Aprendizaje Automático para la Selección de Grupos de Control en Evaluación de Impacto

por

Benjamín Bas Peralta

Presentado ante la **FACULTAD DE MATEMÁTICA, ASTRONOMÍA, FÍSICA Y COMPUTACIÓN** como parte de los requerimientos para la obtención del grado de Licenciado en Ciencias de la Computación de la **UNIVERSIDAD NACIONAL DE CÓRDOBA**

Marzo, 2025

Directores: Dr. Martín Domínguez

Mgter.David Giuliodori

Resumen: (...)

Palabras Clave: Evaluación de Impacto, Grupo de Control, Aprendizaje Automático, Redes Neuronales, Series de Tiempo.

Abstract: (...)

Keywords: Impact Assesment, Control Group, Machine Learning, Neural Networks, Time Series.

Agradecimientos

Índice general

1. Introducción **9**

2. Marco Teórico **11**
2.1. Evaluación de Impacto 12
2.1.1. ¿Qué es una Evaluación de Impacto? 12
2.1.2. Estimación del Tratamiento 13
2.1.3. El Grupo de Control como Estimador del Contrafactual 14
2.1.4. Evaluaciones Experimentales o Aleatorias 17
2.1.5. Evaluaciones Cuasi-experimentales 19
2.2. Inteligencia Artificial 24
2.2.1. Aprendizaje Automático 24
2.2.2. Aprendizaje Supervisado 26
2.3. Redes Neuronales Artificiales 31
2.3.1. Breve Contexto Histórico 32
2.3.2. Red Neuronal Feedforward 32
2.4. Redes Neuronales Convolucionales 40
2.5. Redes Neuronales Recurrentes 45
2.5.1. Long Short-Term Memory 49
2.5.2. Gated Recurrent Unit 49
2.6. Clasificación de Series de Tiempo 49
2.6.1. Clasificación de Series de Tiempo con Aprendizaje Automático . . 49

3. Presentación del problema **51**

4. Marco Experimental **53**
4.1. Conjuntos de Datos 53
4.2. Arquitecturas de Redes Neuronales 54
4.3. Herramientas 54
4.3.1. Hardware 55
4.3.2. Python 55
4.3.3. PyTorch 55

4.3.4. Pandas 55
4.3.5. Numpy 55
4.3.6. Jupyter 55
4.3.7. Optuna 55
4.3.8. MLflow 55

5. Resultados **57**

6. Conclusiones y Trabajos Futuros **59**

Capítulo 1

Introducción

Capítulo 2

Marco Teórico

En este capítulo, presentamos los conceptos teóricos necesarios para entender el contexto del problema y cuál es la solución que planteamos.

Primeramente, explicamos la Evaluación de Impacto, que es el campo de aplicación sobre el que vamos a trabajar. Describimos qué es una Evaluación de Impacto, en qué consiste, cuáles son los problemas involucrados, y las distintas técnicas que se utilizan para llevarla a cabo. Al final de esta sección, presentamos el método de *Propensity Score Matching*, que es el que tomaremos como parámetro para evaluar nuestra solución propuesta.

Luego, nos abordamos al campo de la Inteligencia Artificial, enfocándonos particularmente en el campo de los algoritmos de Aprendizaje Automático. Explicamos cómo funcionan estos y cuáles son los elementos presentes en ellos haciendo especial enfásis en los algoritmos de Aprendizaje Supervisado, dentro de los cuales se enmarca nuestra propuesta.

A continuación, hablamos sobre un *modelo* particular dentro del Aprendizaje Automático que ha cobrado particular importancia en los últimos años: las Redes Neuronales Artificiales. Partimos explicando las de tipo *Feedforward* para Luego introducir otras más específicas como son las Convolucionales y las Recurrentes, de las cuales haremos uso en nuestros experimentos.

Por último, y para combinar el método de solución propuesto junto con el área de aplicación, hablamos sobre las series de tiempo. Mencionamos qué son y cuáles son los parámetros involucrados en ellas, y repasamos los últimos trabajos llevados a cabo que utilizan Inteligencia Artificial para resolver el problema de clasificación de series de tiempo.

2.1. Evaluación de Impacto

2.1.1. ¿Qué es una Evaluación de Impacto?

Los programas y políticas de desarrollo suelen estar diseñados para cambiar resultados, como aumentar los ingresos, mejorar el aprendizaje o reducir las enfermedades [4]. Saber si estos cambios se logran o no es una pregunta crucial para las políticas públicas [4], y el método para responderla es lo que se conoce como **evaluación de impacto**.

Una evaluación de impacto mide los cambios en el bienestar de los individuos que se pueden atribuir a un proyecto, un programa o una política específicos [4]. Su sello distintivo radica en que pueden proporcionar **evidencia** robusta y creíble sobre si un programa concreto ha alcanzado o está alcanzando sus objetivos [4].

Estas evaluaciones ponen un fuerte énfasis en los resultados y buscan responder una pregunta específica de causa y efecto: ¿cuál es el impacto (o efecto causal) de un programa? Más precisamente, intentan identificar y cuantificar los cambios directamente atribuibles al tratamiento [4] sobre un conjunto de **variables de resultado** o **de interés** en un conjunto de individuos [1]. Estas variables son aquellas sobre las cuales se espera que el programa tenga un efecto en los beneficiarios [1]. Podrían ser por ejemplo la estatura y peso de los individuos, la cantidad de empleados o de ventas en una empresa, o indicadores de salud de un paciente.

Por lo tanto, el objetivo final de la evaluación de impacto consiste en establecer lo que se conoce como **efecto del tratamiento**, que es la diferencia entre la variable de resultado del individuo participante en el programa en presencia del programa, y la variable de resultado de ese individuo en ausencia del programa [1]. Sin embargo, es evidente que la respuesta a qué habría pasado con los beneficiarios si no hubieran recibido el tratamiento se refiere a una situación que no es observable. Este resultado hipotético se denomina **contrafactual** y es lo que se debe estimar en cualquier evaluación de impacto.

Algunas de las principales razones por las que se debería promover el uso de estas evaluaciones como herramientas de gestión provienen del hecho que permiten mejorar la rendición de cuentas, la inversión de recursos públicos o la efectividad de una política, obtener financiamiento, como así también probar modalidades de programas alternativos o innovaciones de diseño [4] y revelar la realidad de muchas políticas públicas para de esta forma contribuir a la fiscalización mediática [1].

Un ejemplo claro de por qué son necesarias las evaluaciones de impacto es el que describe Howard White con respecto al Programa Integrado de Nutrición en Bangladesh (PINB) [17]. Este programa identificaba, mediante mediciones en campo, a los niños desnutridos y los asignaba a un tratamiento que incluía alimentación suplementaria a los menores y educación nutricional a las madres [1]. Inicialmente, el programa fue considerado como un éxito ya que los datos de monitoreo mostraban caídas importantes en los niveles de desnutrición. El Banco Mundial decidió, con base en esta evidencia y previo a cualquier tipo de evaluación, aumentar los recursos destinados al programa. Sin embargo,

las primeras evaluaciones de impacto, realizadas por el Grupo Independiente de Evaluación del mismo Banco Mundial y por la ONG inglesa *Save the Children*, mostraron que la mejoría de los indicadores de los beneficiarios era similar o inferior a la de otros niños con características comparables que no hacían parte del programa [1]. Estos resultados reflejaron que las percepciones de los administradores del programa y de las entidades financiadoras eran erradas, y sugirieron algunos correctivos al programa [1].

2.1.2. Estimación del Tratamiento

El marco teórico estándar para formalizar el problema de la evaluación de impacto se basa en el **modelo de resultados potenciales** o **modelo causal de Rubin** [13].

Formalmente, se definen dos elementos para cada individuo $i = 1, ..., N$, donde N denota la población total:

- Por un lado, el indicador de tratamiento D_i, tal que $D_i = 1$ implica que el individuo i participó del tratamiento, y $D_i = 0$ en caso contrario.
- Por otro lado, las variables de resultado las definimos como $Y_i(D_i) = Y_i|D_i$ - se lee como "el valor de Y_i *dado* D_i". De esta forma, $Y_i(1)$ es la variable de resultado si el individuo i es tratado, e $Y_i(0)$ es la variable de resultado si el individuo i no es tratado. Estos valores son los resultados potenciales.

Con esto, el **efecto del tratamiento** para un individuo i se puede escribir como:

$$\tau_i = Y_i(1) - Y_i(0) = (Y_i|D_i = 1) - (Y_i|D_i = 0) \tag{2.1}$$

Según esta fórmula, el impacto causal (τ) de un programa (D) en una variable de resultado (Y) para un individuo i es la diferencia entre la variable con el programa ($Y_i(1)$) y la misma variable sin el programa ($Y_i(0)$).

De nuevo, el problema fundamental de la evaluación de impacto es que se intenta medir una variable en un mismo momento del tiempo para la misma unidad de observación pero en dos realidades diferentes. Sin embargo, claramente solo se da uno de los dos resultados potenciales $Y_i(1)$ o $Y_i(0)$, pero no ambos. Es decir, en los datos queda solamente registrado $Y_i(1)$ si $D_i = 1$ e $Y_i(0)$ si $D_i = 0$; no se dispone de $Y_i(1)$ si el individuo no fue tratado ($D_i = 0$), ni tampoco de $Y_i(0)$ si el individuo fue tratado ($D_i = 1$). De esta manera, el **resultado observado de** Y_i se puede expresar como:

$$Y_i = D_i Y_i(1) + (1 - D_i)Y_i(0) = \begin{cases} Y_i(1) \text{ si } D_i = 1 \\ Y_i(0) \text{ si } D_i = 0 \end{cases} \tag{2.2}$$

Si nos concentramos en las unidades tratadas, en la Ecuación 2.1, el término $Y_i(0) = (Y_i|D_i = 0)$ representa la situación contrafactual, es decir *cuál habría sido el resultado si*

la unidad no hubiera participado en el programa. Al ser imposible observar directamente el contrafactual, es necesario **estimarlo**. La forma más directa de solucionar este problema sería hallando un "clon perfecto" para cada uno de los individuos participantes del programa pero que no haya participado, lo cual resulta bastante difícil. Por lo tanto, el primer paso para lograr esta estimación consiste en **desplazarse desde el nivel individual al nivel del grupo** [4], concentrando el análisis en el **impacto** o **efecto *promedio*** (y no individual).

En primera instancia, se puede estimar el **impacto promedio del programa *sobre la población*** (o ATE, del inglés *Average Treatment Effect*):

$$ATE = \mathbb{E}\left[Y_i(1) - Y_i(0)\right] \tag{2.3}$$

donde el operador $\mathbb{E}$ representa la media de una variable aleatoria.

El ATE se interpreta como el cambio promedio en la variable de resultado cuando un individuo escogido al azar pasa aleatoriamente de ser participante a ser no participante [1], es decir representa el efecto esperado del tratamiento si toda la población fuera tratada. Es importante notar que para calcularlo, habría que estimar un contrafactual tanto para los tratados como para los no tratados.

Sin embargo, en la mayoría de los casos, el tratamiento solo está disponible para un subconjunto de la población, ya sea por restricciones presupuestarias o criterios de elegibilidad, entre otras razones. Además, también ocurre que no todos los que fueron seleccionados para recibirlo efectivamente participan o se inscriben. En estos casos, suele ser más relevante estimar el efecto únicamente en quienes efectivamente recibieron el tratamiento.

Por lo tanto, se define el **impacto promedio del programa *sobre los tratados*** (o ATT, del inglés *Average Treatment Effect on the Treated*), que es en general el parámetro de mayor interés en una evaluación de impacto, y representa el efecto esperado del tratamiento en el subconjunto de individuos que fueron efectivamente tratados:

$$ATT = \mathbb{E}\left[Y_i(1) - Y_i(0)|D_i = 1\right] = \mathbb{E}\left[Y_i(1)|D_i = 1\right] - \mathbb{E}\left[Y_i(0)|D_i = 1\right] \tag{2.4}$$

Es decir, el ATT es la diferencia entre la media de la variable de resultado en el grupo de los participantes y la media que hubieran obtenido los participantes si el programa no hubiera existido [1].

2.1.3. El Grupo de Control como Estimador del Contrafactual

En la ecuación 2.4, el término $\mathbb{E}\left[Y_i(1)|D_i = 1\right]$ es un resultado observable mientras que $\mathbb{E}\left[Y_i(0)|D_i = 1\right]$ es el promedio contrafactual que debemos aproximar. Para lograrlo, se construye lo que se conoce como **grupo de control** o **de comparación**, formado por individuos que no participan del programa pero que, idealmente, son estadísticamente similares [4] al **grupo de tratamiento**, compuesto por quienes sí recibieron el programa.

La Figura 2.1 ilustra esta distinción. A partir de ahora, denotaremos con C_i a un individuo i que pertenezca al grupo de control.

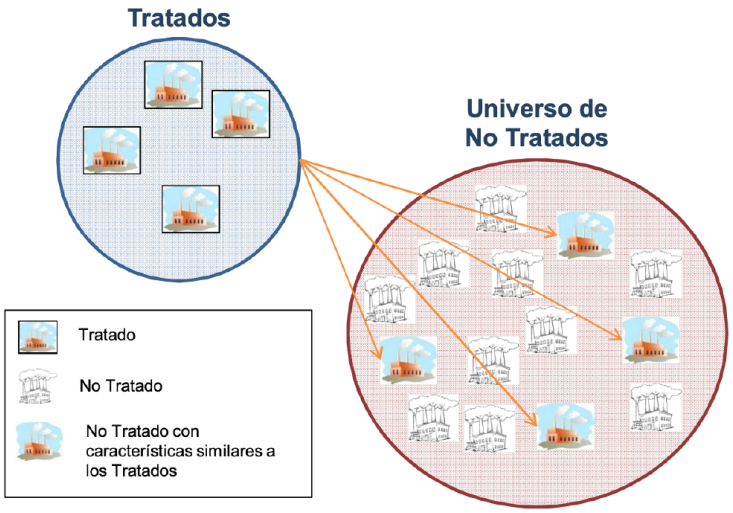


Figura 2.1: El grupo de control debería estar formado idealmente por individuos que no recibieron el tratamiento pero que en promedio poseen características similares a los tratados.

Por lo tanto, en la práctica, el reto está en definir un *buen* grupo de control, es decir uno que sea estadísticamente idéntico al de tratamiento, en promedio, en ausencia del programa [4]. Si se lograra que los dos grupos sean idénticos, con la única excepción que un grupo participa del programa y el otro no, entonces sería posible estar seguros que cualquier diferencia en los resultados tiene que deberse al programa [4].

Para que un grupo de comparación sea válido, debe satisfacer lo siguiente [4]:

- Las características *promedio* del grupo de tratamiento y del grupo de comparación deben ser idénticas en ausencia del programa. Cabe resaltar que no es necesario que las unidades individuales en el grupo de tratamiento tengan clones perfectos en el grupo de control.
- No debe ser afectado por el tratamiento de forma directa ni indirecta.
- El grupo de comparación debería reaccionar de la misma manera que el grupo de tratamiento si fuera objeto del programa.

Cuando el grupo de comparación no produce una estimación precisa del contrafactual, no se puede establecer el verdadero impacto del programa. A continuación, se presentan dos situaciones en las que esto ocurre.

Comparaciones antes-después

Este tipo de comparaciones intenta establecer el impacto de un programa a partir de un seguimiento de los cambios en los resultados en los participantes del programa a lo largo del tiempo. Consideran el contrafactual como el resultado para el grupo de tratamiento antes que comience la intervención, momento también conocido como **línea de base**. Esta comparación supone que si el programa no hubiera existido, el resultado para los participantes del programa habría sido igual a su situación antes del programa, lo cual en la mayoría de los tratamientos no puede sostenerse [4].

