INSTITUTO MEXICANO DEL TRANSPORTE

Implementación de un modelo de predicción de tráfico con aprendizaje profundo

José Alejandro Ascencio Laguna
Carlos Daniel Martner Peyrelongue
Agustín Bustos Rosales

Publicación Técnica No. 707
Sanfandila, Qro.
2022

ISSN 0188-7297

Esta investigación fue realizada en la Coordinación de Transporte Integrado y Logística del Instituto Mexicano del Transporte, por el Mtro. José Alejandro Ascencio Laguna, el Dr. Carlos Daniel Martner Peyrelongue y el Dr. Agustín Bustos Rosales.

Esta investigación es el producto final del proyecto de investigación TI 03/22 Modelo de predicción de volumen de tráfico a través de técnicas de aprendizaje profundo.

Contenido

Página

Índice de figuras

Índice de tablas

Sinopsis

Dada la demanda en la automatización de procesos, el acceso a grandes cantidades de información y la disponibilidad de tecnologías emergentes orientadas a la gestión del tránsito, es esencial el desarrollo de algoritmos que permitan predecir de manera oportuna el tráfico. El aprendizaje profundo tiene la capacidad de gestionar el espacio temporal, ampliando su memoria para recordar y olvidar sucesos importantes en el pasado, esto permitirá generar pronósticos robustos que contemplen periodos de tiempo y por consecuente la posibilidad de planear acciones que minimicen los efectos negativos de la congestión vehicular.

El aprendizaje profundo consiste en simular el funcionamiento del sistema nervioso humano, aprender y extraer los patrones de los datos disponibles para producir resultados óptimos, las redes neuronales profundas se organizan en múltiples capas de neuronas que gestionan una característica determinada, y entre más profundas sean (mayor número de capas); las características de las características, lo que le da sentido a su denominación y a la gran capacidad para comprender las cosas del mundo real.

Esta publicación describe una metodología de aprendizaje profundo para predecir el tráfico, evaluación de los resultados y descripción de su utilidad en un entorno de producción de ingeniería software y gestión en tiempo real.

Abstract

Given the demand for process automation, access to large amounts of information and the availability of emerging technologies aimed at traffic management, the development of algorithms that prevent traffic in a timely manner is essential. Deep learning has the ability to manage the temporal space, expanding its memory to remember and forget important events in the past, this will allow the generation of robust forecasts that contemplate periods of time and therefore the possibility of planning actions that minimize the negative effects of the vehicular congestion.

Deep learning consists of simulating the functioning of the human nervous system, learning and extracting patterns from the available data to produce optimal results, deep neural networks are organized in multiple layers of neurons that manage a certain characteristic, and the deeper they are (higher number of layers); the characteristics of the characteristics, which gives meaning to its name and the great ability to understand things in the real world.

This publication describes a deep learning methodology for predicting traffic, evaluating the results, and describing its usefulness in a real-time management and software engineering production environment.

Resumen ejecutivo

En el presente estudio se describen los fundamentos del aprendizaje profundo, su estructura, sus capacidades y limitantes, concentrándose especialmente en las redes neuronales recurrentes, que a diferencia de las redes neuronales tradiciones que tienen ciertas limitantes para capturar las relaciones espaciales entre los datos, estas no tratan cada entrada de manera independiente y por consecuente generan relaciones secuenciales como sucede en el mundo real, este tipo de técnica ha demostrado éxito en el análisis de texto y series de tiempo.

Una de las características de las redes neuronales recurrentes (RNR) y que se considera como desventaja, es que las modificaciones a los pesos en el proceso de propagación hacía atrás (*backpropagation*) tienden a actualizarlos a cero, para este inconveniente han surgido variaciones de las mismas, las más populares son las redes con capas de memoria a corto plazo (*Long Short Term Memory* LSTM) y su principal variación que son las unidades recurrentes con compuertas (*Gated Recurrent Unity* GRU), donde la segunda presenta ventajas en velocidad debido a que no maneja un estado independiente de la activación tradicional y modifica directamente la de capa anterior, por lo tanto, se puede concluir que LSTM tiene más posibilidad de mejorar el ajuste pero en tiempo mayores a GRU.

Con base en la importancia de las redes neuronales recurrentes, este documento también describe la manera de usar LSTM para la predicción de volúmenes de tráfico, la estructura y el tratamiento de los datos de entrenamiento, cabe mencionar que como todo procesamiento de aprendizaje profundo o en inglés *Machine Learning* (ML) se usará una porción de los datos para la evaluación, con la cual será posible generar una matriz de confusión y otras herramientas para comprobar el ajuste de la red, su precisión y el factor de error.

Se ha demostrado en la literatura que es posible predecir el tráfico a través de técnicas de ML, por tales motivos esta publicación describe como trabajos a futuro la utilidad del pronóstico vehicular en un entorno de producción de ingeniería de software para gestionar los datos generados en tiempo real.

La estructura de este estudio consiste en documentar los fundamentos del aprendizaje profundo como primer capítulo, elementos de las RNR como segundo, la implementación de LSTM con su evaluación como tercero y, por último, en el cuarto capítulo, los trabajos futuros como descripción de entornos en tiempo real.

Introducción

La línea de investigación de sistemas inteligentes o nuevas tecnologías del transporte de la Coordinación de Transporte Integrado y Logística del Instituto Mexicano del Transporte, busca la combinación de las técnicas informáticas y de telecomunicación para resolver problemas actuales en el transporte, algunos de los estudios más importantes en dicha área se concentran en la implementación de tecnologías ITS (*Intelligent Transport System*), simulación, procesamiento en tiempo real y evaluación de nuevas tecnologías.

No cabe duda que existe un incremento del volumen de tráfico de manera exponencial, particularmente en vías urbanas, esto combinado con las necesidades actuales del usuario y de la dependencia de transportase de una manera eficiente y eficaz, causa problemas considerables no sólo en el medio ambiente, sino que también a nivel social. No siempre las soluciones tradicionales como ampliaciones y modificaciones estructurales son las óptimas y mucho menos las más factibles (Panduro & Martín, 2017), la innovación tecnológica dota de inteligencia artificial y de algoritmos robustos que permiten la reducción de costos, la generación de procesos reutilizables y métodos reproducibles en entornos parecidos.

Una de las principales técnicas de la inteligencia artificial es el aprendizaje profundo o en ingles *Deep Learning*, este es un tipo de aprendizaje automático que se ha usado ampliamente para este tipo de problemas, muestra grandes ventajas en el reconocimiento de patrones y la clasificación (Jia et al., 2017), existen estudios pioneros que utilizan este tipo de tecnologías, por ejemplo, el de (W. Huang et al., 2014), que proponen una arquitectura de aprendizaje multitarea con redes de creencia profunda *Deep Belief Network* (DBN) para la predicción de flujo de tráfico; y el de (Lv et al., 2015), donde utiliza modelos de apilado codificador Stacked *Autoencoder Model* (SAE) para una predicción espacial y de correlación temporal. También es posible encontrar estudios actuales en el uso de modelos de este tipo, por ejemplo el de (Abduljabbar et al., 2021) y (Wang et al., 2021), que utilizan aprendizaje LSTM unidireccional y/o bidireccional para la predicción espacio-temporal de flujos en autopistas.

Con base en el texto anterior, es posible concluir que el uso de aprendizaje profundo generó y sigue generando buenos resultados para la predicción de volúmenes de tráfico, es por eso que se siguen estudiando sus variantes y su evolución.

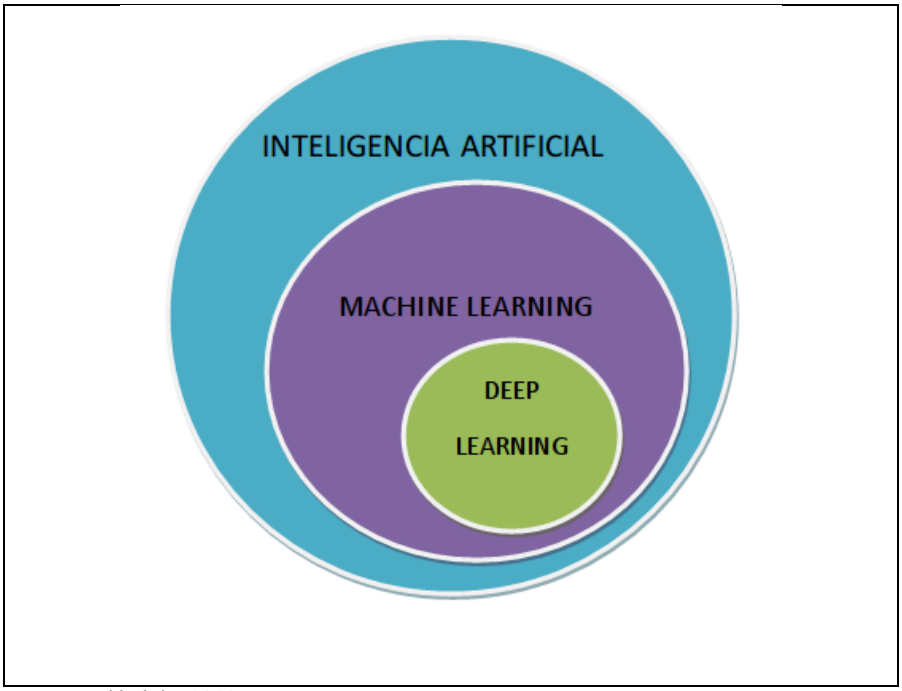
Siendo uno de las estrategias prioritarias del Programa Sectorial de Comunicaciones y Transportes 2020-2024, específicamente la estrategia 2.6.5, “Fomentar la implementación de sistemas inteligentes de transporte”, alineado al quinto objetivo del Programa para un Gobierno Cercano y Moderno (PGCM),

“Establecer una Estrategia Digital Nacional que acelere la inserción de México en la Sociedad de la Información y del Conocimiento” (Secretaría de Comunicaciones y Transportes, 2020), este proyecto pretende generar una revisión bibliográfica de los métodos *Deep Learning* para la predicción de volúmenes de tráfico, plantear una metodología y presentar una experimentación de solución.

1. Aprendizaje profundo

El aprendizaje profundo o mejor conocido en inglés como *Deep Learning* (DL), es el área de investigación más popular de la inteligencia artificial (IA), nuevas investigaciones y con resultados sorprendentes en los campos de procesamiento del lenguaje natural, predicción y visión artificial se basan en modelos DL (github.io, 2022).

Se suele utilizar de manera indistinta los conceptos de ML, IA o DL, sin embargo, y aunque relacionadas existen sus diferencias que son bastante palpables, IA es contexto más general que engloba a ML y DL, además de contemplar otros grandes tópicos, tales como, algoritmos de búsqueda, razonamiento simbólico y lógico, la propia estadística, agentes inteligentes, etc. En la Figura 1.1 se observa dicha representación.



Fuente: github.io, 2022

Figura 1.1 El nivel de DL.

