

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

Agosto, 2025

Directores: Dr. Martín Domínguez

Mgter.David Giuliadori



Este trabajo se distribuye bajo una licencia Creative Commons
Atribución - No Comercial - Compartir Igual 4.0 Internacional.

Resumen

La evaluación de impacto es una metodología que se utiliza para cuantificar los efectos que ha tenido la implementación de una política o tratamiento sobre un conjunto de variables objetivo de la población receptora. Su objetivo principal es estimar la diferencia entre los resultados observables en los tratados y lo que hubiera ocurrido con ellos en ausencia del tratamiento. Esta última situación contrafáctica es aproximada utilizando un grupo de control o comparación, que idealmente debe estar libre del sesgo de auto-selección. En tratamientos sin asignación aleatoria, lograr esto es un gran desafío para el cual existen diferentes técnicas, siendo una de ellas el pareamiento por puntaje de propensión (PSM). Tomando este método como punto de comparación, en este trabajo se exploran alternativas basadas en redes neuronales del tipo *long short-term memory* (LSTM) y convolucionales para la identificación de individuos adecuados para formar parte del grupo de control en escenarios específicos donde la asignación al programa depende potencialmente de resultados pasados y la implementación se hace en múltiples cohortes.

Palabras clave: Evaluación de impacto, grupo de control, pareamiento por puntaje de propensión, aprendizaje automático, redes neuronales, series de tiempo.

Abstract

Impact evaluation is a methodology used to quantify the effects that the implementation of a policy or treatment has had on a set of target variables of the receiving population. Its main objective is to estimate the difference between the observable results in the treated and what would have happened to them in absence of the treatment. This last counterfactual situation is approximated by using a comparison or control group, which ideally should be free from self-selection bias. In non-randomized treatments, achieving this presents a great challenge for which different techniques exist, being one of them the propensity score matching (PSM). Taking this method as a benchmark, this work explores alternatives based on long short-term memory (LSTM) and convolutional neural networks to identify suitable individuals for the control group in specific scenarios where program assignment potentially depends on past outcomes and the implementation occurs across multiple cohorts.

Keywords: Impact evaluation, control group, propensity score matching, machine learning, neural networks, time series.

Agradecimientos

A David y Martín, por acompañarme a lo largo de estos meses enseñándome y guiándome, y por su disposición a responder todas las dudas que fueron surgiendo.

A Pancho y Luis por aceptar ser miembros del tribunal y por sus sugerencias para para mejorar este trabajo.

A mi familia, Andrés, María, Sofi y Feli, por su comprensión y apoyo incondicional a lo largo de toda la carrera, festejando mis logros y siempre motivándome a seguir adelante y dar lo mejor de mí; por ayudarme cuando los tiempos no alcanzaban y por darme la posibilidad de estudiar sin otras preocupaciones.

A Juli y Toni, mis amigos y compañeros desde el cursillo de ingreso hasta el último día de cursado. Hicieron que este recorrido sea mucho más lindo; gracias por cada trabajo, explicación, consejo, charla y momento compartido. Espero que dure para toda la vida.

A mis amigos que me dio la facu, en especial a Seba, Valen, Lucas, Anna, Bruno, Lueme, Martín y Leo, por hacer más llevaderos los días largos, y ayudarme siempre que lo necesitaba.

Al resto de mis amigos, por siempre darme un espacio en donde poder distraerme cuando aparecía el estrés, festejar mis logros, apoyarme y muchas veces hacerme ver las cosas de otra manera.

A todos los profesores y ayudantes que me crucé en estos años. Admiro profundamente su vocación y su esfuerzo constante por formar grandes profesionales y científicos, ayudando a los alumnos en absolutamente todo. Ojalá siempre me toquen docentes como los de FAMAF.

A cada persona que pasó por mi vida durante este tiempo, y que de forma directa o indirecta contribuyó a que hoy yo esté acá.

Por último, a la universidad pública, gratuita y de calidad; que me dio la oportunidad de formarme con una educación de un altísimo nivel y que abre las puertas a todas las personas para que puedan alcanzar sus sueños. Es un recurso invaluable que nos ofrece nuestro país y que hay que defenderlo siempre.

Reconocimientos

Este trabajo utilizó recursos computacionales de UNC Supercómputo (CCAD) de la Universidad Nacional de Córdoba (<https://supercomputo.unc.edu.ar>), que forman parte del Sistema Nacional de Computación de Alto Desempeño (SNCAD) de la República Argentina.

Índice general

1. Introducción	11
2. Marco teórico	15
2.1. Evaluación de impacto	15
2.1.1. Estimación del efecto del tratamiento	17
2.1.2. El grupo de control como estimador del contrafactual	19
2.1.3. Evaluaciones experimentales o aleatorias	23
2.1.4. Evaluaciones cuasi-experimentales	24
2.2. Inteligencia artificial	32
2.2.1. Aprendizaje automático	33
2.2.2. Aprendizaje supervisado	35
2.3. Redes neuronales artificiales	41
2.3.1. Breve contexto histórico	42
2.3.2. Red neuronal feedforward	42
2.4. Redes neuronales convolucionales	51
2.5. Redes neuronales recurrentes	57
2.5.1. Long short-term memory	61
3. Presentación del problema	65
4. Marco experimental	69
4.1. Conjuntos de datos	69
4.2. Arquitecturas de redes neuronales	73
4.2.1. Arquitectura 1: Red LSTM	74
4.2.2. Arquitectura 2: Red Convolucional	75
4.2.3. Arquitectura 3: Red Convolucional + LSTM	75
4.3. Métricas	76
4.3.1. Exactitud	77
4.3.2. Precisión	78
4.3.3. Sensibilidad o cobertura	78
4.3.4. Puntaje F1	79

4.4.	Entrenamiento, evaluación y comparación	79
4.5.	Herramientas	81
4.5.1.	Hardware	81
4.5.2.	Python	82
4.5.3.	Jupyter	82
4.5.4.	Pandas	82
4.5.5.	NumPy	83
4.5.6.	PyTorch	83
4.5.7.	Scikit-learn	83
4.5.8.	Optuna	83
4.5.9.	MLflow	84
4.5.10.	Stata	84
5.	Resultados	85
5.1.	Experimento 1: 45 períodos observados, 20 de dependencia	86
5.2.	Experimento 2: 45 períodos observados, 10 de dependencia	88
5.3.	Experimento 3: 45 períodos observados, 5 de dependencia	89
5.4.	Experimento 4: 45 períodos observados, 0 de dependencia	90
5.5.	Experimento 5: 35 períodos observados, 15 de dependencia	93
5.6.	Experimento 6: 25 períodos observados, 10 de dependencia	94
5.7.	Experimento 7: 15 períodos observados, 8 de dependencia	95
6.	Conclusiones y trabajos futuros	97
6.1.	Conclusiones generales	97
6.2.	Aplicación en casos reales	98
6.3.	Trabajos futuros	100
	Bibliografía	101
	Anexo	105

Capítulo 1

Introducción

Cuando se invierten recursos en un proyecto, suele haber un propósito claro por detrás y, con el paso del tiempo, surge la necesidad de cuestionarse si lo invertido tuvo algún efecto real sobre el objetivo que se estaba tratando de lograr. Por ejemplo, si uno decide dedicarle más horas fuera de clase al estudio de una materia, es normal que nos surja la pregunta de si realmente esas horas sirvieron. En el caso de una empresa, quizás se espera que contratando más empleados, se aumenten la productividad y las ganancias. Las políticas públicas tampoco son una excepción: tomemos el caso de un programa de subsidios para la compra de alimentos en un determinado sector de la población; aquí resulta aún más importante analizar si luego de un tiempo, esto ayudó a mejorar, por ejemplo, los niveles de salud de dicho sector.

En este contexto, cobra relevancia la metodología conocida como **evaluación de impacto** [5][9]. La evaluación de impacto es una técnica que permite estimar cuantitativamente los efectos — positivos o negativos — que ha tenido la implementación de un programa, proyecto o tratamiento sobre determinadas variables de interés — las variables objetivo — de la población receptora del mismo. La forma en la que esto se logra es a través de un **análisis contrafactual**: ¿qué hubiera pasado con las variables de la población receptora — los “tratados” — si el programa no hubiera existido?

La situación anterior, que recibe comúnmente el nombre de “el contrafactual”, es sin dudas un escenario hipotético, no observable en la realidad; y es por esto que es necesario aproximarlos de alguna forma. Para ello, una técnica consiste en construir un **grupo de control** o **comparación**. Este debería estar formado por unidades que no han sido beneficiarias del programa pero que, idealmente, antes del inicio del mismo, eran muy similares a quienes sí lo fueron. Con el grupo de control identificado, se puede estimar el **efecto promedio del programa en los tratados** (*ATT*) comparando los resultados de ambos grupos [34].

Dicho esto, el principal desafío en una evaluación de impacto es construir un grupo de control *válido*. Esto es, uno que permita asegurar que la diferencia observada en los resultados se debe pura y exclusivamente al tratamiento en cuestión, y no a otros facto-

res. Si la asignación al programa es aleatoria, este grupo es todo el conjunto de los no tratados (dentro de la población elegible). Sin embargo, en políticas donde la asignación no tiene reglas claras, y se desconocen cuáles son las verdaderas razones que han llevado a las unidades a participar o no del programa, tomar a los no tratados como grupo de comparación puede producir estimaciones sesgadas, resultantes de no haber considerado las diferencias preexistentes entre ambos grupos. Este problema es conocido como **sesgo de autoselección** [5][3].

Existen diferentes métodos que buscan mitigar este sesgo, siendo uno bastante utilizada el **pareamiento por puntaje de propensión** (PSM) [32][12]. El PSM consiste en estimar el puntaje de propensión — también llamado probabilidad de participación — tanto para tratados y no tratados, y construir un grupo de control emparejando a cada tratado con uno o varios individuos no tratados que tengan puntajes similares. Para el cálculo de esta probabilidad, los evaluadores deben seleccionar ciertas variables que consideren determinantes en la decisión de participar.

Ahora bien, cuando se cree que esta decisión puede estar influida por comportamientos observados a lo largo del tiempo en períodos previos a la implementación del programa, los métodos estadísticos utilizados para calcular el puntaje de propensión no son los más efectivos para capturar posibles patrones y dinámicas, y hacer un emparejamiento adecuado.

Es en estos escenarios donde consideramos que los modelos de **aprendizaje automático** [11][36] pueden contribuir a una mejor selección de unidades pertenecientes al grupo de control. El aprendizaje automático es un campo de la inteligencia artificial cuyo enfoque está en desarrollar técnicas que permitan a las computadoras aprender automáticamente a partir de los datos, sin la necesidad de tener que proveerlas de reglas. Esta estrategia facilita la identificación de patrones, incluso cuando pueden parecer muy difíciles de identificar a simple vista.

Cuando los datos a partir de los cuales los algoritmos aprenden consisten de pares de “entrada-salida esperada” — donde la entrada es un conjunto de características que identifican a cada unidad y la salida esperada puede ser por ejemplo un número o una categoría —, se utiliza el término **aprendizaje supervisado**.

Por ejemplo, si se intenta predecir si un alumno va a aprobar o no un examen, cada unidad sería justamente un alumno, y la entrada podría estar compuesta por sus condiciones socioeconómicas, su promedio de notas del año anterior, y el nivel de educación de sus padres. La salida esperada sería “el alumno aprobó o no”. De esta forma, el algoritmo aprende a partir de muchos ejemplos de estos para poder predecir la aprobación del examen de un nuevo alumno.

Un método muy potente dentro del aprendizaje automático que ha tenido un gran desarrollo en los últimos años es el conocido como **redes neuronales artificiales** [23], cuya inspiración estuvo en el intento de modelar el aprendizaje en el cerebro humano. Dentro de estas, existen algunos tipos específicos adecuados para el procesamiento de datos temporales, como las **redes neuronales recurrentes** (en particular, las de tipo

long short-term memory[13]) y las **redes neuronales convolucionales** [28].

En el presente trabajo, proponemos la utilización de estos dos tipos de redes neuronales para la identificación de grupos de control válidos en contextos donde la asignación al programa es no aleatoria, potencialmente dependiente de compartamientos temporales pasados, y cuya implementación se desarrolla en múltiples períodos de tiempo. Con esto, buscamos aportar una herramienta alternativa para mejorar la calidad de las evaluaciones de impacto en este tipo de tratamientos.

Para la evaluación de nuestro enfoque, generamos datos sintéticos diseñados para imitar situaciones reales, y comparamos el desempeño de las redes con el del PSM. Los resultados obtenidos en los diferentes escenarios modelados indican que las redes superan consistentemente al PSM, aunque no siempre con valores aceptables. Específicamente, los mejores valores se vieron en situaciones con una alta cantidad de períodos observados y tendencias prolongadas con “ruido” y en aquellas con una menor cantidad de datos temporales pero con tendencias monótonas. De esta manera, consideramos que nuestro enfoque es prometedor y tiene potencial para aplicarse en casos reales.

A continuación, se describe la estructura del trabajo, presentando a grandes rasgos sobre qué trata cada uno de los capítulos.

Estructura del trabajo

En el Capítulo 2, se presentan los conceptos fundamentales necesarios para comprender el desarrollo de este trabajo. En particular, explicamos en profundidad la metodología de la evaluación de impacto, los problemas que aparecen y algunas de las técnicas utilizadas. Luego, nos enfocamos en el aprendizaje automático, centrándonos en las redes neuronales, y más particularmente en las de tipo LSTM y convoluciones.

Posteriormente, en el Capítulo 3, se describen algunas de las limitaciones que presenta el PSM en los escenarios estudiados en este trabajo y argumentamos por qué consideramos que el enfoque propuesto debería de superarlas.

En el Capítulo 4, se detalla el marco experimental utilizado. Se mencionan todas las decisiones tomadas en las diferentes etapas, desde la generación de datos sintéticos, la búsqueda de hiperparámetros, las métricas de importancia y la forma de comparar el desempeño de las redes neuronales con el del PSM.

En el Capítulo 5, se muestran los resultados obtenidos en los diferentes experimentos llevados a cabo, haciendo especial énfasis en el puntaje F_1 , la precisión y la cobertura, y en los valores de los hiperparámetros hallados en las diferentes simulaciones.

Finalmente, en el Capítulo 6, se presentan las conclusiones derivadas a partir de los resultados, se propone una manera de aplicar nuestra alternativa en casos reales, y se introducen algunos aspectos de nuestro desarrollo que podrían mejorarse en trabajos futuros.

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 pareamiento por puntaje de propensión, que es el que tomaremos como referencia para evaluar nuestra solución propuesta.

Luego, abordamos el área de la inteligencia artificial, enfocándonos particularmente en el campo de los algoritmos de aprendizaje automático. Explicamos cómo funcionan y cuáles son los elementos presentes en ellos haciendo especial énfasis 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.

2.1. Evaluación de impacto

Los programas y las políticas públicas de desarrollo suelen tener un objetivo puntual con respecto a la población receptora, como aumentar los ingresos, mejorar el rendimiento académico o reducir las enfermedades[9]. Saber si estos resultados se logran o no es una pregunta crucial para quienes diseñan e implementan dichos programas, 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” [9]. Su sello

distintivo es que pueden proporcionar **evidencia** robusta y creíble sobre si un programa concreto ha alcanzado o está alcanzando sus objetivos [9]. Algunas de las principales razones por las que se debería promover su uso 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 [9].

Estas evaluaciones 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 [9] sobre un conjunto de **variables de interés o de resultado** — también llamadas variables objetivo — en un conjunto de individuos [5]. Estas variables son aquellas sobre las cuales se espera que el programa tenga un efecto en los beneficiarios [5]. Podrían ser por ejemplo en el caso de un programa de nutrición, la estatura y peso de los individuos; en una política de financiamiento, la cantidad de empleados o de ventas en una empresa; o en un tratamiento de acceso a medicamentos, 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 [5]. 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.

Un ejemplo claro de por qué son necesarias las evaluaciones de impacto¹ es el que describe Howard White con respecto al Programa Integrado de Nutrición en Bangladesh (PINB) [42]. Este programa identificaba a los niños desnutridos y los asignaba a un tratamiento que incluía alimentación suplementaria a los menores y educación nutricional a las madres [5]. Inicialmente, el programa fue considerado como un éxito ya que los datos de monitoreo mostraban caídas importantes en los niveles de desnutrición. Con base en esta evidencia y previo a cualquier tipo de evaluación, el Banco Mundial decidió aumentar los recursos destinados al programa. Sin embargo, las primeras evaluaciones de impacto mostraron que la mejoría de los indicadores no era significativa con respecto a los niveles de otros niños con características comparables que no formaban parte del programa. Estos resultados reflejaron que las percepciones de los administradores del programa y de las entidades financiadoras eran erradas, y sugirieron algunos correctivos al programa [5].

Otro ejemplo que evidencia la importancia de estas evaluaciones es proporcionado por [18], donde se evaluó el efecto de un programa de preescolar comunitario en zonas rurales de Mozambique, lanzado en 2008 por la organización *Save the Children*. El programa consistió específicamente en el financiamiento de la construcción, equipamiento y

¹Se puede encontrar una variedad de evaluaciones de impacto en la página oficial del Banco Mundial y en la de *International Initiative for Impact Evaluation*.

entrenamiento de 67 aulas en 30 comunidades [18]. El objetivo principal era verificar si un desarrollo en la infancia temprana contribuía a preparar a los niños para la escuela y las etapas posteriores de la vida. Los resultados mostraron no solo que la inscripción a la escuela primaria se incrementó en un 24 % en las comunidades tratadas, sino también que los niños beneficiarios dedicaban un promedio de 7.2 horas adicionales por semana a actividades escolares, y redujeron el tiempo destinado a trabajos en las granjas familiares y a asistir a encuentros comunitarios. Además, la participación en el programa preescolar mostró mejoras consistentes en habilidades motoras finas, cognitivas, y de resolución de problemas [18]. Todos estos resultados contribuyeron a reforzar la idea que los programas preescolares constituyen una política prometedora para mejorar la preparación escolar de niños pobres y desfavorecidos en áreas rurales de África.

Habiéndonos enfocado en el *qué* es una evaluación de impacto y su relevancia, a continuación, presentamos *cómo* se lleva a cabo una, detallando los elementos presentes y los problemas que pueden surgir al hacerlo.

2.1.1. Estimación del efecto del tratamiento

Como mencionamos anteriormente, el resultado deseado al llevar a cabo una evaluación de impacto es el efecto del tratamiento sobre un determinado individuo, que se plantea como la diferencia entre el resultado observado y el contrafactual. El marco teórico estándar para formalizar este problema se conoce como **modelo de resultados potenciales** o **modelo causal de Rubin** [34].

En este, se definen dos elementos para cada individuo $i = 1, \dots, N$, donde N denota la población total bajo análisis:

- Por un lado, el indicador de tratamiento D , tal que $D_i = 1$ implica que el individuo i participó del tratamiento, y $D_i = 0$ que no lo hizo.
- Por otro lado, para las variables objetivo se define $Y_i(D_i) = (Y_i|D_i)$ - se lee como “el valor de Y_i dado D_i ”. De esta forma, $Y_i(1)$ es el valor de la variable objetivo si el individuo i es tratado, e $Y_i(0)$ es la variable objetivo 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:

$$\tau_i = Y_i(1) - Y_i(0) = (Y_i|D_i = 1) - (Y_i|D_i = 0) \quad (2.1)$$

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 el valor de 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, solamente uno de los resultados potenciales es observado en la realidad. Más precisamente, 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:

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

Si nos concentramos en las unidades que efectivamente resultaron tratadas, en la Ecuación 2.1, el término $Y_i(0) = (Y_i | D_i = 0)$ es la situación contrafactual ya que representa cuál habría sido el resultado si la unidad no hubiera participado en el programa. Al ser imposible observar directamente el contrafactual, es necesario **estimarlos**.

La forma más directa de solucionar este problema sería hallando un “clon perfecto” para cada uno de los beneficiarios del programa en el grupo de individuos no tratados. Sin embargo, esto puede resultar bastante complicado si son múltiples las características que distinguen a los individuos o si cada una puede tomar muchos valores, lo que se conoce como el *el problema de la dimensionalidad*. Por lo tanto, el primer paso para lograr esta estimación consiste en **desplazarse desde el nivel individual al nivel del grupo** [9], enfocando el análisis en el **impacto o efecto promedio**.

En primera instancia, se puede estimar el **impacto promedio del programa sobre la población** (o *ATE*, del inglés *average treatment effect*):

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

donde el operador \mathbb{E} representa la esperanza de una variable aleatoria. En este caso, la esperanza se toma sobre toda la población elegible para el tratamiento en cuestión.

“El *ATE* se interpreta como el cambio promedio en la variable de interés cuando un individuo escogido al azar pasa aleatoriamente de ser participante a ser no participante” [5]. Es decir, representa el efecto esperado del tratamiento si toda la población recibiera el tratamiento.

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. Es por esto que suele ser más relevante estimar el efecto únicamente en quienes realmente se hicieron beneficiarios del tratamiento.

De esta manera, 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:

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

donde en este caso la esperanza se toma sobre los individuos tratados. 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 [5]. Ahora bien, ¿cómo podemos intentar estimar esta última situación? A continuación, presentamos una estrategia utilizada para esto.

2.1.2. El grupo de control como estimador del contrafactual

En la Ecuación 2.4, el término $\mathbb{E}[Y_i(1)|D_i = 1]$ es un resultado observable mientras que $\mathbb{E}[Y_i(0)|D_i = 1]$ es el promedio contrafactual que debemos aproximar. Para lograrlo, se selecciona lo que se conoce como **grupo de control** o **de comparación**. Idealmente, este grupo debería estar formado por individuos que no han participado del programa pero que, en promedio, eran estadísticamente similares [9] al **grupo de tratamiento** en la línea de base, es decir en el momento previo al tratamiento. La Figura 2.1 ilustra esta distinción.

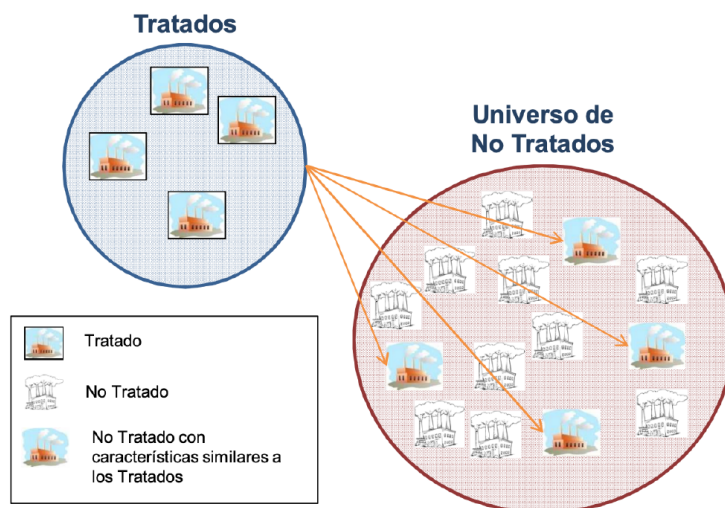


Figura 2.1: El grupo de control debería estar formado idealmente por individuos que no recibieron el tratamiento — “no tratados” — pero que en promedio poseían características similares a los tratados en la línea de base.

Por lo tanto, en la práctica, para obtener una buena estimación del ATT , el reto está en definir un grupo de control *válido*, es decir uno que sea estadísticamente idéntico al de tratamiento, en promedio, en ausencia del programa [9]. 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 [9].

Para que un grupo de comparación sea válido, debe satisfacer lo siguiente [9]:

- 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 comunes en las que esto ocurre.

Comparaciones antes-después

Este tipo de comparaciones intenta establecer el impacto de un programa como la diferencia en los resultados del mismo grupo antes y después de participar de un programa. Estiman el contrafactual con el resultado promedio para el grupo de tratamiento antes que comience la intervención, es decir en la línea de base. Si bien una ventaja de este método es que no es necesario hallar un grupo de control, 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 mismo, lo cual en la mayoría de los tratamientos no puede sostenerse [9].