Comparaciones de inscritos y no inscritos: Sesgo de autoselección

Como explicamos anteriormente, el término $\mathbb{E}[Y_i(0)|D_i = 1]$ de la Ecuación 2.4 representa el contrafactual, que no es observable. Sin embargo, lo que sí se puede medir es la variable de resultado entre los no inscritos, aquellos individuos que no participaron del programa, es decir $\mathbb{E}[Y_i(0)|D_i = 0]$.

Por lo tanto, podríamos tomar a todo el conjunto de los no participantes como grupo de control y solucionar el problema del contrafactual utilizando $\mathbb{E}[Y_i(0)|D_i = 0]$ como estimador, es decir podríamos asumir lo siguiente:

$$\mathbb{E}[Y_i(0)|D_i = 1] \ = \mathbb{E}[Y_i(0)|D_i = 0] \tag{2.5}$$

Esto quiere decir que el valor esperado de la variable de resultado en ausencia del programa ($\mathbb{E}[Y_i(0)]$) es idéntico para el grupo de tratados ($D = 1$) y para el de no tratados ($D = 0$). Indica que los individuos que participan en el programa no son sistemáticamente distintos de los que no lo hacen [1] o, en otras palabras, que la decisión de participar en el programa no está determinada por factores que también influyen en la variable de resultado.

De esta forma, podríamos calcular el ATT como:

$$ATT = \mathbb{E}[Y_i(1)|D_i = 1] - \mathbb{E}[Y_i(0)|D_i = 0] \tag{2.6}$$

Sin embargo, el supuesto de la Ecuación 2.5 se viola toda vez que la participación en el programa es una *elección* del individuo elegible [1]. La razón es que los participantes y los no participantes generalmente son diferentes, aún en ausencia del programa, y son justamente esas diferencias que llevan a que algunos individuos escojan participar y otros no. Por lo tanto, si en estos casos se toma como grupo de control a todos aquellos que no decidieron participar y se mide el impacto con respecto a ellos, puede ocurrir que la diferencia en los resultados se deba a características propias de los individuos que llevaron a unos a anotarse y a otros no.

Este problema se conoce como **sesgo de autoselección**, ya que los integrantes del grupo de control se *autoseleccionaron* para no participar del programa. Más concretamente, este sesgo se produce cuando los motivos por los que un individuo participa en

un programa, usualmente no observables y difíciles de medir, están correlacionados con los resultados [4], y por ende, es muy probable que la variable de resultado del grupo de tratamiento y del grupo de control sean diferentes, *aún si el programa no existiera* [1]. Intuitivamente, si hay variables que explican tanto la participación como la variable de resultado, la comparación de medias puede estar atribuyendo al programa un efecto que en realidad se debe a las diferencias preexistentes entre el grupo de tratamiento y el grupo de control [1].

Un buen ejemplo de esta situación lo presenta [1], en el que se plantea un programa de nutrición infantil. Podría ocurrir que las madres de familias participantes del programa sean más proactivas respecto al desarrollo de sus hijos, por lo cual se preocuparon en lograr la participación en el programa. El problema de autoselección en este caso radica en que la motivación de las madres (que no se observa y es difícil de medir) afecta no solo la probabilidad de participar en el programa, sino también el estado nutricional de los niños ya que estas podrían vigilar mejor la dieta de sus hijos. De esta forma, la diferencia observada en el estado nutricional de los niños de los dos grupos podría deberse parcialmente a la diferencia en el nivel de compromiso de las madres, y no exclusivamente a que un grupo participa en el programa y el otro no.

Podemos plantear este problema más formalmente. Notemos que podemos reescribir la fórmula del ATT (2.4) de la siguiente manera:

$$ATT + \mathbb{E}\left[Y_i(0)|D_i = 1\right] = \mathbb{E}\left[Y_i(1)|D_i = 1\right] \tag{2.7}$$

Si ahora restamos $\mathbb{E}\left[Y_i(0)|D_i = 0\right]$ a ambos lados, obtenemos:

$$ATT + \mathbb{E}\left[Y_i(0)|D_i = 1\right] - \mathbb{E}\left[Y_i(0)|D_i = 0\right] \ = \mathbb{E}\left[Y_i(1)|D_i = 1\right] - \mathbb{E}\left[Y_i(0)|D_i = 0\right] \tag{2.8}$$

De esta forma, el sesgo de autoselección aparece en el término $\mathbb{E}\left[Y_i(0)|D_i = 1\right] - \mathbb{E}\left[Y_i(0)|D_i = 0\right]$.

Dicho esto, el gran desafío de la evaluación de impacto es determinar las condiciones o supuestos bajo los cuales $\mathbb{E}\left[Y_i(0)|D_i = 0\right]$ se puede utilizar como una estimación válida del contrafactual $\mathbb{E}\left[Y_i(0)|D_i = 1\right]$ [1], y por lo tanto utilizarse para poder aproximar el valor del ATT. Asegurarse de que el impacto estimado esté libre de sesgo de autoselección es uno de los principales objetivos de cualquier evaluación de impacto, y plantea importantes dificultades [4].

Las reglas de un programa para seleccionar a los participantes, también llamadas **reglas de asignación**, constituyen el parámetro clave para determinar el método de evaluación de impacto. A continuación, presentamos las distintas reglas y los métodos de evaluación que se utilizan en la práctica al trabajar con cada una de ellas.

2.1.4. Evaluaciones Experimentales o Aleatorias

Este tipo de evaluaciones se utiliza cuando los beneficiarios del programa en cuestión son seleccionados al azar, es decir mediante un sorteo. La asignación aleatoria se

considera la regla de oro de la evaluación de impacto ya que no solo proporciona a los administradores del programa una regla imparcial y transparente para asignar recursos escasos entre poblaciones igualmente merecedoras de ellos, sino que también representa el método más sólido para evaluar el impacto de un programa [4].

La asignación aletoria trae dos consecuencias importantes: las unidades ya no pueden *elegir* si participar o no, y además todas tienen la misma probabilidad de ser seleccionadas para el programa. De esta forma, el resultado potencial de cada individuo es independiente de sus condiciones previas, ya que la asignación al tratamiento no está determinada por sus características, sino por el azar.

Como explicamos anteriormente, el grupo de comparación ideal sería lo más similar posible al grupo de tratamiento en todos los sentidos, excepto con respecto a su participación en el programa que se evalúa. Ahora bien, el proceso de asignación aleatoria producirá dos grupos que tienen una alta probabilidad de ser estadísticamente idénticos en todas sus características (observables y no observables), siempre que el número de unidades sea suficientemente grande [4]. Esto es así ya que en general, si la población de unidades elegibles es lo suficientemente grande, este mecanismo asegura que cualquier rasgo de la población se transfiere a ambos grupos. Sin embargo, en la práctica, este supuesto debería comprobarse empíricamente con los datos de línea de base de ambos grupos.

Dicho esto, lo que ocurre en estos casos es que todos aquellos individuos que no resultaron ser beneficiarios del programa conforman un grupo de control que resulta ser naturalmente un excelente estimador del contrafactual $\mathbb{E}\left[Y_i(0)|D_i = 1\right]$. Es decir, mediante este tipo de asignación, se asegura que se cumple el supuesto de la Ecuación 2.5.

La asignación aleatoria puede utilizarse como regla de asignación de un programa en dos escenarios específicos: cuando la población elegible es mayor que el número de plazas disponibles del programa, o cuando sea necesario ampliar un programa de manera progresiva hasta que cubra a toda la población elegible [4]. Además, la asignación aleatoria podría hacerse a nivel individual (por ejemplo, a nivel de hogares), o a nivel de conglomerado (por ejemplo, comunidades) [1].

Una vez que se haya asignado el tratamiento de manera aleatoria, gracias a **??**, es bastante sencillo estimar el impacto del programa. Después de que el programa ha funcionado durante un tiempo, tendrán que medirse los resultados de las unidades de tratamiento y de comparación. El impacto del programa es sencillamente la diferencia entre el resultado promedio para el grupo de tratamiento y el resultado promedio para el grupo de comparación, es decir:

$$\mathbb{E}\left[Y_i(1)|D_i = 1\right] - \mathbb{E}\left[Y_i(0)|D_i = 0\right]$$

Resulta claro que las evaluaciones experimentales tienen varias ventajas: por diseño, resuelven el problema del sesgo de selección, y además los resultados obtenidos son transparentes, intuitivos y no es necesario utilizar herramientas econométricas sofisticadas para

alcanzarlos [1]. Esto hace que contribuyan a la transparencia de las políticas públicas y a la rendición de cuentas [1].

Sin embargo, este tipo de evaluaciones sufre varias limitaciones que hace que no sea tan común en la práctica. Por un lado, pueden ser costosas y de difícil implementación [1]. Esto se debe a que en general, el experimento aleatorio se hace específicamente para evaluar una intervención, por lo que es necesario destinar recursos para la prueba piloto, la recolección de información, los seguimientos, y a veces incluso la implementación del programa [1]. Otro problema que tienen, y probablemente el más importante, es que por pertenecer al grupo de control, se *excluye* a un segmento de la población, igualmente vulnerable y elegible, de los beneficios de la intervención [1]. Además, dada la imposibilidad de negar los beneficios a un grupo de control por largos períodos, con frecuencia es imposible estimar los impacto de largo plazo [1].

2.1.5. Evaluaciones Cuasi-experimentales

Cuando la asignación aleatoria de los participantes no es factible o plantea problemas éticos, se emplean las llamadas evaluaciones o técnicas cuasi-experimentales, que son aquellas que buscan estimar el impacto causal, pero a diferencia de las experimentales, no se basan en la asignación aleatoria de la intervención [4]. En estos casos, la regla de asignación suele ser menos clara, por lo que se convierte en una incógnita adicional en la evaluación, acerca de la cual se deben formular supuestos [4]. Para ello, estos métodos intentan *simular* la aleatorización de un diseño experimental controlando las diferencias entre los individuos tratados y no tratados bajo diferentes hipótesis.

Si bien los métodos cuasi-experimentales suelen ser más adecuados en algunos contextos operativos, su desventaja es que requieren más condiciones para garantizar que el grupo de comparación provea una estimación válida del contrafactual, como veremos a continuación al profundizar en algunos de ellos.

Diferencias en Diferencias

El modelo de diferencias-en-diferencias (DD) es una manera de controlar la estimación del impacto por las diferencias preexistentes entre el grupo de tratamiento y el de control, que estará formado por todos aquellos individuos elegibles que no recibieron el tratamiento. Es decir, el DD no propone una forma de construir un grupo de control adecuado sino una manera de aproximar el impacto que tenga en cuenta las diferencias entre participantes y no participantes. A grandes rasgos, combina las comparaciones antes-después con las comparaciones de inscritos y no inscritos, ambas discutidas anteriormente.

Dada una variable de resultado Y, este modelo establece el impacto simplemente como el cambio esperado en Y entre el período posterior y el período anterior a la implementación del tratamiento en el grupo de tratamiento, menos la diferencia esperada en Y en el grupo de control en el mismo período [1]. Es decir, **compara los cambios en los**

resultados a lo largo del tiempo entre unidades participantes y no participantes.

Si denotamos con Y_1 al valor de la variable Y en la línea de base, es decir justo antes de la implementación del programa, y con Y_2 al valor de la misma variable en una período posterior a la implementación (también llamado período de seguimiento), entonces el impacto del programa por el método de DD estaría dado por:

$$\tau_{DD} = (\mathbb{E}[Y_2|D=1] - \mathbb{E}[Y_1|D=1]) - (\mathbb{E}[Y_2|D=0] - \mathbb{E}[Y_1|D=0]) \tag{2.9}$$

Es importante señalar que el contrafactual que se estima en este caso es el cambio en los resultados del grupo de tratamiento, y su estimación se logra midiendo el cambio en los resultados del grupo de comparación [4]. En lugar de contrastar los resultados entre los grupos de tratamiento y comparación después de la intervención, la técnica de DD estudia las tendencias entre los grupos de tratamiento y comparación [4].

Como explicamos con anterioridad, el principal problema de las comparaciones antes-después en el grupo de tratamiento es que se asume que lo único que hizo cambiar la variable de resultado fue el programa, cuando en realidad puede haber sido por factores externos. El DD busca solucionar este problema basándose en la idea que la diferencia en los resultados antes-después para el grupo de tratamiento controla por factores que son constantes a lo largo del tiempo en ese grupo, y la manera de capturar los factores variables en el tiempo es medir el cambio antes-después en los resultados de un grupo que no se inscribió en el programa pero que estuvo expuesto al mismo conjunto de condiciones ambientales. De esta forma, si se "limpia" la primera diferencia de otros factores variables en el tiempo que influyen en el resultado de interés sustrayendo la segunda diferencia, se habrá eliminado una fuente de sesgo [4].

Por otro lado, en las comparaciones de inscritos y no inscritos, el gran inconveniente era el sesgo de autoselección. Para entender mejor la solución que el DD propone a esto, resulta conveniente reescribir el estimador como:

$$\tau_{DED} = (\mathbb{E}[Y_2|D=1] - \mathbb{E}[Y_2|D=0]) - (\mathbb{E}[Y_1|D=1] - \mathbb{E}[Y_1|D=0]) \tag{2.10}$$

De aquí se puede ver lo que plantea es restarle al cambio visto en la variable entre tratados y no tratados luego de la implementación del programa (Y_2), las diferencias que ya pueda haber habido entre ellos en la línea de base (Y_1). Es decir, desde este punto de vista, se utiliza a $\mathbb{E}[Y_1|D=1] - \mathbb{E}[Y_1|D=0]$ como un estimador de las diferencias preexistentes entre el grupo de tratamiento y el grupo de control.

Ahora bien, volviendo a la Ecuación 2.9, el supuesto que permite utilizar la diferencia en el grupo de no inscritos para controlar los factores externos que afectan a los inscritos es que, en ausencia del programa, dichos factores habrían impactado a los tratados de la misma manera que a los no tratados. En otras palabras, se asume que, sin la intervención, los resultados en el grupo tratado habrían evolucionado en paralelo con los del grupo de control, aumentando o disminuyendo en la misma proporción. Esto se conoce como **supuesto de tendencias paralelas**, y es sobre el que hay que basarse al trabajar con

este método ya que claramente no es posible observar qué habría ocurrido con el grupo tratado en ausencia del programa.

Propensity Score Matching

Otro enfoque para lograr una buena aproximación del contrafactual consiste en construir un grupo de comparación artificial, a partir del conjunto de individuos que no recibieron el tratamiento. En esto se basa el método conocido como **pareamiento** o **emparejamiento**, que utiliza técnicas estadísticas y grandes bases de datos para construir el mejor grupo de comparación posible sobre la base de características observables [4]. La principal utilidad de este método es que puede aplicarse en el contexto de casi todas las reglas de asignación de un programa, siempre que se cuente con un grupo que no haya participado en el mismo [4].

El supuesto que utilizan estas técnicas para basarse en características observables al construir el grupo de control es que tanto la participación en el programa como los resultados potenciales están únicamente determinados por estas. O, en otras palabras que no están determinados por variables no observables (o no medidas).