DL usa la estructura jerárquica de las redes neuronales artificiales, similar a la del cerebro humano, con nodos de neuronas conectadas como una red o una telaraña, lo que permite tratar el análisis de los datos de manera no lineal, DL aprende a realizar principalmente regresiones y clasificación, por ejemplo, reconocer imágenes, texto o sonido. Los modelos en DL son entrenados con un gran conjunto

de datos etiquetados y arquitecturas de redes con muchas capas, lo que permite darle profundidad neuronal (Centeno, 2019).

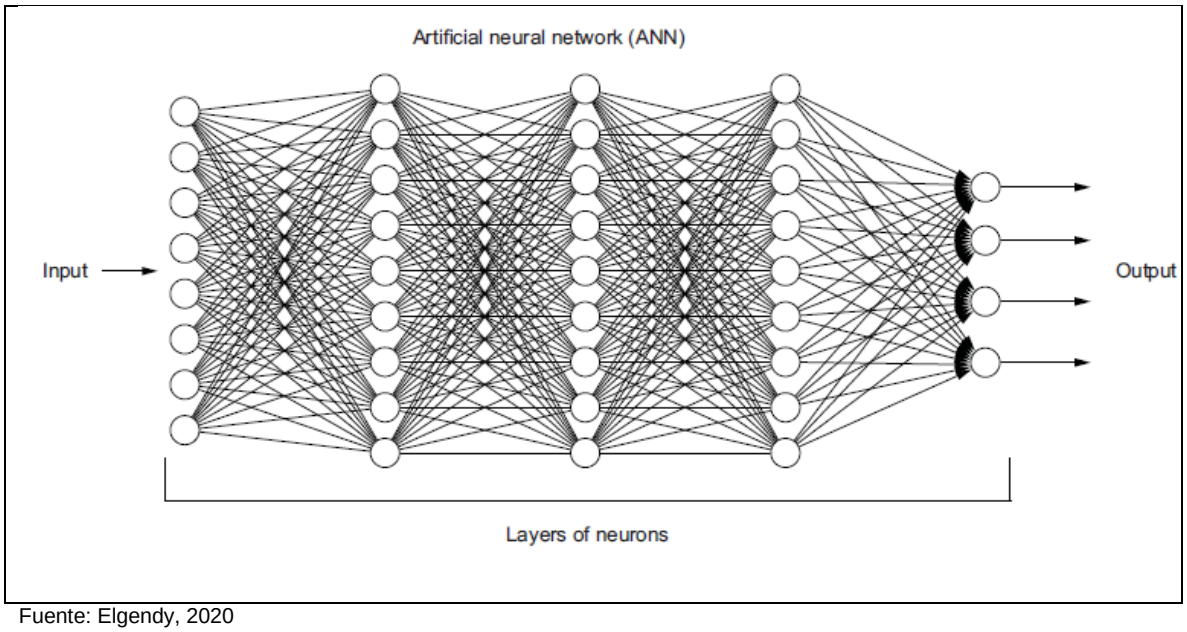
Algunas de las principales restricciones del DL son:

- Requiere de grandes cantidades de datos etiquetados para su buen funcionamiento.
- Requiere gran potencia de cálculo y principalmente en el contexto de visión artificial (VA), el uso de unidades de procesamiento gráfico (*Graphics Processing Unity* GPU) para computo en paralelo y para problemas muy grandes la combinación con clústeres.

Los dos principales conceptos que permitirán comprender DL son redes neuronales artificiales (RNA's) y la propagación hacía atrás.

1.1 Redes neuronales artificiales

Las RNA's o en inglés ANN (*Artificial Neuronal Network*) están compuestas por muchas neuronas estructuradas en capas que realizan algún tipo de cálculo y que tienen por objetivo predecir una salida, en la Figura 1.2 podemos observar un ejemplo genérico de dicha arquitectura.



Fuente: Elgendy, 2020

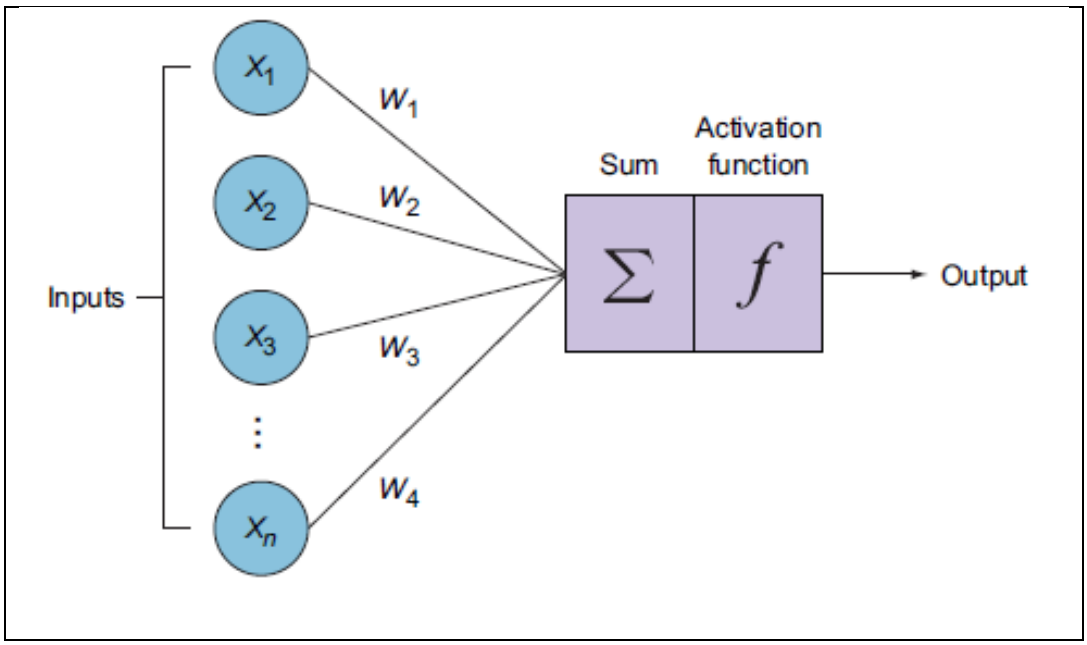
Figura 1.2 Perceptrón Multicapa.

Como se puede observar en la Figura 1.2, se tiene una capa de entrada, un conjunto de capas ocultas y una de capa de salida, todas compuestas por neuronas. Los conceptos RNA's y perceptrón multicapa (*Multilayer Perceptron MLP*) son usados indistintamente, y es obvio porque las RNA's consisten en una estructura de perceptrones multicapa.

El perceptrón es el elemento más simple de la red neuronal, consiste en una única neurona y funciona de manera similar a la neurona biológica, recibe señales eléctricas en varias cantidades, y cuando excede un umbral se dispara una señal de salida a través de su sinapsis, dicha salida alimenta a otra neurona y así sucesivamente.

La neurona artificial realiza especialmente dos funciones consecutivas, la sumatoria de pesos, que representa la fuerza total de las señales de entrada y aplica una función de paso para determinar si la salida se dispara (valor 1) o no (valor 0) con respeto a un umbral determinado.

Cada característica de entrada x_1 tiene asignado su propio peso w_1 que refleja la importancia del proceso de toma de decisión, las entradas con mayor peso tienen mayor efecto en la salida, por lo tanto, sí el peso es alto, se amplifica la señal de entrada, si es bajo, la disminuye. Tradicionalmente las redes neuronales representan los pesos con líneas o bordes desde el nodo de entrada al del perceptrón. Por ejemplo, si se está prediciendo el precio de una casa con base en un conjunto de características, tales como: tamaño, vecindario y número de habitaciones, entonces hay tres características de entrada (x_1, x_2 y x_3), cada una tiene diferente valor de peso que representa su efecto en la decisión final. Supóngase que el tamaño de la casa tiene doble efecto en el precio comparado con el vecindario, y el vecindario doble efecto con respecto al número de habitaciones, por lo tanto, tendremos pesos como 8,4 y 2 respectivamente.



Fuente: Elgendy, 2020

Figura 1.3 Elementos de un Perceptrón Multicapa.

En la Figura 1.3 se representan los principales elementos de un MLP, su descripción se indica a continuación:

- **Vector de entrada (*inputs*)**. El vector de entrada alimenta a la neurona, usualmente denotado por X mayúscula que representa las entradas $x_1, x_2, \dots, x_n$.
- **Vector de pesos (*W*)**. Cada x_i tiene asignado un valor de peso w_i que representa su importancia para distinguirse entre otras entradas.
- **Funciones de las neuronas (*Activation function & Sum*)**. Los cálculos realizados dentro de la neurona modulan la señal de entrada: la suma ponderada de pesos y la función de activación.
- **Salida**. La salida es 0 o 1 (otras funciones de activación podrían producir salidas flotantes), por lo tanto, el nodo de salida representa la predicción del perceptrón.

La suma ponderada de pesos, también conocida como combinación lineal, consiste en la suma de todas las entradas multiplicada por sus pesos, a la cual se agrega el termino sesgo o en inglés *bias*. Esta función produce una línea recta representada por la siguiente función.

$$z = \sum x_i \cdot w_i + b$$

$$z = x_1 \cdot w_1 + x_2 \cdot w_2 + x_3 \cdot w_3 + \cdots + x_n \cdot w_n + b$$

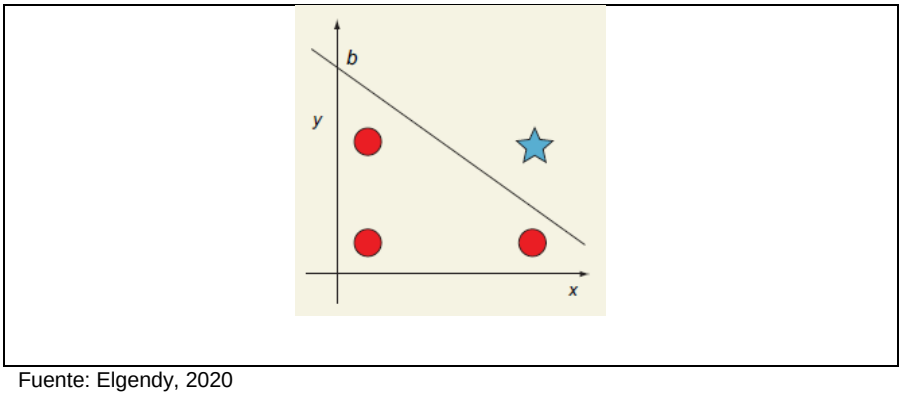
Donde,

b Es el sesgo.

x_i Es la entrada.

w_i Es el peso de la entrada.

El sesgo permite mover la línea arriba y abajo en el eje de las y para ajustar de mejor manera la predicción con los datos, sin este sesgo la línea siempre pasará a través del punto de origen (0,0) y conseguirá un ajuste poco robusto.



Fuente: Elgendy, 2020

Figura 1.4 El sesgo.