Comparaciones de tratados y no tratados: 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 tratados en ausencia 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:

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

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 ($D_i = 1$) y para el de no tratados ($D_i = 0$). Indica que los individuos que participaron en el programa no son sistemáticamente distintos de los que no lo hicieron [5] o, más específicamente, que la decisión de

participar en el programa no estuvo determinada por factores que también pueden influir en la variable de resultado.

De esta forma, podríamos calcular el ATT como:

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

Sin embargo, el supuesto de la Ecuación 2.5 se deja de cumplir toda vez que la participación en el programa es una *elección* del individuo elegible [5]. 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.

El problema descrito 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 [9], y por ende, es muy probable que la variable de resultado difiera entre el grupo de tratamiento y no tratamiento, *aún si el programa no existiera* [5]. Así, comparar las medias de ambos grupos podría atribuir al programa un efecto que en realidad se debe a las diferencias preexistentes entre el grupo de tratamiento y el grupo de control [5].

Un buen ejemplo de esta situación lo presenta [5], en el que se plantea un programa de nutrición infantil. Podría ocurrir por ejemplo 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 está 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 posiblemente estén más pendientes de 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:

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

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

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

De esta forma, es claro que si los individuos tratados son diferentes de los no tratados, aún en ausencia del tratamiento (esto es $\mathbb{E}[Y_i(0)|D_i = 1] - \mathbb{E}[Y_i(0)|D_i = 0] \neq 0$), entonces

la diferencia entre sus medias (el lado derecho de la igualdad) será igual al ATT más la diferencia preexistente entre los grupos (el lado izquierdo de la igualdad) [5]. Es decir, esta comparación de medias será una combinación del efecto directo del tratamiento, capturado en el ATT , y las diferencias preexistentes. Y, sin información adicional, no es posible distinguir qué parte se debe a qué.

Tomar un grupo de control que permita asegurar la ausencia del sesgo de autoselección y que los cambios observados se deben exclusivamente a la aplicación del tratamiento es uno de los grandes desafíos en cualquier evaluación de impacto.

La Tabla 2.1 resume algunas de comparaciones relevantes que se observan en una evaluación de impacto una vez que se ha definido el grupo de control. Allí, usamos Y_{C_i} para denotar al valor de la variable Y en un individuo C_i perteneciente al grupo de control

Lo que se desea medir: el ATT .	$\mathbb{E}[Y_i(1) - Y_i(0) D_i = 1]$
Lo que se observa: el promedio de la diferencia entre los resultados de los tratados y el grupo de control.	$\mathbb{E}[Y_i(1) D_i = 1] - \mathbb{E}[Y_{C_i}(0) D_i = 0]$
El promedio de la diferencia entre lo que se observa y lo que se desea medir .	$\mathbb{E}[Y_{C_i}(0) D_i = 0] - \mathbb{E}[Y_i(0) D_i = 1]$

Tabla 2.1: Distinción entre las diferentes comparaciones presentes en una evaluación de impacto.

Teniendo en cuenta las fórmulas expuestas en la Tabla 2.1, la calidad de una evaluación de impacto dependerá de los supuestos necesarios para asegurar que el grupo de control es una buena aproximación del contrafactual, es decir que $\mathbb{E}[Y_{C_i}(0)|D_i = 0] - \mathbb{E}[Y_i(0)|D_i = 1] = 0$.

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

2.1.3. Evaluaciones experimentales o aleatorias

Este tipo de evaluaciones se utiliza cuando los beneficiarios del programa en cuestión son seleccionados de manera completamente aleatoria². Los tratamientos con esta asignación son también llamados ensayos controlados aleatorios.

La asignación aleatoria trae dos consecuencias importantes: por un lado, las unidades ya no pueden *elegir* si participar o no, lo cual **elimina el sesgo de autoselección**, y por el otro, 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. Esto hace que la asignación aleatoria sea el método más sólido para evaluar el impacto de un programa [9].

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. El proceso de asignación aleatoria cumple justamente con este objetivo: siempre que el número de unidades sea lo suficientemente grande, este mecanismo tiende a producir dos grupos que tienen una alta probabilidad de ser estadísticamente idénticos en todas sus características[9]³.

En consecuencia, lo que sucede 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 $\mathbb{E}[Y_i(0)|D_i = 1]$. Es decir, mediante este tipo de asignación, se asegura que se cumple el supuesto de la Ecuación 2.5.

Una vez que se haya asignado el tratamiento de manera aleatoria, gracias a la eliminación del sesgo de autoselección y el consecuente cumplimiento de 2.5, es bastante directo estimar el impacto del programa. Después que el programa ha funcionado durante un tiempo, se miden los resultados promedio de las unidades de tratamiento y de no tratamiento, y el impacto del programa se estima como la diferencia entre ambos, es decir:

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

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 [9]. Además, la asignación aleatoria podría hacerse a nivel individual (por ejemplo, a nivel de hogares), o a nivel de

²Cabe aclarar que la asignación aleatoria se produce dentro de la población de unidades elegibles, que está compuesta por aquellos individuos para los cuales interesa conocer el impacto del programa [9]. Por ejemplo, si se está implementando un programa de formación para los empleados del área comercial de una empresa, aquellos que sean de otros sectores no formarían parte del conjunto de unidades elegibles.

³En la práctica, este supuesto debería comprobarse empíricamente con los datos de línea de base de ambos grupos.

conglomerado (por ejemplo, comunidades) [5].

Las evaluaciones experimentales tienen varias ventajas: por diseño, al quitarles la posibilidad de elección a los individuos sobre si participar o no en el programa, resuelven el problema del sesgo de autoselección, y además los resultados obtenidos son transparentes, intuitivos y no es necesario utilizar herramientas econométricas sofisticadas para alcanzarlos [5].

Sin embargo, este tipo de experimentos sufre varias limitaciones que hace que no sean tan comunes en la práctica. Por un lado, pueden ser costosos y de difícil implementación ya que en general se hacen 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 [5]. Otro problema que tienen, y probablemente el más importante, es que se excluye aleatoriamente a un segmento de la población, igualmente vulnerable y elegible, de los beneficios de la intervención [5]. Además, dada la imposibilidad de negar los beneficios a un mismo grupo de control por largos períodos, con frecuencia es difícil estimar los impactos de largo plazo [5].

2.1.4. 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 [9]. En estos casos, el programa que se intenta evaluar no tiene una regla de asignación clara que explique por qué ciertos individuos se inscribieron en el programa y otros no lo hicieron, por lo que se convierte en una incógnita adicional en la evaluación, acerca de la cual se deben formular supuestos [9]. 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 como el cambio esperado en Y en los tratados entre el período posterior y el anterior a la implementación del tratamiento, menos la diferencia esperada en Y en los no tratados en el mismo período [5]. 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 un período posterior a la implementación (también llamado período de seguimiento), entonces el impacto del programa por el método de DD está dado por⁴ :

$$ATT_{DD} = (\mathbb{E}[Y^{(2)}|D=1] - \mathbb{E}[Y^{(1)}|D=1]) - (\mathbb{E}[Y^{(2)}|D=0] - \mathbb{E}[Y^{(1)}|D=0]) \quad (2.9)$$

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 [9]. 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 ambos grupos [9].

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. Sin embargo, el cambio observado puede haber sido por factores externos o incluso características propias de los individuos que se mantuvieron durante el programa y afectaron el resultado final.

El DD busca solucionar este problema partiendo de la idea que la diferencia en los resultados antes-después para el grupo de tratamiento refleja tanto el efecto del tratamiento como el impacto de factores constantes a lo largo del tiempo en ese grupo. Para “limpiar” este cambio de otros factores que sí varían en el tiempo, se lo compara con el cambio antes-después observado en un grupo que no participó del programa pero que estuvo expuesto al mismo conjunto de condiciones ambientales.

Ahora bien, 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.

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,

⁴Omitimos el subíndice i por comodidad.

resulta conveniente reescribir el estimador de la Ecuación 2.9 como:

$$ATT_{DD} = (\mathbb{E}[Y^{(2)}|D=1] - \mathbb{E}[Y^{(2)}|D=0]) - (\mathbb{E}[Y^{(1)}|D=1] - \mathbb{E}[Y^{(1)}|D=0]) \quad (2.10)$$

De la Ecuación 2.10 se puede ver que 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 puedan haber existido 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.

Pareamiento por puntaje de propensión

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 [9]. 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 [9].

El supuesto que utilizan estas técnicas para basarse en determinadas 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.

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), se logra simular una aleatorización y entonces 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 forma⁵:

$$\mathbb{E}[Y(0)|D=1, X] = \mathbb{E}[Y(0)|D=0, X], \quad (2.11)$$

y se lo conoce como **supuesto de independencia condicional** (IC).

Por ejemplo, si se considera que las características en cuestión son el nivel socioeconómico (bajo, medio y alto) y la cantidad de hijos, entonces lo que supone este método es que individuos con el mismo nivel socioeconómico y la misma cantidad de hijos tendrían, en promedio, el mismo resultado si no hubieran recibido el tratamiento, independientemente de si efectivamente lo recibieron o no.

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 [5]. Un clon en este caso quiere decir, una vez fijadas las características

⁵Nuevamente, omitimos el subíndice i por comodidad.

que el investigador cree que explican la decisión de inscribirse en el programa, un individuo (o grupo de individuos) con exactamente las mismas características observables X .

Sin embargo, como mencionamos anteriormente, 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 complejo identificar una pareja para cada una de las unidades del grupo de tratamiento.

Para solucionarlo, se utiliza un método denominado **pareamiento por puntaje de propensión** (*propensity score matching*, PSM), presentado por primera vez en 1983 [32]. Consiste en emparejar individuos pero no con base en X , sino en su probabilidad estimada de participación en el programa, dadas sus características observables, es decir en:

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

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, la pareja adecuada para cada individuo del grupo de tratamiento será aquel individuo del grupo de no tratados con una probabilidad de participación en el programa suficientemente cercana [5]. 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 [5]. 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 [9].

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 [5]. 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) [5]. Es decir, si por ejemplo para un vector de variables \hat{X} , tenemos que $P(D = 1|\hat{X}) = 1$, (o $\simeq 1$), entonces no podremos encontrar ningún no tratado con esas características, y consecuentemente ningún control.

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.

La Figura 2.2 muestra un gráfico de densidad de los puntajes de propensión del grupo de tratados y del grupo de no tratados (suponiendo que ya se han calculado en base a un determinado conjunto de características), en donde se puede ver que hay falta de rango común. Lo que se hace en estos casos es simplemente restringir el análisis a los individuos que se encuentran en el área de soporte común aunque esto, como mencionaremos más adelante, puede traer problemas.

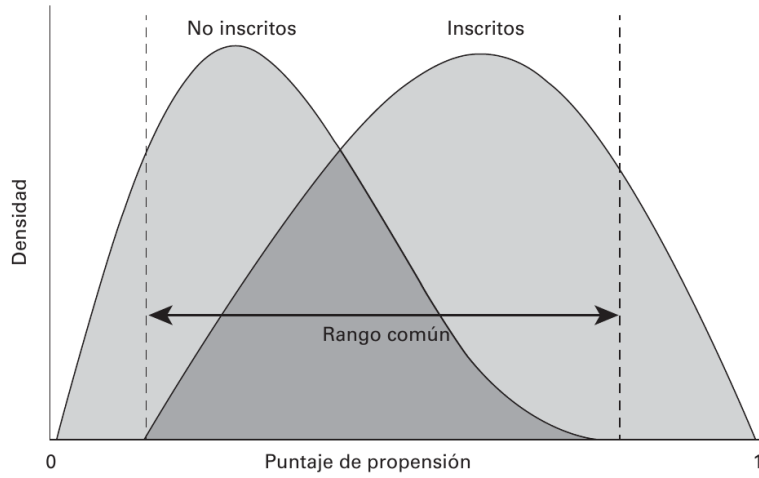


Figura 2.2: Fuente: [9]. Gráfico de densidad de los puntajes de propensión de los tratados y los no tratados. El área entre las líneas punteadas es la región de soporte común, en donde se encuentran tanto inscritos como no inscritos con puntajes de propensión similares. En el área a la izquierda de dicha región solamente hay no inscritos con bajos puntajes de propensión. Es decir, lo que ocurre aquí es que $P(D = 1|\hat{X}) \simeq 0$. Similarmente, en el área a la derecha de la región de soporte común, solamente hay inscritos con altas probabilidades de participar. Es decir, lo que ocurre aquí es que $P(D = 1|\hat{X}) \simeq 1$, y para estos no será posible encontrar ningún individuo de control.

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

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

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 [5]. Es decir, un promedio ponderado de las diferencias entre llaves, donde los ponderadores son funciones de la probabilidad de participación en el programa [5].

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

1. Identificar las unidades que fueron tratadas y las que no.
2. Definir el conjunto de variables observables X que se utilizarán para calcular el puntaje de propensión $P(X)$.
3. Calcular los puntajes de propensión $P(X)$ de todos los individuos, 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. Calcular 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 explican brevemente algunos de estos pasos.

En cuanto a la elección de variables que se incluirán para calcular el puntaje de propensión (paso **2**), si bien es importante poder parear utilizando un gran número de características, lo es más aún hacerlo sobre la base de aquellas que **determinan la inscripción** [9]. En términos estadísticos, estas características reciben el nombre de **covariables, variables independientes o variables explicativas**, y son justamente aquellas que posiblemente predicen el resultado bajo estudio.

Cuanto más se comprenda acerca de los criterios utilizados para la selección de los participantes, en mejores condiciones se estará de construir un buen grupo de covariables [9]. Para tener una mejor comprensión, los investigadores se pueden guiar por los modelos económicos que describen el fenómeno bajo estudio, investigaciones previas y conocimiento del diseño institucional [5]. Además, como dijimos anteriormente, estas características deben ser medidas en la línea de base, y no una vez aplicado el programa, ya que se pueden haber visto afectadas por el mismo.

Con respecto al cálculo del puntaje de propensión (paso **3**), la idea es especificar un modelo $P(D = 1|X) = f(X)$ que permita estimar la probabilidad de participación como una función de las características observables de los individuos, X [5]. Para esto, los modelos de regresión logística (*logit*) o de probabilidad (*probit*) son los más comúnmente utilizados [5]. En este trabajo, utilizaremos la regresión logística, por lo que resulta conveniente saber de qué se trata.

Para empezar, una regresión es un método en el cual se propone una relación fun-

cional entre una variable dependiente y una o varias variables independientes. Una vez establecida la relación, se estiman los coeficientes o parámetros, que representan el peso de cada variable explicativa sobre la variable de interés [10].

Cuando la variable dependiente es categórica (por ejemplo, toma solo los valores 0 o 1), se utiliza la regresión logística. Este modelo permite estimar la probabilidad de que un individuo pertenezca a una determinada categoría en función de sus características observadas. Puntualmente, la regresión logística propone como relación funcional la llamada función logística o sigmoide:

$$f(z) = \frac{e^z}{1 + e^z}$$

El gráfico de esta función se puede ver en la Figura 2.3, donde se observa que siempre da como resultado un valor real en el rango $[0,1]$, por lo que es adecuada para modelar probabilidades.

En la práctica, se toma $z = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_k x_k$, siendo x_j las variables independientes y β_j los coeficientes a estimar, correspondientes a la ponderación que tiene cada una de las variables.

En el caso del PSM, la variable dependiente es D , que toma dos valores: 1 si un individuo fue tratado, y 0 en caso contrario. Y las independientes son aquellas que el investigador seleccionó en el paso **2**, que denotamos con X . Con esto, lo que modela la regresión logística es la probabilidad que un individuo con ciertas características observables sea tratado, es decir $P(D = 1|X = \mathbf{x})$, donde \mathbf{x} corresponde a valores específicos de las variables en X .

Para una explicación más detallada de la regresión logística y de cómo se calculan los coeficientes, se puede consultar la Sección 17.1 de [44].

En lo que refiere al algoritmo de emparejamiento (paso **5**), estos suelen establecer tanto la manera en que se encuentra la pareja (que puede ser también un grupo de individuos) para cada tratado como así también la manera en que se pondera la diferencia a la hora de hacer la comparación. También se los llama estimadores ya que en fin, determinan una forma de estimar el *ATT*.

Uno de los más usados y más simples de entender es el de vecino más cercano (o *NN matching*, por sus siglas en inglés, *nearest neighbor*) [5]. En el *NN matching*, se empareja cada individuo de tratamiento con aquel del grupo de no tratados que tenga la probabilidad de participación más cercana. Es decir, la unidad no tratada tal que la distancia entre su puntaje y el del individuo tratado sea mínima. En este caso, una vez que se haya emparejado a cada individuo tratado, el *ATT* se calcula como el promedio simple de las diferencias entre las variables de resultado en cada pareja. Esta es la técnica que utilizamos en este trabajo ⁶.

⁶Una cuestión a tener en cuenta al utilizar este algoritmo es que la diferencia de puntajes entre un tratado y su vecino más cercano no sea lo suficientemente pequeña como para asegurar que ambos son muy parecidos, y aún así la diferencia entre sus variables de resultado contribuirá al cálculo del *ATT*.

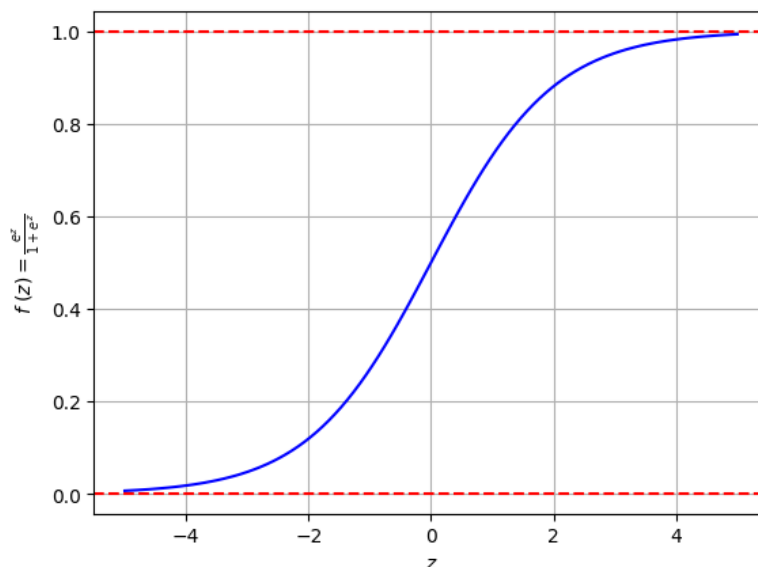


Figura 2.3: Gráfico de la función logística para valores de z en el rango $[-5, 5]$.

Existen otras opciones como tomar los k vecinos más cercanos (con $k > 1$), el emparejamiento por *kernel*, y por regresión lineal local. La principal diferencia con el método descrito en el párrafo anterior es que cada tratado ya no tiene solamente un individuo de comparación, sino un grupo de individuos. Para más detalles sobre cada uno de estos, se puede consultar el Capítulo 6 de [5].

Una evaluación en la que se utilizó la técnica de PSM, particularmente con el algoritmo del vecino más cercano, fue la llevada a cabo a partir del año 2021 por la Unidad de Evaluación Integral de la Calidad y Equidad Educativa de la Ciudad Autónoma de Buenos Aires (CABA) sobre el impacto del programa “Jornada Extendida” en el clima escolar [29], aplicado sobre diferentes escuelas primarias de la ciudad. El PSM fue empleado para seleccionar las escuelas del grupo de control en base a tres características: el ISSAP (Índice de Situación Socioeconómica de los/as Alumnos/as en Escuelas Primarias), la tasa de sobreedad y la tasa de repitencia. Otro análisis en el que se utilizó PSM se encuentra en [2].

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, por lo que se emplea con frecuencia [5].

Sin embargo, es muy importante notar que los resultados arrojados por el PSM serán confiables solamente si se cumple el supuesto de IC. 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 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 [5].

Por otro lado, si bien el cálculo del ATT se puede restringir a aquellos individuos que se encuentran en la región de soporte común entre tratados y no tratados, esta restricción puede volverse preocupante cuando hay una fracción importante de individuos tratados que quedan sin emparejar. Esto implica que existirá un conjunto de individuos para el cual no se puede decir nada acerca del efecto del programa [5].

En este trabajo, tomamos como punto de partida el PSM con regresión logística y el algoritmo de (un) vecino más cercano para compararlo con nuestro enfoque, lo cual se explicará con más detalle en el Capítulo 3 y en la Sección 4.4.

A continuación, introducimos el área de inteligencia artificial, dentro de la cual se enmarcan los algoritmos propuestos en este trabajo.

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, una que la resume adecuadamente es aquella que la relaciona con la capacidad de computar cómo actuar de la mejor manera posible en una determinada situación [36].

En sus comienzos, los métodos desarrollados en el área de la IA estaban principalmente **basados en conocimiento**, es decir en 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 [11].

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 au-**

tomático, 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 [11] han hecho que esta área sea la que mayor desarrollo e impacto ha tenido en el último tiempo.

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 [20] 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 [11]. 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. Siguiendo con el ejemplo, podríamos querer que, en base a las features de las casas, el sistema nos provea un estimado de su valor (regresión) o nos diga si corresponde a una vivienda adecuada para una familia o no (clasificación).

La **experiencia** hace referencia al tipo de información que el algoritmo “puede ver” durante su proceso de aprendizaje o *entrenamiento* [8]. A esta información se la conoce como “conjunto (de datos) de entrenamiento”, y es un subconjunto del *dataset*, que es 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. Cada ejemplo contiene sus características pero también su “etiqueta”, que vendría a ser la “respuesta correcta” para dicha entrada. De esta forma, 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 F_1 , 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 ajuste a los **patrones** aprendidos durante el entrenamiento.

Resulta conveniente introducir en este punto un término muy utilizado en el ML, el de **modelo**. Un modelo es simplemente una ecuación matemática, que se presenta como una forma simplificada de describir hechos de la realidad. En este contexto, es una manera de intentar capturar las relaciones entre los datos, con el objetivo de hacer predicciones basadas en los patrones aprendidos de ejemplos previos. 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:

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

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 [36].

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:

$$\begin{aligned}\mathbf{X} &= (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N) \\ \mathbf{Y} &= (\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_N)\end{aligned}$$

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 (es decir $h(\mathbf{x}_i) \simeq \mathbf{y}_i$ para $i = 1, \dots, N$), entonces se espera que va a realizar predicciones adecuadas 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 \mathbf{w} , y a los que también se suele llamar “**pesos**”. De esta forma, este espacio fija 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 $\mathbf{w} = (w_0, w_1)$. Este caso es comúnmente conocido como regresión lineal univariada (ya que la entrada está compuesta solamente por una variable). Dentro de este espacio, distintas asignaciones de valores a los parámetros generan distintos modelos. Por ejemplo, tomando la asignación $\mathbf{w} = (13, 5)$, tenemos el modelo $h(x) = 5x + 13$, y tomando $\mathbf{w} = (1.25, \pi)$, $h(\mathbf{x}) = \pi \mathbf{x} + 1.25$, 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 $\mathbf{w} = (w_0, w_1, w_2)$. A este caso se lo suele llamar regresión polinomial (de segundo grado) univariada.

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:

$$h(\mathbf{x}) = w_1 x_1 + w_2 x_2 + w_3 x_3 + w_0$$

y sus parámetros $\mathbf{w} = (w_0, w_1, w_2, w_3)$ ⁷.

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:

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

Y lo que queremos idealmente es hallar el vector de parámetros \mathbf{w}^* que hagan que el modelo resultante cumpla:

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

para cada \mathbf{x} en el conjunto de entrenamiento.

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 [28]. Un algoritmo de aprendizaje toma el conjunto de entrenamiento y manipula los parámetros de forma iterativa hasta que las predicciones dadas por el modelo resultante sean lo más cercanas posible a las etiquetas verdaderas.