De esta forma, se asume que al condicionar los resultados potenciales de tratados y no tratados en determinadas características observables (pertinentes en el programa bajo estudio), el sesgo de autoselección es igual a cero. Formalmente, si denotamos con X a los rasgos observables que se están teniendo en cuenta, este supuesto se escribe de la siguiente forma:

$$\mathbb{E}\left[Y_i(0)|D_i = 1, X\right] = \mathbb{E}\left[Y_i(0)|D_i = 0, X\right], \tag{2.11}$$

y se lo conoce como **supuesto de Independencia Condicional** (IC).

Con esto en mente, la versión más directa de esta metodología consiste en encontrar un "clon" *para cada individuo tratado* en el grupo de no tratados y contrastar las variables de resultado de ambos [1]. Un clon en este caso quiere decir, una vez fijadas las características que el investigador cree que explican la decisión del de inscribirse en el programa, un individuo (o grupo de individuos) con exactamente las mismas características observables X.

Sin embargo, en la práctica esto resulta difícil. Si la lista de características observables relevantes es muy grande, o si cada característica adopta muchos valores, puede resultar computacionalmente complejo identificar una pareja para cada una de las unidades del grupo de tratamiento. Esta situación es conocida como *el problema de la dimensionalidad*.

Este problema puede solucionarse utilizando un método denominado pareamiento por puntajes de propensión (*Propensity Score Matching*, PSM). Consiste en emparejar individuos ya no con base en X, sino en su probabilidad estimada de participación en el programa, dadas sus características observables, es decir en:

$$P(X) = P(D = 1|X)$$

[1] donde P denota el operador de probabilidad de un evento. Este valor es conocido como **probabilidad de participación** o **puntaje de propensión**.

De esta forma, el clon adecuado para cada individuo del grupo de tratamiento será aquel del grupo de no tratados con una probabilidad de participación en el programa suficientemente cercana [1]. La ventaja de emparejar a partir de $P(X)$, a diferencia de X, es que $P(X)$ es un número, mientras que X puede tener una dimensión muy grande [1]. Cabe aclarar que para calcular este puntaje, solo deberían utilizarse las características en la línea de base, ya que post-tratamiento, estas pueden haberse visto afectadas por el propio programa [4].

Ahora bien, puede surgir la siguiente pregunta: ¿es posible encontrar (al menos) una pareja para todos los individuos tratados? Y la respuesta es no, y para aquellos individuos no será posible estimar el impacto del programa. Lo que puede ocurrir en la práctica es que para algunas unidades inscritas, no haya unidades en el conjunto de no inscritos que tengan puntajes de propensión similares. En términos técnicos, puede que se produzca una falta de **rango o soporte común**.

La condición de soporte común (SC) establece que individuos con el mismo vector de variables X tienen probabilidad positiva de ser tanto participantes como no participantes del programa [1]. Esto es, fijado un vector de variables $\hat{X}$:

$$0 < P(D = 1|\hat{X}) < 1$$

Si esta condición no se cumple, sería posible encontrar una combinación particular de características que predice perfectamente la participación en el programa, y por tanto, no existiría un individuo que fuera un buen control (o viceversa) [1].

El SC implica que solo se utilizarán en la estimación aquellos individuos tratados para los cuales se pueda encontrar uno no tratado con un puntaje de propensión similar. Por ejemplo, si existen individuos del grupo de tratamiento con una probabilidad de participación muy alta, pero ningún individuo no participante exhibe un puntaje tan alto, entonces estos individuos tratados se descartarán a la hora de hacer el emparejamiento.

Asumiendo que se cumplen las condiciones de CI y SC, el estimador del ATT por PSM estaría dado por:

$$ATT_{PSM} = \mathbb{E}_{P(X)|D=1} \{\mathbb{E}[Y(1)|D = 1, P(X)] - \mathbb{E}[Y(0)|D = 0, P(X)]\} \tag{2.12}$$

donde $\mathbb{E}_{P(X)|D=1}$ es el valor esperado con respecto a la probabilidad de participación $P(X)$, condicional en ser participante del programa [1]. Es decir, un promedio ponderado de las diferencias entre corchetes, donde los ponderadores son funciones de la probabilidad de participación en el programa [1].

Los pasos a seguir a la hora de aplicar la técnica de PSM se pueden resumir en los siguientes puntos:

[1] Notar que se omite el subíndice i por simplicidad.

1. Identificar las unidades que se inscribieron en el programa y las que no lo hicieron.

2. Definir el conjunto de variables observables X que se utilizarán para calcular el puntaje de propensión $P(X)$.

3. Calcular el puntaje de propensión $P(X)$ para cada individuo, tratados y no tratados.

4. Restringir la muestra al soporte común con respecto a la distribución del puntaje de propensión.

5. Seleccionar un algoritmo de emparejamiento para emparejar a cada individuo tratado con un individuo o grupo de individuos no tratados que tenga una probabilidad de participación similar.

6. Se calcula el impacto del programa como el promedio apropiadamente ponderado de la diferencia entre la variable de resultado de los tratados y sus parejas no tratadas (Ecuación 2.12).

A continuación, se detalla un poco más en profundidad algunos de estos pasos.

Con respecto al cálculo del puntaje de propensión, la idea es especificar un modelo $P(D = 1|X) = f(X)$, es decir, la probabiliad de participación como una función que puede ser lineal o no lineal en las características observables de los individuos, X [1]. Con frecuencia se prefieren los modelos de regresión logística (*logit*) o de probabilidad (*probit*) [1].

Como mencionamos anteriormente, la gran ventaja del método de PSM es que es flexible, pudiéndose utilizar en numerosos contextos, independientemente de la regla de asignación de un programa. Además, se puede aplicar a un único levantamiento de información, siempre y cuando exista información para los grupos de tratamiento y de no tratamiento [1]. Por lo tanto, se emplea con frecuencia [1].

Sin embargo, es muy importante notar que los resultados arrojados por el PSM serán confiables solamente si se cumple el supuesto de IC [2]. Es decir, cuando existan razones para pensar que las variables no observables o no disponibles en la base de datos no son un determinante fundamental tanto de la participación en el programa como de las variables de resultado potenciales. En otros términos, cuando no haya diferencias sistemáticas en las características no observables entre las unidades tratadas y las de comparación pareadas que puedan influir en el resultado.

Cabe aclarar que el supuesto de IC no puede ser demostrado, ya que no se puede verificar si lo no observado afecta o no la decisión de participación, pero su plausibilidad puede ser argumentada basándose en el conocimiento sustantivo del área de estudio. Otra alternativa que se lleva a cabo en la práctica es calcular las diferencias en las variables

[2]Si bien es deseable que se cumpla el supuesto de SC, para "solucionarlo" basta con restringir la muestra a aquellos individuos tratados para los que sí se pudo encontrar un control adecuado.

observables X; si los dos grupos son demasiados diferentes en estas, esto podría ser evidencia de que es probable que también existan diferencias entre los dos grupos en características no observadas [1].

En este trabajo, utilizaremos al PSM como referencia para compararlo con nuestro enfoque, como explicaremos más en detalle en la sección 2.3.

2.2. Inteligencia Artificial

La definición más general acerca de la Inteligencia Artificial (IA) establece que es el área de las Ciencias de la Computación que se enfoca tanto en entender como en crear sistemas que simulen comportamientos *inteligentes*. Si bien existen distintas perspectivas sobre qué significa que una computadora actúe de manera inteligente, la que más ha prevalecido a lo largo de los años es aquella que refiere con la capacidad de computar cómo actuar de la mejor manera posible en una determinada situación [15].

En sus comienzos, los métodos desarrollados en el área de la IA estaban principalmente **basados en conocimiento**, es decir reglas matemáticas formales que permitían a las computadoras llevar a cabo inferencias lógicas y de esta forma resolver problemas que eran intelectualmente difíciles para los humanos [5].

Sin embargo, determinar reglas que describan la complejidad y diversidad de la realidad no era una tarea fácil. De esta manera, con el objetivo de hacer a estos sistemas más flexibles y capaces de adaptarse y entender diferentes situaciones, se transicionó hacia un enfoque en el que estos pudieran obtener su propio conocimiento aprendiendo patrones directamente a partir de los datos en lugar de depender exclusivamente de reglas predefinidas.

Este cambio de paradigma dio lugar a lo que hoy se conoce como **Aprendizaje Automático**, la subdisciplina de la IA que permite a los algoritmos mejorar su desempeño en una tarea específica automáticamente a partir de la experiencia. Diversos factores como la creciente disponibilidad de datos, el aumento en la capacidad computacional, y los avances en los algoritmos de optimización [5] han hecho que esta área sea la que mayor desarrollo e impacto ha tenido durante las últimas décadas.

Actualmente, la IA abarca una diversidad de tareas, que van desde lo general, como son las habilidades de aprendizaje, razonamiento, y percepción, entre otras; hasta lo específico, como probar teoremas matemáticos, manejar vehículos, mejorar procesos industriales o incluso diagnosticar enfermedades.

2.2.1. Aprendizaje Automático

El Aprendizaje Automático, más conocido por su nombre en inglés *Machine Learning* (ML), es un campo dentro de la IA cuyo objetivo es desarrollar técnicas que permitan que las computadoras *aprendan* automáticamente a partir de la *experiencia* - los datos -,

sin la necesidad de ser explícitamente programadas para hacerlo.

Un ejemplo de estos algoritmos puede ser un clasificador de correo spam, que aprende a distinguir correos spam de regulares viendo ejemplos de cada tipo de correo. Otro ejemplo puede ser un sistema que aprenda a predecir la edad de una persona a partir de una imagen habiendo experimentado previamente algunos ejemplos de imagen-edad.

En 1997, Tom Mitchell definió en su libro Machine Learning [9] el concepto de "aprender" de la siguiente manera: "Se dice que un programa de computadora aprende de la experiencia E con respecto a una clase de tareas T y una medida de desempeño P, si su desempeño en las tareas de T, medido por P, mejora con la experiencia E". Para tener un mejor entendimiento, nos enfocamos a continuación en cada uno de estos tres componentes.

Las tareas de ML se describen usualmente en términos de cómo el sistema debería procesar un *ejemplo* o *entrada*, entendiendo a esta como un conjunto de características (o en inglés, *features*) medidas cuantitativamente a partir de un cierto objeto o evento [5]. Por ejemplo, una entrada puede estar compuesta por los datos de un hogar, como la cantidad de habitaciones y de baños, si tiene patio o no, el tamaño de la cocina, etcétera. Dentro de las tareas que pueden ser resueltas por un sistema de ML se encuentran la clasificación, la regresión, la traducción, la detección de objetos en imágenes, la generación de nuevos datos, la detección de valores atípicos, entre muchas otras.

La experiencia hace referencia al tipo de información que el algoritmo "puede ver" durante su proceso de aprendizaje o *entrenamiento* [3]. A esta información se la conoce como "conjunto (de datos) de entrenamiento", y es un subconjunto del *dataset*, que es simplemente el conjunto de todos los ejemplos o datos con los que se cuenta. En base a la experiencia, los algoritmos de ML se clasifican en dos grandes categorías:

- **Algoritmos de Aprendizaje Supervisado**: experimentan un dataset que contiene pares de entrada-salida, esto es cada ejemplo contiene sus features pero también su "etiqueta", que vendría a ser la "respuesta correcta" para dicha entrada. El algoritmo aprende una función que asigna (*mapea*) entradas a salidas. Las tareas más comunes llevadas a cabo con este tipo de aprendizaje son las de regresión, en donde la etiqueta corresponde a un valor continuo; y clasificación, en donde la solución viene dada por la categoría (dentro de un conjunto predefinido de categorías) a la que pertenece un ejemplo.

- **Algoritmos de Aprendizaje No Supervisado**: ven un dataset que cuenta solamente con características de cada entrada pero sin etiquetas, e intentan aprender automáticamente patrones y propiedades útiles de la estructura de los datos. Algunas tareas que se llevan a cabo con este tipo de aprendizaje son *clustering*, reducción de dimensionalidad, y detección de anomalías.

También existen otros tipos de algoritmos, como los de **Aprendizaje Semi-supervisado**, en donde el dataset contiene algunos ejemplos etiquetados y otros sin etiquetas; y los de

Aprendizaje por Refuerzo, en donde el algoritmo aprende la mejor estrategia para una situación a partir de la interacción con su entorno en forma de recompensas y castigos.

Por último, para medir el desempeño de estos algoritmos, se definen métricas cuantitativas que dependen de la tarea que se esté realizando y del objetivo que se intente lograr. Por ejemplo, en clasificación, una de las más comunes es la de exactitud (*accuracy*), que es la proporción de ejemplos para los cuales se predijo la salida correcta (dada por el dataset); aunque también existen otras como la precisión, sensibilidad, especificidad, y el puntaje F1, que pueden resultar más adecuadas según el dominio del problema. Por otro lado, en tareas de regresión, algunas métricas que se suelen utilizar son el Error Cuadrático Medio y el Error Absoluto Medio.

El objetivo fundamental del ML es que un algoritmo actúe correctamente ante nuevas entradas desconocidas, es decir que **generalice** más allá de los ejemplos del conjunto de entrenamiento. Por ello, aunque las métricas anteriores pueden calcularse durante el entrenamiento para verificar que el algoritmo esté mejorando, su verdadero desempeño se evalúa en un subconjunto del dataset distinto al de entrenamiento, denominado "conjunto de test". Como buena práctica, este conjunto debe permanecer completamente separado del proceso de aprendizaje para obtener una estimación realista de la capacidad de generalización del algoritmo.

Habiendo presentado una idea general sobre cuál es el objetivo de los algoritmos de Aprendizaje Automático, ahora nos enfocaremos en los de Aprendizaje Supervisado, que son los que usaremos en este trabajo.

2.2.2. Aprendizaje Supervisado

Como mencionamos anteriormente, en el Aprendizaje Supervisado el algoritmo aprende una función que mapea entradas a salidas a partir de un conjunto de entrenamiento compuesto por datos etiquetados. El objetivo es que al presentarle a esta función una nueva entrada no vista previamente, esta sea capaz de computar la salida que mejor se ajusta a los **patrones** aprendidos durante el entrenamiento.

Resulta conveniente introducir en este punto un término muy utilizado en el ML: **modelo**. Un modelo es simplemente una ecuación matemática, que se presenta como una forma simplificada de describir hechos de la realidad. En este contexto, será una manera de intentar capturar las relaciones entre los datos, con el objetivo de hacer predicciones basadas en los patrones aprendidos de ejemplos previos. Habiendo presentado este concepto, en general se suele utilizar la palabra modelo para referirse a la función de la que hablamos en el párrafo anterior.

Formalmente, una tarea de Aprendizaje Supervisado es la siguiente:

Dado un conjunto de entrenamiento de N ejemplos de entrada-salida:

$$(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), ..., (\mathbf{x}_N, \mathbf{y}_N)$$

donde cada $\mathbf{x}_i$ es un vector de características ($\mathbf{x}_i \in \mathbb{R}^n$ para algún $n \in \mathbb{N}$) e $\mathbf{y}_i$ es la salida correspondiente ($\mathbf{y}_i \in \mathbb{R}^m$ para algún $m \in \mathbb{N}$), y cada par fue generado por una **función desconocida** $\mathbf{y} = f(\mathbf{x})$, descubrir una función h que aproxime a la función real f [15].
Denotaremos con $\mathbf{X}$ al vector de entradas del conjunto de entrenamiento, y con $\mathbf{Y}$ al vector de salidas, ambos de tamaño N y bien ordenados. Es decir:

$$\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N)$$
$$\mathbf{Y} = (\mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_N)$$

De esta forma, el modelo es la función h, que toma un vector de entrada $\mathbf{x}$ y devuelve una salida $\mathbf{y}$, y se presenta como una **hipótesis** de f. La suposición sobre la que se trabaja es que si el modelo funciona bien para los pares de entrenamiento, entonces se espera que va a realizar buenas predicciones para nuevas entradas cuya etiqueta se desconoce.