Se puede observar en la Figura 1.4 que la ecuación de la línea recta está dada por: $y = mx + b$, donde b es la intersección en el eje de las y, para ser capaz de definir una línea se requieren dos cosas, la pendiente y un punto de la línea, el sesgo es ese punto en el eje de las y (*y-axis*). Para visualizar la importancia del sesgo, se puede intentar separar los círculos de las estrellas usando una línea recta que pasa a través del origen (0,0), lo cual no es posible.

La función de activación es la unidad de toma de decisión del cerebro, en RNAs esta toma la misma suma pondera de la entrada y activa la neurona sí es más alta que un cierto umbral. Existen distintos tipos de función de activación, su implementación depende del contexto y tipo de problema. La función de activación más simple usada por el perceptrón es la de paso, esta produce una salida binaria (0 o 1), básicamente consiste en activar la neurona si la suma ponderada es > 0, en caso contrario ≤ 0 no se activa.

Cuando se está construyendo una red neuronal, una de las principales decisiones que se debe tomar es qué función de activación usar, la función de activación se refiere a las funciones de transformación (*transfer functions*) o la eliminación de la linealidad (*nonlinearities*), ya que las funciones de activación convierten la suma de pesos ponderada en un modelo no lineal. Una función de activación toma lugar al final de cada perceptrón que decide si se activa o no la neurona.

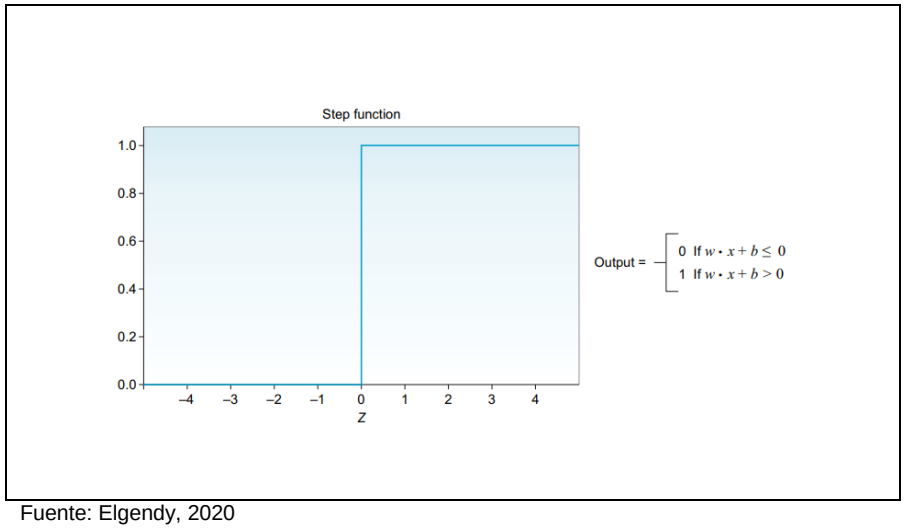
El propósito de la función de activación es introducir no linealidad en la red, sin ellas es como si el perceptrón multicapa fuera un único perceptrón, sin importar cuantas capas agreguemos. La función de activación requiere restringir el valor de salida a un cierto valor finito para poder gestionarse de una manera sencilla.

Hay una infinidad de funciones de activación, de echo en los últimos años se ha visto mucho progreso en la creación de estados del arte sobre la función de activación, sin embargo, todavía se considera que hay pocas funciones de activación con respecto a las necesidades de ajuste en los tipos de conjuntos de datos (Elgendy, 2020). A continuación, los más populares:

- Función de trasferencia lineal (*Linear transfer function*). También llamada función de identidad (*identity function*), indica que la función pasa una señal sin cambios, en términos prácticos la salida será igual que la entrada.

$$activation(z) = z = wx + b$$

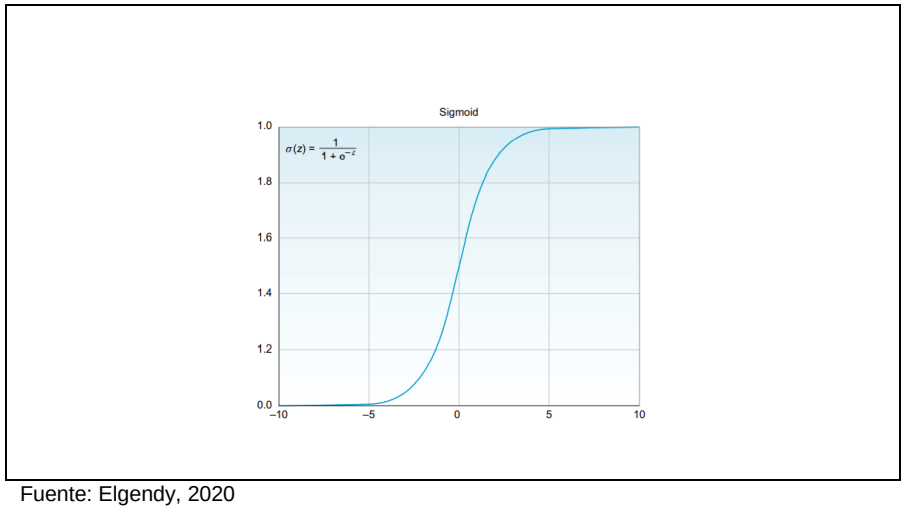
- Función de escalón lateral (*Heaviside step function* o *binary classifier*). Es la función de paso y genera una salida binaria.



Fuente: Elgendy, 2020

Figura 1.5 Clasificación binaria.

- Función logística (*Sigmoid/logistic function*). Esta es una de las más comunes, a menudo se usa en clasificadores binarios para predecir la probabilidad de que un ejemplo pertenezca a alguna de las dos clases. Esta función limita todos los valores a una probabilidad de entre 0 y 1, lo cual reduce significativamente los valores o *outliers (ejemplos anormales o valores atípicos)* en los datos sin removerlos. Dicha función de activación convierte las variables continuas en un rango de $-\infty\ to\ \infty$ en una simple probabilidad de entre 0 y 1. También, es denominada curva de forma *s* (en inglés *S-shape curve*) porque cuando se grafica produce una curva de ese tipo, a diferencia de la función de paso que usa una respuesta discreta, esta produce una probabilidad de paso.



Fuente: Elgendy, 2020

Figura 1.6 Sigmoid.

- Función Softmax (*Softmax function*). Es una generalización de la función *sigmoide* y se usa para obtener una clasificación de probabilidades cuando se tiene más de dos clases, fuerza a que las salidas de una red neuronal

sumen 1, comúnmente es usada para problemas de DL donde se requiere predecir una sola clase entre muchas opciones (más de dos). A continuación, la ecuación:

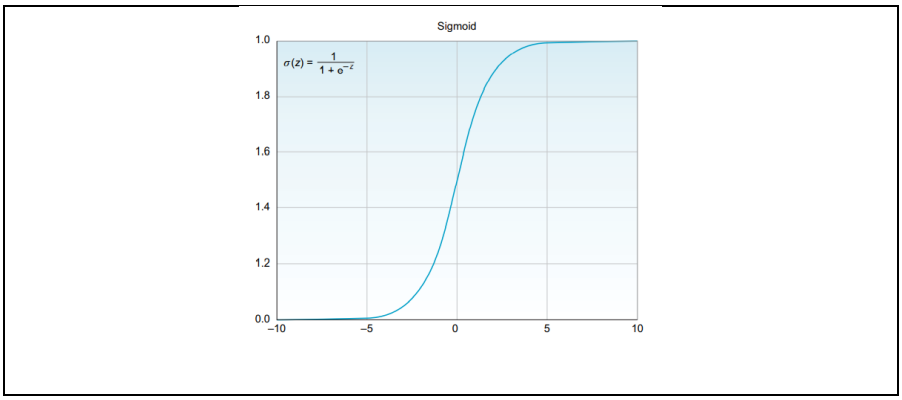
$$\sigma(x_j) = \frac{e^{x_j}}{\sum_i e^{x_i}}$$

donde x son los elementos del vector de entrada y e es la función exponencial.

- Función de tangente hiperbólica (*Hyperbolic tangent function tanh*). Es una versión modificada de la función *sigmoide*, ésta en lugar de pasar la señal de valores entre 0 y 1, los aplana a valores entre -1 y 1. Esta función de activación trabaja mejor que la función *sigmoide* en capas ocultas porque tiene el efecto de centrar los datos para que la media sea cercana a cero en lugar de 0.5 lo que facilita un poco al aprendizaje en la siguiente capa. A continuación, el modelo:

$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

donde x son los valores del vector de entrada, $\sinh$ es el seno hiperbólico y $\cosh$ el coseno hiperbólico.



Fuente: Elgendy, 2020

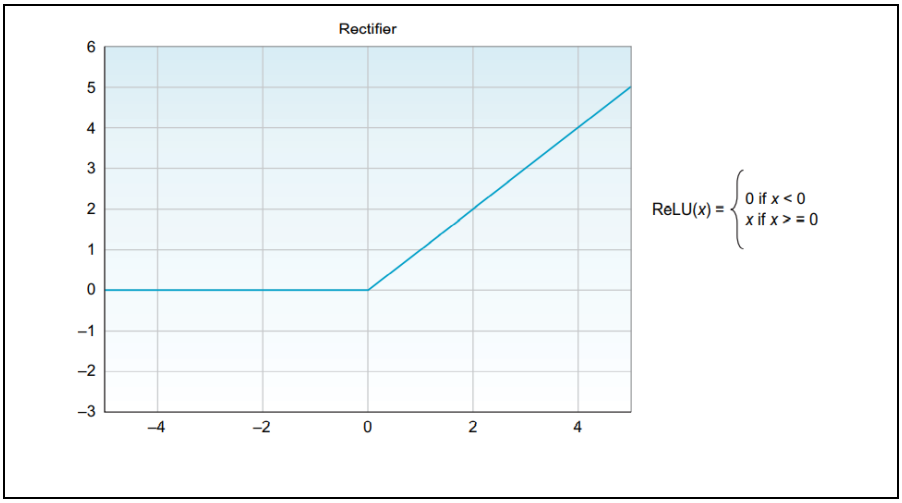
Figura 1.7 Tanh.

Uno de los inconvenientes de las funciones *sigmoide* y tangente hiperbólica es que, si z es muy grande o muy pequeño, entonces el gradiente (la derivada) es muy pequeño (muy cercano a cero), lo que hará que el descenso del gradiente se alente.

- Unidad lineal rectificada (*Rectified linear unit ReLU*). Esta función de activación activa una neurona sólo si la entrada está por encima de cero. Si la entrada es por debajo de cero, la salida siempre será cero, pero cuando la entrada es mayor que cero, se tiene una relación lineal con la variable de salida. A continuación, la función:

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

Donde x es el vector de los valores de entrada y max es la función para obtener el máximo valor. Es considerada la función de activación en el estado del arte porque trabaja bien en muchas situaciones diferentes y tiende a entrenar mejor que las funciones *sigmoide* y tangente hiperbólica en capas ocultas.