⁷Para 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: $h(\mathbf{x}) = \mathbf{x} \cdot \mathbf{w}$, donde el operador \cdot representa el producto escalar. Retomaremos esta idea más adelante al hablar sobre redes neuronales artificiales.

Para que el algoritmo sepa cómo modificar estos parámetros para mejorar sus predicciones, necesita una forma de conocer 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 \mathbf{w}' actuales, en términos de qué tan lejos están sus predicciones de las respuestas reales del conjunto de entrenamiento.

Como venimos enfatizando, lo que caracteriza al modelo en cada iteración es el valor de sus parámetros. Por lo tanto, tiene sentido tratar a la función de pérdida como una que depende de estos, es decir⁸ $L(\mathbf{w})$. A continuación, se mencionan dos ejemplos de funciones de error:

- Una función de pérdida que es común usar en problemas de regresión es la de error cuadrático medio (ECM), dada por:

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

- Para problemas de clasificación binaria como el que tratamos en este trabajo, vamos a tener (generalmente) $\mathbf{y}_i \in \{0, 1\}$, y se suele emplear la entropía cruzada binaria (ECB), dada por:

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

donde \ln es la función de logaritmo natural. A grandes rasgos, 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.

Para comprender un poco mejor cómo funciona de manera intuitiva, tomemos un ejemplo simple. Supongamos que tenemos una entrada \mathbf{x}' cuya etiqueta es $\mathbf{y}' = 1$, y que la salida del modelo es $h(\mathbf{x}', \mathbf{w}') = 0.95$. En primera instancia, vamos a tener que el lado derecho del término de la sumatoria directamente se anula (ya que $1 - \mathbf{y}' = 1 - 1 = 0$). Y, para ver qué pasa con el lado izquierdo, notemos que $\ln(h(\mathbf{x}', \mathbf{w}')) = \ln(0.95) \approx -0.05$. Multiplicando por el (-1) del comienzo de la fórmula, vamos a tener como resultado 0.05, lo cual es bajo e indica una buena predicción del modelo. Si en cambio la predicción hubiera sido $h(\mathbf{x}', \mathbf{w}') = 0.25$, tendríamos $\ln(h(\mathbf{x}', \mathbf{w}')) = \ln(0.25) \approx -1.39$, dando una pérdida de 1.39.

⁸En realidad la función de pérdida también depende de los datos de entrenamiento, por lo que si somos estrictos, deberíamos escribir $L(\{\mathbf{X}, \mathbf{Y}\}, \mathbf{w})$. Sin embargo, como estos datos están fijos durante todo el proceso de aprendizaje, podemos omitirlos.

Lo que se puede ver es que el lado izquierdo influye cuando la salida esperada para la entrada actual es 1, y el lado derecho lo hará cuando la salida esperada es 0.

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

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

donde el operador $\arg \min_{\alpha} g(\alpha)$ da el valor de α que minimiza $g(\alpha)$.

Si luego de la minimización la pérdida es baja, quiere decir que hemos encontrado parámetros que hacen que el modelo prediga precisamente las salidas de entrenamiento \mathbf{y}_i a partir de las entradas \mathbf{x}_i (con $i = 1, \dots, N$). 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 \mathbf{w}^* , que se calcule analíticamente.

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

El algoritmo por defecto que se utiliza para resolver este problema de optimización es el llamado **descenso por el gradiente** (DG), y es muy potente ya que se puede aplicar a una gran variedad de funciones de pérdida [36].

La idea del DG es ir modificando los parámetros iterativamente hasta eventualmente llegar a un “valle” de la función de pérdida, en donde su valor es bajo. El algoritmo se basa en que la dirección dada por el gradiente⁹ 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 \mathbf{w} .

Concretamente, se empieza con un vector de parámetros $\mathbf{w}^{(0)}$ con valores arbitrarios. Luego, con el objetivo de reducir el valor de la función de pérdida, estos valores se van mejorando tomando un paso a la vez en dirección opuesta al gradiente. Este proceso se

⁹Dada una función g de varias variables, es decir, $g(v_1, v_2, \dots, v_M)$, el **gradiente** de g es el vector formado por las derivadas parciales de g con respecto a cada uno de sus parámetros:

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

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.

realiza hasta que el algoritmo *converja* a un mínimo (local o global) de la función de pérdida.

Un hiperparámetro¹⁰ determinante en el algoritmo del DG es el tamaño de los pasos, llamado **tasa de aprendizaje**, que denotaremos con la letra η . Si el valor de η es muy pequeño, entonces el algoritmo va a tener que realizar muchas iteraciones para converger [8]. 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 [8].

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

$$\mathbf{w}^{(p+1)} = \mathbf{w}^{(p)} - \eta \nabla L(\mathbf{w}^{(p)})$$

donde p indica el número de iteración actual, y $\nabla L(\mathbf{w}^{(p)})$ es el gradiente de la función de error con respecto a los pesos del modelo, evaluado en los valores de los pesos de la iteración actual.

Hiperparámetros

La mayoría de los algoritmos de ML tienen **hiperparámetros**, que son ajustes que permiten controlar tanto la capacidad expresiva del modelo como el proceso de entrenamiento. Los valores de los hiperparámetros no son optimizados por el algoritmo de aprendizaje [11], sino que se fijan de antemano y no se modifican a lo largo del entrenamiento.

Con lo visto hasta ahora, podemos pensar en dos ejemplos: el grado del polinomio con el cual ajustar los datos, y la tasa de aprendizaje del DG. En la sección siguiente, veremos que existen varios más.

Los hiperparámetros 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. La respuesta a esto es un proceso llamado **búsqueda u optimización de hiperparámetros**.

Esta búsqueda se hace de forma empírica: se entrenan distintos modelos con diferentes hiperparámetros sobre el mismo conjunto de entrenamiento, se mide el desempeño de cada uno de ellos, y se elige el mejor. Ahora bien, el desempeño¹¹ no se mide sobre el conjunto de test, ya que esto provocaría que la evaluación final sobre la capacidad de generalización del modelo esté sesgada; no sabríamos realmente cómo funciona ante entradas nunca vistas.

Por lo tanto, una técnica para realizar correctamente esta búsqueda consiste en construir una tercera partición del conjunto de datos, además de la de entrenamiento y la

¹⁰Hablaremos a continuación 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 \mathbf{w} .

¹¹Cómo se mide exactamente el desempeño es una decisión de diseño. Generalmente interesa maximizar o minimizar una métrica puntual; y así lo que “guiará” la optimización de hiperparámetros serán los valores de esa métrica obtenidos por los distintos modelos.

de test, llamada **conjunto de validación**. De esta forma, en la búsqueda de hiperparámetros, el modelo se sigue entrenando con el conjunto de entrenamiento, pero su rendimiento se mide en el de validación, dejando al de test como si no existiese. Una vez que se han elegido los mejores hiperparámetros, se entrena el modelo con el conjunto de entrenamiento y se mide su capacidad real de generalización sobre el conjunto de test.

Otra estrategia para llevar a cabo esta optimización se denomina **validación cruzada** (*cross-validation*). Consiste en dividir el conjunto de entrenamiento en k partes (disjuntas), y luego entrenar el modelo k veces, cada vez utilizando $k - 1$ partes para el entrenamiento y la parte restante para la validación. En este caso, habiendo seleccionado una métrica particular, el rendimiento del modelo con ciertos hiperparámetros va a estar dado por el promedio del puntaje obtenido en cada partición. Al igual que antes, una vez que se han elegido los mejores hiperparámetros, se entrena el modelo con el conjunto de entrenamiento (sin particionar) y se mide su capacidad real de generalización sobre el conjunto de test.

Algo a tener en cuenta es que aunque el espacio de hiperparámetros suele ser más pequeño que el de los parámetros del modelo en sí, puede seguir siendo lo suficientemente grande como para probar cada combinación de manera exhaustiva [28]. Ante esto, existen algoritmos de optimización de hiperparámetros que eligen inteligentemente qué combinaciones ir probando, con base en resultados anteriores [28]. Sin embargo, cabe notar que esta parte del entrenamiento de un modelo es una de las más costosas computacionalmente y en tiempo.

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.
- 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 w .
- Una **función de pérdida** $L(w)$, que da una medida de qué tan bien está funcionando el modelo y ayuda a guiar el entrenamiento.
- Un **algoritmo de optimización**, que busca minimizar la función de pérdida, siendo el más común el descenso por el gradiente.
- **Hiperparámetros**, que permiten controlar el proceso de entrenamiento y la capacidad expresiva del modelo.

Con esto, el objetivo es encontrar los parámetros w^* que minimicen la función de pérdida en el conjunto de entrenamiento. Para esto, se comienza con pesos $w^{(0)}$ arbitrarios

y, haciendo uso del algoritmo de optimización, se los va modificando iterativamente hasta llegar a un mínimo (local o global) de $L(\mathbf{w})$.

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 \mathbf{w}^* , se espera que el modelo encontrado tendrá un buen desempeño en el conjunto de test.

Como venimos enfatizando, el principal desafío del ML es hacer que un modelo tenga una buena capacidad de generalización, es decir que actúe de manera correcta ante entradas aún no vistas. Por un lado, esto depende de qué tan representativo es el conjunto de entrenamiento. Pero por otro, también depende de la elección de un espacio de funciones adecuado, cuyos modelos sean lo suficientemente *expresivos* como para capturar la distribución de los datos.

Sin embargo, más allá de la expresividad que otorgue el espacio de funciones y de la representatividad del conjunto de entrenamiento, existe otra variable presente, que es la dimensionalidad de los datos. La capacidad de generalizar se vuelve aún más complicada cuando los datos son de altas dimensiones, es decir cada entrada contiene muchas características.

En este contexto, aparecen los modelos conocidos como **redes neuronales**, que no solo permiten describir conjuntos de funciones complejas, expresivas, y no lineales sin la necesidad de tener un conocimiento profundo sobre la estructura subyacente de los datos, sino que también son capaces de trabajar con espacios de alta dimensión. En la Sección siguiente, explicaremos el funcionamiento de estos modelos.

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 [11]. 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 [8], y que sobre todo parecían extremadamente difíciles de resolver. Como mencionamos anteriormente, su avance se vio muy favorecido por la creciente disponibilidad de datos y el aumento en la capacidad computacional.

A continuación, damos un breve contexto histórico y luego ahondamos en su funcionamiento.

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* [19] un modelo computacional simplificado utilizando lógica proposicional acerca de cómo funcionan las neuronas de cerebros animales en conjunto para llevar a cabo cómputos complejos [8]. 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 formas o *arquitecturas* más simples de RNAs: el “Perceptrón” [33]. Este consistía de solamente una neurona artificial que recibía entradas, las combinaba en una suma pesada y si el resultado era mayor a un determinado umbral, daba como salida un valor de 1, y en caso contrario de (-1). La gran limitación del Perceptrón fue que servía solamente para resolver problemas de clasificación en donde los datos son linealmente separables. Sin embargo, el aporte más notorio de Rosenblatt fue la definición de un algoritmo para el entrenamiento del Perceptrón que le permitía mejorar automáticamente sus parámetros internos (los “pesos” de la suma pesada) para poder llegar a la solución óptima.

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”, también conocido actualmente como “red neuronal de propagación directa”¹² (del inglés *feedforward neural network*) [11].

Para explicar la idea y los elementos presentes detrás de estos algoritmos, tomaremos como referencia las redes neuronales feedforward, y en particular las llamadas “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” [36]. 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 expresivos como para poder capturar la complejidad de los datos del mundo real [36].

Más concretamente, estos modelos son llamados *redes* porque el espacio de funciones que proveen está formado en realidad por la composición de varias funciones [11]. Por ejemplo, podríamos tener la composición de tres funciones $f^{(1)}$, $f^{(2)}$ y $f^{(3)}$ para formar la

¹²Si bien estos términos se suelen usar indistintamente, la realidad es que las redes neuronales feedforward presentan algunas diferencias con respecto al Perceptrón Multicapa.

siguiente función:

$$\hat{\mathbf{y}} = f(\mathbf{x}) = f^{(3)}(f^{(2)}(f^{(1)}(\mathbf{x}))) \quad (2.13)$$

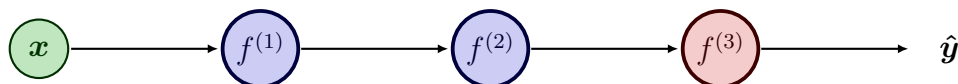
donde $\mathbf{x} \in \mathbb{R}^n$ para algún $n \in \mathbb{N}$, $\hat{\mathbf{y}} \in \mathbb{R}^m$ para algún $m \in \mathbb{N}$, y asumiremos que los espacios de dominio y llegada de las diferentes funciones son “compatibles”.

Usualmente, se dice que las redes están organizadas en **capas**. De esta forma, en la ecuación anterior, a \mathbf{x} se la conoce como **capa de entrada**, a las funciones $f^{(1)}$ y $f^{(2)}$ como **capas 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 hace referencia al término **aprendizaje profundo**.

Recordemos que uno de los elementos presentes en el aprendizaje supervisado es el conjunto de entrenamiento, compuesto por pares de entrada-etiqueta. Ahora bien, lo interesante de estos modelos es que si bien este conjunto especifica qué tiene que producir la capa de salida ante cada entrada particular, no determina cuál debe ser el comportamiento de las otras capas [11]. 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 \mathbf{x} va a la función $f^{(1)}$, la salida de $f^{(1)}(\mathbf{x})$ va a $f^{(2)}$, y la salida de $f^{(2)}(f^{(1)}(\mathbf{x}))$ va directamente a $f^{(3)}$ para de esa forma producir el resultado final¹³ $\hat{\mathbf{y}} = f(\mathbf{x}) = f^{(3)}(f^{(2)}(f^{(1)}(\mathbf{x})))$.

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,

¹³Usamos $\hat{\mathbf{y}}$ para referirnos a la aproximación dada por el modelo de \mathbf{y} , la etiqueta real para una entrada \mathbf{x} .

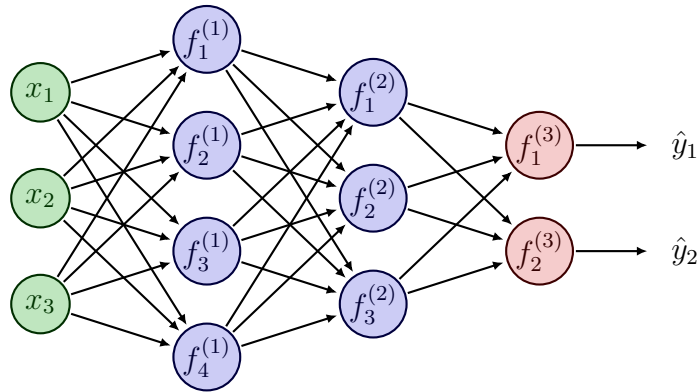


Figura 2.4: Red neuronal de tipo feedforward y totalmente conectada, con la capa de entrada formada por 3 neuronas, dos capas ocultas, cada una formada por 4 y 3 neuronas respectivamente, y la capa de salida, formada por dos neuronas. En este caso, la red “acepta” entradas $\mathbf{x} \in \mathbb{R}^3$ y produce salidas $\hat{\mathbf{y}} \in \mathbb{R}^2$. Adaptado de [22].

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 en base a estas, computan su propio valor de activación [11].

De esta forma, la capa de entrada va a tener tantas neuronas como la dimensión de los datos de entrada (\mathbf{x}). Es decir, si \mathbf{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, “pasándole” su valor computado a todas ellas.

Con esto en mente, podemos concretizar un poco más el ejemplo con el que venimos trabajando suponiendo que \mathbf{x} es un vector de dimensión 3, las capas ocultas dadas por $f^{(1)}$ y $f^{(2)}$ tienen 4 y 3 neuronas respectivamente y la capa de salida tiene 2. Así, suponiendo que nuestra red es totalmente conectada, nuestro grafo resultaría en el de la Figura 2.4, donde el superíndice de cada nodo indica el número de capa y el subíndice hace referencia al número de neurona en esa capa.

A partir de la Figura 2.4, se puede ver que cada neurona de la capa de entrada representa un elemento del vector de entrada, pero las neuronas de tanto las capas ocultas como la de salida reciben las salidas de las neuronas de la capa anterior. Veamos entonces qué hace concretamente 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** para producir su salida. Esta función se denomina **función de activación**, y el hecho que sea no lineal es importante ya que de no ser así, cualquier composición de unidades podría representarse mediante una función lineal [36]. Como mencionamos anteriormente, es justamente esta no linealidad lo que permite a estos modelos representar funciones arbitrarias [36] y complejas. En general, se asume que todas las neuronas de una capa tienen la misma función de activación, pero puede ocurrir que diferentes capas tengan diferentes funciones de activación.

Más precisamente, una función de activación es una función no lineal que toma cualquier valor real como entrada y produce como resultado un número en un determinado rango. Algunas funciones de activación comunes son las siguientes:

- **Sigmoide**: produce un valor entre 0 y 1. Es por esto que se suele usar en problemas de clasificación binaria para que una única neurona en la capa de salida represente la probabilidad de que la entrada pertenezca a la clase “positiva”.

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

- **ReLU** (abreviatura de *Rectified Linear Unit*): produce un valor entre 0 e ∞ . Esta y sus variantes son las más utilizadas actualmente en las neuronas de capas ocultas, siendo una de las razones la simplicidad de su cálculo.

$$\text{ReLU}(x) = \max(0, x)$$

- **Tangente hiperbólica**: produce un valor entre -1 y 1. Lo particular de esta es que mantiene el signo de la entrada.

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

Un aspecto a notar en este punto es que en las redes neuronales aparecen nuevos hiperparámetros, que ya no tienen tanto que ver con el proceso de entrenamiento en sí, sino más bien con el diseño y la capacidad de la red, como son la cantidad de capas ocultas, la cantidad de neuronas en cada capa oculta, e incluso la función de activación de cada capa. Para todos estos, se puede encontrar el valor óptimo utilizando las técnicas mencionadas en la sección anterior.

Para formalizar el cómputo de una neurona, es necesario introducir cierta notación. Denotaremos con:

- $n^{(k)}$ la cantidad de neuronas de la capa k .

- $s_j^{(k)}$ a la salida de la unidad j de la capa k
- $a_j^{(k)}$ a la función de activación 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 $(k + 1)$,

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

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

Volviendo a la Fórmula 2.14, 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 “ficticia” (*dummy*) de la capa anterior, para la cual utilizaremos el subíndice 0. El valor de salida de esta neurona se fija en 1, y el peso asociado con una neurona j de una capa k será $w_{0,j}^{(k)}$. Este peso se suele llamar **sesgo** o **bias**¹⁴ y permite que la entrada a dicha neurona sea distinta de 0 incluso cuando todas las salidas de la capa anterior sean 0 [36]. Agregándolo, podemos escribir la Ecuación 2.14 de forma vectorizada:

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

donde:

- $\mathbf{w}_j^{(k-1)}$ 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 $w_{0,j}^{(k-1)}$)
- $\mathbf{s}_j^{(k-1)}$ 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).

Con esta notación, si tomamos nuevamente a $f_1^{(2)}$, los vectores involucrados van a ser $\mathbf{w}_1^{(1)}$ y $\mathbf{s}_1^{(1)}$, dados en este caso por:

$$\mathbf{w}_1^{(1)} = (w_{0,1}^{(1)}, w_{1,1}^{(1)}, w_{2,1}^{(1)}, w_{3,1}^{(1)}, w_{4,1}^{(1)}), \mathbf{s}_1^{(1)} = (1, s_1^{(1)}, s_2^{(1)}, s_3^{(1)}, s_4^{(1)})$$

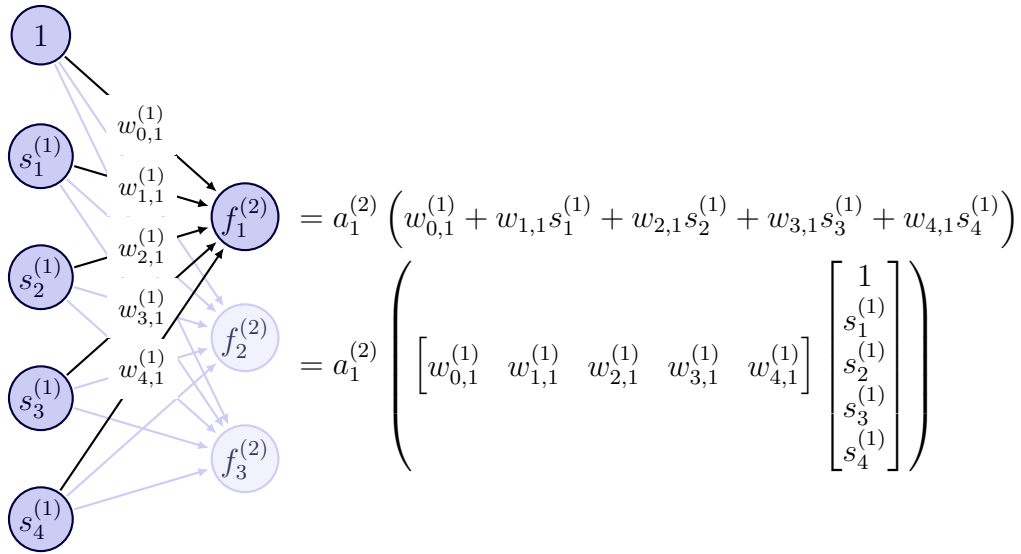


Figura 2.5: Entradas a la neurona $f_1^{(2)}$, junto con sus pesos asociados. Se incluyen tanto la neurona dummy de la capa anterior como su bias. Cabe aclarar que se usan indistintamente las letras s y f para denotar a las neuronas. Adaptado de [22].

Esto se puede ver mejor gráficamente haciendo foco en dicha neurona, como lo ilustra la Figura 2.5.

Así como utilizamos vectores para describir el cómputo de una neurona, podemos emplear matrices para describir el comportamiento de toda una capa. De hecho, las librerías que implementan estos modelos se valen de esto ya que suelen estar optimizadas para cálculos con matrices.

En resumen, las redes neuronales constituyen una forma de modelar funciones altamente expresivas, capaces de capturar patrones complejos a través de la composición de capas de neuronas que actúan en conjunto. Estas neuronas están conectadas entre sí con distintas intensidades que, como veremos, son justamente los parámetros aprendibles de las redes. Además, introducen nuevos hiperparámetros referentes a la arquitectura. A continuación, describimos cómo se lleva a cabo su entrenamiento, siempre dentro del contexto del aprendizaje supervisado.

Entrenamiento

Los parámetros a optimizar en las redes son las intensidades de las conexiones entre las neuronas, que también venimos llamando pesos. Los algoritmos de optimización que

¹⁴En 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 con $b_j^{(k)}$.

se emplean actualmente se basan en la regla del DG, aunque con algunas mejoras.

Como vimos anteriormente, el DG requiere calcular el gradiente de la función de pérdida con respecto a cada uno de los pesos del modelo. Ahora bien, si prestamos atención a los ejemplos presentados de funciones de pérdida, estas recorren todas las muestras del conjunto de entrenamiento. Por lo tanto, si este conjunto tiene un gran número de ejemplos, calcular el gradiente sobre todos ellos puede volverse una tarea computacionalmente costosa, y más aún si se lleva a cabo en cada paso de actualización.

Para reducir esta carga, en la práctica se recurre a una aproximación: en lugar de usar todas las muestras, se selecciona aleatoriamente un subconjunto de ellas, llamado **lote** (*batch*), y se calcula el gradiente usando únicamente ese lote. Luego, se actualizan los pesos en base a esta 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. Esta técnica suele llamarse optimización por **mini-lotes** o **estocástica**, ya que introduce cierto grado de aleatoriedad en el cálculo del gradiente.

Otra mejora aplicada sobre el método del DG estándar, diseñada para acelerar el aprendizaje, es la técnica conocida como **momentum**. A grandes rasgos, consiste en actualizar los pesos mirando no solo el gradiente de la iteración actual, sino también prestando atención a la acumulación de gradientes de las iteraciones anteriores. Esto permite sobre todo acelerar las actualizaciones cuando se encuentran sucesivos gradientes que apuntan en la misma dirección [11]. Un optimizador ampliamente utilizado actualmente, y que es el que empleamos en este trabajo, es **Adam** [17]. Este combina momentum con una tasa de aprendizaje adaptativa para cada parámetro.