Ahora bien, esta función h se obtiene a partir de un **espacio de funciones**, que es elegido por quien diseña el modelo. Este espacio podría ser el conjunto de funciones lineales, de polinomios de grado 2, de polinomios de grado 3, etcétera. Por lo tanto, el espacio determina la forma o "**arquitectura**" que va a tener la función h, y consecuentemente cuáles son sus parámetros, cuyo vector denotaremos con la letra $\boldsymbol{w}$, y a los que también se suele llamar ""**pesos**". De esta forma, este espacio determina la familia de posibles relaciones entre entradas y salidas, y los parámetros especifican la relación particular (el modelo).

Por ejemplo, si suponemos que cada entrada $\mathbf{x} \in \mathbb{R}$, y establecemos como espacio el conjunto de funciones lineales, entonces la forma de h será:

$$h(\mathbf{x}) = w_1\mathbf{x} + w_0$$

y sus parámetros $\boldsymbol{w} = (w_0, w_1)$. Este caso es comúnmente conocido como regresión lineal unidimensional (ya que la entrada es simplemente un número real). Dentro de este espacio, distintas asignaciones de valores a los parámetros generan distintos modelos. Por ejemplo, tomando la asignación , tenemos el modelo $h(x) = 5\mathbf{x}+13$, y tomando , $h(\mathbf{x}) = \pi\mathbf{x}+1{,}25$, entre muchos otros.

En cambio, si tomamos el conjunto de polinomios de grado 2 y seguimos suponiendo $\mathbf{x} \in \mathbb{R}$, entonces la forma de h será:

$$h(\mathbf{x}) = w_2\mathbf{x}^2 + w_1\mathbf{x} + w_0$$

y sus parámetros $\boldsymbol{w} = (w_0, w_1, w_2)$. A este caso se lo suele llamar regresión polinómica de segundo grado unidimensional.

Otro caso un poco más "complejo" podría ser en el que $\mathbf{x} \in \mathbb{R}^3$, es decir $\mathbf{x} = (x_1, x_2, x_3)$ y seguimos tomando una relación lineal. Aquí, h sería:

$$h(\mathbf{x}) = w_1x_1 + w_2x_2 + w_3x_3 + w_0$$

y sus parámetros $\boldsymbol{w} = (w_0, w_1, w_2, w_3)$[3].

Notemos entonces que, fijado un espacio de funciones, quienes van a determinar qué modelo es "mejor" que otro en este espacio, es decir qué modelo aproxima mejor a f, son los valores de los parámetros w_i. Por lo tanto, h en realidad es una función de la entrada $\mathbf{x}$ pero también de los parámetros, establecidos por el espacio de funciones elegido:

$$h(\mathbf{x}, \boldsymbol{w})$$

Y lo que querremos idealmente es:

$$f(\mathbf{x}) \approx h(\mathbf{x}, \boldsymbol{w})$$

Entonces, cuando un modelo se entrena o aprende, lo que realmente hace es intentar encontrar los valores de los parámetros que describan la verdadera relación entre entradas y salidas [11]. Un algoritmo de aprendizaje toma el conjunto de entrenamiento y manipula los parámetros hasta que las predicciones dadas por él sean lo más cercanas posible a las etiquetas verdaderas.

Para que el algoritmo sepa cómo modificar estos parámetros para mejorar sus predicciones, necesita una forma de saber cómo está siendo su desempeño. Para esto, se define lo que se conoce como **función de pérdida** o **función de error**, que denotaremos con la letra L y que justamente hace eso: retorna un número que resume qué tan bien o mal está funcionando el modelo con sus parámetros $\boldsymbol{w}$ actuales, en términos de qué tan lejos están sus predicciones de las respuestas reales del conjunto de entrenamiento.

Una vez establecida su forma, la *performance* del modelo va a estar dada por los valores actuales de sus parámetros. Por lo tanto, tiene sentido tratar a la función de pérdida como una que depende de los parámetros del modelo, es decir $L(\boldsymbol{w})$[4]. A continuación, se mencionan dos ejemplos de funciones de error:

- Una función de pérdida que se es común usar en problemas de regresión es la de Error Cuadrático Medio (ECM), dada por:

$$ECM(\boldsymbol{w}) = \frac{1}{N} \sum_{i=1}^{N} \left(h(\mathbf{x}_i, \boldsymbol{w}) - \mathbf{y}_i\right)^2$$

- Para problemas de clasificación binaria como el que tratamos en este trabajo, vamos

[3]Para simplificar la notación, lo que se suele hacer es asumir que la entrada $\mathbf{x}$ tiene un componente adicional constante x_0 con el valor 1, es decir $\mathbf{x} = (x_0, x_1, x_2, x_3) = (1, x_1, x_2, x_3)$. De esta forma, podríamos escribir: $h(\mathbf{x}) = \mathbf{x} \cdot \boldsymbol{w}$, donde el operador $\cdot$ representa el producto escalar. Retomaremos esta idea más adelante.

[4]En realidad la función de pérdida también depende de los datos de entrenamiento, por lo que si somos estrictos deberíamos escribir $L(\{\mathbf{X}, \mathbf{Y}\}, \boldsymbol{w})$. Sin embargo, como estos datos están fijos durante todo el proceso de aprendizaje, podemos omitirlos.

a tener $\mathbf{y}_i \in \mathbb{R}$, y se suele emplear la Entropía Cruzada Binaria (ECB), dada por:

$$ECB(\boldsymbol{w}) = -\frac{1}{N}\sum_{i=1}^{N}[\mathbf{y}_i \; log(h(\mathbf{x}_i, \boldsymbol{w})) + (1 - \mathbf{y}_i) \; log(1 - h(\mathbf{x}_i, \boldsymbol{w}))]$$

Sin entrar en detalles, lo que mide esta función es la diferencia entre dos distribuciones de probabilidad, que en este caso son la distribución real de los datos dada por el conjunto de entrenamiento y la distribución predicha por el modelo en su estado actual.

En general, se asume que un valor más bajo de $L(\boldsymbol{w})$ indica un mejor desempeño del modelo, y por lo tanto el objetivo del proceso de entrenamiento se convierte en **encontrar los parámetros $\boldsymbol{w}^*$ que minimicen la función de pérdida** (en los datos del conjunto de entrenamiento):

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} L(\boldsymbol{w})$$

Si luego de la minimización la pérdida es baja, quiere decir que hemos encontrado parámetros que predicen precisamente las salidas de entrenamiento $\mathbf{y}_i$ a partir de los entradas $\mathbf{x}_i$ (con $i = 1, ..., N$). Sin embargo, como explicamos previamente, la evaluación final del modelo se debe hacer con el conjunto de test, sobre el cual se puede también calcular la pérdida.

La pregunta que surge entonces es cómo hacer para lograr esta minimización. Dependiendo del espacio de funciones y de la función de costo seleccionada, puede que exista una fórmula cerrada para $\boldsymbol{w}^*$, que se calcule analíticamente. Por ejemplo, en el caso de la regresión lineal multivariada y tomando como función de pérdida el ECM, se puede demostrar que la solución cerrada es:

$$\boldsymbol{w}^* = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{Y}$$

Sin embargo, si la función de costo tiene muchas variables (parámetros) o los modelos son complejos, resulta más útil recurrir a métodos numéricos para aproximar este mínimo.

El algoritmo más genérico que se utiliza para resolver este problema de optimización es el llamado **Descenso por el Gradiente** (DG). Es muy potente ya que se puede aplicar a cualquier función de pérdida [15], sin importar el espacio de funciones elegido.

La idea del DG es ir modificando los parámetros iterativamente hasta eventualmente llegar a un "valle" de la función de pérdida. El algoritmo se basa en que la dirección dada por el gradiente[5] de una función en un punto es la de máximo crecimiento, y por lo tanto su opuesta es la de máximo decrecimiento. En este caso, la función de la cual se calcula

el gradiente es la de error, con respecto a los parámetros en $\boldsymbol{w}$.

Concretamente, se empieza con un vector de parámetros $\boldsymbol{w}$ con valores arbitrarios que va mejorando gradualmente, tomando un paso a la vez en dirección opuesta al gradiente, con el objetivo de reducir el valor de la función de pérdida, hasta que el algoritmo *converja* a un mínimo (local) de la función de pérdida.

Un hiperparámetro[6] determinante en el algoritmo del DG es el tamaño de los pasos, llamado **tasa de aprendizaje**, que denotaremos con la letra α y supondremos por el momento que se mantiene constante en todo el proceso. Si el valor α es muy pequeño, entonces el algoritmo va a tener que realizar muchas iteraciones para converger [3]. Si en cambio el valor es muy alto, entonces puede ser que vayamos "saltando" de un lado a otro de un valle de la función, haciendo que el algoritmo diverja [3].

De esta forma, la regla del DG está dada por:

$$\boldsymbol{w}_{p+1} = \boldsymbol{w}_p - \alpha \nabla L(\boldsymbol{w})$$

donde p indica el número de iteración actual, ∇L representa el gradiente de la función L, y $\nabla L(\boldsymbol{w})$ es el gradiente evaluado en los pesos del modelo.

Algo importante a notar es que en la forma que lo presentamos hasta ahora, el cómputo del gradiente de la función de peso involucra recorrer todas las muestras del conjunto de entrenamiento, lo cual computacionalmente es "pesado" y puede demorar el tiempo de entrenamiento. Es decir, en cada iteración, el cálculo del gradiente requiere evaluar la contribución de cada una de las muestras, lo que puede ser caro cuando N es muy grande. Para mitigar este problema, existen variantes del DG que buscan aproximar el gradiente utilizando subconjuntos del conjunto de entrenamiento. Retomaremos este tema en la proxima sección de Redes Neuronales.

Resumen

Hasta aquí hemos hablado de cómo está compuesto prácticamente cualquier algoritmo de Aprendizaje Supervisado. Los distintos elementos presentes son:

- Un **conjunto de entrenamiento**, que contiene ejemplos de entrada-salida.

[5]Dada una función g de varias variables, es decir, $g(v_1, v_2, ..., v_M)$, el **gradiente** de g es el vector formado por las derivadas parciales de g con respecto a cada uno de sus parámetros:

$$\nabla g = \left(\frac{\partial g}{\partial v_1}, \frac{\partial g}{\partial v_2}, \ldots, \frac{\partial g}{\partial v_M} \right)$$

donde la notación $\frac{\partial g}{\partial v_i}$ representa la **derivada parcial** de g con respecto a v_i, que mide cómo cambia g cuando se varía v_i mientras se mantienen constantes las demás variables. Algo a notar es que el vector resultante es un vector de funciones, y cuando se lo evalúa en un punto, da la dirección de mayor crecimiento de g desde ese punto.

[6]Hablaremos más adelante sobre qué es un hiperparámetro, pero lo importante es que no es un parámetro que se aprende, como lo son los contenidos en el vector $\boldsymbol{w}$.

- Un **conjunto de test**, que servirá para evaluar el desempeño real del modelo, y nos dará una noción de su capacidad de **generalización** una vez entrenado.
- Un **espacio de funciones**, que determina la forma de la función h y los parámetros $\boldsymbol{w}$.
- Una **función de pérdida** $L(\boldsymbol{w})$, que mide el desempeño del modelo.
- Un **algoritmo de optimización**, que busca minimizar la función de pérdida, siendo el más común el Descenso por el Gradiente.

Con esto, el objetivo es encontrar los parámetros $\boldsymbol{w}^*$ que minimicen la función de pérdida en el conjunto de entrenamiento. Para esto, se comienza con pesos $\boldsymbol{w}$ arbitarios y, haciendo uso del algoritmo de optimización, se los va modificando iterativamente hasta llegar a un mínimo (local o global) de $L(\boldsymbol{w})$.

Se trabaja sobre la hipótesis que minimizando el error en el conjunto de entrenamiento, el modelo tendrá una buena capacidad de generalización. Es decir, habiendo llegado a $\boldsymbol{w}^*$, se espera que el modelo encontrado tendrá un buen desempeño en el conjunto de test.

Algo que no hemos discutido hasta el momento es de la correspondencia del espacio de funciones con el conjunto de datos con el que estamos tratando.

El éxito del Aprendizaje Supervisado depende en gran medida de elegir un espacio de funciones adecuado que sea capaz de capturar la distribución de los datos. Por ejemplo, si elegimos un modelo lineal cuando los datos tienen una relación cuadrática, el modelo no podrá representar correctamente la estructura del problema. Por otro lado, si seleccionamos un espacio de funciones altamente flexibles - o *expresivas* -, como polinomios de grado alto, puede ocurrir que el modelo termine describiendo peculiaridades de los datos de entrenamiento que son atípicas y llevan a predicciones inusuales [11].

Si bien existen otros modelos que permiten identificar relaciones no lineales y no polinómicas, varios de ellos requieren tener un buen conocimiento sobre la estructura de los datos, y muchas veces su desempeño está condicionado a ciertas transformaciones, proceso que se conoce por su nombre en inglés *feature engineering*.

En este punto es donde aparecen los modelos conocidos como **Redes Neuronales**, que permiten describir un espacio de funciones complejas, expresivas, y no lineales sin la necesidad de tener un conocimiento profundo sobre los datos. En cambio, son capaces de aprender las representaciones subyacentes automáticamente. En la sección siguiente, explicaremos el funcionamiento de estos algoritmos.

2.3. Redes Neuronales Artificiales

Las Redes Neuronales Artificiales (RNAs) son un modelo específico dentro del ML, cuya estructura y funcionamiento estuvieron inspirados inicialmente por el intento de ser

modelos computacionales del aprendizaje biológico, es decir modelos de cómo el aprendizaje podría ocurrir en el cerebro [5]. Durante los últimos años, ha sido el área que más desarrollo e impacto ha tenido, principalmente gracias a su versatilidad, potencia y escalabilidad, cualidades que hacen que estos modelos sean capaces de enfrentar problemas grandes y complejos [3] que hasta el momento parecían extremadamente difíciles de resolver, como son el reconocimiento de voz y la detección de objetos.

2.3.1. Breve Contexto Histórico

Aunque su gran éxito ha sido reciente, lo cierto es que la idea de las RNAs data desde 1943, cuando Warren McCulloch y Walter Pitts presentaron en su artículo *A Logical Calculus of Ideas Immanent of Nervous Activity* [8] un modelo computacional simplificado utilizando lógica proposicional acerca de cómo funcionan las neuronas de cerebros animales en conjunto para llevar a cabo cómputos complejos [3]. Presentaron una versión muy simplificada de la neurona biológica, que solamente tenía una o más entradas binarias y una salida binaria.

Posteriormente, en 1958, Frank Rosenblatt presentó una de las arquitecturas más simples de RNAs: el "Perceptrón" [12], el cual servía principalmente para resolver problemas de clasificación en donde los datos son linealmente separables. Su aporte más notorio es que definió un algoritmo para el entrenamiento del Perceptrón que le permitía mejorar automáticamente sus parámetros internos para poder llegar a una solución.

Más tarde, se descubrió que los problemas que no podían ser resueltos por el Perceptrón, sí podían ser resueltos "apilando" múltiples perceptrones, lo cual llevó a la invención del "Perceptrón Multicapa" (PMC), también conocido actualmente como "Red Neuronal de Propagación Directa" [7](del inglés *Feedforward Neural Networks*, FFNN) [5] que conforman el punto de partida de las redes neuronales actuales.