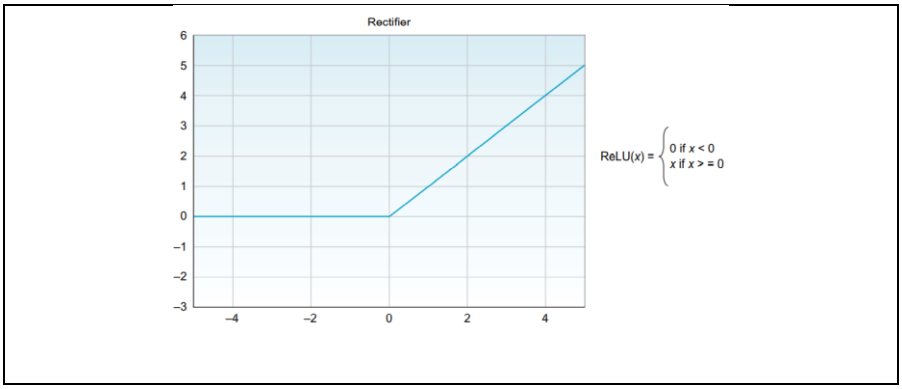
Fuente: Elgendy, 2020

Figura 1.8 ReLU.

- Unidad lineal rectificada defectuosa (*Leaky ReLU*). *Una de las desventajas de la* activación *ReLU es que la* derivada es cero cuando (x) es enegativo. *Leaky ReLU* es una variación de *ReLU* que busca mitigar dicha desventaja, en vez de tener una función que sea igual a cero, cuando $x < 0$, *Leaky ReLU* introduce un pequeño valor negativo (alrededor de 0.01). Usualmente trabaja mejor que la función *ReLU* sin embargo no es usada mucho en la práctica.

$$f(x) = \max(0.01x, x)$$

El valor no necesariamente debe ser 0.01, de hecho, los desarrolladores suelen usar este parámetro como hiperparámetro a calibrar.



Fuente: Elgendy, 2020

Figura 1.9 Leaky ReLU.

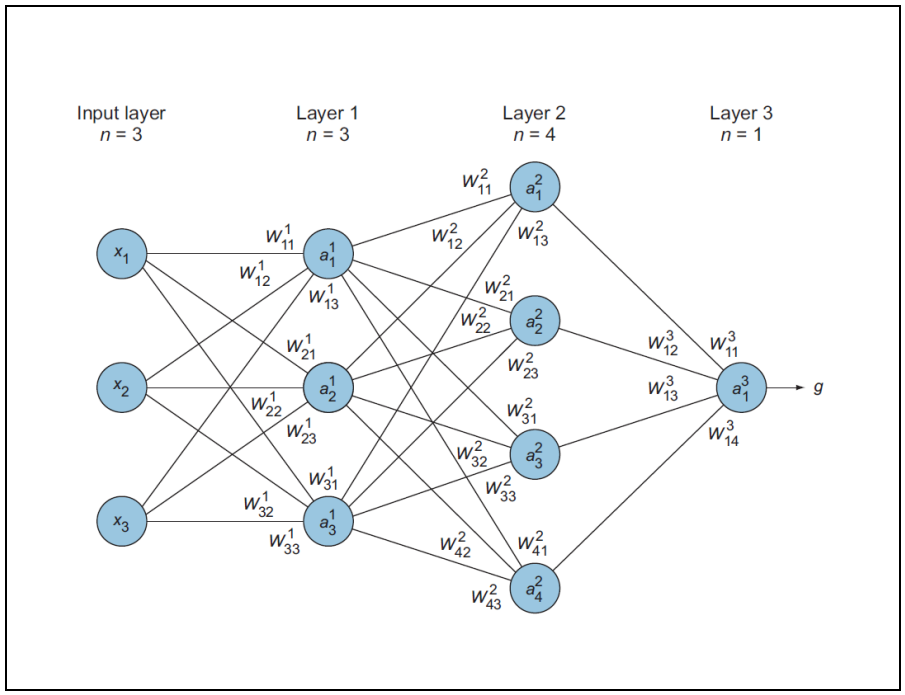
1.2 Retroalimentación (*Feedforward*)

El proceso de calcular la combinación lineal y aplicar la función de activación se le denomina retroalimentación o en inglés *feedforward*, este implica la dirección de avance en la información que fluye desde la capa de entrada a través de las capas ocultas hasta llegar a la capa de salida.

Feedforward ocurre a través de la implementación consecutiva de dos funciones:

- Suma ponderada.
- Función de activación.

En la siguiente figura se observa un ejemplo de una RNA de tres capas:



Fuente: Elgendy, 2020

Figura 1.10 RNA tres capas.

Los componentes de la RNA representados en la figura 1.10 son:

- **Capas o en inglés *Layers***. Consiste en una entrada con tres características y tres capas ocultas con 3,4 y 1 neuronas.
- **Pesos y sesgos o en inglés *Weights y biases*** $(\boldsymbol{w}, \boldsymbol{b})$. Los pesos son asignados de manera aleatoria y son denotados por $W_{ab}^{(n)}$, donde n indica el número de la capa y ab el peso conectado desde la neurona a en la capa (n) con la neurona b en la capa anterior $(n-1)$. Por ejemplo, $W_{23}^{(2)}$ es el peso que conecta el segundo nodo en la capa 2 con el tercer nodo en la capa 1, es importante mencionar que las denotaciones en la literatura son variadas, sin embargo, esta es la más utilizada. Los sesgos se tratan de manera similar

a los pesos porque se inicializan aleatoriamente y sus valores se aprenden durante el entrenamiento, por lo tanto, estos se representan con la misma denotación.

- ***Función de activación o en inglés Activación function*** $\boldsymbol{\sigma(x)}$. En este ejemplo se usa la función *sigmoide* como la función de activación.
- **Valores del nodo o en inglés *Node values*** $(\boldsymbol{a})$. Se calcula la suma ponderada, aplicamos la función de activación y asignamos el valor al nodo a_m^n, donde n es el número de la capa y m es el número de la neurona, por ejemplo, a_2^3 es el nodo 2 de la capa 3.

Los cálculos *feedforward* del ejemplo se presentan en la figura 1.10:

- *Eel cálculo de la capa 1:*

$$\boldsymbol{a_1^{(1)} = \sigma(w_{11}^{(1)} x_1 + w_{12}^{(1)} x_2 + w_{13}^{(1)} x_3)}$$
$$\boldsymbol{a_2^{(1)} = \sigma(w_{21}^{(1)} x_1 + w_{22}^{(1)} x_2 + w_{23}^{(1)} x_3)}$$
$$\boldsymbol{a_3^{(1)} = \sigma(w_{31}^{(1)} x_1 + w_{32}^{(1)} x_2 + w_{33}^{(1)} x_3)}$$

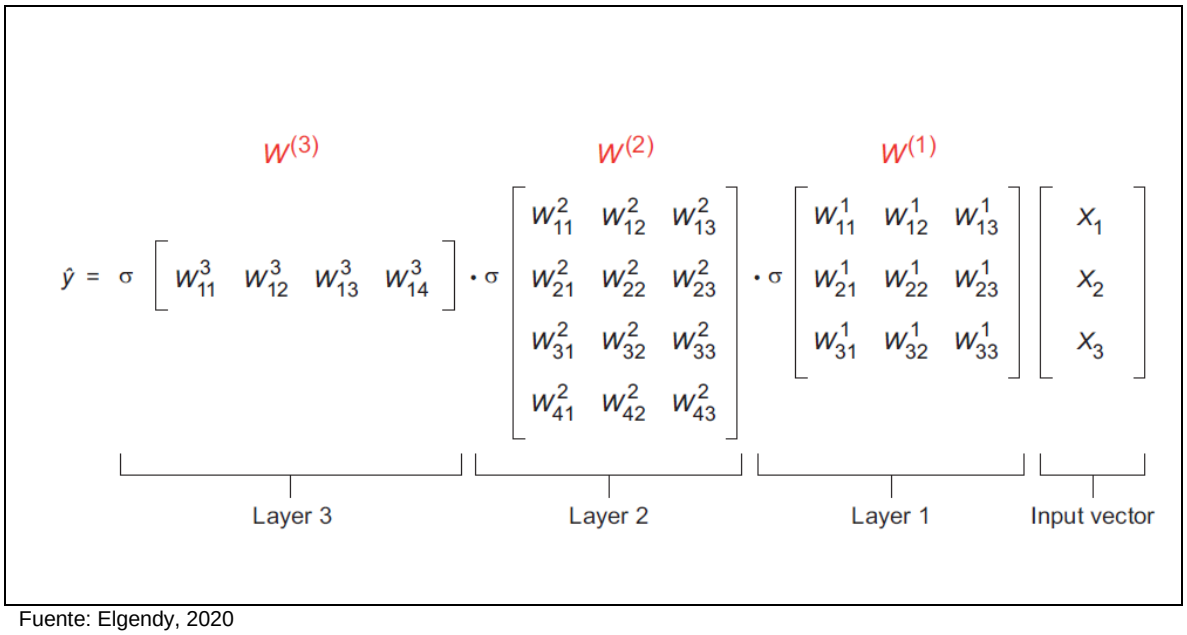
- *El cálculo de la capa 2:*

$$\boldsymbol{a_1^{(2)}, a_2^{(2)}, a_3^{(2)} \; y \; a_4^{(2)}}$$

- *El cálculo de la capa 3:*

$$\boldsymbol{\hat{y} = a_1^{(3)} = \sigma(w_{11}^{(3)} a_1^{(2)} + w_{12}^{(3)} a_2^{(2)} + w_{13}^{(3)} a_3^{(2)} + w_{14}^{(3)} a_4^{(2)})}$$

Se observa que una red pequeña requiere un gran número de cálculos repetitivos, sin embargo, existen procesos matriciales que reducen la complejidad operativa y por consecuente la reducción en el tiempo de ejecución, como se aprecia en la Figura 1.11.



Fuente: Elgendy, 2020

Figura 1.11 Manejo eficiente de matrices.

1.3 Función de error

¿Cómo se evalúa la predicción que la RNA produce? Más importante aún, ¿Cómo saber que tan cerca está la predicción de la respuesta correcta?

La respuesta es la función de error, la selección de la función de error es otro aspecto importante para el diseño de la red neuronal; estas funciones también son denominadas funciones de costo (*cost functions*) o funciones de perdida (*loss functions*), denominaciones usadas en la bibliografía de DL (Elgendy, 2020).