Para entender cómo progresa el entrenamiento, es útil definir el concepto 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. Este valor escalar no se utiliza directamente en el entrenamiento, pero se emplea comúnmente para monitorear el progreso del aprendizaje.
5. Calcular el gradiente utilizando solamente las muestras del lote.
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 detuvo 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 error** (*backpropagation*) y fue presentado en el año 1986 [35].

El backpropagation se basa fundamentalmente en la regla de la cadena para derivadas de funciones compuestas y se divide en dos etapas: primero, el paso hacia adelante (*forward pass*) y luego, el paso hacia atrás (*backward pass*). Para entender estas etapas, supondremos que solamente una entrada es provista a la red.

En el paso hacia adelante, la red simplemente lleva a cabo una predicción para la entrada, realizando todo el cómputo intermedio necesario para llegar a producir una salida, y guardando la salida de cada neurona.

El paso hacia atrás 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 de cada conexión al error total [8]. Este paso se basa fuertemente en la idea que un pequeño cambio en uno de los pesos afecta indirectamente el resultado final a través de todas las capas sucesivas [28]. Para verlo, supongamos que tenemos una red neuronal con tres capas ocultas, que denotaremos con $\mathbf{h}^{(1)}$, $\mathbf{h}^{(2)}$ y $\mathbf{h}^{(3)}$ (la cantidad de neuronas en cada capa es irrelevante), y veamos cómo computaríamos el efecto de un cambio en un peso de manera intuitiva [28]:

- 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 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 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 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. Por lo tanto, no es necesario recomputarlos, lo cual hace que el algoritmo sea eficiente. Calcular de este modo los efectos de los cambios —es decir, las

derivadas parciales de la pérdida con respecto a cada peso— es lo que constituye el paso hacia atrás del algoritmo.

Finalmente, con todos los gradientes calculados, se actualizan los pesos utilizando la regla del DG (con las optimizaciones que se hayan incluido).

Para finalizar, hablaremos de un problema bastante conocido al entrenar redes neuronales, conocido como **sobreajuste** (*overfitting*).

A grandes rasgos, los factores que determinan qué tan bueno es un modelo son sus capacidades para [11]:

- Reducir el error en el conjunto de entrenamiento.
- Reducir la brecha entre el error en el conjunto de entrenamiento y en el de test.

En muchos casos, las redes neuronales son capaces de lograr lo primero, principalmente gracias a su habilidad de modelar una gran variedad de funciones. Sin embargo, no siempre logran alcanzar lo segundo, y cuando esto ocurre, hablamos de sobreajuste. Es decir, este fenómeno se produce cuando la brecha entre el error en el conjunto de entrenamiento y en el de test es considerable. En general, esto es una señal que el modelo ha aprendido los patrones vistos durante su entrenamiento “de memoria” y no está siendo capaz de generalizar correctamente.

Por suerte, para solucionar este problema, existen técnicas llamadas **de regularización**. Una regularización es cualquier modificación hecha sobre un algoritmo de aprendizaje que tiene como objetivo reducir el error de generalización (i.e. en el conjunto de test) pero no el de entrenamiento [11].

Existen varias estrategia de regularización, pero una puntual que utilizaremos en este trabajo y que ha probado ser efectiva es llamada **dropout**. El dropout se puede aplicar sobre la capa de entrada o las capas ocultas, pero no sobre la de salida. Una ventaja es que su implementación es relativamente sencilla: en cada paso de entrenamiento¹⁵, cada neurona de la capa sobre la que se esté aplicando el dropout tiene una probabilidad p de ser temporalmente descartada, lo cual significa que su salida va a ser ignorada en este paso [8], normalmente multiplicándola por 0. Aquí se introduce un nuevo hiperparámetro, p , llamado “tasa de dropout”.

Este método obliga a las neuronas a aprender no solo a ser útiles por sí solas, pero sino también a ser compatibles con muchos conjuntos posibles de neuronas que pueden estar o no activas en la red [36]. En resumen, contribuye a que la red no dependa de unidades específicas y la fuerza a aprender diversas explicaciones para cada entrada [36].

Luego del entrenamiento, el dropout deja de aplicarse y todas las neuronas de la red contribuyen a la inferencia. Sin embargo, como en este momento la red tiene más neuronas activas que durante el entrenamiento, lo que se hace para compensar este hecho es multiplicar cada peso por $(1 - p)$ [28].

¹⁵Cuando se utiliza DG estocástico, un paso de entrenamiento equivale a procesar un lote.

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 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 (RNCs) son un tipo especializado de redes neuronales pensadas para el procesamiento de datos que tienen una estructura de grilla [11]. 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 bidimensional 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.

Una **serie de tiempo univariada o unidimensional** se puede representar como un vector ordenado U de valores reales que reflejan la evolución a lo largo del tiempo de un determinado fenómeno:

$$U = (x_1, x_2, \dots, x_N)$$

donde N es la cantidad total de observaciones, y x_i corresponde el valor observado en el período i , para $i = 1, \dots, N$.

Las series de tiempo están presentes en una gran variedad de situaciones de la realidad. Algunos ejemplos incluyen la evolución del precio de un activo financiero, la variación de indicadores de salud como el ritmo cardíaco, los registros diarios de temperatura y precipitaciones, el seguimiento de indicadores económicos o sociales como la tasa de empleo o niveles de pobreza, entre muchos otros. En el contexto de este trabajo, estas series se utilizarán como entradas de las redes neuronales.

El término *convolucionales* proviene del hecho que lo que caracteriza a estas redes es la utilización de una operación matemática llamada **convolución**¹⁶ [11]. 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

¹⁶En realidad, la operación que está presente en estas redes es una relacionada con la convolución, llamada **correlación cruzada**, pero el término que se utiliza sigue siendo convolucionales. El símbolo que se suele utilizar es $*$.

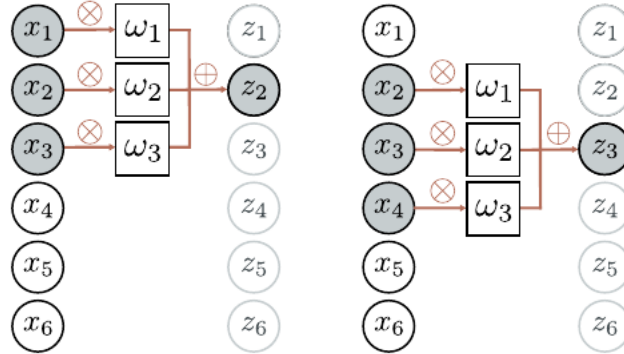


Figura 2.6: Fuente: [28]. Ejemplo de convolución unidimensional aplicada sobre una entrada de tamaño 6 con un filtro $\mathbf{w} = (w_1, w_2, w_3)$ de tamaño 3. El símbolo \otimes indica la multiplicación, y el \oplus la suma.

números reales, también comúnmente llamados tensores [11]. 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 vector¹⁷ $\mathbf{x} = (x_1, x_2, \dots, x_m)$ ($m \in \mathbb{N}$), si tomamos un filtro de tamaño 3, dado por $\mathbf{w} = (w_1, w_2, w_3)$, 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” a x_i :

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

En la Figura 2.6, se puede ver cómo se computan z_2 y z_3 en un ejemplo en donde la entrada es de tamaño 6 y el filtro de tamaño 3. Un detalle a notar aquí es que al calcular z_2 y z_3 , el filtro “entra” por completo en una región de la entrada, pero si quisiéramos calcular z_1 , pareciera que w_1 no tiene un valor de entrada con el cual multiplicarse. Existen alternativas para solucionar este problema que detallaremos más adelante.

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:

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

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 [36].

La salida de la convolución suele llamarse **mapa de características** (*feature map*) [11], y resalta las áreas de la entrada que son “similares” a la característica que el filtro está tratando de capturar [8]. Para ver esto, tomemos otro ejemplo.

¹⁷Por conveniencia, el primer elemento de los vectores será indexado en 1, y no en 0.

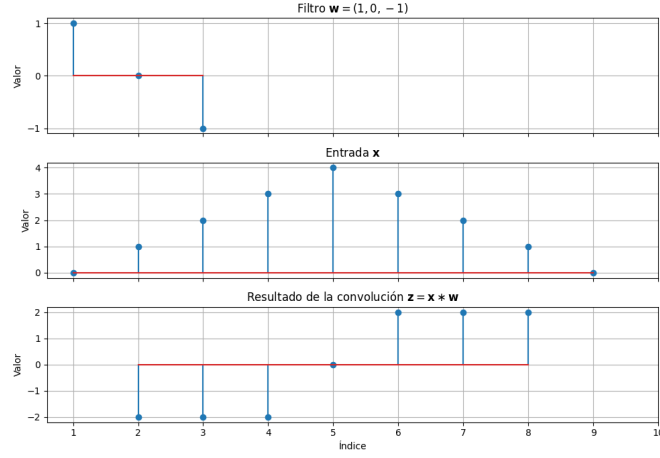


Figura 2.7: Ejemplo de convolución unidimensional en donde se aplica el filtro $\mathbf{w} = (-1, 0, 1)$ sobre una entrada de tamaño 9. El resultado de la convolución se calcula únicamente en aquellas posiciones donde el filtro puede superponerse completamente con la entrada. Se puede ver que en las regiones donde los valores vecinos presentan un aumento o descenso, el resultado de la convolución es distinto de 0. En cambio, en la posición 5, donde los vecinos tienen el mismo valor, el resultado es 0.

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

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

O sea, dada una posición i en la entrada \mathbf{x} , el filtro calcula la diferencia entre el “vecino” izquierdo (x_{i-1}) y el derecho (x_{i+1}). De esta forma, si para un i el valor resultante de aplicar el filtro es positivo, quiere decir que hubo una transición de un valor mayor a uno menor (una “bajada”), y si es negativo, viceversa (una “subida”). Si el valor es cercano a 0, significa que la entrada no cambia significativamente de valor entre los dos vecinos. En este caso, se podría decir que el filtro “detecta” regiones de la entrada en donde hay aumentos o descensos. Este ejemplo se puede ver más claramente en la Figura 2.7.

A partir de esto, podemos ver que cada filtro se encarga de identificar una característi-

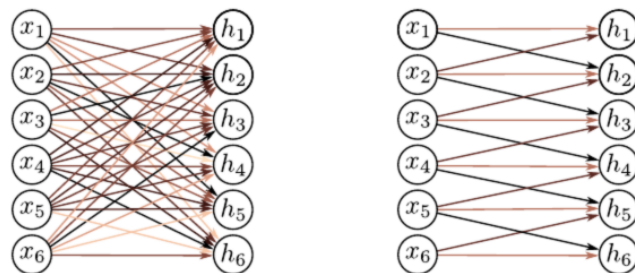


Figura 2.8: Fuente: [28]. Capas de una red totalmente conectada (izquierda) y capas convolucionales (derecha). Denotamos con x a la capa de la izquierda, y con h a la de la derecha, en ambos casos. En la arquitectura totalmente conectada, cada neurona de la capa x está conectada con todas las neuronas de la capa h , habiendo un total de $6 \cdot 6 = 36$ pesos entre ellas. En cambio, en la convolucional, asumiendo que se aplica un filtro de tamaño $l = 3$, cada neurona de la capa h computa una suma pesada de $l = 3$ neuronas consecutivas de la capa x , usando los mismos pesos. Esto reduce la cantidad de parámetros entre ambas capas de 36 a 3.

ca específica en la entrada, y que dicha característica depende de los valores del filtro. En las RNCs, estos valores — los pesos involucrados en cada filtro — son justamente los parámetros a optimizar durante el proceso de entrenamiento. El objetivo es que el modelo aprenda exactamente qué filtros le son más útiles para capturar los rasgos relevantes presentes en los datos.

Para entender cómo se aplican estos filtros en las RNCs, es importante describir el funcionamiento de las capas convolucionales, que son las responsables de realizar la operación de convolución. A diferencia de las redes totalmente conectadas, donde cada neurona de una capa está conectada con todas las de la capa anterior, lo que ocurre en las capas convolucionales es que cada neurona se conecta únicamente con un subconjunto de neuronas (consecutivas) de la capa anterior. Este subconjunto recibe comúnmente el nombre de **campo receptivo** y es sobre el que cada neurona aplicará el filtro correspondiente. Por ejemplo, en la Figura 2.6, si vemos a la columna de la izquierda como las entradas, y la de la derecha como una capa convolucional, el campo receptivo de la neurona z_2 está formado por x_1 , x_2 y x_3 , y el de la neurona z_3 por x_2 , x_3 y x_4 .

Además, es importante notar que cada neurona de una capa convolucional aplica el mismo filtro, por lo que todas las neuronas de una misma capa comparten los mismos pesos. Esto se puede observar claramente en la Figura 2.6, donde tanto la neurona z_2 como la z_3 aplican el mismo filtro (w_1, w_2, w_3) sobre sus respectivos campos receptivos. Como consecuencia, la cantidad de parámetros a aprender en una red convolucional es menor que en una red totalmente conectada. Esta diferencia puede verse en la Figura 2.8.

Con esta idea de campo receptivo, podemos pensar en una red con dos capas convolucionales consecutivas, donde cada neurona aplica un filtro de tamaño 3. Lo que va a

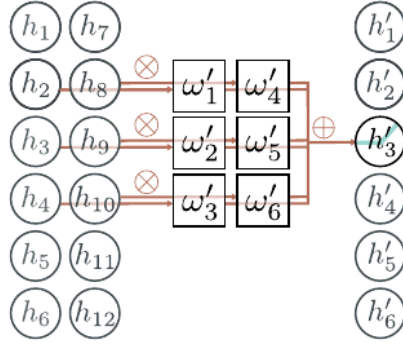


Figura 2.9: Fuente: [28]. La capa de neuronas h_i tiene dos canales: aquel formado por las neuronas h_1 a h_6 y aquel formado por las neuronas h_7 a h_{12} . La capa de neuronas h'_i aplica un filtro de tamaño $l = 3$ sobre cada uno de los canales de la capa anterior. Por lo tanto ahora termina habiendo $2 \cdot 3 = 6$ pesos entre cada neurona de la capa h'_i y la capa h_i .

ocurrir entonces es que las neuronas de la primera capa oculta toman una suma pesada sobre conjuntos de 3 neuronas consecutivas de la capa de entrada. Luego, las unidades de la segunda capa oculta hacen lo mismo sobre conjuntos de 3 neuronas consecutivas de la primera capa oculta, que a su vez ya representan combinaciones locales de la entrada. Esto implica que la primera capa oculta se concentra en características de bajo nivel, más específicas, mientras que las capas posteriores aprenden representaciones progresivamente más abstractas como resultado de integrar los patrones detectados en las capas anteriores [8].

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 mapa de características. 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 mapa para cada nivel profundidad, cada uno capturando una característica puntual de la entrada. 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 la capa anterior tiene C canales y el filtro tiene tamaño l , entonces al aplicarlo, ahora va a haber una cantidad de pesos $l \cdot C$. Esto se puede ver en la Figura 2.9.

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

entrada.

- 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 suele usar el 0 para este relleno (*zero-padding*). Con esto, se puede computar el valor de z_1 que presentamos en la Figura 2.6.
- 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 [28].

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 [8]. 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*).

De esta forma, 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 el tamaño de la salida [11]. Y así, una RNC profunda se compone de varias de estas capas.

Una técnica particular que ha demostrado tener un efecto positivo al entrenar redes convolucionales es una conocida como **normalización por lotes** (*batch normalization*) [14] [11] [8], introducida en 2015. Esta consiste en reescalar los valores generados en las capas intermedias de la red a partir de los ejemplos en el lote actual [36]. La normalización suele agregarse antes de la aplicación de la función de activación y es importante notar que agrega nuevos parámetros a optimizar durante el entrenamiento de la red. Para ver los detalles concretos de esta técnica, se puede consultar el Capítulo 11 de [8].

En este trabajo, empleamos este tipo de redes para resolver la tarea de **clasificación de series de tiempo**. Esto es: dada una serie de tiempo como entrada de la red, se busca que la red sea capaz de predecir a qué categoría corresponde. Para ello, lo que se suele hacer es aplanar (*flatten*) la salida de la última capa convolucional. Esta operación simplemente convierte el tensor resultante de dicha capa en uno unidimensional, por lo que no agrega nuevos parámetros aprendibles durante el entrenamiento. Por ejemplo, si la salida consiste de k mapas de características de longitud n cada uno, entonces aplanarla dará como resultado un vector unidimensional de tamaño $k \times n$.

El paso anterior es necesario ya que las capas siguientes que se agregan para hacer la clasificación son totalmente conectadas que, como vimos anteriormente, aceptan úni-

camente entradas unidimensionales. Entonces, lo que ocurre es que el vector de tamaño $k \times n$ es procesado por estas últimas hasta finalmente llegar a la de salida.

En problemas de clasificación con más de dos categorías, la capa de salida suele tener tantas neuronas como clases haya. Sin embargo, para tareas de clasificación binaria, como la que tratamos aquí, se utiliza habitualmente una única neurona con una función de activación sigmoide, cuya salida representa la probabilidad de que la secuencia de entrada pertenezca a la clase positiva (usualmente asociada con la etiqueta 1).

2.5. Redes neuronales recurrentes

Las redes neuronales recurrentes (RNRs) son una familia de redes neuronales pensadas específicamente para trabajar con **datos secuenciales**, en donde el orden y la dinámica importan. Es por esto que han sido y continúan siendo muy útiles en tareas como análisis de series temporales y procesamiento de lenguaje natural. Se emplean por ejemplo para predecir cuáles van a ser los próximos valores en una serie de tiempo (tarea comúnmente conocida como *forecasting*), cuáles van a ser las palabras que continúan una oración, y también para clasificar series de tiempo, que es para lo que las usaremos nosotros¹⁸.

Una limitación de las redes que hemos presentado hasta el momento (totalmente conectadas y convolucionales) es que no tienen memoria. Si quisiéramos procesar una secuencia temporal de datos con estas redes, deberíamos “mostrársela” entera como una entrada fija, perdiendo la dependencia temporal que existe entre los elementos de la secuencia. Lo que ocurriría es que las distintas características de la entrada representarían los pasos de tiempo, pero la red las trataría como si fueran atributos independientes.

Las RNRs proponen una forma alternativa de trabajar con datos de este tipo. Procesan las secuencias iterando sobre los distintos pasos de tiempo manteniendo una **memoria** que contiene información relacionada con lo que ha visto hasta el momento¹⁹. Esto se logra introduciendo ciclos en los grafos de estas redes, que permiten que las neuronas puedan tomar como entradas sus propias salidas de pasos anteriores, y provoca que entradas recibidas en pasos más tempranos afecten la respuesta de la red ante la entrada actual [36]. Además, al procesar cada paso de tiempo por separado, permiten trabajar con secuencias de longitud variable, lo cual es una gran ventaja a comparación de las totalmente conectadas y las convolucionales.

Para comprender un poco más este comportamiento, tomemos una RNR simple, compuesta por una única neurona (oculta) f que recibe la entrada correspondiente al tiempo t (x_t), produce una salida (y_t) y se la envía tanto a la capa de salida como a sí misma [8], como se puede ver a la izquierda de la Figura 2.10. Así, en cada paso t , la neurona “recurrente” f recibe no solo la entrada x_t sino también su propia salida computada en

¹⁸En “*The Unreasonable Effectiveness of Recurrent Neural Networks*” se pueden encontrar algunas aplicaciones de RNRs.

¹⁹Cabe aclarar que este estado interno se reinicia entre el procesamiento de diferentes secuencias.

el paso anterior, (f_{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 2.10 y se asemeja a la de una feedforward.

Otro detalle a tener en cuenta en estas redes es que, como se puede ver en la Figura 2.10, en cada paso de tiempo de una misma secuencia, la red genera una salida. Es decir, si tenemos una secuencia de largo T y la red produce en cada paso de tiempo una salida de tamaño o , entonces al terminar de procesar una secuencia entera, se tendría una salida “total” de tamaño $T \times o$. Sin embargo, como la salida en cada paso de tiempo contiene información de los pasos de tiempo previos, se suele utilizar la producida en el último paso.

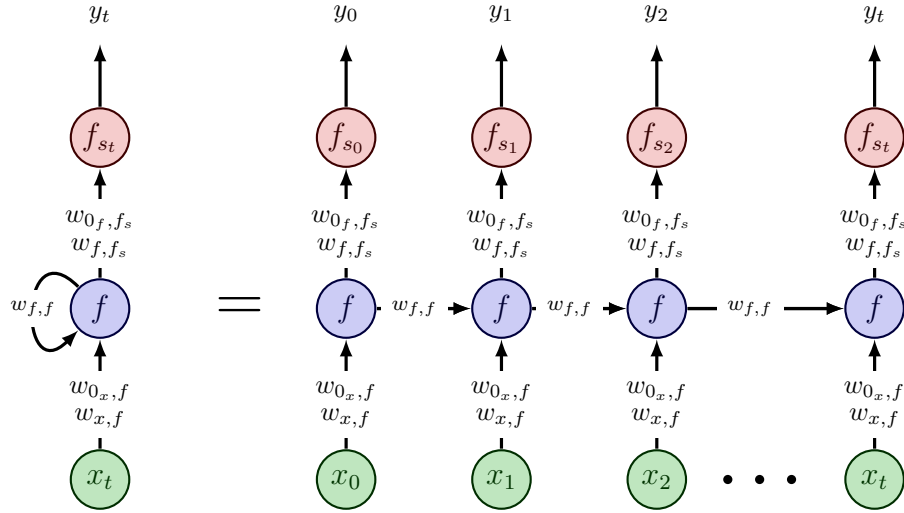


Figura 2.10: A la izquierda, se ve el grafo correspondiente a una RNR simple, con la capa de entrada, una capa oculta formada solamente por una neurona con su respectiva conexión recurrente, y la capa de salida. A la derecha, se ve la misma red pero “desenrollada” a lo largo del tiempo. Asumiremos por simplicidad que tanto la entrada como la salida en cada momento t son simplemente números reales, es decir $x_t, y_t \in \mathbb{R} \forall t$. Denotamos con $w_{0_x,f}$ y $w_{x,f}$ al sesgo y al peso respectivamente que van desde la entrada a la neurona oculta, con w_{0_f,f_s} y w_{f,f_s} a los que van desde la neurona oculta hasta la capa 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 feedforward, en el sentido que computa una suma pesada de sus entradas y aplica una función de activación sobre esta. Sin embargo, a partir de la Figura 2.10, podemos notar algunas 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 ($w_{x,f}$) y el sesgo correspondiente ($w_{0_x,f}$). Este peso extra, denotado con

$w_{f,f}$, es el que se aplica sobre f_{t-1} , la salida de la neurona en el paso anterior.

- Por otro lado, la red utiliza los mismos pesos $w_{x,f}$, $w_{f,f}$ y w_{f,f_s} , y sesgos $w_{0_x,f}$ y w_{0_f,f_s} 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:

$$f_t = a_f(w_{x,f}x_t + w_{f,f}f_{t-1} + w_{0_x,f}) \quad (2.17)$$

$$y_t = f_{s_t}(w_{f,y}, f_t, w_{0_f,f_s}) = a_y(w_{f,y}f_t + w_{0_f,y}) \quad (2.18)$$

donde a_f denota la función de activación de la capa oculta y a_y la de la capa de salida. Algo que vale la pena aclarar es que para la entrada x_0 , correspondiente al primer paso de tiempo, lo que sería la salida del paso anterior se establece manualmente, por lo general con un valor nulo.

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 la memoria de la que venimos hablando, a la que se suele llamar **estado oculto** o **interno** 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 2.11.