Para explicar la idea y los elementos presentes detrás de estos algoritmos, tomaremos como referencia las redes neuronales Feedforward, y en particular las totalmente conectadas (*fully connected*), que definiremos a continuación.

2.3.2. Red Neuronal Feedforward

Como dijimos anteriormente, las redes neuronales son un modelo particular de Aprendizaje Automático. Aquí, las funciones hipótesis se caracterizan por incorporar **no linealidad** y toman la forma de circuitos algebraicos complejos con conexiones que pueden tener diferentes "intensidades" [15]. La idea principal en estos circuitos es que el "camino" recorrido al realizar el cómputo tenga varios pasos como para permitir que las variables de entrada puedan interactuar de formas complejas. Esto hace que sean lo suficientemente

[7]Si bien estos términos se suelen usar indistintamente, la realidad es que las redes neuronales feedforward tienen leves diferencias con el Perceptrón Multicapa.

expresivos como para poder capturar la complejidad de los datos del mundo real [15].

Más concretamente, estos modelos son llamados *redes* porque el espacio de funciones que proveen está formado en realidad por la composición de varias funciones [5]. Por ejemplo, podríamos tener la composición de tres funciones $f^{(1)}$, $f^{(2)}$ y $f^{(3)}$ para formar la siguiente función:

$$f(\boldsymbol{x}) = f^{(3)}(f^{(2)}(f^{(1)}(\boldsymbol{x}))) \tag{2.13}$$

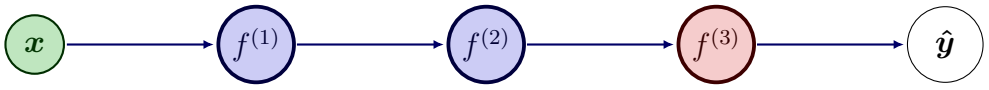
donde $\boldsymbol{x}$ es un vector de dimensión n de números reales: $\boldsymbol{x} = (x_1, x_2, ..., x_n)$, es decir $\boldsymbol{x} \in \mathbb{R}^n$.

Usualmente, se dice que las redes están organizadas en **capas**. De esta forma, en la ecuación anterior, a $\boldsymbol{x}$ se la conoce como **capa de entrada**, a las funciones $f^{(1)}$ y $f^{(2)}$ como **capa ocultas o intermedias**, y a $f^{(3)}$, que es la que produce el resultado final, como **capa de salida**. La longitud de esta cadena de funciones es la que va a dar la **profundidad** de la red.

Las RNAs tienen una capa de entrada y una de salida, pero el número de capas ocultas depende de quien la diseñe. Cuando tienen una capa oculta, se las llama "superficiales" (o poco profundas, del inglés *shallow*), y cuando tienen más de una *profundas*. Es por esto que al hablar de redes neuronales, muchas veces se hablahace referencia al término **Aprendizaje Profundo**.

Lo interesante de estos modelos es que si bien el conjunto de entrenamiento especifica qué tiene que producir la capa de salida ante cada entrada particular, no determina cuál debe ser el comportamiento de las otras capas [5]. En cambio, es el algoritmo de aprendizaje el que tiene que decidir cómo usarlas para lograr una buena aproximación de la función desconocida.

Una forma muy común y más intuitiva de pensar estos modelos es a través de grafos dirigidos cuyas flechas describen cómo están compuestas las funciones y cómo fluye la información a través de ellas. Si hay una flecha que une a dos nodos, diremos que están "conectados". Por ejemplo, la Ecuación 2.13 se representaría de la siguiente manera:



En este caso, la entrada $\boldsymbol{x}$ va a la función $f^{(1)}$, la salida de $f^{(1)}(\boldsymbol{x})$ va a $f^{(2)}$, y la salida de $f^{(2)}(f^{(1)}(\boldsymbol{x}))$ va directamente a $f^{(3)}$ para de esa forma producir el resultado final $\hat{y} = f(\boldsymbol{x}) = f^{(3)}(f^{(2)}(f^{(1)}(\boldsymbol{x})))$.

Es justamente el comportamiento anterior el que caracteriza a las redes neuronales de tipo ***feedforward***: los datos y resultados fluyen en una sola dirección; cada nodo computa su resultado y se lo pasa a su sucesor (o sucesores, como veremos más adelante). En el grafo, esta situación se refleja en el hecho que no hay ciclos, por lo que las redes de este tipo se representan por medio de grafos dirigidos y acíclicos.

Ahora bien, ¿qué es exactamente una capa? En la terminología de redes neuronales, una capa es un conjunto de **unidades** o, tomando en cuenta su inspiración biológica,

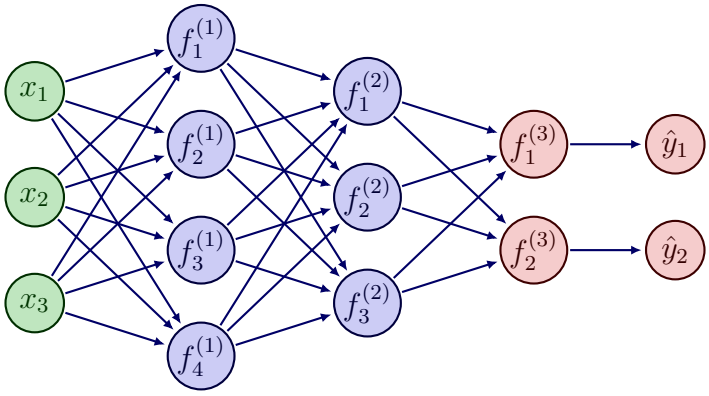


Figura 2.2: Red Neuronal de tipo Feedforward con la capa de entrada formada por 3 neuronas, dos capas ocultas, cada una formada por 4 y 3 neuronas respectivamente, y la capa de salida, formada por dos neuronas. En este caso, la red "acepta" entradas $\boldsymbol{x} \in \mathbb{R}^3$ y produce salidas $\hat{\boldsymbol{y}} \in \mathbb{R}^2$. Adaptado de [10].

neuronas, que actúan en paralelo. Cada unidad representa una función que toma un vector y retorna un escalar, y se asemejan a las neuronas biológicas en el sentido que reciben entradas (o estímulos) de otras unidades y computan su propio valor de activación [5].

Cabe aclarar que la capa de entrada va a tener tantas neuronas como la dimensión de los datos de entrada ($\boldsymbol{x}$). Es decir, si $\boldsymbol{x}$ es de dimensión n, entonces la capa de entrada va a tener n unidades. Sin embargo, la cantidad de neuronas de cada capa oculta depende del diseño de la red, y la de la capa de salida depende sobre todo del problema que se esté tratando de resolver. Si se trata por ejemplo de un problema de clasificación, entonces la capa de salida va a tener en general tantas neuronas como categorías existan en el dominio del problema.

Teniendo el concepto de neuronas, podemos introducir el de redes **totalmente conectadas** (*fully connected*), que son aquellas en las que cada unidad de una capa se conecta con todas las de la capa siguiente.

Con esto en mente, podemos concretizar un poco más el ejemplo con el que venimos trabajando suponiendo que $\boldsymbol{x}$ es un vector de dimensión 3, las capas ocultas dadas por $f^{(1)}$ y $f^{(2)}$ tienen 3 y 4 neuronas respectivamente y la capa de salida tiene 2. Así, suponiendo que nuestra red es fully connected, nuestro grafo resultaría en: donde el superíndice de cada nodo indica el número de capa y subíndice hace referencia al número de neurona en esa capa.

Vemos entonces que cada neurona de la capa de entrada se encarga de recibir una "parte" del vector de entrada[8], pero las neuronas de tanto la capa oculta como la de salida reciben las salidas de las neuronas de la capa anterior. Veamos entonces qué hace precisamente una neurona o unidad.

Una neurona simplemente calcula una suma pesada de sus entradas, provenientes de las unidades de la capa anterior, y luego aplica una función **no lineal** que denotaremos con a para producir su salida. Formalmente, si denotamos con:

- $s_j^{(k)}$ a la salida de la unidad j de la capa k
- $a_j^{(k)}$ a la función no lineal de la unidad j de la capa k
- $w_{i,j}^{(k)}$ la intensidad o **peso** de la conexión entre la neurona i de la capa k y la j de la capa $(k+1)$,

tenemos que:

$$s_j^{(k)} = a_j^{(k)} \left(\sum_i w_{i,j}^{(k-1)} s_i^{(k-1)} \right) \tag{2.14}$$

donde el ínidice i de la sumatoria va a recorrer todas las neuronas de la capa anterior.

La función $a_j^{(k)}$ se denomina **función de activación**, y el hecho que sea no lineal es importante ya que de no ser así, cualquier composición de unidades estaría representando una función lineal [15]. Es justamente esta no linealidad lo que permite a estos modelos representar funciones arbitrarias [15] y complejas. En general, se asume que todas las neuronas de una capa tienen la misma función de activación.

Algunas funciones de activación comunes son las siguientes:

- **Sigmoide**: $\sigma(x) = \frac{1}{1+e^{-x}}$
- **ReLU** (abreviatura de *rectified linear unit*): $\text{ReLU}(x) = \max(0, x)$
- **Tangente hiperbólica**: $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

Actualmente, la más utilizada es la ReLU y variantes de ella.

Si en nuestro ejemplo tomamos la neurona $f_1^{(2)}$, tenemos que su salida estará dada por:

$$\begin{aligned} s_1^{(2)} &= a_1^{(2)} \left(\sum_{i=1}^{3} w_{i,1}^{(1)} s_i^{(1)} \right) \\ &= a_1^{(2)} \left(w_{1,1}^{(1)} s_1^{(1)} + w_{2,1}^{(1)} s_2^{(1)} + w_{3,1}^{(1)} s_3^{(1)} \right) \end{aligned}$$

En las redes, se estipula que cada unidad tiene una entrada extra desde una neurona *dummy* de la capa anterior, para la cual utilizaremos el subíndice 0, cuyo valor de salida se fija en 1 y su peso correspondiente es $w_{0,j}^{(k)}$ para una neurona j de una capa k[9]. Este peso se suele llamar **bias** y permite que la entrada a dicha neurona sea distinta de 0 incluso cuando todas las salidas de la capa anterior sean 0 [15]. Agregándolo, podemos

escribir la Ecuación 2.14 de forma vectorizada:

$$s_j^{(k)} = a_j^{(k)} \left(\left(\boldsymbol{w}_j^{(k-1)} \right)^T \boldsymbol{s}_j^{(k-1)} \right) \tag{2.15}$$

donde $\boldsymbol{w}_j^{(k-1)}$ es el vector de todos los pesos que salen de las neuronas de la capa $(k-1)$ y se dirigen a la unidad j de la capa k (incluyendo $w_{0,j}$) y $\boldsymbol{s}_j^{(k-1)}$ es el vector de todas las salidas de la capa anterior que se dirigen a la unidad j de la capa k (incluyendo el 1 fijo de la neurona dummy).

Si tomamos nuevamente a $f_1^{(2)}$, los vectores involucrados van a ser $\boldsymbol{w}_1^{(1)}$ y $\boldsymbol{s}_1^{(1)}$, dados por:

$$\boldsymbol{w}_1^{(1)} = (w_{0,1}^{(1)}, w_{1,1}^{(1)}, w_{2,1}^{(1)}, w_{3,1}^{(1)}, w_{3,1}^{(1)}),\ \boldsymbol{s}_1^{(1)} = (1, s_1^{(1)}, s_2^{(1)}, s_3^{(1)}, s_4^{(1)})$$

Gráficamente:

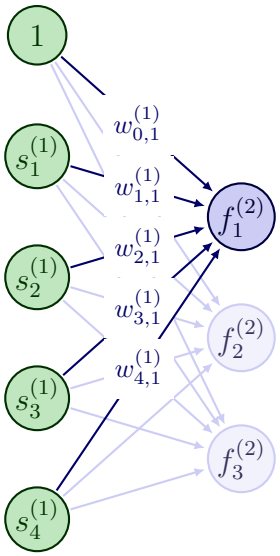


Figura 2.3: ... Adaptado de [10].

Así como utilizamos vectores para describir el cómputo de una neurona, podemos emplear matrices para describir el comportamiento de toda una capa. Para ello, tomemos la siguiente notación, para lo cual resulta conveniente fijar una capa k, que tiene m neuronas:

- $\boldsymbol{s}^{(k)}$ es el vector columna formado por las salidas de la capa k. Es decir: $\boldsymbol{s}^{(k)} = (s_0^{(k)}, s_1^{(k)}, ..., s_m^{(k)})^T = (1, s_1^{(k)}, ..., s_m^{(k)})^T$.

[9] En la bibliografía sobre redes neuronales, también se suele presentar a este peso como un elemento "aparte" de la neurona, no como un peso extra, y se lo denota como $b_j^{(k)}$.

- $\boldsymbol{a}^{(k)}$ es la función de activación de la capa k, con una aplicación elemento a elemento. Es decir: $\boldsymbol{a}^{(k)}(x_1, x_2, ..., x_m) = (a^{(k)}(x_1), a^{(k)}(x_2), ..., a^{(k)}(x_m))$.
- $\mathbf{W}^{(k)}$ es la matriz de pesos que salen de la capa k. Cada fila j de esta matriz corresponde a los pesos que salen de cada neurona de la capa k (incluyendo el bias) y se dirigen a la neurona j de la capa $(k+1)$. Es decir cada fila es $\boldsymbol{w}_j^{(k)}$ con $j = 1, ..., n$, y n la cantidad de neuronas de la capa $(k+1)$. Así, esta matriz tiene dimensiones $n \times m$.