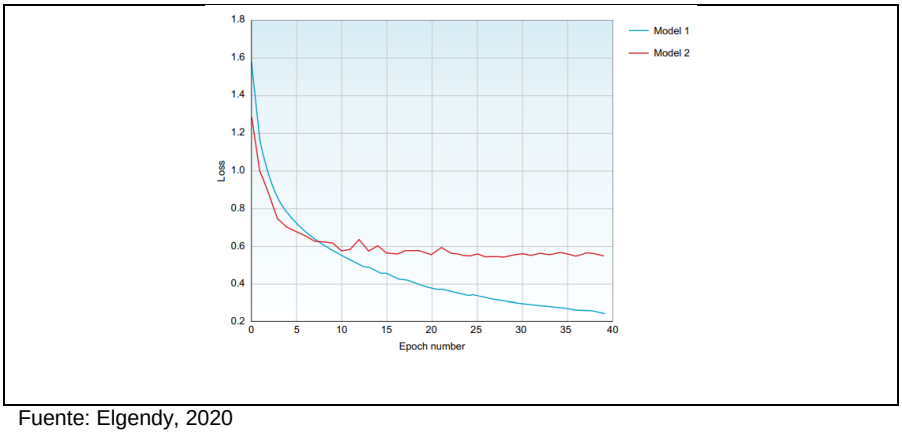
La función de error es una medida que indica que tanto falla la predicción de la red neuronal con respecto a la salida (*the label*). Esta cuantifica que tan cerca se está de la solución correcta, por ejemplo, si tiene una perdida alta, entonces el modelo no está teniendo un buen desempeño, si la perdida es muy pequeña entonces el modelo está haciendo un buen trabajo.

Con base en lo anterior se puede decir que, si se tiene una perdida alta, entonces el modelo requiere ser entrenado para incrementar su precisión.

Calcular el error es un problema de optimización, los problemas de optimización se concentran en definir una función de error e intentar optimizar los parámetros con el objetivo de minimizarlo.

En ML el problema de optimización consiste en encontrar un óptimo en los pesos de las variables, minimizando el error tanto como se pueda, si no hay forma de saber que tan cerca estamos del resultado, entonces no es posible saber que modificar en la siguiente iteración.

Diferentes funciones de error resultan en diferentes errores en la misma predicción. En la Figura 1.12 se representa como el error es optimizado con diferentes funciones de error sobre los mismos datos, podemos ver que el modelo 1 presenta mejor desempeño.



Fuente: Elgendy, 2020

Figura 1.12 Diferentes funciones de error.

Las funciones de error más comunes son:

- **Error cuadrático medio o en *inglés Mean Squared Error (MSE)*** que es usado normalmente para problemas de regresión. Requiere una salida de valor real (como el precio de la casa), en lugar de sólo comparar la salida de la predicción con la etiqueta $(\hat{y}_i - y_i)$, el error es elevado al cuadrado y promediado sobre el número de puntos de datos, tal y como se ve en la siguiente ecuación:

$$E(W, b) = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2$$

  Donde,

W	Matriz de pesos
b	Vector de sesgo
N	Número de ejemplos de entrenamiento
$\hat{y}_i$	Predicción de salida
y_i	La salida correcta (objetivo o *target*)
$(\hat{y}_i - y_i)$	Normalmente se le denomina residual

- **Entropía cruzada o en inglés *cross-entropy*** para problemas de clasificación. Es comúnmente usada para problemas de clasificación porque cuantifica la diferencia entre dos distribuciones de probabilidad, por ejemplo, supóngase que para una instancia especifica de entrenamiento, se intenta clasificar a un perro en una imagen sobre tres posibles clases (perros, gatos y peces), la distribución para esta instancia de entrenamiento es la siguiente:

$$P(cat) = 0.0; P(dog) = 1.0; P(fish) = 0.0$$

  Se puede interpretar que la probabilidad de que sea de la clase A es 0%, de la clase 2 100% y de la clase 3 0%.

  El error de la entropía cruzada permite determinar qué tan cerca está la distribución de la predicción, con respecto a la distribución real. Se estima mediante la siguiente fórmula:

$$E(W, b) = -\sum_{i=1}^{m}\hat{y}_i \log(p_i)$$

  Donde,

$\hat{y}_i$	Es la probabilidad del *target*
p_i	Es la probabilidad de la predicción
m	Es el número de clases

Se requiere minimizar el error tanto como se pueda, cero es ideal, el más bajo error resultará en la más alta precisión en la predicción de valores, pero ¿Cómo se minimiza el error?

Supóngase que la entrada $x = 0.3$ y su etiqueta (valor real) es $y = 0.8$, la predicción es calculada de la siguiente manera:

$$\hat{y}_i = w \cdot x = w \cdot 0.3$$

El error en su simple forma es calculado para comparar la predicción de la etiqueta:

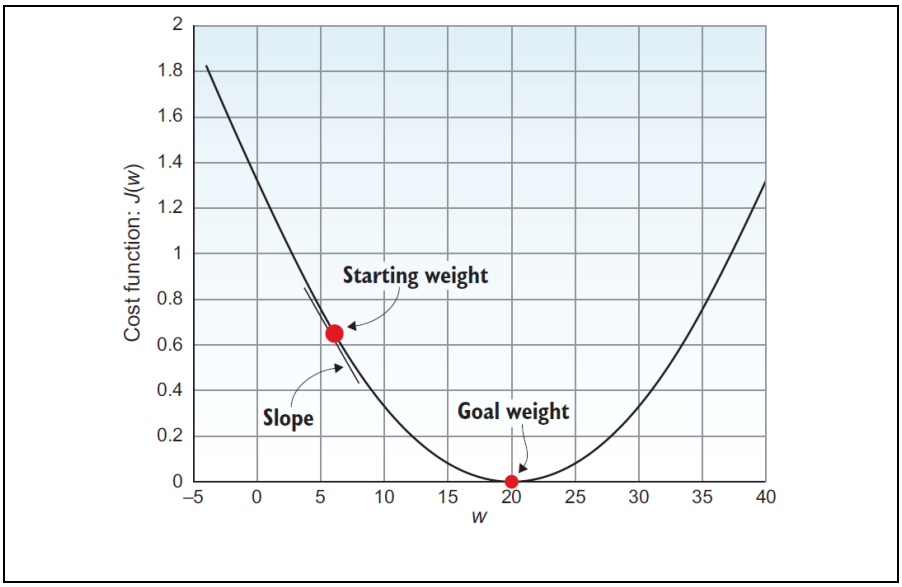
$$erro = |\hat{y} - y|$$

$$= |(w \cdot x) - y|$$

$$= |w \cdot 0.3 - 0.8|$$

Con esto se puede observar que la entrada (x) y la etiqueta (y) son valores fijos, estos nunca cambiarán para ese ejemplo específico, quiere decir que los únicos valores que se pueden cambiar son: el error y el peso, ahora si se desea conseguir el error mínimo, se debe ajustar el peso arriba o abajo para conseguir el óptimo.

El gráfico de la función de error con respecto al peso puede observarse en la Figura 1.13.



Fuente: Elgendy, 2020

Figura 1.13 Función de error.

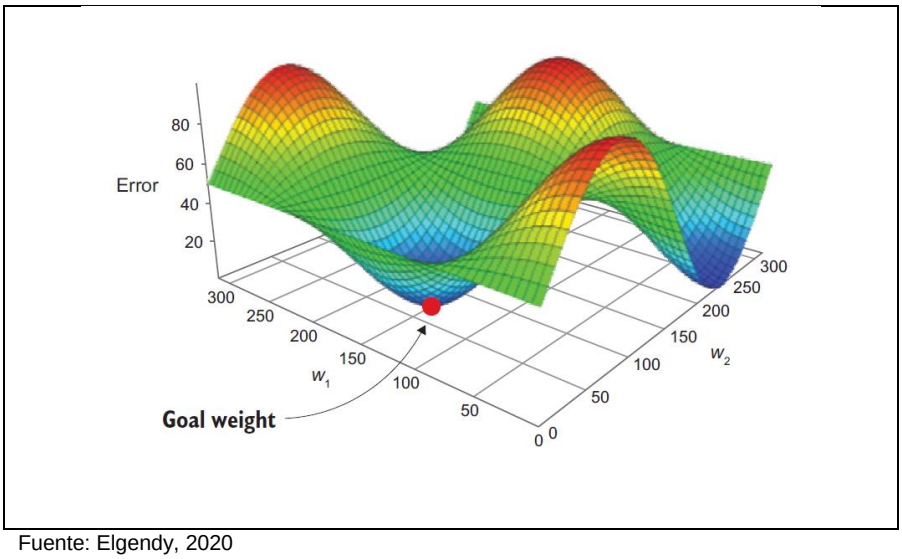
1.4 Algoritmos de optimización

Entrenar una red neuronal implica mostrarle muchos ejemplos a la misma (*trainig dataset*), la red hace predicciones a través de *feedforward* y las compara con las etiquetas reales (*target*) para calcular el error. Finalmente, la red neuronal necesita ajustar los pesos hasta minimizar el error, lo que significa maximizar la precisión

(*accuracy*), por lo tanto, es necesario construir algoritmos que encuentren los pesos óptimos.

En RNAs la función de optimización de error significa actualizar los pesos y los sesgos hasta encontrar pesos óptimos, dicho de otra forma, encontrar el mejor valor de pesos que produzca un mínimo error.

En la Figura 1.12 se observa la forma más simple de la RNA, donde se tiene una única entrada y un único peso, graficando el error (lo que se está intentando minimizar) con respecto al peso, se representa en una curva 2D. Pero si se tuvieran dos pesos y se quisieran graficar todos los posibles valores de dos pesos, se tendría un plano 3D del error (los dos pesos y el error).



Fuente: Elgendy, 2020

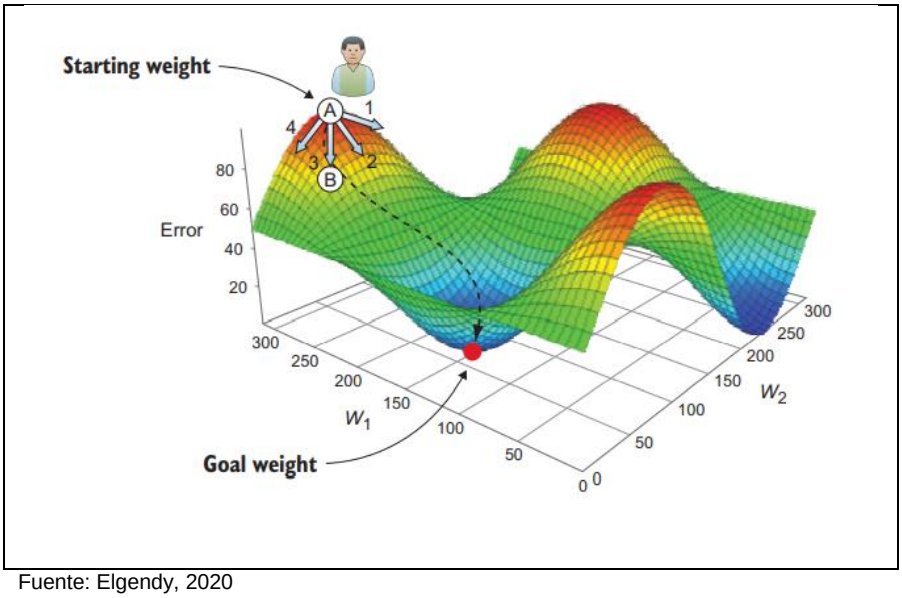
Figura 1.14 Función de error 3D.