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 mismo tiempo, vamos a tener que:

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

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,

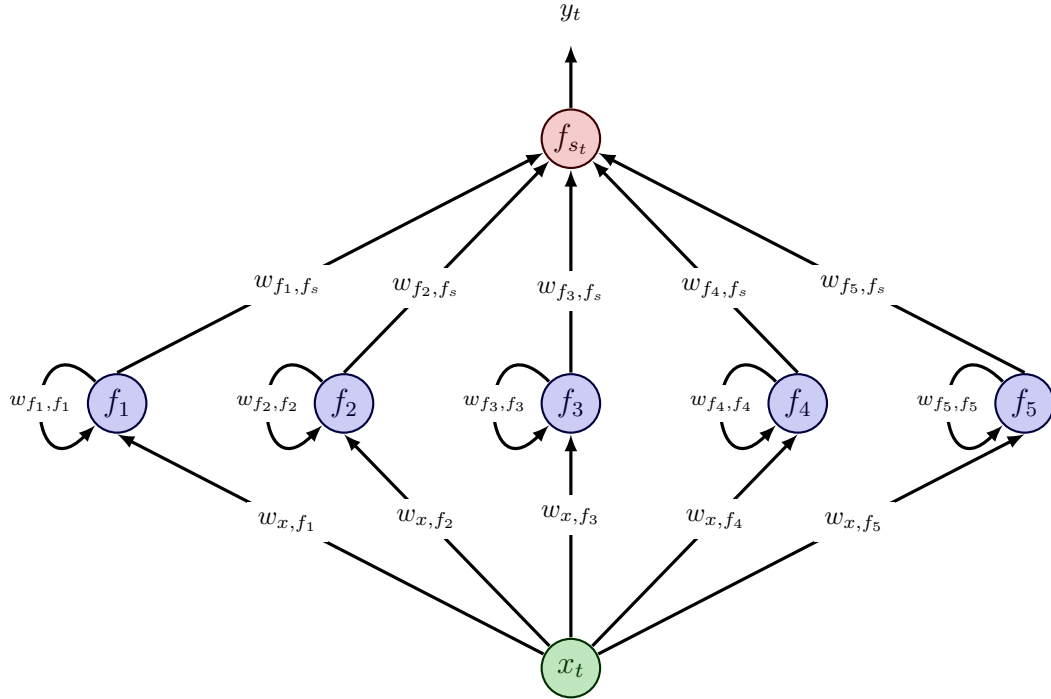


Figura 2.11: RNR con una capa oculta de 5 neuronas recurrentes, cada una con su ciclo. Similarmente a la Figura anterior, denotamos con w_{x, f_i} al peso que va desde la capa de entrada a la neurona oculta f_i , con w_{f_i, f_i} al peso de cada neurona oculta a sí misma, y con w_{f_i, f_s} al que va desde la neurona oculta f_i a la capa de salida ($i = 1, 2, 3, 4, 5$). En este caso, omitimos los sesgos.

lo que va a ocurrir es que el estado oculto producido por las neuronas de una capa va a ser la entrada de las neuronas de la siguiente capa, estando las capas totalmente conectadas entre sí.

Para entrenar este tipo de redes, el truco está en “desenrollarlas” como vimos anteriormente y luego utilizar el algoritmo de retropropagación del error presentado en la sección de redes neuronales. Esta estrategia es conocida como **retropropagación del error a lo largo del tiempo** (*backpropagation through time*).

Ahora bien, estas redes en su forma estándar sufren de un problema conocido como **el problema del gradiente desvaneciente** [4]. Intuitivamente, este ocurre cuando al calcular el gradiente de la función de pérdida con respecto a un parámetro determinado, este resulta ser muy pequeño. Como consecuencia, la actualización de dicho parámetro durante el proceso de entrenamiento es insignificante, lo que puede provocar que la red

aprenda muy lentamente o incluso que deje de entrenarse por completo²⁰.

El gradiente desvaneciente aparece principalmente cuando se aplica el algoritmo de retropropagación en redes muy profundas, es decir de muchas capas. Este es justamente el caso de las redes recurrentes en su forma desenrollada, en las que aún se acentúa más ya que los pesos son compartidos en todos los pasos de tiempo [11]. Por lo tanto, mientras más largas sean las series de tiempo, más chances hay que nos enfrentemos a este problema. Esto provoca que estas redes sean incapaces de capturar dependencias de largo plazo al procesar secuencias de muchos pasos de tiempo²¹.

2.5.1. Long short-term memory

Las redes conocidas como *long short-term memory* (LSTM) son un tipo particular de RNRs que fueron introducidas en el año 1997 [13] para solucionar el problema del gradiente desvaneciente que tenían las RNRs estándar, haciéndolas capaces de aprender dependencias de largo plazo. Esta arquitectura ha sido utilizada con éxito en una gran variedad de problemas, y los mayores avances obtenidos con redes recurrentes se han logrado empleando este tipo de red [25].

Como hemos visto hasta el momento, todas las redes recurrentes tienen una estructura de cadena de “módulos” o “bloques” [25] — las neuronas recurrentes — que se repiten a lo largo de la secuencia temporal. En una RNR estándar, este módulo es simplemente una aplicación de una función de activación, particularmente la tangente hiperbólica, sobre una suma pesada de sus entradas. En las LSTM, estas neuronas se complejizan para incorporar mecanismos que permiten mantener una memoria a largo plazo en la red. Las llamaremos “celdas” o neuronas LSTM.

El principal componente que introducen las LSTM para poder retener información por largos períodos de tiempo es el **estado de celda**. La particularidad de este estado es que se transmite a lo largo del tiempo de manera casi inalterada, salvo por algunas interacciones lineales [25] [36]. Esto es lo que permite justamente solucionar el desvanecimiento del gradiente y aprender dependencias de largo plazo.

Así, vamos a tener dos estados que interactúan entre sí para realizar predicciones:

- Por un lado, el **estado de celda**, que representa la **memoria a largo plazo** de la red, y no es afectado de manera directa por ninguna activación no lineal ni ningún peso o sesgo.
- Por otro lado, el **estado oculto** (del que hablamos previamente), que representa la **memoria a corto plazo** de la red, y es el que se utiliza para generar las salidas

²⁰También está el problema del gradiente explosivo, que ocurre cuando el gradiente de la función de pérdida con respecto a un parámetro determinado es muy grande, causando que el aprendizaje sea muy inestable.

²¹La Sección 22.6 de [36] provee un ejemplo simple con cálculos para entender exactamente cómo se produce el problema del gradiente desvaneciente en las RNRs estándar.

de la red en cada paso de tiempo.

Ahora bien, con esto, la idea es que la red aprenda qué almacenar en el estado de celda, qué descartar y qué leer de él [8], y a esto lo hace a través de otros nuevos elementos llamados **compuertas**. En particular, se introducen tres tipos de compuertas en la celda LSTM:

- **La compuerta de olvido f** (del inglés *forget*): determina si cada elemento del estado de celda es recordado, copiándolo al paso siguiente, u olvidado, fijándolo en 0.
- **La compuerta de entrada i** (del inglés *input*): determina cuál va a ser la información que se va a utilizar para actualizar el estado de celda a partir de la entrada actual y del estado oculto actual.
- **La compuerta de salida o** (del inglés *output*): determina qué partes de la memoria de largo plazo son transferidas a la memoria de corto plazo.

De esta forma, las compuertas son una manera de regular el paso de información [25]. La salida de cada compuerta es un vector de valores, todos en el rango $[0, 1]$. Estos se obtienen como la salida de una pequeña red neuronal totalmente conectada en la que las entradas son los datos del paso de tiempo actual y el estado oculto anterior, y cuya capa de salida aplica una función de activación sigmoide.

A continuación, describimos los elementos presentes en el cómputo efectuado por una celda LSTM. Utilizaremos la siguiente notación para un tiempo t :

- x_t para el vector de entrada.
- h_t para el estado oculto.
- C_t para el estado de celda.
- \tilde{C}_t para el “potencial” estado de celda (veremos enseguida qué significa esto).
- f_t, i_t, o_t para las salidas de las compuertas de olvido, entrada y salida respectivamente.

Cabe aclarar que $h_t, C_t, \tilde{C}_t, f_t, i_t$ y o_t son todos vectores de la misma dimensión. Con estos elementos, podemos dar un diagrama de una celda LSTM, que se encuentra en la Figura 2.12.

Como se puede deducir a partir de dicha Figura, ahora vamos a tener varios conjuntos de pesos involucrados, cada uno de los cuales se puede representar como una matriz, como se mencionó en la Sección 2.3, y todas se aprenden durante el entrenamiento. Dejando de lado sus dimensiones, y asumiendo que todas incluyen los sesgos, las denotaremos de la siguiente forma:

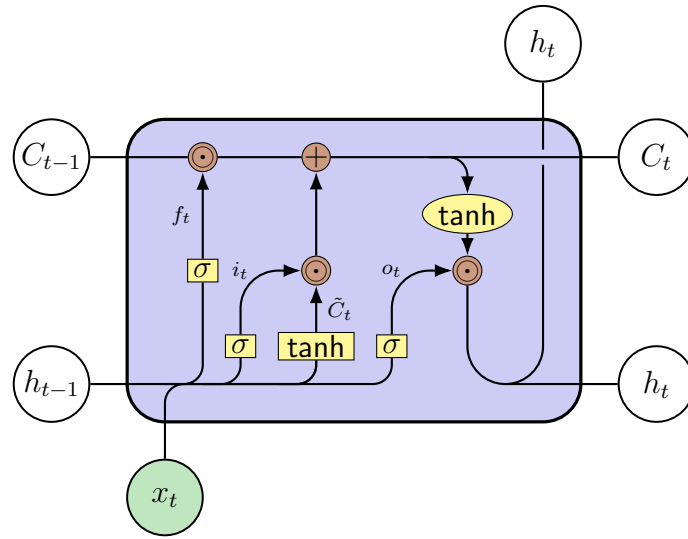


Figura 2.12: Diagrama de una celda LSTM adaptado de [25]. σ representa la función sigmoide, \tanh la tangente hiperbólica, y el signo \odot el producto de Hadamard, también conocido como “multiplicación elemento a elemento”. Toma como entradas x_t , el vector de variables del paso de tiempo t ; C_{t-1} , el estado de celda del paso anterior; y h_{t-1} , el estado oculto del paso anterior. Produce como salidas un nuevo estado de celda C_t y un nuevo estado oculto h_t , que se usa tanto en el paso de tiempo siguiente como en una hipotética capa sucesiva.

- $W_{x,f}$ y $W_{h,f}$ para los pesos de la compuerta de olvido.
- $W_{x,i}$ y $W_{h,i}$ para los pesos de la compuerta de entrada.
- $W_{x,o}$ y $W_{h,o}$ para los pesos de la compuerta de salida.
- $W_{x,\tilde{C}}$ y $W_{h,\tilde{C}}$ para los pesos de la potencial memoria a largo plazo.

En cada una, los subíndices indican de dónde hacia dónde van los pesos. Por ejemplo, $W_{x,f}$ es la matriz de pesos que va desde la entrada x_t hacia la compuerta de olvido f_t , y $W_{h,f}$ es la matriz de pesos que va desde el estado oculto h_{t-1} hacia la compuerta de olvido f_t . Con esto, veamos exactamente qué hace una neurona LSTM.

Para empezar, la celda toma tres entradas: el estado de celda del paso anterior C_{t-1} , el estado oculto del paso anterior h_{t-1} , y el vector de entrada del paso actual x_t . En cada paso, las tres compuertas actúan “internamente” para producir dos salidas: el nuevo estado de celda C_t y el nuevo estado oculto h_t .

La primera compuerta que actúa es la de olvido, para determinar que información ya presente en el estado de celda se “sigue recordando”. Esta decisión se toma aplicando una función sigmoide sobre el estado oculto anterior h_{t-1} , y la entrada actual x_t , produciendo

un número entre 0 y 1 para cada valor presente en C_{t-1} :

$$f_t = \sigma(\mathbf{W}_{x,f}x_t + \mathbf{W}_{x,h}h_{t-1})$$

El próximo paso es decidir qué nueva información ingresa a C_t utilizando la compuerta de entrada. Esta etapa se puede ver como compuesta por dos partes:

- Por un lado, se calcula una potencial memoria a largo plazo \tilde{C}_t . Esto se hace combinando el estado oculto del paso anterior h_{t-1} y la entrada actual x_t , y aplicando una tangente hiperbólica:

$$\tilde{C}_t = \tanh(\mathbf{W}_{x,\tilde{C}}x_t + \mathbf{W}_{h,\tilde{C}}h_{t-1})$$

- Por otro lado, a través de la compuerta de entrada, se calcula cuáles van a ser los valores de la memoria a largo plazo que se van a actualizar:

$$i_t = \sigma(\mathbf{W}_{x,i}x_t + \mathbf{W}_{h,i}h_{t-1})$$

Con estas tres cantidades (f_t, \tilde{C}_t, i_t) , es momento de efectivamente actualizar el estado de celda:

1. En primer lugar, se multiplica elemento a elemento (\odot) C_{t-1} por f_t . Esto provoca que un valor cercano a 0 en f_t haga que el valor correspondiente en C_{t-1} se “olvide”, y que un valor cercano a 1 haga que se siga recordando.
2. Y luego, se le suma la nueva información, dada por la multiplicación elemento a elemento entre \tilde{C}_t e i_t , que intuitivamente representa qué valores se van a actualizar y en qué medida.

De esta forma, la nueva memoria a largo plazo está dada por:

$$C_t = C_{t-1} \odot f_t + \tilde{C}_t \odot i_t$$

Por último, es necesario determinar cuál va a ser el nuevo estado oculto de la neurona. Este se construye como una versión “filtrada” del nuevo estado de celda, haciendo uso del valor de la compuerta de salida:

$$o_t = \sigma(\mathbf{W}_{x,o}x_t + \mathbf{W}_{h,o}h_{t-1})$$

Combinando o_t con una aplicación de tangente hiperbólica sobre el nuevo estado de celda C_t , que coloca a todos los valores allí presentes en el rango $[-1, 1]$, da la nueva memoria a corto plazo:

$$h_t = o_t \odot \tanh(C_t)$$

Una aclaración es que cuando estas redes se utilizan para llevar a cabo clasificación de secuencias, lo que se hace es tomar el estado **oculto** del último paso y, similarmente a lo que se hace en las redes convolucionales, se agregan una o más capas totalmente conectadas que procesen este vector para hacer la clasificación final.

Capítulo 3

Presentación del problema

El objetivo principal de toda evaluación de impacto es calcular el efecto de un programa o intervención sobre la población receptora. Para ello, es necesario obtener una estimación apropiada del contrafactual: ¿qué habría pasado con la variable de resultado de los beneficiarios del programa si el mismo no hubiera existido? Como esta es una situación hipotética, no observable en la realidad, se recurre a la conformación de un grupo de control, que debería estar formado, idealmente, por individuos que eran estadísticamente similares a los tratados antes de la implementación del programa — es decir, en la línea de base — pero que no fueron expuestos a la intervención en cuestión.

En esta tarea, aparece el problema del sesgo de autoselección, que ocurre cuando no se toman en cuenta las diferencias preexistentes entre tratados y no tratados. Estas diferencias pueden haber influido tanto en la decisión de participar en el programa como en los resultados potenciales posteriores, por lo que no considerarlas podría afectar la validez de las comparaciones entre ambos grupos.

En evaluaciones sobre programas con asignación aleatoria, el grupo de control se obtiene directamente tomando todos los individuos que no fueron tratados y el problema de autoselección se soluciona naturalmente. Sin embargo, en las evaluaciones cuasi-experimentales, hechas sobre intervenciones en donde la asignación al tratamiento no fue aleatoria y se desconocen los motivos que han llevado a los individuos a inscribirse o recibir el tratamiento, seleccionar un grupo de control adecuado constituye un gran desafío. Este es uno de los temas centrales de este trabajo.

Una técnica para abordar este problema es el PSM, discutida anteriormente y que es la que tomamos como punto de partida aquí. Este método empareja individuos tratados con no tratados basándose en su probabilidad estimada de participación, calculada a partir de un conjunto de variables que se consideran influyentes en el programa bajo cuestión.

Estas variables son seleccionadas por los evaluadores y pueden representar tanto características observadas en un instante determinado como también compartimientos a lo largo del tiempo. Por ejemplo, si consideramos un programa de financiamiento a empresas, se puede tomar como variable el número de empleados y los ingresos mensuales de sola-

mente el mes anterior al inicio del programa, o se podría tomar esta cantidad observada durante diez meses previos al comienzo, siempre y cuando el investigador considere que es relevante hacerlo. De esta forma, es posible incluir series de tiempo en el cálculo del puntaje de propensión, tomando como variables explicativas varios períodos de la misma característica. Esto permite capturar no solo un estado puntual de las unidades sino también cómo ha sido su evolución antes del inicio del programa, lo cual puede contribuir a un mejor emparejamiento y consecuentemente, a una estimación más precisa del contrafactual.

Otra aspecto a tener en cuenta a la hora de identificar los individuos de control es la forma de implementación del programa a lo largo del tiempo. En algunos casos, el ingreso al tratamiento por parte de los individuos se realiza en un único instante de tiempo, pero en otros la entrada al programa ocurre de manera secuencial. En este último caso, se habla de una **cohorte** o una **camada** para referirse al conjunto de individuos que ingresaron al programa en el mismo momento. Por ejemplo, si los ingresos al programa se dieron en dos meses distintos de un determinado año, entonces los individuos que ingresaron en primer mes conforman una cohorte, y los que ingresaron en el segundo forman otra cohorte distinta.

Habiendo presentado estas cuestiones, nuestro trabajo se enfoca en programas con entrada secuencial y en los que suponemos que los individuos tratados resultaron ser tratados (o decidieron inscribirse al programa) debido a la dinámica temporal observada en la variable objetivo¹ de cada uno en períodos anteriores a la asignación al programa. Bajo estas circunstancias, el PSM presenta ciertas limitaciones:

- Por un lado, cuando existen múltiples cohortes, la forma en la que se aplica la técnica es de a una cohorte a la vez. Es decir, se estima un modelo para calcular la probabilidad de participación por camada, y se buscan controles por separado para cada una de ellas. Dicho de otra manera, el proceso completo de emparejamiento se realiza tantas veces como cohortes haya.
- Por otro lado, cuando las variables tenidas en cuenta en realidad representan una misma característica observada en distintos períodos de tiempo, la regresión logística — que es una de las formas más comunes en la que se calcula el puntaje de propensión (y la que usamos en este trabajo) — no es capaz de capturar la relación temporal entre ellas, sino que las trata como si fueran independientes.

Con estos problemas en mente, el objetivo de este trabajo es explorar una alternativa basada en la utilización de redes neuronales para la identificación de grupos de control bajo los supuestos mencionados previamente. La estrategia propuesta se apoya en distintos tipos de arquitecturas, concretamente: redes convolucionales, redes LSTM, y la combinación de ellas. Como explicamos en el capítulo anterior, estas incorporan naturalmente la

¹Recordemos que la variable objetivo es aquella sobre la que se espera que el programa tenga un efecto.

relación temporal de los datos, permitiendo capturar patrones a lo largo del tiempo. En los escenarios bajo estudio, donde suponemos que la dinámica previa a la intervención es un factor clave para entender la participación en el programa, consideramos que estas redes pueden resultar especialmente útiles.

Para evaluar la metodología propuesta, desarrollamos diferentes conjuntos de datos sintéticos de tipo panel², ya que en los casos reales algunos comportamientos no pueden observarse directamente. En estos escenarios, variamos la cantidad de **períodos observados** (previos al inicio del programa), la cantidad de **períodos de dependencia temporal** — que son aquellos dentro de los períodos observados en los que modelamos una tendencia específica —, y la **dinámica temporal** observada en la variable. Teniendo esto en cuenta, los objetivos de nuestro trabajo se pueden resumir en los siguientes puntos:

- Obtener resultados que permitan comparar el desempeño de las diferentes arquitecturas de redes neuronales propuestas con el del PSM en los escenarios simulados.
- Evaluar la capacidad de las redes neuronales para identificar individuos de control ante variaciones en la cantidad de períodos observados, los períodos de dependencia y la dinámica temporal de la variable observada.
- Superar las limitaciones observadas del PSM utilizando las redes neuronales. Esto es: incorporar de manera más efectiva la dependencia temporal de las covariables y agilizar la identificación de los grupos de control de cada cohorte.

Nuestras hipótesis son las siguientes:

- Con una cantidad fija de períodos observados pre-tratamiento, esperamos que las redes neuronales sean más capaces de capturar la tendencia de los períodos de dependencia para obtener un mejor rendimiento que el PSM. Sin embargo, a medida que reducimos la cantidad de períodos de dependencia temporal, consideramos que su desempeño se degradará y se asemejará al del PSM.
- A medida que haya más períodos pre-tratamiento — en otras palabras, mayor disponibilidad de datos —, ambos métodos mejorarán sus resultados, pero siempre el de las redes será consistentemente superior.
- Las redes neuronales permitirán entrenar un único modelo que pueda utilizarse en diferentes cohortes, a diferencia del PSM, que requiere una estimación por cohorte.
- Las redes LSTM superarán el desempeño de las convolucionales, y la combinación de ambas dará un mejor rendimiento que el de cada una por separado.

²Los datos de tipo panel son aquellos en los que a cada unidad considerada le corresponde una serie de tiempo.

A continuación, presentamos el marco sobre el cual llevamos a cabo los experimentos. Detallamos el alcance de nuestro trabajo, el diseño de las diferentes simulaciones, las arquitecturas de redes propuestas, las métricas de interés para evaluar los modelos, la estrategia de comparación entre los dos enfoques, y las herramientas empleadas en todo este proceso.

Capítulo 4

Marco experimental

El alcance del trabajo desarrollado comprende las siguientes etapas, de las cuales hablaremos en este capítulo:

1. Generación de datos sintéticos.
2. Diseño de arquitecturas de redes neuronales y búsqueda de hiperparámetros.
3. Selección de métricas de desempeño relevantes para la evaluación de los modelos.
4. Entrenamiento, evaluación y comparación de las técnicas.

También nombraremos las diferentes herramientas utilizadas para llevar a cabo estas tareas.

4.1. Conjuntos de datos

Como mencionamos en el Capítulo 3, los experimentos fueron realizados con datos generados sintéticamente, diseñados para imitar situaciones reales. Para comprender cómo se construyeron estos datos, recordemos que nuestro objetivo es identificar unidades pertenecientes al grupo de control en escenarios donde asumimos que la asignación de individuos (o su inscripción) al tratamiento presenta las siguientes características:

- Es no aleatoria.
- Es secuencial, es decir que hay cohortes.
- Es dependiente de los comportamientos temporales observados en la variable objetivo del programa durante los períodos previos a su inicio.

Dicho esto, la síntesis de los datos involucró la creación de una serie de tiempo univa-

riada para cada individuo o unidad, que incorpora componentes autorregresivos¹, efectos fijos y efectos temporales², resultando así en un conjunto de datos de panel. Distinguimos entre tres tipos de individuos:

- **Individuos tratados:** son aquellos que han recibido el tratamiento en algún período, es decir que forman parte de alguna de las cohortes.
- **Individuos de control:** son aquellos que, en los datos sintéticos, sabemos que pertenecen al grupo de control para una determinada cohorte de tratados. Los llamaremos “controles” y son justamente aquellos que los modelos deberán identificar.
- **Individuos “NiNi”** (“ni tratados ni controles”): son aquellas unidades que no han sido tratadas y que, en los datos sintéticos, sabemos que no forman parte del grupo de control de ninguna cohorte.

Para los tratados y los controles, agregamos una característica extra además de la serie de tiempo, a la que llamamos “*inicio de programa*”³. En el caso de los tratados, esta variable indica el momento en que un individuo recibió el tratamiento (relativo a la cantidad de períodos observados de los individuos). Por ejemplo, si se consideran 20 períodos de tiempo previos al tratamiento y existen 3 cohortes, entonces los inicios de programa serán 21, 22 y 23. Por otro lado, para cada unidad de control, el inicio de programa representa el momento en que se aplicó la intervención a los individuos tratados para los cuales actúa como control.