Con esto, tenemos que la salida de una capa k está dada por:

$$\boldsymbol{s}^{(k)} = \boldsymbol{a}^{(k)}\left(\mathbf{W}^{(k-1)}\boldsymbol{s}^{(k-1)}\right)$$

Con todo esto en mente, veamos cuál es la función que describe la red neuronal presentada como ejemplo:

$$\begin{aligned}
\hat{\boldsymbol{y}} &= \boldsymbol{s}^{(3)} \\
&= \boldsymbol{a}^{(3)}\left(\mathbf{W}^{(2)}\boldsymbol{s}^{(2)}\right) \\
&= \boldsymbol{a}^{(3)}\left(\mathbf{W}^{(2)}\boldsymbol{a}^{(2)}\left(\mathbf{W}^{(1)}\boldsymbol{s}^{(1)}\right)\right) \\
&= \boldsymbol{a}^{(3)}\left(\mathbf{W}^{(2)}\boldsymbol{a}^{(2)}\left(\mathbf{W}^{(1)}\boldsymbol{a}^{(1)}\left(\mathbf{W}^{(0)}\boldsymbol{s}^{(0)}\right)\right)\right) \\
&= \boldsymbol{a}^{(3)}\left(\mathbf{W}^{(2)}\boldsymbol{a}^{(2)}\left(\mathbf{W}^{(1)}\boldsymbol{a}^{(1)}\left(\mathbf{W}^{(0)}\boldsymbol{x}^{T}\right)\right)\right)
\end{aligned}$$

Y si queremos profundizar aún más esta ecuación para ver dónde aparece cada peso:

$$\begin{aligned}
\hat{\boldsymbol{y}} = \; & \boldsymbol{a}^{(3)}\left(\mathbf{W}^{(2)}\boldsymbol{a}^{(2)}\left(\mathbf{W}^{(1)}\boldsymbol{a}^{(1)}\left(\mathbf{W}^{(0)}\boldsymbol{x}^{T}\right)\right)\right) \\
= \; & \boldsymbol{a}^{(3)}\left(\mathbf{W}^{(2)}\boldsymbol{a}^{(2)}\left(\mathbf{W}^{(1)}\boldsymbol{a}^{(1)}\left(\mathbf{W}^{(0)}\begin{bmatrix}1\\x_1\\x_2\\x_3\end{bmatrix}\right)\right)\right) \\
= \; & \boldsymbol{a}^{(3)}\left(\mathbf{W}^{(2)}\boldsymbol{a}^{(2)}\left(\mathbf{W}^{(1)}\boldsymbol{a}^{(1)}\left(\begin{bmatrix}w_{0,1}^{(0)} & w_{1,1}^{(0)} & w_{2,1}^{(0)} & w_{3,1}^{(0)}\\ w_{0,2}^{(0)} & w_{1,2}^{(0)} & w_{2,2}^{(0)} & w_{3,2}^{(0)}\\ w_{0,3}^{(0)} & w_{1,3}^{(0)} & w_{2,3}^{(0)} & w_{3,3}^{(0)}\\ w_{0,4}^{(0)} & w_{1,4}^{(0)} & w_{2,4}^{(0)} & w_{3,4}^{(0)}\end{bmatrix}\begin{bmatrix}1\\x_1\\x_2\\x_3\end{bmatrix}\right)\right)\right) \\
= \; & \boldsymbol{a}^{(3)}\left(\mathbf{W}^{(2)}\boldsymbol{a}^{(2)}\left(\mathbf{W}^{(1)}\boldsymbol{a}^{(1)}\left(\begin{bmatrix}w_{0,1}^{(0)} + w_{1,1}^{(0)}x_1 + w_{2,1}^{(0)}x_2 + w_{3,1}^{(0)}x_3\\ w_{0,2}^{(0)} + w_{1,2}^{(0)}x_1 + w_{2,2}^{(0)}x_2 + w_{3,2}^{(0)}x_3\\ w_{0,3}^{(0)} + w_{1,3}^{(0)}x_1 + w_{2,3}^{(0)}x_2 + w_{3,3}^{(0)}x_3\\ w_{0,4}^{(0)} + w_{1,4}^{(0)}x_1 + w_{2,4}^{(0)}x_2 + w_{3,4}^{(0)}x_3\end{bmatrix}\right)\right)\right)
\end{aligned}$$

$$
\begin{aligned}
&= \boldsymbol{a}^{(3)} \left(\mathbf{W}^{(2)} \boldsymbol{a}^{(2)} \left(\mathbf{W}^{(1)} \begin{bmatrix} a^{(1)} \left(w_{0,1}^{(0)} + w_{1,1}^{(0)} x_1 + w_{2,1}^{(0)} x_2 + w_{3,1}^{(0)} x_3 \right) \\ a^{(1)} \left(w_{0,2}^{(0)} + w_{1,2}^{(0)} x_1 + w_{2,2}^{(0)} x_2 + w_{3,2}^{(0)} x_3 \right) \\ a^{(1)} \left(w_{0,3}^{(0)} + w_{1,3}^{(0)} x_1 + w_{2,3}^{(0)} x_2 + w_{3,3}^{(0)} x_3 \right) \\ a^{(1)} \left(w_{0,4}^{(0)} + w_{1,4}^{(0)} x_1 + w_{2,4}^{(0)} x_2 + w_{3,4}^{(0)} x_3 \right) \end{bmatrix} \right) \right) \\
&= (\dots)
\end{aligned}
$$

Entrenamiento

En el caso de las redes neuronales, los parámetros a optimizar van a ser las intensidades de las conexiones entre las neuronas, también llamadas pesos. La gran mayoría de los optimizadores que se utilizan actualmente se basan en la regla del DG, aunque con algunas mejoras que describiremos a continuación.

Como mencionamos en la sección anterior, lo costoso de la regla del DG presentada hasta el momento es que requiere evaluar la función de pérdida considerando *todas* las muestras del conjunto de entrenamiento. Para reducir esta carga, lo que se suele usar en realidad es una técnica conocida como **Descenso por el Gradiente Estocástico** (DGE), que permite obtener una aproximación del gradiente a partir del gradiente promedio de un subconjunto del conjunto de entrenamiento, que recibe el nombre de **lote**. Luego, se actualizan los pesos en base a este estimación. Cabe aclarar que el tamaño de lote, es decir la cantidad de ejemplos que se eligen cada vez, se mantiene fijo en todo el entrenamiento y constituye un hiperparámetro.

Teniendo esto en cuenta, podemos definir el concepto de **época**. Una época se refiere al proceso completo en el cual el modelo se entrena utilizando **todos** los datos disponibles en el conjunto de entrenamiento una vez. A grandes rasgos, los pasos involucrados en una época son los siguientes:

1. Mezclar el conjunto de entrenamiento. Este paso en realidad es opcional pero evita que el modelo aprenda patrones no deseados debido al orden de los datos.
2. Seleccionar un lote del tamaño predefinido del conjunto de datos de entrenamiento.
3. Para cada ejemplo del lote:
   a) Computar la predicción del modelo.
   b) Calcular la pérdida.
4. Promediar el error a lo largo de todos los ejemplos del lote.
5. Calcular el gradiente.
6. Actualizar los pesos.
7. Volver a 2.

8. Una vez que se han recorrido todos los lotes, finaliza la época.

Ahora bien, un paso fundamental en el entrenamiento de las redes neuronales y del que vale la pena profundizar es el del cálculo del gradiente. De hecho, no encontrar una solución eficiente para lograrlo frenó el avance de estos modelos durante varios años. El algoritmo que vino a dar respuesta y que es el utilizado hasta hoy es conocido como **retropropagación** (del inglés *backpropagation* y fue presentado en el año 1986 [14]).

El backpropagation se divide en dos etapas: primero, el paso hacia adelante (*forward pass*) y luego, el paso hacia atras (*backward pass*). Para entender estas etapas, supondremos que solamente una entrada es provista a la red.

En el forward pass, la red simplemente lleva a cabo una predicción para la entrada, realizando todo el cómputo intermedio necesario para llegar a producir una salida.

El backward pass comienza por calcular el error cometido por la red para la entrada dada, y consiste en propagar este error desde la capa de salida hasta la de entrada, midiendo la contribución al error de cada conexión [3]. Este paso se basa fuertemente en la idea que un pequeño cambio en uno de los pesos causa un efecto cadena sobre el cómputo restante de la red [11]. Para verlo, supongamos que tenemos una red neuronal con tres capas ocultas, que denotaremos con $\mathbf{h}_1$, $\mathbf{h}_2$ y $\mathbf{h}_3$, y veamos cómo computaríamos el efecto de un cambio en un peso [11]:

- Para calcular cómo un cambio en un peso que se dirige a $\mathbf{h}_3$ modifica el valor de la pérdida, necesitamos saber (i) cómo un cambio en $\mathbf{h}_3$ modifica la salida $\mathbf{f}$ del modelo y (ii) cómo un cambio en la salida modifica la pérdida.

- Para calcular cómo un cambio en un peso que se dirige a $\mathbf{h}_2$ modifica el valor de la pérdida, necesitamos saber (i) cómo un cambio en $\mathbf{h}_2$ afecta a $\mathbf{h}_3$, (ii) cómo un cambio en $\mathbf{h}_3$ modifica la salida $\mathbf{f}$ del modelo y (iii) cómo un cambio en la salida modifica la pérdida.

- Para calcular cómo un cambio en un peso que se dirige a $\mathbf{h}_1$ modifica el valor de la pérdida, necesitamos saber (i) cómo un cambio en $\mathbf{h}_1$ afecta a $\mathbf{h}_2$, (ii) cómo un cambio en $\mathbf{h}_2$ afecta a $\mathbf{h}_3$, (iii) cómo un cambio en $\mathbf{h}_3$ modifica la salida $\mathbf{f}$ del modelo y (iv) cómo un cambio en la salida modifica la pérdida.

Mientras nos movemos hacia las primeras capas de la red, vemos que la mayoría de los términos que se necesitan ya han sido calculados en pasos anteriores, por lo que no es necesario recomputarlos. Justamente calcular los efectos de los cambios (que son básicamente las derivadas parciales) de esta manera es conocido como el backward pass.

El paso final del backpropagation es actualizar los pesos con los gradientes calculados en el backward pass y utilizando la regla del DG.

Hiperparámetros

Al entrenar un modelo, existen ciertos ajustes que son utilizados para controlar el proceso de entrenamiento y son llamados **hiperparámetros**. Los valores de los hiperparámetros no son aprendidos por el modelo [5], sino que se fijan de antemano y no se modifican a lo largo del entrenamiento.

Algunos de estos hiperparámetros son los siguientes:

- La cantidad de capas ocultas de la red.
- La cantidad de neuronas en cada capa oculta.
- La función de activación de cada capa.
- La tasa de aprendizaje del optimizador.
- El número de épocas de entrenamiento.
- El tamaño de lote.

Todos estos pueden contribuir a que el modelo mejore su desempeño, por lo que surge el problema de cómo hacer para encontrar el valor óptimo de cada uno. Una técnica para hacerlo consiste en construir una tercera partición del conjunto de datos, además de la de entrenamiento y la de test, llamada **conjunto de validación**.

Otra estrategia se denomina **validación cruzada** (*cross-validation*) y consiste en dividir el conjunto de entrenamiento en k partes, y luego entrenar el modelo k veces, cada vez utilizando $k-1$ partes para el entrenamiento y la parte restante para la validación.

En las siguientes secciones, hablaremos sobre dos tipos particulares de redes neuronales de las que hacemos uso en este trabajo: las Redes Neuronales Convolucionales y las Redes Neuronales Recurrentes. Cada una fue diseñada originalmente para trabajar con un tipo específico de datos. Las convolucionales son ideales para procesar datos estructurados en forma de grilla, mientras que las recurrentes son adecuadas para secuencias temporales. Presentaremos la intuición sobre detrás de ellas y nos concentraremos en su aplicación sobre series de tiempo, relevante para nuestro trabajo.

2.4. Redes Neuronales Convolucionales

Como mencionamos anteriormente, las Redes Neuronales Convolucionales (RNC) son un tipo especializado de redes neuronales pensadas para el procesamiento de datos que tienen una estructura de grilla [5]. Ejemplos de estos datos son las series de tiempo, que pueden pensarse como una grilla unidimensional y las imágenes, que pueden verse como una grilla bidimiensional de píxeles.

Si bien las RNCs se han usado principalmente para el procesamiento de imágenes, en este trabajo nos enfocaremos en su uso para analizar series de tiempo, en particular unidimensionales. Veremos cómo su funcionamiento permite capturar patrones temporales en los datos.

Sin entrar en detalles[10], una **serie de tiempo univariada o unidimensional** se puede representar como un vector ordenado U de valores reales, en el que cada valor corresponde a una medición en el tiempo sobre un determinado fenómeno:

$$U = (x_1, x_2, ..., x_N)$$

donde N es la dimensión del vector.

Ahora bien, el término *convolucionales* proviene del hecho que lo que caracteriza a estas redes es la utilización de una operación matemática llamada **convolución**[11][5]. Las capas de la red que aplican esta operación son llamadas *capas convolucionales*.

La convolución toma dos argumentos: por un lado, la *entrada* y por otro lado, un *filtro* o *kernel*. En el contexto del ML, ambos son usualmente arreglos multidimensionales de números reales, también comúnmente llamados tensores [5]. Intuitivamente, la operación consiste en "deslizar" el filtro sobre la entrada, y en cada momento calcular el producto escalar. Para verlo más claramente en el caso unidimensional, tomemos un ejemplo.

Dado como entrada a la convolución un vector $\mathbf{x}$, si tomamos un filtro de tamaño 3, dado por $\boldsymbol{w} = (w_1, w_2, w_3)^T$, entonces el resultado de la convolución es un vector $\mathbf{z}$ en donde cada componente z_i es una suma pesada de las entradas "cercanas":

$$z_i = w_1 x_{i-1} + w_2 x_i + w_3 x_{i+1}$$

En la Figura 2.4, se puede ver cómo se computan z_2 y z_3.

Si generalizamos y tomamos un vector $\mathbf{x}$ de tamaño n correspondiente a la entrada y un vector $\mathbf{w}$ de tamaño l correspondiente al kernel, entonces el resultado de la convolución es un vector $\mathbf{z}$ donde:

$$z_i = \sum_{j=1}^{l} w_j x_{j+i-(l+1)/2} \tag{2.16}$$

En otras palabras, para generar cada componente i de la salida, se toma el producto escalar entre el kernel $\mathbf{w}$ y una parte de $\mathbf{x}$ de largo l centrada en x_i [15].

A partir de aquí, se puede ver que aplicar una convolución resulta en una salida usualmente de menor tamaño que la entrada. La salida de la convolución suele llamarse **mapa de características** (*feature map*) [5], que resalta las áreas de la entrada que son

[10]En una sección posterior, hablaremos más en profundidad sobre las series de tiempo unidimensionales.

[11]En realidad, la operación que está presente en estas redes es una relacionada con la convolución, llamada **correlación cruzada**.

[3]Por conveniencia, el primer elemento de los vectores será indexado en 1, y no en 0.

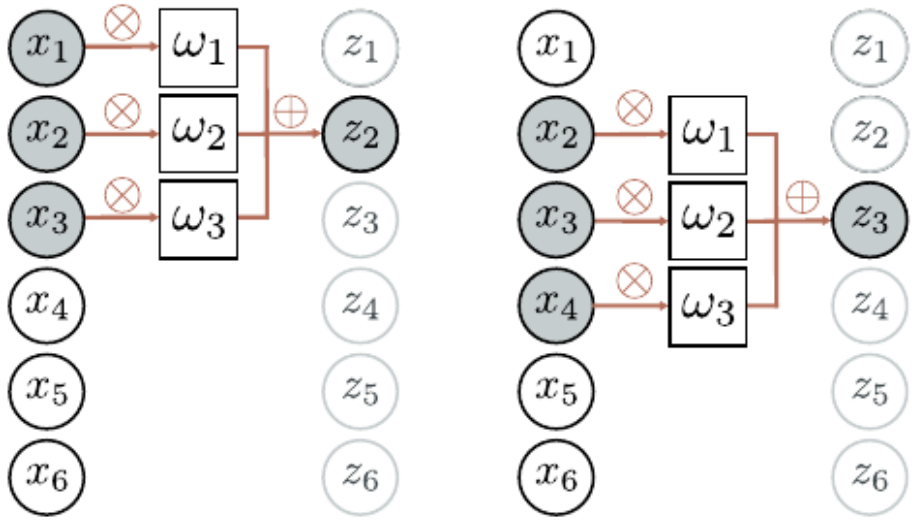


Figura 2.4: Fuente: [11]. Ejemplo de convolución unidimensional con un filtro $\boldsymbol{w}$ de tamaño 3.

"similares" a la característica que el filtro está tratando de capturar [3]. Para ver esto, tomemos otro ejemplo.

Supongamos que tenemos el filtro $\boldsymbol{w} = (1, 0, -1)$ (de tamaño $l = 3$). Entonces, usando la Ecuación 2.16, veamos qué calcula este filtro en cada posición i de la salida:

$$\begin{aligned} z_i =& \sum_{j=1}^{3} w_j x_{j+i-(3+1)/2} \\ =& \sum_{j=1}^{3} w_j x_{j+i-2} \\ =& w_1 x_{1+i-2} + w_2 x_{2+i-2} + w_3 x_{3+i-2} \\ =& 1 \times x_{i-1} + 0 \times x_i + (-1) \times x_{i+1} \\ =& x_{i-1} - x_{i+1} \end{aligned}$$

O sea, lo que hace es dada una posición i, calcular la diferencia entre el "vecino" izquierdo de i y el derecho. De esta forma, si para un i el valor es positivo, quiere decir que hubo una transición de un valor mayor a uno menor, y si es negativo, viceversa. En este caso, se podría decir que el filtro "detecta" regiones de la entrada en donde hubo un cambio de tendencia.