El principal objetivo del proceso de optimización es investigar el espacio para encontrar los mejores pesos que logren el error más bajo posible, por cada nuevo atributo se tendría una nueva dimensión, por lo que se podría concluir que encontrar los mínimos locales con fuerza bruta en problemas reales podría ser muy complejo o hasta imposible, por tales motivos se requieren técnicas de optimización, las más populares en DL y basadas en el descenso del gradiente son:

- **Descenso del gradiente o en inglés *Gradient Descent (GD)***. La definición general de un gradiente (también conocido como la derivada), es esa función a la que se le llama pendiente o tasa de cambio de línea, que es la tangente a la curva en cualquier punto dado. En pocas palabras es la pendiente o inclinación de la curva. El descenso del gradiente simple significa actualizar los pesos de manera iterativa para descender la pendiente de la curva del error hasta conseguir un mínimo error. La función de error con respecto a los pesos consiste un punto inicial (un valor de peso), se calcula la derivada de la función de error para conseguir la pendiente (dirección) del próximo paso

y se repite el proceso para dar pasos debajo de la curva hasta encontrar el mínimo error.
Para visualizar cómo funciona el descenso del gradiente, se grafica en un esquema 3D (ver en Figura 1.14) y se pasa por el proceso paso a paso. El peso aleatorio inicial es en un punto A, el objetivo es descender de la cima del error hasta los valores de peso objetivo w_1 y w_2 que producen el valor de error mínimo. Esto se hace dando pasos hacia abajo de la curva hasta obtener ese error mínimo, para tal proceso se requiere:

- La dirección de paso (*gradient*). El proceso del descenso del gradiente consiste en calcular la derivada del error con respeto al peso $\left(\frac{dE}{dw}\right)$, ahora hay una cosa que queda pendiente, el gradiente sólo determina la dirección y se requiere el tamaño de paso, puede ser un pie de paso o 100 pies de paso.
- El tamaño de paso (*learning rate*). Es el tamaño de cada paso en la RNA que hace que se descienda en el error, usualmente es denotado por α. Es uno de los hiperparámetros más importantes que se puede calibrar en el proceso de entrenamiento. Un tamaño de paso grande significa que la RNA aprenderá más rápido (descender de la cima del error a pasos agigantados) y un tamaño de paso pequeño significa un aprendizaje más lento.

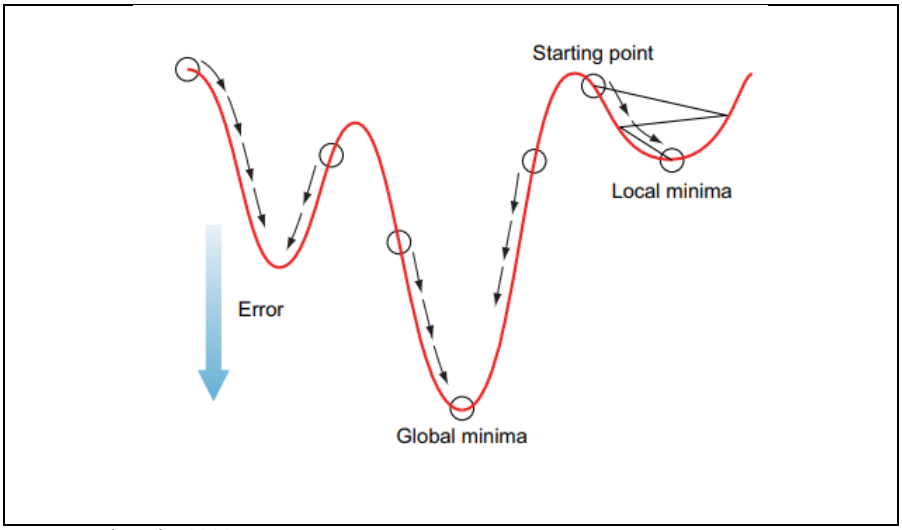


Fuente: Elgendy, 2020

Figura 1.15 Descenso del gradiente.

- **Descenso del gradiente estocástico o en inglés *Stochastic gradient descent (SGD)***. Este algoritmo selecciona de manera aleatoria puntos de dato y va a través del descenso del gradiente con un punto de dato a la vez. Este mecanismo proporciona muchos puntos de partida de diferentes pesos y desciende todas las cimas de error para calcular sus mínimos locales, con base en ello, el valor mínimo de estos mínimos locales será el mínimo global.

Debido a que se da un paso después de calcular el gradiente para todos los datos de entrenamiento en el lote GD, se observa que el camino hacia el error es suave y casi una línea recta. Por el contrario, debido a la naturaleza estocástica (aleatoria) de SGD, se verá que el camino hacia el mínimo de costo global no es directo, pero puede zigzaguear si visualizamos la superficie de costo en un espacio 2D (ver Figura 1.15). Esto se debe a que en SGD, cada iteración intenta ajustarse mejor a un solo ejemplo de entrenamiento, lo que lo hace mucho más rápido, pero no garantiza que cada paso nos lleve hacia abajo en la curva.



Fuente: Elgendy, 2020

Figura 1.16 SGD.

Estocástico es justo una palabra elegante para aleatorio, *el descenso del gradiente estocástico* es probablemente el algoritmo de optimización más usado en ML y DL. Mientras que el descenso del gradiente mide la perdida y calcula el gradiente sobre todo el conjunto de entregamiento para dar paso hacía el mínimo error, SGD elige aleatoriamente una instancia en el conjunto de entrenamiento para cada paso y calcula el gradiente basándose sólo en esa única instancia.

- **Descenso del gradiente por mini-lote o en inglés *Mini-batch Gradient Descent (MB-GD)***. Es una combinación entre BGD y SGD, en vez de ejecutar el gradiente desde un ejemplo (SGD) o todos los ejemplos (GD), se divide el conjunto de datos de entrenamiento en mini-lotes para ejecutar el gradiente (comúnmente un mini-lote es $k = 256$). MB-GD converge en menos iteraciones que BGD porque actualiza los pesos con mayor frecuencia, por otro lado, MB-GD permite usar operaciones vectorizadas, que generalmente producen mejor rendimiento que SGD.

Es importante mencionar que durante los últimos años se ha estudiado el descenso del gradiente, siendo un área muy activa y con muchas variaciones, algunas de las más populares son (Elgendy, 2020):

- *Nesterov accelerated gradient.*
- *RMSprop*
- *Adam*
- *Adagrad.*

1.5 Propagación hacía atrás (*Backpropagation*)

La propagación hacía tras (*Backpropagation)* es el núcleo del aprendizaje en las RNAs, hasta ahora se sabe que entrenar una red neuronal consiste en los siguientes pasos:

- *Feedforward*. Que realiza una combinación lineal (suma ponderada) y aplica la función de activación para conseguir la predicción $(\hat{y})$.

$$\hat{y} = \sigma \cdot W^{(3)} \sigma \cdot W^{(2)} \sigma \cdot W^{(1)} \cdot (x)$$

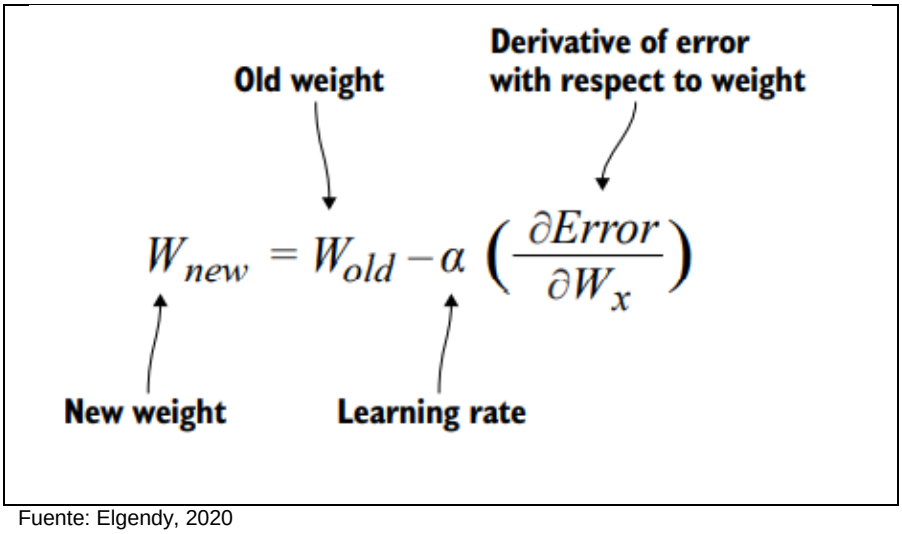
- Comparar la predicción con la etiqueta para calcular la función de error.

$$E(W, b) = \frac{1}{N} \sum_{i=1}^{N} |\hat{y}_i - y_i|$$

- Usar el descenso del gradiente para optimizar el algoritmo de la función de error.

$$\Delta w_i = -\propto \frac{dE}{dw_i}$$

- *Backpropagation* de Δw a través de la red actualiza los pesos (ver figura 1.15).



Fuente: Elgendy, 2020

Figura 1.17 Denotación *Backpropagation*.

Donde el nuevo peso es la resta entre el viejo peso y el valor del descenso del gradiente

Backpropagation también denominada paso hacía tras (*backward pass)*, significa propagar la derivada del error con respecto a cada peso en específico, desde la última capa hasta la primera para ajustar los pesos. Para propagar Δw desde la predicción del nodo $(\hat{y})$ en todos los caminos a través de las capas ocultas y hasta llegar a la capa de entrada, los pesos son actualizados:

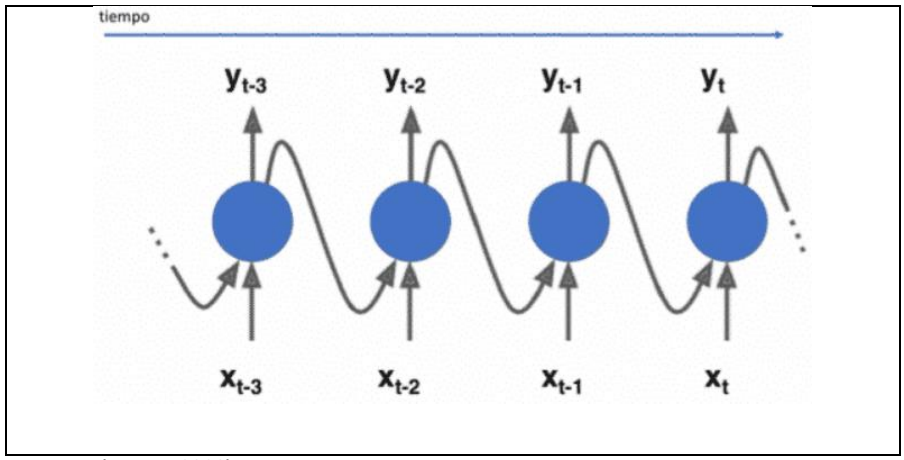
$$(w_{next-step} = w_{current} + \Delta w)$$

Esto llevaría el error un paso más abajo en la cima de errores, el ciclo comienza de nuevo para actualizar los pesos y tomar el error en otro paso más abajo, esto hasta conseguir el mínimo error.