Siguiendo lo explicado en capítulos anteriores, nuestro trabajo se enmarca dentro del aprendizaje supervisado. Por lo tanto, también se incluye la etiqueta que le corresponde a cada individuo: “1” para tratados y controles, y “0” para los NiNi. La razón por la cual tratados y controles comparten la misma etiqueta es que buscamos que los modelos identifiquen, durante su entrenamiento, los patrones presentes en los individuos tratados

¹Un **proceso autorregresivo de orden uno**, abreviado como **AR(1)** es un modelo de series de tiempo en donde el valor actual de la serie depende linealmente de su valor más reciente más una perturbación impredecible [44]. La fórmula de un AR(1) es la siguiente:

$$y_t = \rho \cdot y_{t-1} + e_t$$

donde a ρ se lo denomina coeficiente autorregresivo y e_t para $t = 0, 1, \dots, T$ es una secuencia de valores independiente e idénticamente distribuidos con media cero y varianza σ_e^2 . Para más información, se puede consultar el Capítulo 11.1 de [44].

²En el modelado de series de tiempo, una forma de capturar los efectos no observables que influyen en la variable dependiente y son constantes en el tiempo consiste en incorporar un factor denominado **efecto fijo**, el cual permanece invariante a lo largo de todos los pasos de tiempo. Para una explicación más detallada, se puede consultar el Capítulo 13.3 de [44]. Similarmente, los efectos no observables que sí cambian con el tiempo pero que afectan a todos los individuos por igual se conocen como **efectos temporales**.

³La forma en que incluimos la característica de “inicio de programa” en los NiNi se detallará en la Sección 4.4.

para luego poder clasificar como 1 a aquellos que se les asemejen. Estos serán justamente los que el modelo identifique como potenciales individuos de control.

La motivación detrás de esta construcción artificial es forzar que tanto tratados como controles compartan una determinada dinámica en su comportamiento temporal, mientras que los NiNi no. Cabe aclarar que la presencia de la dinámica exacta que modelamos constituye un supuesto que en la práctica puede resultar difícil de verificar. Nuestro interés radica en evaluar si al simularla, los algoritmos son capaces de capturar tales patrones.

Como mencionamos anteriormente, para cada individuo i generamos una serie de tiempo univariada y_i de una longitud L . La fórmula que utilizamos para simular cada valor $y_{i,t}$ de las series, con $t = 0, 1, \dots, L$ indicando el período generado, fue la siguiente:

$$y_{i,0} = \mu + u_{i,0} \cdot \sigma \quad (4.1)$$

$$y_{i,t} = \phi \cdot y_{i,t-1} + (1 - \phi) \cdot \mu + u_{i,t} \cdot \sigma \quad (t \geq 1) \quad (4.2)$$

donde:

$$\mu = \mu_{EF} + EF_i + \mu_{ET}$$

es una constante que depende del individuo i y:

- μ_{EF} representa la media de los efectos fijos para todos los individuos de un mismo grupo. Denotamos con μ_{EF_T} , μ_{EF_C} y $\mu_{EF_{NiNi}}$ la media de los efectos fijos para tratados, controles y NiNi respectivamente. En todos nuestros experimentos, tomamos $\mu_{EF_T} = \mu_{EF_C} = 10$, y el que sí variamos es $\mu_{EF_{NiNi}}$.
- μ_{ET} representa la media de los efectos temporales en todos los pasos de la serie y es la misma para todos los grupos.
- EF_i representa el efecto fijo del individuo i , constante en todos los pasos de tiempo.
- ϕ es el componente autorregresivo asociado a la variable generada para todos los individuos de un mismo grupo. Representa el grado de dependencia temporal entre el valor actual de la serie y el anterior. En nuestros experimentos, usamos $\phi = 0.9$ para todos los grupos.
- $u_{i,t}$ es un valor aleatorio proveniente de una distribución normal estándar (se genera uno en cada paso de tiempo y para cada individuo).
- σ es la desviación estándar del término de ruido de las series, y es la misma para todos los grupos. En este trabajo, tomamos $\sigma = 5$.

Generamos diversos escenarios, todos con 3 cohortes, que se diferencian entre sí por ciertos aspectos:

1. La longitud de las series de tiempo generadas, que denotamos con L . Esta representa la cantidad de períodos observados para cada individuo **previos al inicio del programa**.
2. La cantidad de períodos previos al inicio del programa en los que se observa un comportamiento específico en tratados y controles. A estos períodos, como mencionamos previamente, los llamamos “**períodos de dependencia** (temporal)”, y corresponden siempre a los últimos períodos pre-tratamiento. Denotamos la cantidad de períodos de dependencia con n_{pd} , y en nuestros escenarios, se va a cumplir siempre que $0 \leq n_{pd} < L$.
3. En los períodos de dependencia, modelamos un comportamiento al que llamamos “**decreciente con ruido**”: en dichos períodos, forzamos a que cada nuevo valor generado de la serie sea estrictamente menor al anterior, salvo en — a lo sumo — una cantidad m de ocasiones en donde permitimos que haya subidas o incrementos. Este m también varía entre escenarios. Consideramos que esta tendencia es representativa de fenómenos reales, y que puede influir en la decisión que los individuos reciban un tratamiento o participen de un programa.
4. La media de los efectos temporales para los individuos NiNi ($\mu_{EF_{NiNi}}$). Puntualmente, reducir este valor a la hora de generar los datos provoca que la media de la variable dependiente en los NiNi en cada uno de los períodos esté por debajo de la de tratados y controles⁴.

Una vez fijados los parámetros de un escenario, generamos 100 simulaciones para garantizar que los resultados obtenidos sean estadísticamente significativos.

A modo de ejemplo, la Figura 4.1 muestra el gráfico de 15 series de tiempo generadas para individuos tratados con $L = 45$, $n_{pd} = 20$ y $m = 6$. Como tenemos 3 cohortes y de cada individuo estamos tomando los 45 períodos antes que cada uno haya recibido el tratamiento (independientemente de la cohorte a la que pertenezcan), los inicios de programa, aunque no se visualizan explícitamente, son 46, 47 y 48 (uno para cada cohorte).

En todos los escenarios y simulaciones, la distribución de individuos fue la siguiente:

- Cantidad de individuos tratados: 1000, divididos en las 3 cohortes (334 en la primera y la segunda, y 332 en la restante).
- Cantidad de individuos de control: 1000, divididos en las 3 cohortes (334 en la primera y la segunda, y 332 en la restante).
- Cantidad de individuos NiNi: 3500.

En la Sección 4.4, explicamos cómo a partir de los datos generados, construimos los conjuntos de entrenamiento y test, y cómo fueron utilizados para obtener los resultados.

A continuación, describimos las diferentes arquitecturas de redes neuronales que uti-

⁴Este efecto se verá con mayor claridad en el Experimento 4 del Capítulo 5.

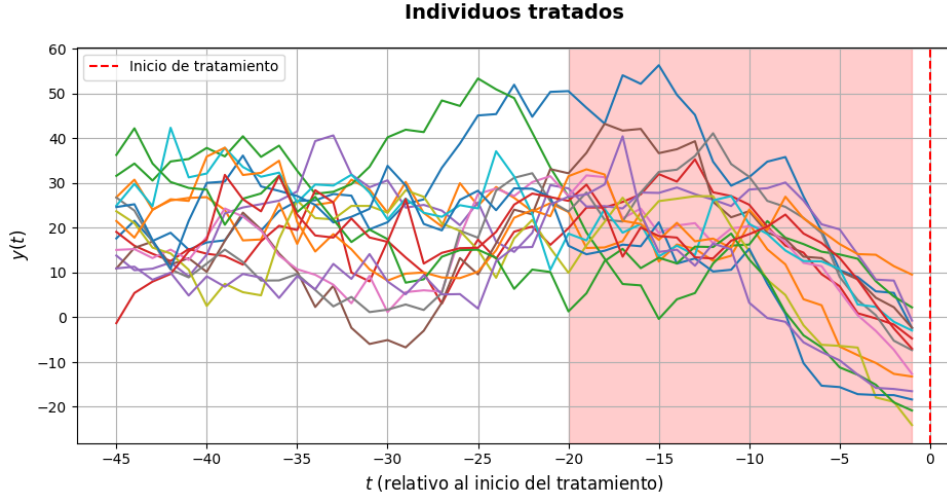


Figura 4.1: Gráfico de 15 series de tiempo generadas para individuos tratados con $L = 45$, $n_{pd} = 20$, y $m = 6$. Los tratados están divididos en cohortes, cada una de las cuales tiene su correspondiente inicio de programa, por lo que en el eje x se encuentra el período relativo al inicio de programa de cada individuo (por ejemplo, -5 indica el quinto período antes que una unidad reciba el tratamiento, independientemente de cuándo lo recibió). El eje y representa el valor de la variable en cada período. La línea vertical punteada de color rojo indica el inicio de tratamiento de cada individuo, y los períodos con fondo rojo son los períodos de dependencia.

lizamos durante los experimentos y los valores particulares que fueron incluidos en la búsqueda de hiperparámetros.

4.2. Arquitecturas de redes neuronales

Para empezar, un aspecto a notar sobre nuestros datos es que cada muestra contiene una característica temporal (la serie de tiempo univariada) y una “estática” (el inicio de programa). En todas las arquitecturas implementadas, se procesó la serie temporal por un lado y, para realizar la clasificación, combinamos el resultado de dicho procesamiento con la entrada estática.

La salida de cada modelo es un número real que representa la probabilidad que la entrada dada⁵ — caracterizada por la serie de tiempo y el inicio de programa — pertenezca

⁵En realidad, lo que producen las redes como salida es un valor real, y no una probabilidad propiamente dicha. Para interpretarlo como tal, se debe aplicar una función sigmoide. En nuestro caso, esta función no se incluye en la propia red porque durante el entrenamiento, PyTorch — la librería que utilizamos para entrenar las redes neuronales — provee una función de pérdida llamada `BCEWithLogitsLoss` que realiza internamente esta “conversión”.

a la categoría con etiqueta 1, que en nuestro caso representa formar parte del grupo de control. Para obtener la clase final (0 o 1), se aplica — por fuera del modelo propiamente dicho — un **umbral de decisión** sobre esta probabilidad: si la probabilidad es mayor que dicho umbral, se considera que el modelo ha clasificado a la entrada como perteneciente a la clase 1. En este trabajo, fijamos este umbral en 0.5⁶.

Dejamos algunos hiperparámetros fijos para el entrenamiento y las arquitecturas de las redes:

- Número de épocas de entrenamiento: 100.
- Número de capas para el procesamiento de series temporales: 2.
- Optimizador: Adam.

Y aquellos sobre los que llevamos a cabo una búsqueda fueron los siguientes:

- Tamaño de lote: probamos con 32, 64 y 128.
- Número de neuronas por capa: probamos 32, 64 y 128.
- Tasa de aprendizaje: probamos con 0.01, 0.001 y 0.0001.
- Dropout: probamos con 0.3, 0.5 y 0.7.

Para hacer la búsqueda de hiperparámetros, utilizamos la técnica de validación cruzada sobre los conjuntos de entrenamiento con 3 particiones, cada una manteniendo las proporciones originales de 0s y 1s del conjunto de entrenamiento. La métrica que se buscó optimizar fue el puntaje F_1 sobre la clase positiva — discutido en la sección siguiente — promediado sobre las 3 iteraciones de la validación cruzada.

En lo que sigue, incluimos ilustraciones y detallamos en profundidad las arquitecturas utilizadas. Para diseñarlas, tomamos como base las propuestas en [41], [16] y [38].

4.2.1. Arquitectura 1: Red LSTM

Como se observa en la Figura 4.2, en la primera arquitectura propuesta, la serie de tiempo es procesada por dos capas LSTM con una capa intermedia de dropout. Posteriormente, se realiza una operación de concatenación, en la que se combina la salida del “bloque” LSTM⁷ con la característica correspondiente al inicio del programa, formando un vector unidimensional. Este vector se introduce luego en una red totalmente conectada compuesta por una capa oculta de 128 neuronas con función de activación ReLU, seguida de una capa de dropout, y finaliza en una capa de salida que produce el valor resultante de la red.

⁶Tomamos esta decisión con el objetivo de no introducir un nuevo factor de variabilidad en nuestros experimentos.

⁷Como explicamos en la Sección 2.5, la salida de este bloque corresponde al estado oculto del último paso de tiempo de cada serie.

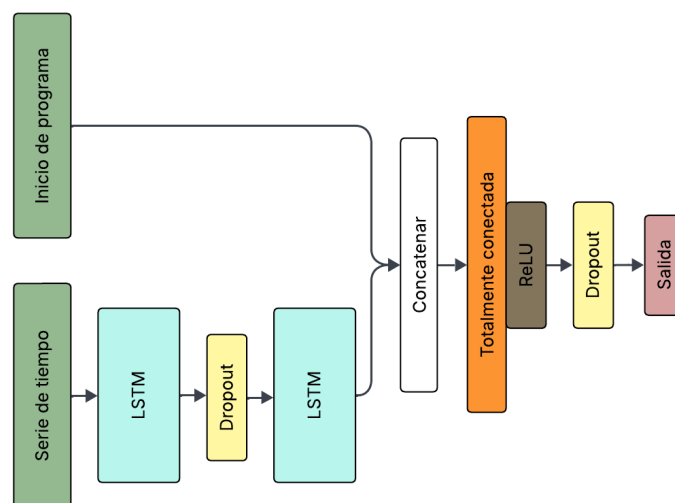


Figura 4.2: Esquema de la arquitectura de la red LSTM propuesta.

4.2.2. Arquitectura 2: Red Convolucional

El segundo modelo propuesto sigue la misma estructura general que la arquitectura anterior, pero reemplazando las capas LSTM por capas convolucionales. Concretamente, ahora el procesamiento de la serie temporal se realiza mediante dos capas convolucionales, cada una con normalización por lotes y activación ReLU, con una capa intermedia de dropout. A estas dos, les sigue una segunda capa de dropout y una operación de average pooling. La primera capa convolucional aplica 128 filtros de tamaño 3 con padding, la segunda aplica 256 filtros de tamaño 2 sin padding, y el filtro del pooling es de tamaño 2. Finalmente, la salida resultante se convierte en un vector unidimensional — se “aplana” —, que se concatena con el inicio de programa y se introduce en las capas posteriores del modelo, encargadas de producir la salida de la red. La Figura 4.3 muestra un diagrama de esta arquitectura.

4.2.3. Arquitectura 3: Red Convolucional + LSTM

Esta última arquitectura combina las dos anteriores: el procesamiento de la serie temporal es llevado a cabo de manera paralela por las capas LSTM de la Arquitectura 1 y las capas convolucionales de la Arquitectura 2. Luego, se combinan las salidas de ambos bloques y el inicio de programa para posteriormente producir la predicción final a través de la red totalmente conectada. La Figura 4.4 muestra un diagrama de esta arquitectura.

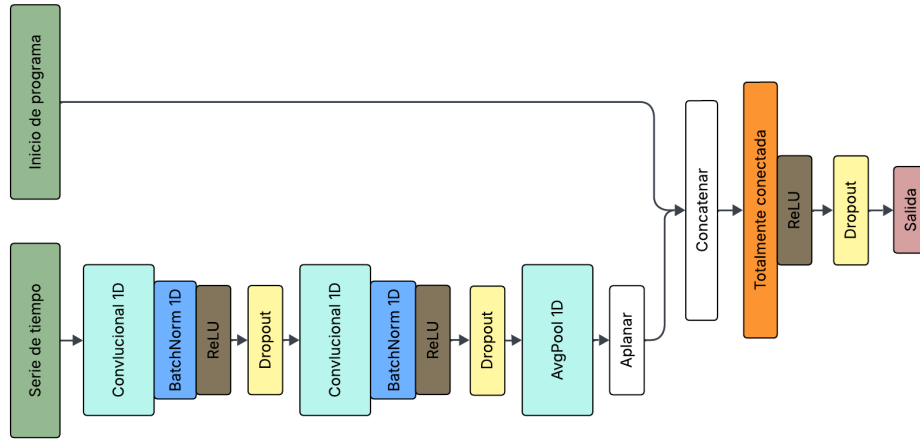


Figura 4.3: Esquema de la arquitectura de la red convolucional propuesta.

4.3. Métricas

En problemas de clasificación binaria como el que tratamos aquí, existen diferentes métricas para evaluar el desempeño del modelo utilizado, siempre sobre el conjunto de test. Las que más relevancia tendrán depende del contexto del problema. Para describirlas, utilizaremos la siguiente notación, asumiendo que la etiqueta 1 corresponde a la clase “positiva” y la 0 a la “negativa”:

- **VP**: Verdaderos Positivos. Es la cantidad de predicciones correctas para la clase positiva. Es decir, son aquellas observaciones cuya etiqueta verdadera es 1 y que el modelo también clasificó como 1. En nuestro caso, los *VP* serán los individuos de control que el modelo haya identificado correctamente como tales.
- **VN**: Verdaderos Negativos. Es la cantidad de predicciones correctas para la clase negativa. Es decir, son aquellas observaciones cuya etiqueta verdadera es 0 y que el modelo también clasificó como 0. En nuestro caso, los *VN* serán los individuos NiNi que el modelo haya identificado correctamente como tales.
- **FP**: Falsos Positivos. Es la cantidad de predicciones incorrectas para la clase negativa. Es decir, son aquellas observaciones cuya etiqueta verdadera es 0 pero que el modelo clasificó como 1. En nuestro caso, los *FP* serán los individuos NiNi que el modelo haya identificado incorrectamente como de control.
- **FN**: Falsos Negativos. Es la cantidad de predicciones incorrectas para la clase positiva. Es decir, son aquellas observaciones cuya etiqueta verdadera es 1 pero que el modelo clasificó como 0. En nuestro caso, los *FN* serán los individuos de control que el modelo haya identificado incorrectamente como NiNi.

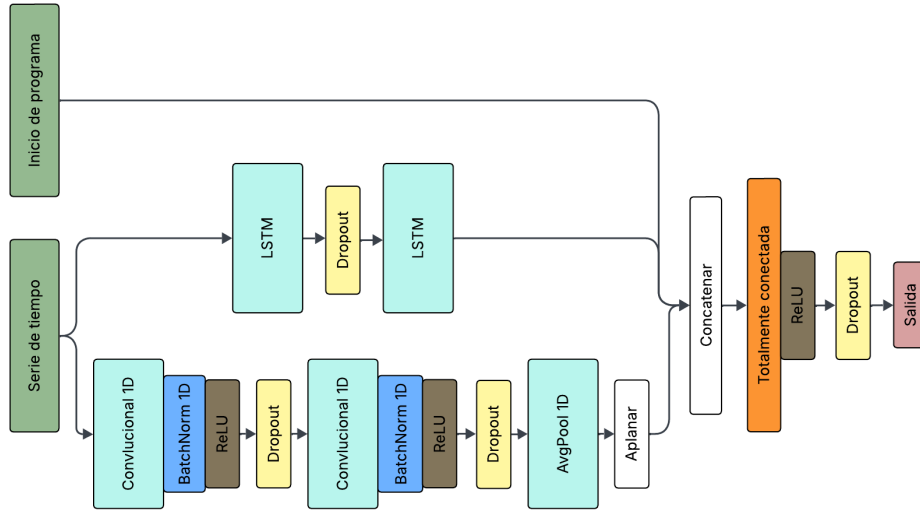


Figura 4.4: Esquema de la arquitectura de la red LSTM + convolucional propuesta.

Todas estas cantidades se suelen presentar de forma conjunta en lo que se denomina **matriz de confusión**, cuya representación se encuentra en la Figura 4.5.

Con estos conceptos en mente, algunas de las métricas que se suelen considerar son las siguientes, cuyo valor mínimo es 0 y el máximo es 1:

4.3.1. Exactitud

Es la proporción de entradas para las cuales el modelo predijo la salida correcta. Su fórmula está dada por:

$$Exactitud = \frac{VP + VN}{VP + VN + FP + FN}$$

Utilizar esta medida no es recomendable en datasets desbalanceados, es decir en aquellos en donde hay un alto porcentaje de una clase pero bajo de otra. Veamos por qué con un ejemplo.

Supongamos que tenemos un dataset de 100 muestras en donde el 90 % de los datos tiene etiqueta 0 y el restante 10 % tiene etiqueta 1. Se puede ver que si el modelo predice un valor de 0 para cualquier entrada, entonces tendría una exactitud del 90 %, que es un porcentaje bastante alto, pero se habrá equivocado en todas las instancias positivas. Por esto, suele ser necesario prestar atención a otros valores.

Verdadero	Clase 0	VN	FP
	Clase 1	FN	VP
		Clase 0	Clase 1
		Predicción	

Figura 4.5: Matriz de confusión.

4.3.2. Precisión

Es la proporción de Verdaderos Positivos sobre todos los positivos detectados por el modelo. Una precisión de 1 indica que cada elemento predicho como 1 efectivamente era un 1, y por ejemplo una de 0.5 indica que de las entradas predichas como 1, solamente la mitad era realmente un 1. Su fórmula está dada por:

$$Precisión = \frac{VP}{VP + FP}$$

Nuevamente, utilizar solamente la precisión puede ser engañoso. Una forma de tener una precisión perfecta es crear un clasificador que siempre prediga el valor 0, excepto en una única instancia de la que esté más seguro de predecir 1. Si esta predicción es correcta, entonces tendríamos $VP = 1$, $FP = 0$, dando de esta forma una precisión de 1, pero habiendo acertado solamente una instancia positiva.

4.3.3. Sensibilidad o cobertura

Es la proporción de instancias positivas verdaderas predichas correctamente por el modelo, por lo que también se la suele llamar tasa de verdaderos positivos. Una cobertura de 1 indica que cada observación que era un 1 fue predicha como 1 por el modelo. Su fórmula está dada por:

$$Sensibilidad = \frac{VP}{VP + FN}$$

Prestar atención solamente a la sensibilidad también es riesgoso. Si volvemos al dataset de 100 muestras donde el 90% del dataset tiene etiqueta 0 y el restante 10% tiene

etiqueta 1, podemos obtener una sensibilidad de 1 simplemente construyendo un modelo que siempre prediga 1, ya que tendríamos $VP = 10$ y $FN = 0$. Sin embargo, tendríamos una precisión de 0.1, ya que $FP = 90$.

Dicho esto, existe una medida que combina a estas dos métricas, y que es justamente a la que más importancia damos en este trabajo.

4.3.4. Puntaje F1

El puntaje F_1 es la media armónica entre la precisión y la sensibilidad, lo que significa que da igual importancia a ambas. Si denotamos con P a la precisión y con S a la sensibilidad, la fórmula del puntaje F_1 está dada por:

$$F_1 = \frac{2}{\frac{1}{P} + \frac{1}{S}} = 2 \cdot \frac{P \cdot S}{P + S}$$

A partir de aquí, se puede ver que el valor de esta métrica será alto solamente si los valores de S y P son altos y que, si al menos uno de los dos es bajo, entonces el F_1 “tenderá” hacia ese valor. Si por ejemplo $P = 1$ pero $S = 0.5$, entonces tendríamos $F_1 = 0.67$. En cambio, si $P = 1$ pero $S = 0.8$, tendríamos $F_1 = 0.89$.

Como mencionamos anteriormente, la métrica a la que más le prestamos atención en este trabajo para evaluar el desempeño de los modelos es el puntaje F_1 . Consideramos que maximizarla resulta adecuado en nuestro contexto ya que nos permite encontrar un equilibrio entre dos objetivos: por un lado, identificar la mayor cantidad posible de individuos de control (es decir, lograr una alta cobertura) y a la vez, minimizar la cantidad de falsos positivos (es decir, aumentar la precisión). De esta forma, el puntaje F_1 guía la búsqueda de hiperparámetros⁸ y también la comparación de los diferentes modelos.