Con esto, podemos ver que un filtro se encarga de identificar una característica específica en la entrada. Y por lo tanto, diferentes filtros actuando en conjunto podrán detectar diferentes características puntuales.

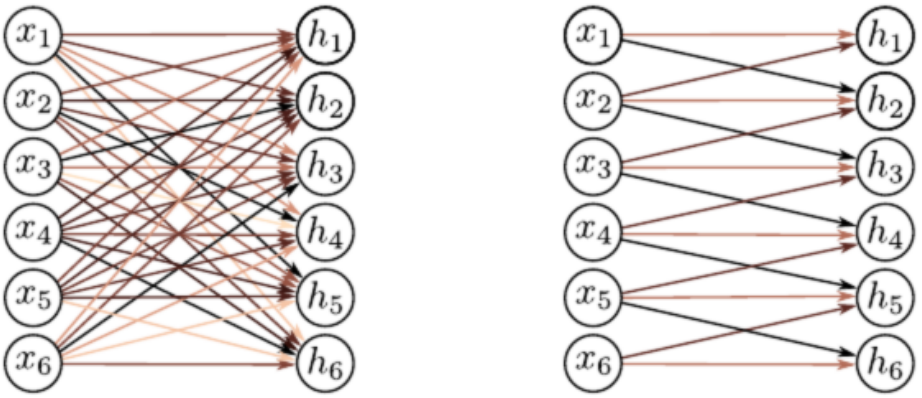


Figura 2.5: Fuente: [11]. Capas de una red fully connected (izquierda) vs capas convolucionales (derecha). Asumimos por comodidad que la capa de las neuronas x_i es la capa de entrada y la de las h_i es la primera oculta. En las capas de la fully connected, vemos que cada neurona de la capa x está conectada con todas las neuronas de la capa h, habiendo un total de $6 \times 6 = 36$ pesos entre ellas. En cambio, en la convolucional, asumiendo que se aplica un filtro de tamaño $l = 3$, tenemos que cada neurona de la capa h computa una suma pesada (con los mismos pesos) de $l = 3$ neuronas consecutivas de la capa x.

Ahora que ya tenemos una intuición clara sobre qué hace una convolución, veamos cómo esta operación se traslada a las redes neuronales. Para empezar, lo que va a ocurrir ahora es que los parámetros a optimizar van a ser los pesos involucrados en cada filtro.

Las redes convolucionales son un ejemplo de red feedforward, ya que la información fluye en una sola dirección. Ahora bien, vimos que en las redes totalmente conectadas, cada neurona de una capa está conectada con todas las de la capa anterior. En cambio, en las capas convolucionales lo que ocurre es que cada neurona está conectada con un subconjunto de neuronas (consecutivas) de la capa anterior, sobre las cuales aplicará el filtro, y que recibe el nombre de **campo receptivo**. Es importante notar que cada neurona de una capa convolucional aplica el mismo filtro, por lo tanto todas ellas comparten los mismos pesos. Esto significa que la cantidad de parámetros a aprender en una red convolucional es mucho menor que en una red totalmente conectada, y es algo que caracteriza a las RNCs. Este comportamiento se puede ver en la Figura 2.5.

Con esta idea de campo receptivo, podemos pensar en una red con dos capas convolucionales seguidas y en las que se aplica un filtro de tamaño 3. Lo que va a ocurrir entonces es que las neuronas de la primera capa oculta toman una suma pesada de conjuntos de 3 neuronas consecutivas de la capa de entrada. Luego, las unidades de la segunda capa oculta toman una suma pesada de conjuntos de 3 neuronas consecutivas de la primera capa oculta, que son a su vez sumas pesadas de 3 neuronas de entrada. Por lo tanto, las neuronas de la segunda capa oculta en realidad tienen un campo receptivo de 5 neuronas. De esta forma, el campo receptivo de las unidades en las capas siguientes va aumentando

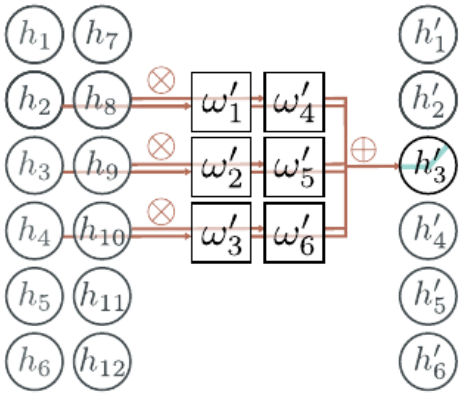


Figura 2.6: Fuente: [11]. La capa de neuronas h_i tiene dos canales: aquel formado por las neuronas h_1 a h_6 y aquel formado por las neuronas h_7 a h_{12}. La capa de neuronas h'_i aplica un filtro de tamaño $l = 3$ sobre cada uno de los canales de la capa anterior. Por lo tanto ahora termina habiendo $2 \times 3 = 6$ pesos entre cada neurona de la capa h'_i y la capa h_i.

[11]. Esto provoca que la primera capa oculta se concentre en características de bajo nivel, más específicas, luego la siguiente en otras de mayor nivel como resultado de integrar las anteriores, y así sucesivamente [3].

Hasta ahora vimos una capa convolucional que aplica solamente un filtro, lo que nos va a dar como resultado solamente un mapa de características. Sin embargo, una misma capa puede aplicar varios filtros (todos del mismo tamaño), cada uno de los cuales va a generar su propio feature map. Para esto, lo que se hace es darle a la capa una "profundidad" de tantos filtros como querramos aplicar sobre la entrada, obteniendo de esta forma un feature map para cada nivel profundidad, cada uno capturando una característica puntual de la imagen. Así, se suele decir que una capa tiene varios *canales*.

Algo que cabe notar en este punto es que cuando una neurona aplica el filtro correspondiente sobre las de la capa anterior, si esta última tiene varios canales, entonces el filtro es aplicado en todos ellos. Es decir, si por ejemplo la capa anterior tenía C canales y el filtro tiene tamaño l, entonces al aplicar el filtro, ahora vamos a tener una matriz de pesos de tamaño $l \times C$. Esto se puede ver en la Figura 2.6.

Además de la cantidad de filtros en una capa, y su tamaño l, existen otros hiperparámetros de las capas convolucionales:

- El paso (*stride*), que representa la distancia entre dos campos receptivos consecutivos. Un stride igual a 1 quiere decir que el filtro se aplica en cada posición de la entrda.
- El relleno (*padding*), que controla cuántos valores "fantasma" se agregan artificialmente a los extremos de la entrada de forma tal que el tamaño de la salida de la convolución no sea tanto menor al de la entrada. En la práctica, se usa el 0 para

este relleno (*zero-padding*).

- La dilatación (*dilation*), que determina cuántos 0s se intercalan en los pesos del filtro. Con este parámetro, se puede convertir un filtro de $l = 5$ a uno "dilatado" de tamaño 3 fijando el segundo y cuarto elementos en 0. Esto permite seguir "integrando" la información de una región de la entrada de tamaño 5 pero requiriendo solamente 3 pesos para hacerlo [11].

Otros de los componentes particulares de las RNCs son las capas de *pooling*, cuyo objetivo principal es reducir el tamaño del mapa producido por la convolución [3]. Es decir, se aplican luego de la convolución y lo que hacen es reemplazar valores contiguos presentes en una determinada sección del mapa por una medida resumen de ellos, entre las cuales algunas comúnmente utilizadas son el máximo (*max pooling*) y el promedio (*average pooling*). Esta reducción de tamaño no solo disminuye la cantidad de parámetros sino que también hace a la red más robusta ante desplazamientos de la imagen [3].

Una capa convolucional típica de una RNC se compone de tres etapas: en primer lugar, se producen las convoluciones, que se encargan de producir activaciones lineales; luego, cada una de estas activaciones pasa por una función de activación no lineal; y finalmente se aplica pooling para reducir (aún más) el tamaño de la salida [5].

De esta froma, una red neuronal convolucional profunda se compone de varias de estas capas.

2.5. Redes Neuronales Recurrentes

Las Redes Neuronales Recurrentes (RNR) son una familia de redes neuronales pensadas específicamente para trabajar con **datos secuenciales** en donde el orden importa. Es por esto que han sido y continúan siendo muy útiles en tareas como análisis de series temporales y procesamiento de lenguaje natural.

A diferencia de las redes feedforward como las que venimos presentando hasta el momento, en donde la información fluye solamente en una dirección, pasando de las neuronas de una capa a las de la capa siguiente, lo que caracteriza a las RNR es la presencia de cíclos en su grafo. Estos cíclos hacen que las neuronas de la red puedan tomar como entradas sus propias salidas de pasos anteriores, permitiéndole a la red de esta forma tener un **estado interno** o **memoria**: entradas recibidas en pasos más empranos afectan la respuesta de la red ante la entrada actual [15].

Para comprender un poco más este comportamiento, tomemos la más simple de las RNR, compuesta por una única neurona (oculta) que recibe la entrada correspondiente al tiempo t, produce una salida y se la envía a sí misma [3], como se puede ver a la izquierda del Diagrama 2.7. Así, en cada paso t, la *neurona "recurrente"* recibe no solo la entrada x_t sino también su propia salida computada en el paso anterior, y_{t-1}. Esto se hace aún más evidente cuando "desenrollamos" la red a lo largo del tiempo, cuya representación se encuentra a la derecha de la Figura 2.7 y se asemeja a la de una feedforward.

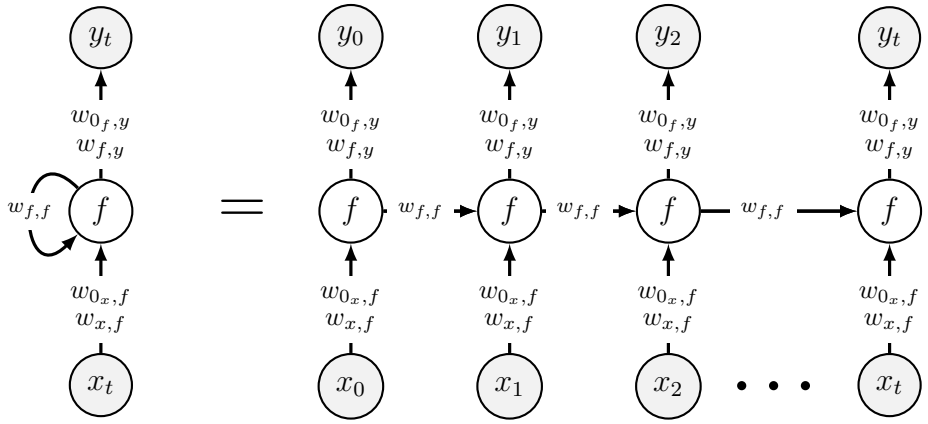


Figura 2.7: A la izquierda, se ve la más simple de las Redes Neuronales Recurrentes, con solamente la capa de entrada, una capa oculta formada por una neurona con su respectiva conexión recurrente, y la capa de salida. A la derecha, se ve la misma red pero "desenrollada" a lo largo del tiempo. Asumiremos por simplicidad que tanto la entrada como la salida en cada momento t son simplemente números real, es decir $x_t, y_t \in \mathbb{R}\ \forall t$. Denotamos con $w_{x,f}$ al peso que va desde la entrada a la neurona oculta, con $w_{0_x,f}$ al asociado a la neurona dummy con valor fijo en 1 que está en la entrada, con $w_{f,y}$ al que va desde la neurona oculta hasta la capa de salida, con $w_{0_f,y}$ al bias entre la oculta y la de salida, y con $w_{f,f}$ al peso que aplica la neurona sobre su propia salida del paso anterior.

Cada neurona realiza lo mismo que en una red feeforward, en el sentido que computan una suma pesada de sus entradas y aplican una función de activación sobre esta. Sin embargo, a partir de la Figura 2.7, podemos notar dos particularidades con respecto a las redes recurrentes:

- Por un lado, cada neurona tiene un peso extra además del que se aplica sobre la entrada x_t, el cual está denotado en la Figura con $w_{x,f}$. Este peso extra corresponde al que se aplica sobre la salida de la neurona en el paso anterior f_{t-1}, denotado con $w_{f,f}$.
- Por otro lado, la red utiliza los mismos pesos $w_{x,f}$, $w_{f,f}$ y $w_{f,y}$ en *todos* los pasos de tiempo.

Con esto en cuenta, vamos a tener que el cómputo llevado a cabo por la red en cada paso de tiempo t está dado por:

$$\begin{aligned} f_t &= a_f\left(w_{x,f}x_t + w_{f,f}f_{t-1} + w_{0,f}\right) \\ y_t &= a_y\left(w_{f,y}f_t + w_{0,y}\right) \end{aligned}$$

donde a_f denota la función de activación seleccionada para la capa oculta y a_y la de la capa de salida. Algo que vale la pena aclarar es que para el primer paso de tiempo, lo

que sería la salida del paso anterior se establace manualmente, como por ejemplo con el valor 0.

Nos concentremos en la salida de la neurona recurrente en el tiempo t, es decir en f_t, dejando la salida de la red y_t de lado. A partir de las Ecuaciones anteriores, podemos ver lo que venimos explicando: f_t es una función de tanto la entrada en el tiempo actual x_t y de su salida en el paso anterior f_{t-1}. Entonces, si tomamos un $t' > 0$ fijo, vamos a tener que:

- $f_{t'}$ es una función de $x_{t'}$ y de $f_{t'-1}$, pero
- $f_{t'-1}$ es a su vez una función de $x_{t'-1}$ y de $f_{t'-2}$, pero
- $f_{t'-2}$ es a su vez una función de $x_{t'-2}$ y de $f_{t'-3}$,
- y así sucesivamente

Este comportamiento hace que $f_{t'}$ sea una función de todas las entradas vistas desde $t = 0$, constituyendo de esta forma una especie *memoria*, que se suele llamar **estado oculto** de la neurona. En este caso, como la red solamente tiene una capa oculta con una neurona recurrente, el estado oculto de la neurona coincide con el estado oculto de la red.

De la misma forma en que lo hicimos anteriormente para las redes feedforward, podemos empezar a complejizar esta red agregando varias neuronas recurrentes en la capa oculta, cada una con su propio ciclo, como se puede ver en la Figura 2.8.

Con esto, vamos a tener que el estado oculto de la red va a ser un vector formado por el estado oculto de las 5 neuronas. O sea, si denotamos con f_t al estado oculto de la red en el tiempo t y con f_{i_t} al estado oculto de la neurona oculta i, en el tiempo t vamos a tener que:

$$f_t = (f_{1_t}, f_{2_t}, f_{3_t}, f_{4_t}, f_{5_t})$$

Y también podemos volver a utilizar la notación matricial para describir el comportamiento de la red:

$$\begin{aligned} f_t &= \boldsymbol{a}_t \left(\mathbf{W}_x x^T + \mathbf{W}_f f_{t-1}^T\right) \\ y_t &= \boldsymbol{a}_y \left(\mathbf{W}_y f_t\right) \end{aligned}$$

donde:

- $\boldsymbol{a}_t$ es la función de activación de la capa oculta, aplicada elemento a elemento, es decir $\boldsymbol{a}_t(x_0, x_1, ..., x_n) = (a_t(x_0), a_t(x_1), ..., a_t(x_n))$.
- $\mathbf{W}_x$ es la matriz de tamaño $k \times (n+1)$ de pesos que salen desde la capa de entrada hasta la capa oculta, incluyendo el bias, donde k es el número de neuronas en la capa oculta y n el tamaño de la entrada.