2. Redes neuronales recurrentes

Las redes neuronales recurrentes (RNR), en inglés *Recurrent Neuronal Network*, son una clase de RNA's que gestionan específicamente las series temporales y permiten administrar la dimensión de tiempo. En procesamiento neuronal artificial tradicional la función de activación se ejecuta en una única dirección (*feedforward*), lo cual quiere decir que no toma en cuenta valores previos, en cambio las RNR's emplea conexiones entre nodos que apuntan hacia atrás en el tiempo, algo parecido a una retroalimentación entre neuronas y sus capas (Torres, 2020).

La neurona recurrente en cada instante de tiempo (*timestep*) recibe la entrada X y la salida Y de la capa anterior para generar su salida, en la Figura 2.1 se puede observar su ejecución genérica.



Fuente: (Torres, 2020)

Figura 2.1 Neurona Recurrente.

Se puede decir que cada neurona recibe dos entradas en cada instante de tiempo, la entrada de la capa anterior y la salida del instante de tiempo anterior de la misma capa, la gestión de los dos conjuntos de parámetros mencionados se denota a continuación:

$$y_t = f(W \cdot X_t + U \cdot y_{t-1} + b)$$

Donde $X = (x_1, \dots, x_T)$ es la secuencia de entrada en la capa anterior, W es la matriz de pesos y b el sesgo en las capas anteriores, para gestionar la recurrencia las RNR's extienden la funcionalidad con una conexión en el tiempo, donde U es la matriz de pesos en el instante $t-1$ y que también es entrenado con el proceso de *backpropagation* junto con W y b (Torres, 2020).

Teóricamente para la gestión de la recurrencia en una RNR se requiere principalmente dos elementos, la celdas de memoria *Memoy cell* y *backpropagation* a través del tiempo, el primero de ellos almacena un estado a través del tiempo, lo que hace que este tipo de soluciones sean adecuadas para problemas de aprendizaje automático con datos secuenciales, su memoria permite recordar información sobre las entradas que recibieron y ser eficientes en la predicción de lo que vendrá después, el segundo elemento consiste en calcular el error en un determinado instante de tiempo y que depende del anterior, este es propagado hacía a tras desde el último hasta el primer instante, lo cual admite un ajuste que considera la trascendencia de los datos.

Los dos principales problemas con las RNR son los gradientes explosivos (*Exploding Gradients)* y fugas de gradiente (*Vanishing Gradients)*, en el primero se asignan importancias exageradamente altas a los pesos sin razones importantes, lo cual afectará al resultado final del entrenamiento, una simple solución a esto es el proceso a prueba y error (*ajuste o calibración*) del factor de aprendizaje (*learning rate lr)* para reducir los gradientes, por otro lado, el segundo implica gradientes demasiado pequeños lo cual conduce a que la red deje de aprender. Es importante mencionar que el gradiente es una derivada parcial con respeto a las entradas y mide cuánto cambia la salida de una función al hacer ajustes a dichas entradas, este parámetro permite también conocer hacia donde debe hacerse el ajuste para minimizar el error, se refiere al aprendizaje, y se puede observar que para salir de un gradiente explosivo puede llevar a entrar a un gradiente de fuga, y aunque el factor de aprendizaje puede sacarnos de uno, se ha desarrollado un proceso bastante eficiente denominado unidades de puerta o en inglés *gate units*, dicha técnica se revisará en el siguiente capítulo con el modelo LSTM.

3. LSTM para la predicción de tráfico y su evaluación.

Las redes neuronales LSTM son un tipo de RNR's con la particularidad de que las actualizaciones en las capas ocultas son reemplazadas por celdas de memoria de propósito específico, lo que les permite encontrar y explotar grandes rangos de dependencia en los datos (Z. Huang et al., 2015).

La memoria en las capas LSTM pueden verse como compuertas que se bloquean o se desbloquean, esto quiere decir que existe un procedimiento que decide si la célula almacena o elimina información, esto en función de la importancia que se asignan a los datos que se están recibiendo a través de los pesos recurrentes, es esto lo que al final permite decidir si dicha información es o no importante. En esta tecnología las neuronas LSTM tienen tres compuertas hacía las celdas de información, la de entrada (*input gate)*, la de olvido (*forget gate)* y la de salida (*output gate)* (Torres, 2020), la celda de memoria LSTM se implementa de la siguiente manera (Z. Huang et al., 2015):

$$i_t = \sigma(W_{xi}xt + W_{hi}ht - 1 + W_{ci}ct - 1 + b_i)$$

$$f_t = \sigma\left(W_{xf}xt + W_{hf}ht - 1 + W_{cf}ct - 1 + b_f\right)$$

$$c_t = f_t c_t - 1 + i_t \tanh(W_{xc}xt + W_{hc}ht - 1 + b_c)$$

$$O_t = \sigma(W_{xo}xt + W_{ho}ht - 1 + W_{co}ct - 1 + b_o)$$

$$h_t = o_t \tanh(c_t)$$

donde,

σ	Es la función logística sigmoidea.
i	Es la compuerta de entrada.
f	Es la compuerta de salida.
o, c	Son los vectores de célula.
h	Vector oculto del mismo tamaño que los vectores de célula.
W_{hi}, W_{xo}	La matriz de puerta de entrada oculta y puerta de salida.
W_{ci}	Matriz de pesos de las celdas a los vectores de puerta.

Para el siguiente ejemplo, se han obtenido los aforos del **Paseo Internacional**, específicamente del tramo *Otay Mesa Dwy*, muy cercano a la frontera entre México y Estado unidos, dicho tramo está formada por las incorporaciones de las carreteras Blvd. Garita de Otay y Acceso A Línea SENTRI de México hacía Estado Unidos, una aproximación a la geolocalización tipo vector latitud-longitud, de dicho lugar es *[32.557626, -116.941546]*.

El modelo de predicción LSTM fue desarrollado con un conjunto de datos que considera 34,835 registros, los cuales consisten en la fecha de captura en periodos de cinco minutos y los aforos por carril, en este caso tres; por otro lado, la información pertenece al conteo de vehículos de enero a abril de 2020, de lunes a domingo y tuvo el objetivo de detectar los patrones por día.

En la Figura 3.1 se puede observar un ejemplo de los datos iniciales y finales del conjunto total.

	5 Minutes	Flow (Veh/5 Minutes)
0	2020-01-01 00:00:00	31.0
1	2020-01-01 00:05:00	39.0
2	2020-01-01 00:10:00	31.0
3	2020-01-01 00:15:00	26.0
4	2020-01-01 00:20:00	36.0
5	2020-01-01 00:25:00	43.0
6	2020-01-01 00:30:00	35.0
7	2020-01-01 00:35:00	35.0
8	2020-01-01 00:40:00	28.0
9	2020-01-01 00:45:00	40.0
10	2020-01-01 00:50:00	40.0
11	2020-01-01 00:55:00	32.0
12	2020-01-01 01:00:00	35.0
13	2020-01-01 01:05:00	54.0
14	2020-01-01 01:10:00	45.0
15	2020-01-01 01:15:00	47.0
16	2020-01-01 01:20:00	51.0
17	2020-01-01 01:25:00	42.0
18	2020-01-01 01:30:00	39.0
19	2020-01-01 01:35:00	53.0

	5 Minutes	Flow (Veh/5 Minutes)
34815	2020-04-30 22:20:00	46.0
34816	2020-04-30 22:25:00	35.0
34817	2020-04-30 22:30:00	41.0
34818	2020-04-30 22:35:00	36.0
34819	2020-04-30 22:40:00	49.0
34820	2020-04-30 22:45:00	38.0
34821	2020-04-30 22:50:00	39.0
34822	2020-04-30 22:55:00	34.0
34823	2020-04-30 23:00:00	30.0
34824	2020-04-30 23:05:00	41.0
34825	2020-04-30 23:10:00	50.0
34826	2020-04-30 23:15:00	42.0
34827	2020-04-30 23:20:00	38.0
34828	2020-04-30 23:25:00	39.0
34829	2020-04-30 23:30:00	40.0
34830	2020-04-30 23:35:00	42.0
34831	2020-04-30 23:40:00	40.0
34832	2020-04-30 23:45:00	40.0
34833	2020-04-30 23:50:00	41.0
34834	2020-04-30 23:55:00	36.0

Figura 3.1 Encabezado y pie de datos.

La metodología general del modelado consiste en:

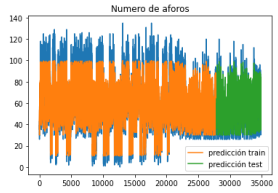
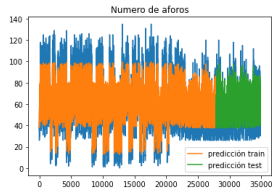
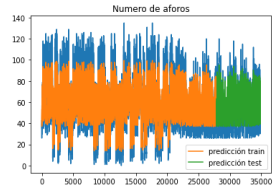
1. **Recopilación de los datos**. Se obtuvo la información de un radar cerca de la frontera México-Estados Unidos.
2. **Preprocesamiento de los datos**. Tales como concatenación de archivos, búsqueda que datos faltantes, detección de valores atípicos (*outliers)*, particiones (s*plit)* de datos para entrenamiento y validación, estandarización

y aplanamiento (agregar una dimensión de matriz, requerimiento del software para el proceso de entrenamiento).

3. **Modelado de la RNA**. Usando una red de tipo secuencial con capas recurrentes y pasando a totalmente conectadas para obtener el valor final de regresión.
4. **Regularización**. Utilizando técnicas tales como las de penalización *Lasso* y *Ridge*, además de la desactivación neuronal *Dropout*.
5. **Calibración**. Modificando los hiperparámetros para mejorar los resultados, tales como, número de neuronas en cada capa, número de capas, factores de regularización, factor de aprendizaje, variaciones LSTM y número de épocas o periodos.
6. **Evaluando el modelo**. Predecir contra datos de entrenamiento y datos de evaluación y graficando el ajuste contra los datos reales.
7. **Implementación**. Se almacena el modelo en forma de archivo de pesos para predecir valores a futuro.

Considerando un 20% del conjunto de datos para la evaluación y el resto para el entrenamiento de la RNA (27,867 registros). Se realizaron tres análisis con diferentes configuraciones de hiperparámetros y se obtuvieron los resultados mostrados en la Tabla 3.1.