4.4. Entrenamiento, evaluación y comparación

Como mencionamos en el Capítulo anterior, uno de nuestros objetivos es obtener resultados que permitan comparar el desempeño de las redes neuronales con el del PSM en los distintos escenarios. Las diferencias metodológicas inherentes a ambas técnicas hacen que esto sea un desafío. En esta sección, describimos la estrategia propuesta en este trabajo para abordarlo.

Para empezar, a partir de los datos generados, construimos los conjuntos de entrenamiento y test de la siguiente manera, con la idea que los modelos aprendan los patrones vistos en los tratados para luego poder identificar los controles:

⁸En el sentido que la mejor combinación de hiperparámetros va a ser aquella para la cual se haya obtenido el máximo F_1 (dentro de las combinaciones probadas).

- **Conjunto de entrenamiento:** individuos tratados de todas las cohortes (1000 en total) y una muestra aleatoria de 1000 NiNi.
- **Conjunto de test:** individuos de control de todas las cohortes (1000 en total) y los 2500 NiNi restantes (de los 3500 generados en cada escenario).

Un detalle a tener en cuenta es que tanto tratados como controles tienen una característica extra: el inicio de programa. Sin embargo, esta no está presente en los NiNi, justamente porque no fueron tratados y porque no forman parte del grupo de comparación en ninguna de las cohortes. Para reflejar esto e intentar “transmitírsele” al modelo, por cada NiNi incluido, generamos versiones múltiples de cada uno, simulando cómo se verían si pertenecieran a distintas cohortes. Puntualmente, a cada uno lo repetimos tantas veces como inicios de programa (o cohortes) haya, utilizando en cada repetición los períodos previos al respectivo inicio. En todas las repeticiones, la etiqueta asignada es 0. Por lo tanto, los conjuntos de entrenamiento y test finales fueron los siguientes:

- **Conjunto de entrenamiento:**
 - Individuos tratados de todas las cohortes (1000) con etiqueta 1.
 - 1000 individuos NiNi, cada uno repetido tantas veces como cohortes haya, con los períodos previos al inicio de programa de cada cohorte (tomando 3 cohortes, terminan siendo 3000), y en todas las repeticiones con etiqueta 0.
- **Conjunto de test:**
 - Individuos de control de todas las cohortes (1000) con etiqueta 1.
 - 2500 individuos NiNi, cada uno repetido tantas veces como cohortes haya, con los períodos previos al inicio de programa de cada cohorte (tomando 3 cohortes, terminan siendo 7500), y en todas las repeticiones con etiqueta 0.

Ahora bien, para evaluar las redes, se toman los resultados obtenidos en el conjunto de test, el cual permanece oculto durante el proceso de entrenamiento. Sin embargo, en el PSM no existe esta noción: todos los individuos disponibles son utilizados para estimar los coeficientes de la regresión logística, calcular las probabilidades y hacer el emparejamiento.

Además, otra distinción fundamental es que las redes, incluyendo la feature de inicio de programa, posibilitan la utilización de un solo modelo para evaluarlo sobre las distintas cohortes, mientras que el PSM requiere realizar una estimación por cohorte.

Debido a estas diferencias identificadas, para poder hacer una comparación razonable, adaptamos los dos métodos. Para el PSM, lo que hicimos fue modificarlo de tal forma que permita incorporar la noción de “entrenamiento” y “test”; y para las redes, cambiamos la forma de evaluarlas para poder hacerlo por cohorte. Los detalles del procedimiento propuesto se encuentran en la Tabla 4.1, donde es importante destacar que, para cada métrica, el valor reportado en una simulación corresponde al promedio de los obtenidos en las tres cohortes.

	Redes neuronales	PSM
Entrenamiento	Se utiliza el conjunto de entrenamiento en su totalidad para construir un único modelo capaz de incorporar diferentes cohortes.	Para cada cohorte, se estima una regresión logística utilizando los tratados de la cohorte y los NiNi del conjunto de entrenamiento correspondientes al inicio de programa de la cohorte.
Evaluación	Se divide al conjunto de test en tantos subconjuntos como cohortes haya. Cada uno de ellos está formado por los controles de una cohorte y los NiNi del conjunto de test correspondientes a esa cohorte. Con el modelo entrenado, se evalúa en cada uno de estos subconjuntos, y el resultado final es el promedio de los obtenidos en cada partición.	Se divide al conjunto de test en tantos subconjuntos como cohortes haya. Cada uno de ellos está formado por los controles de una cohorte y los NiNi del conjunto de test correspondientes a esa cohorte. Con la regresión estimada en cada cohorte, se hace la inferencia de la probabilidad de participación sobre los tratados y la partición de test correspondientes a la cohorte; y con ello se lleva a cabo el emparejamiento. Se evalúa el emparejamiento en cada cohorte, y el resultado final es el promedio de los obtenidos en cada cohorte.

Tabla 4.1: Comparación entre el procedimiento de evaluación de redes neuronales y PSM.

4.5. Herramientas

A continuación, nombramos las herramientas que más 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. En la bibliografía, se incluye el enlace a su documentación.

4.5.1. Hardware

Para el entrenamiento de los modelos y la búsqueda de hiperparámetros, utilizamos recursos computacionales de UNC Supercómputo (CCAD) de la Universidad Nacional de Córdoba [40], que forman parte del Sistema Nacional de Computación de Alto Desempeño

(SNCAD) de la República Argentina. En particular, hicimos uso de la Computadora Nabucodonosor [7], destinada especialmente a potenciar el uso del ML. Cuenta con 2 procesadores Xeon E5-2680 v2 de 10 núcleos cada uno, 64 GiB de RAM, y 1 GPU NVIDIA A30 con 24 GiB de VRAM.

4.5.2. Python

Python [30] es un lenguaje de programación interpretado, orientado a objetos y con tipado dinámico. Es multipropósito, por lo que se puede usar para una diversidad de tareas. Otros de sus aspectos más relevantes son su sintaxis simple e intuitiva con énfasis en la legibilidad, el alto grado de abstracción que provee, la gran cantidad de librerías que existen, y el extenso soporte de la comunidad. Además, es gratis, de código abierto y multiplataforma, por lo que puede ejecutarse en diferentes sistemas operativos.

Durante los últimos años, ha sido el lenguaje más utilizado en las áreas de ciencia de datos y aprendizaje automático, principalmente por el desarrollo de librerías como *pandas*, *NumPy*, *PyTorch*, *TensorFlow*, y *Scikit-learn*, que facilitan el manejo de grandes volúmenes de datos y la utilización de algoritmos de ML.

En nuestro caso, utilizamos Python para toda la implementación de este trabajo.

4.5.3. Jupyter

Jupyter [15] es una plataforma que ofrece un entorno de desarrollo interactivo a través de documentos llamados “*notebooks*”, organizados en unidades denominadas “celdas”. La potencia de estas notebooks radica en que pueden contener al mismo tiempo celdas de código, texto en lenguaje natural escrito en Markdown, imágenes, y diversas otras representaciones, como tablas y diagramas. Estos documentos proporcionan una forma rápida y práctica de prototipar y explicar código, explorar y visualizar datos, y realizar chequeos intermedios observando la salida de cada celda, lo cual contribuye a prevenir errores. Jupyter soporta diferentes lenguajes de programación y permite la utilización de entornos virtuales.

En este proyecto, implementamos un *pipeline* mediante notebooks de Jupyter que abarca desde la carga, visualización y transformación de los datos, hasta la selección, entrenamiento y evaluación de los modelos.

4.5.4. Pandas

Pandas [27] es una librería de Python que provee estructuras de datos rápidas, flexibles y expresivas que permiten trabajar de manera fácil e intuitiva con datos tabulares. Las dos principales estructuras de datos de *pandas* son las **Series** (unidimensionales) y los **DataFrames** (bidimensionales). En cada una, se pueden utilizar diferentes tipos de datos, como números enteros, decimales, palabras, fechas, entre otros. Algunas de sus capaci-

dades destacadas incluyen la lectura y escritura de datos en diferentes formatos (CSV, Microsoft Excel, Stata, etcétera), el manejo de datos faltantes, y la simple y eficiente transformación de datos.

Utilizamos **DataFrames** para almacenar los datos sintéticos en archivos y luego cargarlos, y para construir los conjuntos de entrenamiento y test.

4.5.5. NumPy

NumPy [24] es una librería de Python que permite el cómputo numérico. Ofrece arreglos N-dimensionales, funciones matemáticas variadas, generadores de números aleatorios, rutinas de álgebra lineal, y más. El núcleo de NumPy es código C optimizado, por lo que combina la flexibilidad de Python con la rapidez propia del código compilado. Estas características hacen que sea la base de muchas otras librerías utilizadas en ámbitos muy diversos.

Empleamos NumPy principalmente para crear las series de tiempo de los diferentes individuos en matrices, que luego pasamos a **DataFrames**.

4.5.6. PyTorch

PyTorch [31] es una librería de Python que facilita la utilización de redes neuronales, tanto en su diseño como en su entrenamiento. Provee cómputo acelerado mediante el uso de placas gráficas, e implementa el algoritmo de retropropagación del error de manera eficiente. La mayoría de PyTorch está escrito en C++ y CUDA.

Aquí, utilizamos PyTorch justamente para diseñar las diferentes arquitecturas de redes neuronales, que se implementan como clases en Python, y para llevar a cabo el ciclo de entrenamiento de los distintos modelos.

4.5.7. Scikit-learn

Scikit-learn [37] es una biblioteca de Python de código abierto para aprendizaje automático. Provee varias herramientas útiles que facilitan tareas como el preprocesamiento de datos, el entrenamiento y la evaluación de modelos, el cálculo de métricas, la búsqueda de hiperparámetros, entre otros.

En nuestro caso, usamos Scikit-learn para automatizar el proceso de validación cruzada durante la búsqueda de hiperparámetros, y para la evaluación final de los modelos mediante el cálculo automático de diversas métricas.

4.5.8. Optuna

Optuna [26] es una librería de Python que permite realizar la optimización automática de hiperparámetros de una manera eficiente. Resulta intuitiva de utilizar y su incorpora-

ción en un pipeline es bastante directa. Un aspecto muy útil es que incluye un *dashboard* interactivo que posibilita la visualización en tiempo real de los valores de hiperparámetros evaluados en los diferentes ensayos, como así también de gráficos que reflejan la relación entre estos valores y el resultado obtenido con ellos. Otra característica relevante es que Optuna permite la paralelización de las evaluaciones, lo que contribuye a acelerar el proceso de optimización.

4.5.9. MLflow

MLflow [21] es una librería de Python que permite llevar un registro fino de las diferentes etapas y los diferentes experimentos llevados a cabo en un proyecto de ML. Provee una interfaz con la cual se pueden crear diferentes proyectos, y dentro de cada uno diferentes experimentos. En cada uno de ellos, permite cargar los parámetros y valores que el usuario crea adecuados (como pueden ser los hiperparámetros empleados para entrenar el modelo), los datasets utilizados, y diferentes archivos que pueden incluir gráficos, métricas, y hasta incluso los pesos del modelo entrenado.

4.5.10. Stata

Stata [39] es un paquete de software estadístico que ofrece una gran variedad de herramientas para el análisis exhaustivo de datos, incluyendo tareas de manipulación, exploración, visualización e inferencia estadística. Es utilizado en disciplinas como economía, políticas públicas, educación y salud pública, entre otras. Además, ofrece compatibilidad con Python mediante la biblioteca `pystata`, la cual permite integrar de forma eficiente los objetos `DataFrame` de `pandas` con las funcionalidades de Stata.

En este trabajo, utilizamos Stata para la implementación del PSM, específicamente a través de las funciones `logit` y `psmatch2`.

Capítulo 5

Resultados

En lo que sigue, presentamos los resultados obtenidos con las diferentes arquitecturas de redes neuronales y con el método de PSM en los escenarios modelados. Para cada uno, detallamos los parámetros utilizados: períodos observados (L), períodos de dependencia (n_{pd}), cantidad máxima de incrementos en los períodos de dependencia (m) y media de los efectos fijos de los NiNi ($\mu_{EF_{NiNi}}$).

Recordemos que, fijados los parámetros de un escenario, se generaron 100 simulaciones del mismo con el objetivo de obtener resultados estadísticamente significativos. Tanto para las diferentes arquitecturas de redes como para el PSM, se midió su rendimiento en cada una de las 100 simulaciones, y luego se calculó el promedio.

En el caso de las redes neuronales, cada simulación incluyó una búsqueda de hiperparámetros sobre 16 combinaciones distintas, realizada automáticamente por Optuna. El resultado en cada simulación corresponde a la red entrenada con la mejor combinación de hiperparámetros hallada para esa simulación.

Mostramos los promedios del puntaje F_1 , la precisión y la cobertura en el conjunto de test a lo largo de las 100 simulaciones. Cabe aclarar que, siguiendo lo explicado en la Sección 4.4, dada una métrica determinada, el resultado obtenido en una simulación es el promedio de esa métrica entre las cohortes. Además, para las redes neuronales, de cada hiperparámetro mostramos el valor que fue seleccionado como el óptimo con mayor frecuencia en las simulaciones correspondientes a un escenario.

Dado que nuestra métrica principal es el puntaje F_1 , verificamos si las diferencias observadas en dicha métrica entre cada arquitectura y el PSM son estadísticamente significativas mediante pruebas de hipótesis de diferencia de medias. Los detalles y resultados de estas pruebas se encuentran en el Anexo. En todos los casos, rechazamos la hipótesis nula que plantea la igualdad de las medias poblacionales, lo cual implica que las diferencias observadas resultan estadísticamente significativas.

5.1. Experimento 1: 45 períodos observados, 20 de dependencia

En este primer escenario, tomamos 45 períodos observados previos al inicio del programa para cada individuo con 20 períodos de dependencia, en donde permitimos un máximo de 6 incrementos. Así, la proporción de períodos de dependencia sobre períodos observados es de 0.44, y de subidas sobre períodos de dependencia de 0.3. Al tener 45 períodos observados pre-tratamiento y 3 cohortes, los inicios de programa son 46, 47 y 48. Además, el valor de $\mu_{EF_{NiNi}}$ en este caso fue el mismo que para tratados y controles (10).

Consideramos que este escenario es el que más favorece a las redes ya que es en el que mayor cantidad de períodos observados y de dependencia hay, por lo que lo tomamos como punto de referencia para comparar lo ocurrido en los otros. Asimismo, creemos que tomar 45 períodos de observación como la cantidad máxima en todos los escenarios es acorde con la realidad, en donde en algunas situaciones puede resultar complicado tener información sobre más períodos, como suele ocurrir, por ejemplo, en el caso de pequeñas y medianas empresas.

La Figura 5.1 muestra algunos ejemplos de las series de tiempo generadas para cada uno de los grupos, en donde se ve claramente la tendencia que estamos tratando de capturar en tratados y controles. En ellos, en los últimos 20 de los 45 períodos considerados se observa el comportamiento “decreciente con ruido”, que en los NiNi no está presente.

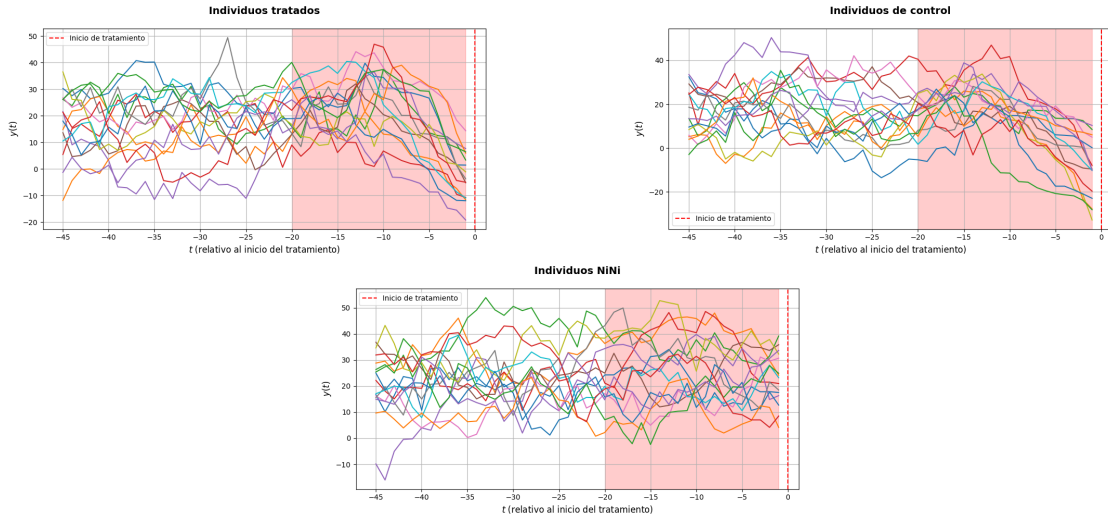


Figura 5.1: Ejemplos de series de tiempo generadas para cada grupo en el Experimento 1. Los períodos con un fondo rojo indican los períodos de dependencia para tratados y controles, en donde se modela la tendencia “decreciente con ruido”. La línea punteada roja indica el inicio de tratamiento para cada individuo.

La Tabla 5.1 muestra los resultados obtenidos en las métricas F_1 , precisión y cobertura por las diferentes redes y el PSM. Cada valor representa el promedio de esa métrica en las 100 simulaciones en el conjunto de test, junto con la desviación estándar.

	Puntaje F_1	Precisión	Cobertura
LSTM	0.80984 ± 0.02679	0.72408 ± 0.04763	0.92219 ± 0.01823
Convolutacional	0.83945 ± 0.01999	0.76293 ± 0.03537	0.93468 ± 0.01383
LSTM + Convolutacional	0.83694 ± 0.01836	0.76084 ± 0.03397	0.93161 ± 0.01418
PSM	0.65882 ± 0.01561	0.65998 ± 0.01556	0.65767 ± 0.01573

Tabla 5.1: Promedios y desviaciones estándar de las métricas F_1 , precisión y cobertura sobre la clase positiva (controles), evaluadas en el conjunto de test en las 100 simulaciones del Experimento 1 con las diferentes arquitecturas de redes neuronales y el PSM.

En primer lugar, se puede ver que el desempeño de todas las redes fue superior al del PSM, y consideramos que los resultados obtenidos con ellas son satisfactorios. En segundo lugar, el rendimiento de las redes fue parecido entre ellas; más aún entre la Convolutacional y la LSTM + Convolutacional, siendo la primera apenas mejor. Algo que también se puede observar es que la cobertura obtenida con todas las redes es bastante alta, lo cual indica que la mayoría de los controles fueron identificados por la red; mientras que la precisión fue consistentemente menor, aunque con valores aceptables.

La Tabla 5.2 muestra, para cada arquitectura, el valor de cada hiperparámetro que fue seleccionado con mayor frecuencia como el óptimo durante la búsqueda en las 100 simulaciones. Consideramos que de utilizar alguna de las arquitecturas, estos valores deberían ser los empleados para entrenarla.

	Tamaño de lote	Neuronas en capas ocultas	Tasa de aprendizaje	Dropout
LSTM	128 (44 %)	128 (69 %)	0.001 (98 %)	0.3 (54 %)
Convolutacional	32 (46 %)	-	0.001 (90 %)	0.3 (66 %)
LSTM + Convolutacional	32 (37 %)	64 (37 %)	0.001 (86 %)	0.3 (72 %)

Tabla 5.2: Valores de hiperparámetros seleccionados con mayor frecuencia en las 100 simulaciones en cada arquitectura en el Experimento 1. Cada celda contiene dicho valor y entre paréntesis el porcentaje de simulaciones en el que resultó ser el mejor, de acuerdo a la optimización realizada por Optuna mediante validación cruzada.

Se puede observar a simple vista que para todas las arquitecturas, la tasa de aprendi-

zaje de 0.001 resultó ser ampliamente la mejor y que en la mayoría de los casos se tomó un dropout de 0.3, que era el menor en el espacio de búsqueda establecido. Otro detalle es que el tamaño de lote parece no haber sido tan relevante, ya que en cada modelo, el valor más seleccionado lo fue en menos de la mitad de las simulaciones. Algo que resulta interesante es que en la LSTM “sola”, el número de neuronas por capa más elegido fue 128, el máximo en el espacio de búsqueda, mientras que cuando se la combinó con la Convolutiva, este número se redujo a la mitad.

5.2. Experimento 2: 45 períodos observados, 10 de dependencia

En este, utilizamos la misma cantidad de períodos observados que en el escenario anterior, pero disminuimos los períodos de dependencia a la mitad, pasando de 20 a 10, y también la cantidad máxima de aumentos en ellos, pasando de 6 a 3. De esta manera, la proporción entre períodos de dependencia y períodos observados es de 0.22, y la de subidas sobre períodos de dependencia se mantiene igual que antes. Aquí también tomamos $\mu_{EF_{NiNi}} = 10$, el mismo valor que para tratados y controles.

Los resultados obtenidos se encuentran en la Tabla 5.3. Vemos que todas las métricas disminuyeron considerablemente en comparación al escenario anterior. El rendimiento del PSM fue muy bajo; y en el caso de las redes resulta particularmente alarmante el caso de la precisión, que en todas se ubicó cerca de 0.5, indicando que de los controles identificados por ellas, aproximadamente la mitad realmente lo era.

Esto confirma en primera instancia nuestra hipótesis que con la misma cantidad de observaciones pero menor cantidad de períodos de dependencia, el rendimiento de los modelos empeora.

	Puntaje F_1	Precisión	Cobertura
LSTM	0.56154 ± 0.01787	0.43601 ± 0.02619	0.79338 ± 0.03664
Convolutiva	0.60380 ± 0.01910	0.48697 ± 0.02697	0.79797 ± 0.03463
LSTM + Convolutiva	0.59771 ± 0.01924	0.47867 ± 0.02921	0.80019 ± 0.03765
PSM	0.36086 ± 0.01464	0.36133 ± 0.01465	0.36040 ± 0.01463

Tabla 5.3: Promedios y desviaciones estándar de las métricas F_1 , precisión y cobertura sobre la clase positiva (controles), evaluadas en el conjunto de test en las 100 simulaciones del Experimento 2 con las diferentes arquitecturas de redes neuronales y el PSM.

Los valores de los hiperparámetros se encuentran en la Tabla 5.3, en donde sí se ve la misma tendencia que en el caso anterior: la mejor tasa de aprendizaje fue por lejos 0.001

en todos los casos, el mejor dropout fue de 0.3 y el número de neuronas en las capas ocultas del bloque LSTM se redujo cuando fue combinado con el convolucional.

	Tamaño de lote	Neuronas en capas ocultas	Tasa de aprendizaje	Dropout
LSTM	32 (39 %)	128 (78 %)	0.001 (99 %)	0.3 (51 %)
Convolucional	32 (42 %)	-	0.001 (78 %)	0.3 (80 %)
LSTM + Convolucional	32 (56 %)	32 (41 %)	0.001 (67 %)	0.3 (84 %)

Tabla 5.4: Valores de hiperparámetros seleccionados con mayor frecuencia en las 100 simulaciones en cada arquitectura en el Experimento 2. Cada celda contiene dicho valor y entre paréntesis el porcentaje de simulaciones en el que resultó ser el mejor, de acuerdo a la optimización realizada por Optuna mediante validación cruzada.

5.3. Experimento 3: 45 períodos observados, 5 de dependencia

Aquí redujimos aún más la cantidad de períodos de dependencia, pasando de 10 a 5, pero esta vez con 0 incrementos. Esto implica que la proporción de períodos de dependencia sobre observados es de 0.11, pero la tendencia modelada en ellos es monótona decreciente, como se observa en la Figura 5.2. Nuevamente, tomamos $\mu_{EF_{NiNi}} = 10$.

La Tabla 5.5 muestra los valores obtenidos de las diferentes métricas. Para nuestra sorpresa, en el caso de las redes, fueron bastante similares — y hasta en algunos casos mejores — que los obtenidos en el Experimento 1, en donde teníamos la misma cantidad de períodos observados, de los cuales casi la mitad mostraban un compartamiento decreciente con ruido. Esto parece indicar que quitar los incrementos en los períodos de dependencia, lo cual disminuye la complejidad del patrón a hallar, favorece en gran medida a las redes neuronales. Sin embargo, el PSM presentó un rendimiento menor al visto en dicho escenario, contrario a nuestra hipótesis.