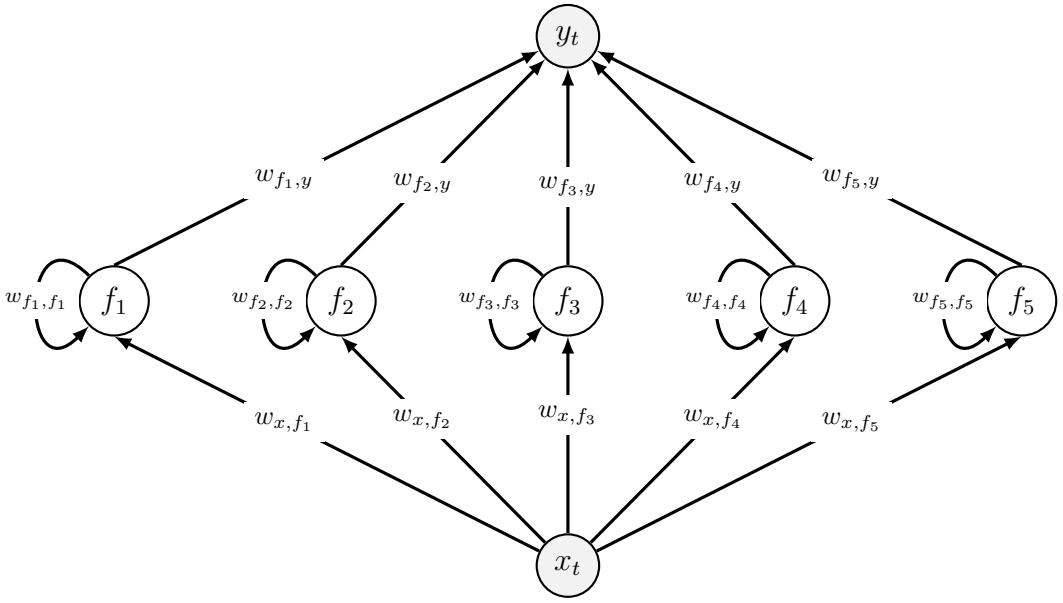


Figura 2.8: Red Neuronal Recurrente con una capa oculta de 5 neuronas recurrentes, cada una con su ciclo. Similarmente al Diagrama anterior, denotamos con ... **TODO**

- x^T es el vector transpuesto de la entrada, junto con el 1 del bias. En este caso, estamos asumiendo $x \in \mathbb{R}$.
- $\mathbf{W}_f$ es la matriz diagonal de tamaño $k \times k$ de pesos recurrentes. En cada elemento de la diagonal i se encuentra el peso que va desde la neurona i a sí misma, es decir w_{f_i,f_i}.
- f_{t-1}^T es el estado oculto de la red en el paso de tiempo anterior.
- $\boldsymbol{a}_y$ es la función de activación de la capa de salida, aplicada elemento a elemento, al igual que $\boldsymbol{a}_t$.
- $\mathbf{W}_y$ es la matriz de tamaño $o \times (k+1)$ de pesos que van desde la capa oculta hasta la capa de salida, incluyendo el bias, donde o es el tamaño de la salida. En nuestra red, tomamos $o = 1$.

Al igual que antes, se pueden agregar más capas de neuronas recurrentes a la red, permitiendo de esta manera que cada capa tenga su propio estado oculto. En este caso, lo que va a ocurrir es que el estado oculto producido por las neuronas de una capa

2.5.1. Long Short-Term Memory

2.5.2. Gated Recurrent Unit

2.6. Clasificación de Series de Tiempo

Una **serie de tiempo** es una secuencia **ordenada** de valores medidos sucesivamente en intervalos de tiempo igualmente espaciados [7]. Este tipo de datos está presente en distintos fenómenos, como lecturas meteorológicas, registros financieros, señales fisiológicas, y observaciones industriales [16].

Formalmente, una serie de tiempo **univariada** U es un conjunto ordenado de valores reales $U = (x_1, x_2, ..., x_N$, y su dimensión está dada por la cantidad N de valores [7].

2.6.1. Clasificación de Series de Tiempo con Aprendizaje Automático

Teniendo una definición de una serie de tiempo, el problema de **Clasificación de Series de Tiempo** (CST) consiste en: dado un conjunto de clases o categorías Y, y un conjunto de datos D formado por series de tiempo univariadas U_i donde a cada una le corresponde una etiqueta $y(U_i) \in Y$, el objetivo es encontrar una función f, a la que llamaremos "clasificador" o "modelo" de forma tal que $f(U) = y(U)$ [7]. Como venimos explicando, un buen modelo será uno que pueda capturar y generalizar el el patrón de las series de datos de forma tal que sea capaz de clasificar correctamente nuevos datos. El repositorio que se suele tomar como referencia para evaluar los diferentes modelos de CST es el de la Universidad de California Riverside, llamado *UCR Time Series Classification Archive* [2].

Durante las últimas décadas, se han propuestos diferentes métodos para afrontar este problema. Los primeros trabajaban directamente sobre las series de tiempo utilizando alguna medida de similitud predefinida que pueda capturar la similitud entre ellas, como por ejemplo la distancia euclideana o la deformación dinámica del tiempo (*Dynamic Time Warping*) [16].

Se han desarrollado también algoritmos basados en características o *features*, en donde cada serie temporal es convertida en un conjunto de features globales, que es posteriormente usando para definir la semejanza entre pares de series de tiempo [7]. Algunos de los más conocidos son ...

El gran problema de todas las estrategias mencionadas es que requieren un gran preprocesamiento de los datos y una ingeniería de atributos (*feature engineering*) [16], para lo cual generalmente se necesita tener un buen conocimiento de campo sobre las series sobre las que se está trabajando.

Es por esto que recientemente, las redes neuronales profundas han comenzado a ser utilizadas para clasificar series temporales [16] ya que permiten trabajar de forma directa

con los datos, y se encargan de detectar automáticamente cuáles son las características que distinguen a las series. Algunas arquitecturas utilizadas hasta el momento son el perceptrón multicapa [16], redes neuronales convolucionales [16], y estas últimas combinadas con unidades LSTM unidireccionales [6] y bidireccionales [7].

Capítulo 3

Presentación del problema

Como explicamos anteriormente, el principal problema en la Evaluación de Impacto de tratamientos sin asignación aleatoria es el de la identificación de un grupo de control o comparación adecuado, que cumpla las características descritas previamente.

Cuando el ingreso al tratamiento por parte de los individuos se realiza en un único instante de tiempo, el grupo de control obtenido mediante la técnica de PSM conforma el mejor estimador del contrafactual. Sin embargo, cuando la entrada al programa ocurre de manera secuencial, los modelos logísticos no son capaces de gestionar esa probabilidad variable en el tiempo. Lo que se hace entonces es estimar la probabilidad por camada de ingreso al programa, es decir se agrupan individuos que hayan ingresado al tratamiento en temporalidades similares, y se buscan controles separadamente para cada una de estas camadas.

En este trabajo, exploramos una alternativa distinta para este último caso. Proponemos la utilización de redes neuronales, específicamente del tipo LSTM **y convolucionales????**, capaces de incorporar la dimensión temporal de los datos, para la detección automática de grupos de control, en el caso particular en que el programa en cuestión se implementa **secuencialmente** y existe una **alta dependencia temporal** entre una variable observada de los individuos y el momento de ingreso al tratamiento. El principal objetivo es evaluar el potencial y la efectividad de estas redes para capturar dependencias dinámicas y características temporales inherentes a los datos observados, para de esta forma mejorar el proceso de inferencia causal en la evaluación de impacto.

Capítulo 4

Marco Experimental

4.1. Conjuntos de Datos

A continuación, detallamos la metodología utilizada para llevar a cabo las simulaciones y medir el desempeño de las redes, y explicamos cómo este problema se termina enmarcando dentro de una clasificación de series de tiempo.

El primer paso para poder llevar a cabo los experimentos es contar con datos. Nuestro entorno - *framework* - de simulación está diseñado para generar **datos sintéticos** que imiten escenarios reales en donde la asignación al tratamiento es no aleatoria, escalonada y potencialmente dependiente de resultados pasados.

El proceso de generación de datos involucra la creación de una serie de tiempo univariada para cada individuo, que incorpora efectos fijos, componentes autoregresivos e impactos del tratamiento. Trabajamos bajo el supuesto que el comportamiento dinámico de la variable modelada para cada unidad es el que determina si esta ingresa al tratamiento o no, y es la misma sobre la que se espera que el tratamiento tenga un efecto.

Los parámetros para la generación de los datos son los siguientes:

- `n_sample`: cantidad de individuos en la simulación.
- `treated_pct`: porcentaje de individuos tratados.
- `control_pct`: porcentaje de individuos de control.
- `T`: cantidad de períodos observados de cada individuo.
- `first_tr_period`: primer período de tratamiento (o único, dependiendo de la cantidad de cohortes).
- `n_cohorts`: cantidad de cohortes.
- `phiT`: persistencia auto-regresiva para los tratados.
- `phiC`: persistencia auto-regresiva para los controles.

- `n_dep_periods`: cantidad de períodos de dependencia para la participación en el tratamiento.

El resultado de la simulación es un conjunto de datos de panel compuesto por series de tiempo univariadas de longitud `T` para los distintos tipos de individuos:

- **Individuos de tipo 1**: son aquellos que han recibido el tratamiento en algún período. Notar que en una situación real, son datos con los que contamos. Los llamaremos también "tratados".
- **Individuos de tipo 2**: son aquellos que en los datos sintéticos, sabemos que forman parte del grupo de control. Es importante notar que en un escenario real, no sabemos quiénes son estos individuos sino que son los que tratamos de identificar. Nos referiremos a ellos también como "controles".
- **Individuos de tipo 3**: son aquellas unidades que no han sido tratadas y que en los datos sintéticos, sabemos que no forman parte del grupo de control. A estos también los mencionaremos como "NiNi" (ni tratado ni control).

En el dataset, cada individuo tiene un identificador y su tipo; y aquellos de tipo 1 y 2 tienen como feature extra el período (desde `first_tr_period` hasta `first_tr_period + n_cohorts`) en el que ingresaron al programa. A continuación, se muestran algunos ejemplos, tomando `T = 6`, `first_tr_period = 3` y `n_cohorts = 2`:

En este punto cabe recordar cuál es nuestra meta: a partir de información sobre los individuos que fueron tratados, queremos identificar de entre los no tratados, quiénes son los que podrían formar parte del grupo de control.

También resulta muy importante tener en cuenta que en los datos generados sintéticamente, sabemos quiénes son los controles, pero en la realidad esto es justamente lo que queremos identificar.

Ahora bien, para entrenar y evaluar a nuestros modelos, tomamos en cuenta lo que ocurriría en un escenario real, en el que sabríamos solamente quiénes son los tratados y quiénes los no tratados. Por lo tanto, construimos el conjunto de entrenamiento con la totalidad de los individuos de tipo 1 con etiqueta 1 y algunos de tipo 3 con etiqueta 0, y en el conjunto de test colocamos a todos los de tipo 2, queriendo predecir en ellos un 1, y al resto de tipo 3, queriendo predecir en ellos un 0.

Más aún, como queremos identificar a los grupos de control de cada cohorte,

4.2. Arquitecturas de Redes Neuronales

4.3. Herramientas

A continuación, nombramos las herramientas que nos ayudaron a llevar a cabo los experimentos, desde la generación de datos sintéticos hasta el diseño, entrenamiento y

evaluación de los modelos, junto con la búsqueda de hiperparámetros óptimos.

4.3.1. Hardware

4.3.2. Python

4.3.3. PyTorch

4.3.4. Pandas

4.3.5. Numpy

4.3.6. Jupyter

4.3.7. Optuna

4.3.8. MLflow

Capítulo 5

Resultados

Capítulo 6

Conclusiones y Trabajos Futuros

Bibliografía

[1] Raquel Bernal y Ximena Peña. *Guía práctica para la evaluación de impacto*. 1.ª ed. Universidad de los Andes, Colombia, 2011. ISBN: bernal2011. URL: http://www.jstor.org/stable/10.7440/j.ctt1b3t82z (visitado 10-02-2025).

[2] Hoang Anh Dau et al. *The UCR Time Series Classification Archive*. https://www.cs.ucr.edu/~eamonn/time_series_data_2018/. Oct. de 2018.

[3] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow*. 1st. O'Reilly Media, 2017. ISBN: 9781491962299.

[4] Paul Gertler et al. *La evaluación de impacto en la práctica: Segunda edición*. Banco Internacional para la Reconstrucción y el Desarrollo/Banco Mundial, 2016. DOI: https://doi.org/10.18235/0006529. URL: https://hdl.handle.net/10986/25030.

[5] Ian Goodfellow, Yoshua Bengio y Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[6] Fazle Karim et al. «LSTM Fully Convolutional Networks for Time Series Classification». En: *IEEE Access* 6 (2018), págs. 1662-1669. ISSN: 2169-3536. DOI: 10.1109/access.2017.2779939. URL: http://dx.doi.org/10.1109/ACCESS.2017.2779939.

[7] Riaz A. et al. Khan M. Wang H. «Bidirectional LSTM-RNN-based hybrid deep learning frameworks for univariate time series classification». En: *The Journal of Supercomputing* (2021), págs. 7021-7045. URL: https://doi.org/10.1007/s11227-020-03560-z.

[8] Warren McCulloch y Walter Pitts. «A logical calculus of the ideas immanent in nervous activity». En: *The bulletin of mathematical biophysics* (19430). URL: https://doi.org/10.1007/BF02478259.

[9] Tom M. Mitchell. *Machine Learning*. McGraw-hill, 1997. ISBN: 0070428077.

[10] Izaak Neutelings. *Neural Networks with TikZ*. URL: https://tikz.net/neural_networks/.

[11] Simon J.D. Prince. *Understanding Deep Learning*. The MIT Press, 2024. URL: http://udlbook.com.

[12] Frank Rosenblatt. «The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain». En: *Psychological Review* 65.6 (1958), págs. 386-408. DOI: 10.1037/h0042519.

[13] Donald Rubin. «ESTIMATING CAUSAL EFFECTS OF TREATMENTS IN EXPERIMENTAL AND OBSERVATIONAL STUDIES». En: *Journal of Education Psychology* 66.5 (1974), págs. 688-701. DOI: https://doi.org/10.1002/j.2333-8504.1972.tb00631.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/j.2333-8504.1972.tb00631.x.

[14] David E. Rumelhart, Geoffrey E. Hinton y Ronald J. Williams. «Learning representations by back-propagating errors». En: *Nature* (1986). DOI: 10.1038/323533a0. URL: https://doi.org/10.1038/323533a0.

[15] Stuart J. Russell y Peter Norvig. *Artificial Intelligence: A Modern Approach*. 4, Global Edition. Pearson Education, 2020. ISBN: 9781292401133.

[16] Zhiguang Wang, Weizhong Yan y Tim Oates. *Time Series Classification from Scratch with Deep Neural Networks: A Strong Baseline*. 2016. arXiv: 1611.06455 [cs.LG]. URL: https://arxiv.org/abs/1611.06455.

[17] Howard White. «Theory-based impact evaluation: principles and practice». En: *Journal of development effectiveness* 1.3 (2009), págs. 271-284.