Tabla 3.1 Configuraciones y resultados

No	Arquitectura	Hiperparámetros	Error	Gráfica de ajuste
1	Model: "sequential" _____ Layer (type) Output Shape Param # ===== gru (GRU) (None, 1, 32) 3360 gru_1 (GRU) (None, 1, 32) 6336 gru_2 (GRU) (None, 32) 6336 dense (Dense) (None, 1) 33 ===== Total params: 16,065 Trainable params: 16,065 Non-trainable params: 0 _____	-Capas RNR GRU -16,065 pesos a ajustar -3 capas de 32 neuronas recurrentes -1 capa densa como salida -20 épocas	Trainning10.62 puntos. Test 9.92 puntos.	
2	Model: "sequential" _____ Layer (type) Output Shape Param # ===== lstm (LSTM) (None, 1, 32) 4352 lstm_1 (LSTM) (None, 1, 32) 8320 lstm_2 (LSTM) (None, 32) 8320 dense (Dense) (None, 1) 33 ===== Total params: 21,025 Trainable params: 21,025 Non-trainable params: 0 _____	-Capas RNR LSTM -21,025 pesos a ajustar -3 capas de 32 neuronas recurrentes -1 capa densa como salida -2 regularizaciones *dropout* tradicionales y dos recurrentes en las primeras dos LSTM 10 épocas	Trainning11.16 puntos. Test 9.83 puntos.	
3	Model: "sequential" _____ Layer (type) Output Shape Param # ===== lstm (LSTM) (None, 1, 32) 4352 lstm_1 (LSTM) (None, 1, 32) 8320 lstm_2 (LSTM) (None, 32) 8320 dense (Dense) (None, 1) 33 ===== Total params: 21,025 Trainable params: 21,025 Non-trainable params: 0 _____	-Capas RNR LSTM -21,025 pesos a ajustar -3 capas de 32 neuronas recurrentes -1 capa densa como salida -2 regularizaciones *dropout* tradicionales y dos recurrentes en las primeras dos LSTM 20 épocas	Trainning11.19 puntos. Test 10.62 puntos	

Nota. El número de parámetros representa la complejidad computacional en el proceso de entrenamiento.

Fuente: Análisis realizados por los autores de esta publicación.

Como se puede observar en la Tabla 3.1 la arquitectura GRU (Modelo 1) presenta mejores resultados que las capas tradicionales LSTM (Modelo 2 y 3), no solamente genera un error menor a los dos experimentos posteriores, sino que su complejidad computacional y velocidad de entrenamiento es mucho menor. Por otro lado, también se verifica que al incrementar el número de épocas en las arquitecturas con capas LSTM tradicional y la implementación de *dropouts* tradicionales-recurrentes no ayuda a mejorar el desempeño del modelo.

En la gráfica de ajuste de la Tabla 3.1 se puede observar que se logra detectar el patrón de aforos a un nivel medio, sin embargo, se ajusta mejor a los días de menor conteo, se puede comprobar a simple vista, ya que tanto los datos de entrenamiento (color naranja) y los datos de prueba (color verde) en aforos menores a 40 vehículos, se separan mucho menos que en aforos mayores a 100 vehículos con respecto al conjunto de datos reales (color azul), por lo cual, para trabajos a futuro sería oportuno realizar la experimentación con BI-LSTM (Hu et al., 2022) y, con la combinación de RN's Convolucionales y LSTM (Hu et al., 2022).

4. La utilidad de la predicción de tráfico en tiempo real

La predicción del tráfico a través de técnicas de series de tiempo tiene como principal objetivo pronosticar la congestión o capacidad de un tramo carretero en los distintos instantes de tiempo a futuro, sin embargo, esta información puede explotarse a su máximo nivel cuando es combinada con otras soluciones de procesos automatizados.

En la actualidad las tecnologías más avanzadas son Big Data e Internet de las Cosas, que en conjunto con las técnicas de la inteligencia artificial han sobrepasado muchas de las barreras que anteriormente eran inalcanzables, por ejemplo, se ha logrado maximizar a gran escala el poder de cómputo en el procesamiento de datos y en la detección de patrones ocultos con técnicas profundas, el procesamiento de imágenes en tiempo real y la correlación de información a través de la nube y las redes sociales.

Los términos “Transformación Digital” e “Industria 4.0” son la visión de todas las organizaciones, pues impulsa la innovación, mejora la eficiencia de los procesos, proporciona capacidad de respuesta veloz y ofrece nuevas oportunidades gracias al análisis de grandes cantidades de datos y la detección de patrones.

Con base en lo descrito anteriormente se está desarrollando una plataforma de punta para la gestión del tráfico, dicha aplicación se denomina Laboratorio de Visión Artificial del Transporte (LabVAT), mediante el cual se busca generar sinergia entre el procesamiento de grandes cantidades de datos y la obtención de información proveniente de distintos dispositivos que se relacionan en la operación del transporte de carga y la sociedad, esto para proponer soluciones en tiempo real que detonen de manera inmediata un conjunto de recomendaciones de manera oportuna y automática, logrando minimizar los efectos negativos de la congestión en tramos carreteros problemáticos, y no sólo esto, a futuro se pretende gestionar también las áreas de aterrizaje y despegue aéreo, de tal forma que con recorridos de drones y el apoyo de la VA se puedan detectar objetos en las pistas que pueden ocasionar problemas operativos y de seguridad.

Conclusiones

El aprendizaje profundo es un subcampo del aprendizaje automático, que este a su vez se encuentra dentro del concepto más general que es la inteligencia artificial. Esto quiere decir que el aprendizaje profundo contempla el procesamiento genérico neuronal de aprendizaje automático, que a través de un conjunto de ejemplos y de una técnica de aprendizaje, la red intenta ajustar y encontrar los patrones en dicho insumo de información, además sigue contemplando la filosofía de la inteligencia artificial para intentar simular el comportamiento del cerebro humano, por ejemplo, a través de un conjunto de imágenes de gatos aprender a discriminar por características cuando lo es y cuando no.

El aprendizaje profundo es un algoritmo que permite a las computadoras aprender con el ejemplo, lo que es natural para el humano, por ejemplo, reconocer una señal de tráfico, distinguir a un peatón, entender una indicación señalada por voz, etc., el aprendizaje profundo ha generado resultados sorprendentes, por tales motivos, muchos investigadores concentran su atención en este tipo de técnicas y por consecuente lograr lo que antes no era posible.

El aprendizaje profundo puede resolver problemas muy complejos cuando se usa el computo en paralelo (GPUs) y clúster para procesamiento distribuido, de tal forma que se pueda reducir el tiempo de entrenamiento de semanas a horas o posiblemente en menos.

Las redes neuronales artificiales demuestran la importancia o la utilidad de una característica en el valor de peso de conexión entre neuronas.

Feedforward consiste en una serie de cálculos a través de las capas para hacer una predicción, es la principal funcionalidad de las redes neuronales artificiales.

Es posible concluir que el número de cálculos que hay que realizar en una red pequeña son muchas, cuando se tiene una red compleja esto se vuelve poco práctico, con un manejo eficiente de matrices se puede pasar múltiples entradas a la vez, con la librería NumPy de lenguaje de programación Python es posible acelerar el procesamiento.

El proceso para encontrar el peso objetivo de la red neuronal por ajuste de peso se realiza a través de un proceso iterativo usando un algoritmo de optimización.

Backpropagation comienza su proceso al final de la red, propagando hacía tras el error, de manera recursiva aplicando la regla de la cadena para calcular el gradiente en todos los caminos hasta las entradas, esto hace que se actualicen los pesos, a lo que se le denomina también aprendizaje.

Una de las principales desventajas de las redes neuronales recurrentes es el hecho de requerir un ajuste adicional de pesos, se refiere al peso recurrente, su costo computacional es considerablemente alto, sin embargo, existen variaciones de dichas redes que permiten mejorar dichos tiempos de respuesta, o simplemente el resultado supera el valor de la desventaja en cuanto a tiempo.

En el preprocesamiento de los datos para el aprendizaje automático se hace una partición automática de entrenamiento y evaluación con un proceso aleatorio a los registros, con el propósito de eliminar las dependencias en el orden de los mismo, sin embargo, para las capas LSTM esta dependencia es necesaria, por lo tanto, esta tarea es eliminada.

El manejo de la regularización en las redes neuronales recurrentes es esencial para evitar el sobre ajuste en los resultados; las técnicas más utilizadas son LASSO, RIDGE y DROPOUT, tanto para pesos tradicionales como para los recurrentes.

Bibliografía

Abduljabbar, R., Dia, H. & Tsai, P. (2021). Unidirectional and Bidirectional LSTM Models for Short-Term Traffic Prediction. *Journal of Advanced Transportation*, *2021*. https://doi.org/https://doi.org/10.1155/2021/5589075

Centeno, F. (2019). *DEEP LEARNING. Tesis de ingeniería*. Universidad de Sevilla.

Elgendy, M. (2020). *Deep Learning for Vision Systems.* (1ª ed.). New York. Manning Publications.

github.io. (2022). *Introducción al Deep Learning* (IAAR). Consultado desde https://iaarbook.github.io/deeplearning/

Huang, W., Song, G., Hong, H. & Xie, K. (2014). Deep architecture for traffic flow prediction: deep belief networks with multitask learning. *IEEE Transactions on Intelligent Transportation Systems*, *15*(5), 2191–2201.

Huang, Z., Xu, W. & Yu, K. (2015). Bidirectional LSTM-CRF Models for Sequence Tagging. *Computation and Language*, 2015. https://doi.org/https://doi.org/10.48550/arXiv.1508.01991

Hu, X., Liu, T. & Lin, C. (2022). Attention-based Conv-LSTM and Bi-LSTM networks for large-scale traffic speed prediction. *The Journal of Supercomputing, 2022*.

Jia, Y., Wu, J. & Xu, M. (2017). Traffic Flow Prediction with Rainfall Impact Using a Deep Learning Method. *Journal of Advanced Transportation*, *2017*. https://doi.org/https://doi.org/10.1155/2017/6575947

Lv, Y., Duan, Y., Kang, W., Li, Z. & Wang, F. (2015). Traffic flow prediction with big data: a deep learning approach. *IEEE Transactions on Intelligent Transportation Systems*, *16*(2), 865–873.

Panduro, D. y Martín, R. (2017). *Propuesta de implementación de un sistema inteligente de transporte para la mejora de las condiciones viales en el tramo del Panamericana Norte entre av. los Alisos y av. Abancay. Tesis de ingeniería*. Universidad Peruana de Ciencias Aplicadas.

Secretaría de Comunicaciones y Transportes [SCT]. (2020). *Programa Sectorial de Comunicaciones y Transportes 2020-2024* (Programa Sectorial Derivado Del Plan Nacional de Desarrollo 2019-2024). Consultado desde https://www.dof.gob.mx/nota_detalle.php?codigo=5596042&fecha=02/07/2020

Torres, J. (2020). *Python Deep Learning.* (1ª ed.). España. Alfaomega.

Wang, J., Ma, Y., Yang, X., Li, T. & Wei, H. (2021). Short-Term Traffic Prediction considering Spatial-Temporal Characteristics of Freeway Flow. *Journal of Advanced Transportation*, *2021*. https://doi.org/https://doi.org/10.1155/2021/5815280

COMUNICACIONES

SECRETARÍA DE INFRAESTRUCTURA, COMUNICACIONES Y TRANSPORTES

Km 12+000 Carretera Estatal 431 “El Colorado Galindo”
Parque Tecnológico San Fandila, Mpio. Pedro Escobedo,
Querétaro, México. C.P. 76703
Tel: +52 (442) 216 97 77 ext. 2610
Fax: +52 (442) 216 9671

publicaciones@imt.mx

http://www.imt.mx/