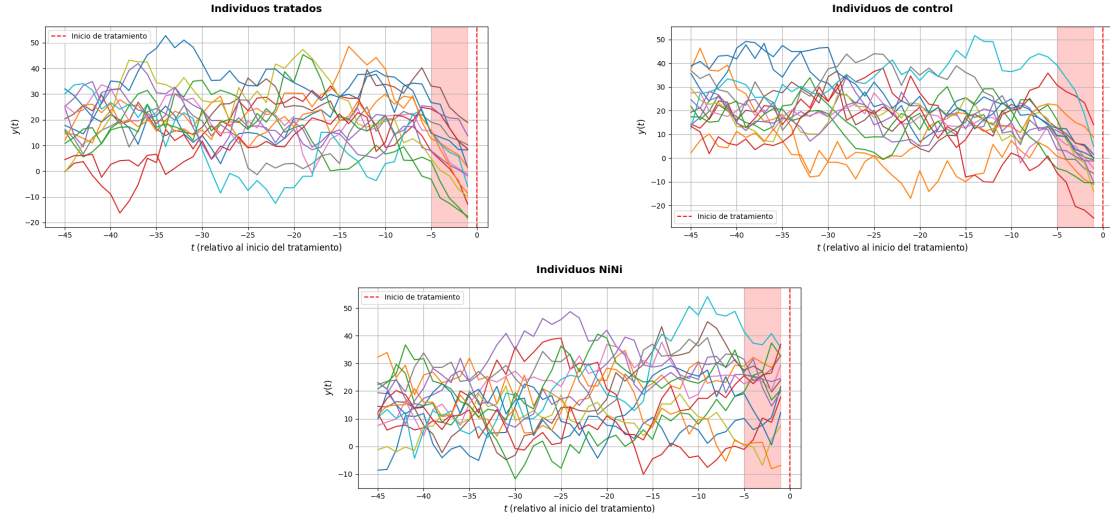


Figura 5.2: Ejemplos de series de tiempo generadas para cada grupo en el Experimento 3. Los períodos con un fondo rojo indican los períodos de dependencia para tratados y controles, en donde se ve el comportamiento monótono decreciente. La línea punteada roja indica el inicio de tratamiento para cada individuo.

	Puntaje F_1	Precisión	Cobertura
LSTM	0.82570 ± 0.01856	0.72740 ± 0.03211	0.95680 ± 0.02282
Convolutacional	0.84641 ± 0.01741	0.75091 ± 0.03095	0.97133 ± 0.01503
LSTM + Convolutacional	0.84759 ± 0.01555	0.75319 ± 0.02809	0.97040 ± 0.01510
PSM	0.53324 ± 0.01634	0.53398 ± 0.01638	0.53251 ± 0.01633

Tabla 5.5: Promedios y desviaciones estándar de las métricas F_1 , precisión y cobertura sobre la clase positiva (controles), evaluadas en el conjunto de test en las 100 simulaciones del Experimento 3 con las diferentes arquitecturas de redes neuronales y el PSM.

En la Tabla 5.6 se reflejan los resultados correspondientes a los hiperparámetros, en donde nuevamente se observa lo mismo que antes.

5.4. Experimento 4: 45 períodos observados, 0 de dependencia

En este experimento, seguimos tomando 45 períodos de observación pero con ninguno de dependencia, por lo que consideramos que es el más extremo y en el que el desempeño

	Tamaño de lote	Neuronas en capas ocultas	Tasa de aprendizaje	Dropout
LSTM	64 (35 %)	128 (57 %)	0.001 (100 %)	0.3 (64 %)
Convolutacional	32 (42 %)	-	0.001 (89 %)	0.3 (76 %)
LSTM + Convolutacional	32 (50 %)	32 (35 %), 64 (35 %)	0.001 (87 %)	0.3 (71 %)

Tabla 5.6: Valores de hiperparámetros seleccionados con mayor frecuencia en las 100 simulaciones en cada arquitectura en el Experimento 3. Cada celda contiene dicho valor y entre paréntesis el porcentaje de simulaciones en el que resultó ser el mejor, de acuerdo a la optimización realizada por Optuna mediante validación cruzada.

de las redes y el PSM debería ser cercano.

Aquí, lo que diferencia a controles y tratados de los NiNi ya no es la presencia de una determinada tendencia o no, sino que es el valor de la media de los efectos fijos, que para los primeros es de 10 mientras que para los NiNi es de 7. Esto provoca que, al graficar la variable para individuos de los diferentes grupos, no se observe ninguna diferencia a simple vista, como se puede observar en la Figura 5.3. Sin embargo, al graficar el promedio por grupo de la variable en cada uno de los períodos de tiempo, los promedios de tratados y controles se ubican siempre por encima de los de los NiNi, como muestra la Figura 5.4.

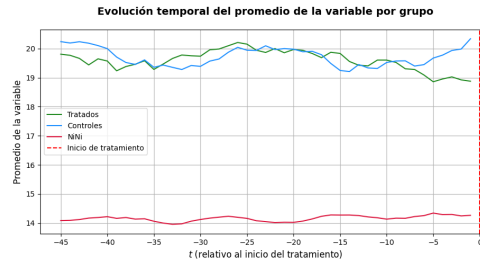


Figura 5.4: Gráfico de la evolución temporal del promedio de la variable para los distintos grupos. El promedio se calcula por período con todos los individuos de cada grupo. Aquí se puede ver claramente que al disminuir el valor de la media de los efectos fijos en el grupo de los NiNi, los promedios por período están todos por debajo de los de los tratados y controles.

Los resultados de este experimento se encuentran en la Tabla 5.7, en donde se observan valores bajos de todas las métricas en todos los modelos. Algo a notar es que a comparación de los escenarios anteriores, la diferencia entre los resultados obtenidos con las redes y con el PSM se redujo bastante, como habíamos planteado en nuestras hipótesis. Más aún, si prestamos atención a la precisión, la obtenida con los modelos es muy similar a la obtenida con el PSM.

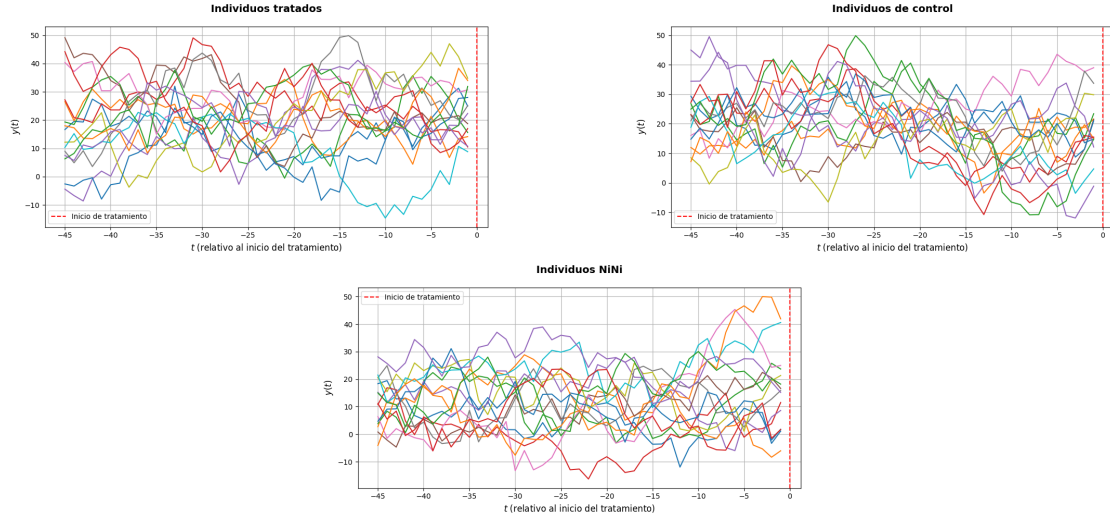


Figura 5.3: Ejemplos de series de tiempo generadas para cada grupo en el Experimento 4. La línea punteada roja indica el inicio de tratamiento para cada individuo. En este caso, no hay períodos de dependencia y no se observa ninguna tendencia ni en tratados ni en controles. A simple vista, no hay nada que diferencie a los individuos pertenecientes a estos dos grupos de los NiNi.

De esta manera, los valores obtenidos indican que quitando los períodos de dependencia, ninguno de los modelos es capaz de capturar una diferencia más “implícita” entre los diferentes individuos, incluso con la mayor cantidad de períodos de observación modelados.

	Puntaje F_1	Precisión	Cobertura
LSTM	0.28642 ± 0.02401	0.19572 ± 0.02442	0.55165 ± 0.06813
Convolutacional	0.29402 ± 0.01193	0.21026 ± 0.00954	0.49111 ± 0.03548
LSTM + Convolutacional	0.29149 ± 0.01031	0.20927 ± 0.01094	0.48416 ± 0.03778
PSM	0.20161 ± 0.01254	0.20182 ± 0.01252	0.20140 ± 0.01255

Tabla 5.7: Promedios y desviaciones estándar de las métricas F_1 , precisión y cobertura sobre la clase positiva (controles), evaluadas en el conjunto de test en las 100 simulaciones del Experimento 4 con las diferentes arquitecturas de redes neuronales y el PSM.

Los valores de los hiperparámetros seleccionados con mayor frecuencia como los óptimos se encuentran en la Tabla 5.8. La principal diferencia con respecto a los escenarios anteriores es que aquí la tasa de aprendizaje de 0.0001 resultó como la más elegida. Sin embargo, en este escenario no se obtuvieron buenos resultados, por lo que una alternativa

sería probar con un espacio de búsqueda más amplio, o incluso con otras arquitecturas de redes neuronales.

	Tamaño de lote	Neuronas en capas ocultas	Tasa de aprendizaje	Dropout
LSTM	128 (37 %)	128 (71 %)	0.001 (99 %)	0.3 (55 %)
Convolutacional	32 (43 %)	-	0.0001 (74 %)	0.3 (97 %)
LSTM + Convolutacional	32 (42 %)	32 (39 %)	0.0001 (69 %)	0.3 (99 %)

Tabla 5.8: Valores de hiperparámetros seleccionados con mayor frecuencia en las 100 simulaciones en cada arquitectura en el Experimento 4. Cada celda contiene dicho valor y entre paréntesis el porcentaje de simulaciones en el que resultó ser el mejor, de acuerdo a la optimización realizada por Optuna mediante validación cruzada.

5.5. Experimento 5: 35 períodos observados, 15 de dependencia

En este experimento, tomamos 35 períodos observados, de los cuales 15 son de dependencia con un máximo de 4 incrementos. Así, el cociente entre períodos de dependencia sobre observados es de 0.43, y el de subidas sobre períodos de dependencia de 0.27, ambos muy parecidos a los del Experimento 1. Tomamos para los NiNi el mismo valor de media de efectos fijos de tratados y controles ($\mu_{EF_{NiNi}} = 10$).

En otras palabras, lo que ocurre es que se reduce la cantidad total de datos temporales pero se mantienen las proporciones previas. La idea de este escenario — y los siguientes — es evaluar el desempeño de los distintos modelos en casos en donde la información disponible sobre los períodos previos al tratamiento es más limitada.

Los resultados obtenidos, resumidos en la Tabla 5.9, muestran solamente un leve deterioro de todos los modelos en comparación con los del Experimento 1. Consideramos que esto resulta bastante alentador, ya que sugiere que los modelos son robustos frente a esta reducción en la cantidad de datos temporales pre-tratamiento.

	Puntaje F_1	Precisión	Cobertura
LSTM	0.78815 ± 0.02252	0.69409 ± 0.03919	0.91444 ± 0.01953
Convolutacional	0.81989 ± 0.02092	0.73371 ± 0.03725	0.93110 ± 0.01520
LSTM + Convolutacional	0.81942 ± 0.01833	0.73744 ± 0.03291	0.92359 ± 0.01573
PSM	0.61062 ± 0.01661	0.61155 ± 0.01669	0.60970 ± 0.01656

Tabla 5.9: Promedios y desviaciones estándar de las métricas F_1 , precisión y cobertura sobre la clase positiva (controles), evaluadas en el conjunto de test en las 100 simulaciones del Experimento 5 con las diferentes arquitecturas de redes neuronales y el PSM.

Los valores de los hiperparámetros, que se reflejan en la Tabla 5.10, siguen mostrando los mismos patrones que antes. La tasa de aprendizaje más elegida fue 0.001, el dropout 0.3, los valores más elegidos de tamaños de lote lo fueron en menos de la mitad de las simulaciones, y las neuronas ocultas del bloque LSTM se reduce cuando se lo combina con el procesamiento convolutacional.

	Tamaño de lote	Neuronas en capas ocultas	Tasa de aprendizaje	Dropout
LSTM	64 (39 %)	128 (62 %)	0.001 (99 %)	0.3 (51 %)
Convolutacional	64 (36 %)	-	0.001 (98 %)	0.3 (77 %)
LSTM + Convolutacional	32 (41 %)	32 (41 %)	0.001 (95 %)	0.3 (78 %)

Tabla 5.10: Valores de hiperparámetros seleccionados con mayor frecuencia en las 100 simulaciones en cada arquitectura en el Experimento 5. Cada celda contiene dicho valor y entre paréntesis el porcentaje de simulaciones en el que resultó ser el mejor, de acuerdo a la optimización realizada por Optuna mediante validación cruzada.

5.6. Experimento 6: 25 períodos observados, 10 de dependencia

En este caso, redujimos aún más la cantidad de información disponible. Pasamos de tener 45 períodos en el escenario más beneficioso a tener 25, casi un 50 % menos de datos. Los períodos de dependencia son 10 con una cantidad máxima de incrementos de 3. Así, las proporciones de períodos de dependencia y observados es de 0.4 y de aumentos sobre períodos de dependencia de 0.33, similares a los del Experimento 1. Tomamos $\mu_{EF_{NiNi}} = 10$, igual que en tratados y controles.

Como esperábamos, los valores obtenidos de las diferentes métricas son peores que los de los experimentos 1 y 5, en donde había más datos temporales; y similares a los del Experimento 2, con la misma cantidad de períodos de dependencia pero 45 de observación. Sin embargo, las redes neuronales siguen superando ampliamente al PSM.

	Puntaje F_1	Precisión	Cobertura
LSTM	0.57106 ± 0.01913	0.44636 ± 0.02609	0.79608 ± 0.02930
Convolutacional	0.62629 ± 0.01736	0.50731 ± 0.02557	0.82156 ± 0.03511
LSTM + Convolutacional	0.61837 ± 0.01957	0.50342 ± 0.03310	0.80676 ± 0.03637
PSM	0.36480 ± 0.01572	0.36531 ± 0.01576	0.36431 ± 0.01569

Tabla 5.11: Promedios y desviaciones estándar de las métricas F_1 , precisión y cobertura sobre la clase positiva (controles), evaluadas en el conjunto de test en las 100 simulaciones del Experimento 6 con las diferentes arquitecturas de redes neuronales y el PSM.

En la Tabla 5.12, que muestra los valores de los hiperparámetros, la principal diferencia con respecto a los casos anteriores es que en el caso de la arquitectura LSTM + Convolutacional, el número de neuronas en la capas ocultas del bloque LSTM fue 128, el máximo en el espacio de búsqueda.

	Tamaño de lote	Neuronas en capas ocultas	Tasa de aprendizaje	Dropout
LSTM	128 (38 %)	128 (77 %)	0.001 (98 %)	0.5 (43 %)
Convolutacional	64 (35 %)	-	0.001 (93 %)	0.3 (78 %)
LSTM + Convolutacional	64 (39 %)	128 (38 %)	0.001 (95 %)	0.3 (86 %)

Tabla 5.12: Valores de hiperparámetros seleccionados con mayor frecuencia en las 100 simulaciones en cada arquitectura en el Experimento 6. Cada celda contiene dicho valor y entre paréntesis el porcentaje de simulaciones en el que resultó ser el mejor, de acuerdo a la optimización realizada por Optuna mediante validación cruzada.

5.7. Experimento 7: 15 períodos observados, 8 de dependencia

En este último experimento, tomamos 15 períodos de observación, de los cuales 8 son de dependencia con una cantidad máxima de 2 incrementos. De esta manera, la proporción de períodos de dependencia y observados es de 0.53, un valor mayor al de los

experimentos anteriores, y de aumentos sobre períodos de dependencia de 0.25, menor al de los otros escenarios. De nuevo, el valor de $\mu_{EF_{NiNi}}$ es el mismo que para tratados y controles (10).

Los resultados, que se reflejan en la Tabla 5.13, muestran una leve mejora con respecto a los del Experimento 6. Esto podría explicarse, al menos en parte, por el hecho de que si bien hay una menor cantidad de datos, poco más de la mitad de los períodos observados son de dependencia. Sin embargo, la baja precisión sigue siendo un aspecto preocupante.

	Puntaje F_1	Precisión	Cobertura
LSTM	0.61563 ± 0.01930	0.49560 ± 0.02659	0.81554 ± 0.03051
Convolutacional	0.67466 ± 0.02309	0.54877 ± 0.03250	0.87960 ± 0.03372
LSTM + Convolutacional	0.66751 ± 0.02285	0.54698 ± 0.03536	0.86140 ± 0.03752
PSM	0.37886 ± 0.01287	0.37936 ± 0.01294	0.37837 ± 0.01282

Tabla 5.13: Promedios y desviaciones estándar de las métricas F_1 , precisión y cobertura sobre la clase positiva (controles), evaluadas en el conjunto de test en las 100 simulaciones del Experimento 7 con las diferentes arquitecturas de redes neuronales y el PSM.

Los hiperparámetros más seleccionados como los óptimos se encuentran en la Tabla 5.14, donde se ve que los valores más elegidos resultaron ser los mismos para todas las arquitecturas. Además, con el número de neuronas de las capas ocultas de la arquitectura LSTM + Convolutacional ocurrió lo mismo que en el experimento anterior, el valor más seleccionado fue el más alto en el espacio de búsqueda.

	Tamaño de lote	Neuronas en capas ocultas	Tasa de aprendizaje	Dropout
LSTM	64 (39 %)	128 (80 %)	0.001 (99 %)	0.3 (49 %)
Convolutacional	64 (39 %)	-	0.001 (99 %)	0.3 (79 %)
LSTM + Convolutacional	64 (40 %)	128 (38 %)	0.001 (98 %)	0.3 (81 %)

Tabla 5.14: Valores de hiperparámetros seleccionados con mayor frecuencia en las 100 simulaciones en cada arquitectura en el Experimento 7. Cada celda contiene dicho valor y entre paréntesis el porcentaje de simulaciones en el que resultó ser el mejor, de acuerdo a la optimización realizada por Optuna mediante validación cruzada.

Capítulo 6

Conclusiones y trabajos futuros

En este trabajo se exploró la utilización de redes neuronales para la selección de grupos de control en evaluaciones de impacto sobre programas cuya asignación de beneficiarios es no aleatoria, depende de resultados observados de la variable objetivo en períodos previos al inicio del tratamiento, y en el que hay múltiples cohortes. Se evaluaron diferentes arquitecturas de redes neuronales y se comparó su desempeño con el del PSM, un método ampliamente utilizado para esta tarea. A continuación, desarrollamos las principales conclusiones obtenidas a partir de los resultados, proponemos una forma de aplicar los métodos explorados en casos reales, y mencionamos algunos aspectos de nuestro enfoque que podrían investigarse en trabajos futuros.

6.1. Conclusiones generales

En primer lugar, el rendimiento de las redes neuronales en las métricas evaluadas superó consistentemente al del PSM en todos los escenarios considerados. Este resultado fue respaldado por pruebas de hipótesis que permitieron concluir que las diferencias observadas fueron estadísticamente significativas, lo que refuerza la validez del enfoque propuesto.

Con respecto a la comparación entre las diferentes arquitecturas, contrario a nuestras expectativas iniciales, con la que mejores resultados obtuvimos en la gran mayoría de los escenarios fue la Convolutiva. Para nuestra sorpresa, la combinación de LSTM + Convolutiva no trajo mejoras sobre la Convolutiva, aunque sí sobre la LSTM. Esto sugiere que, en general, las operaciones aplicadas por las redes convolucionales permitieron una mejor captura de los patrones propuestos en los diferentes escenarios, y que la incorporación de un bloque LSTM no tuvo un efecto significativo (al menos con la arquitectura específica y el espacio de búsqueda de hiperparámetros propuestos).

Sin embargo, es importante destacar una restricción de la arquitectura Convolutiva: solamente puede procesar entradas de tamaño fijo, es decir secuencias de la misma

longitud. En cambio, la LSTM, como se explicó en la Sección 2.5, es capaz de manejar secuencias de longitud variable. Esta característica la convierte en una opción más flexible para su uso en casos reales donde no se cuenta con la información completa de todos los períodos de tiempo considerados para todos los individuos.

Como observamos en las diferentes tablas de hiperparámetros, la tendencia general fue que la tasa de aprendizaje más seleccionada fue de 0.001, el dropout de 0.3, y que los valores de tamaños de lote elegidos con mayor frecuencia lo fueron en menos de la mitad de las simulaciones. Algo a notar es que el valor de dropout 0.3 era el menor dentro del espacio de búsqueda establecido, lo cual puede indicar que con valores aún menores podrían mejorar el desempeño.

En relación al número de neuronas en las capas LSTM, el valor más frecuente en la arquitectura LSTM fue de 128 en la mayoría de los casos, pero en la LSTM + Convolutional, este valor tendió a reducirse. Este hecho refuerza la hipótesis que el procesamiento realizado por las convoluciones resulta más efectivo que el llevado a cabo por las neuronas LSTM. Sin embargo, como en el caso de la LSTM el valor más seleccionado (128) era el más alto en el espacio de búsqueda, se podría considerar el uso de valores más altos.

Teniendo en cuenta las características de los diferentes escenarios, podemos concluir que las redes demostraron un rendimiento prometedor en aquellos donde había una alta cantidad de períodos observados y las tendencias eran largas y ruidosas (experimentos 1 y 5), o cuando eran cortas y con un comportamiento monótono (Experimento 3). En cambio, en el escenario donde no había un patrón claro en los comportamientos pasados, sino que era una característica menos observable la que distinguía a tratados y controles de NiNi (Experimento 4), ninguno de los modelos mostró resultados aceptables.

Por otro lado, los valores obtenidos en los experimentos 6 y 7, donde la cantidad de períodos observados era baja (25 y 15 respectivamente), sugieren que el desempeño de las redes mejora a medida que aumenta la proporción entre períodos de dependencia y períodos observados. Esta tendencia también puede observarse al comparar los experimentos 1 y 2, lo cual refuerza nuestra hipótesis.

6.2. Aplicación en casos reales

En nuestros experimentos, conocemos de antemano cuáles son las unidades que deberían ser identificadas como controles y cuáles como NiNi. Sin embargo, en una situación real, a la hora de llevar a cabo una evaluación de impacto, solamente se conoce quiénes han sido tratados y quiénes no; y dentro de este último grupo, están mezclados NiNi e individuos que potencialmente podrían servir como controles. Esto nos llevó a preguntarnos cómo podría aplicarse la metodología presentada en este trabajo en un entorno real. A continuación, detallamos nuestra propuesta.

Para empezar, deben cumplirse los supuestos considerados en este trabajo sobre los programas utilizados evaluados. Es decir, la asignación a la intervención es no aleatoria,

depende de resultados observados en la variable objetivo en períodos anteriores al inicio del programa, y existe una o más cohortes.

Una vez verificado el cumplimiento de estos supuestos, la manera que proponemos de aplicar nuestra metodología para seleccionar individuos de control es la siguiente:

1. Separar los individuos tratados y no tratados, y para todos tomar los n períodos que haya disponibles antes del tratamiento¹.
2. Etiquetar a **todos** los individuos tratados con 1 y a un subconjunto de los no tratados con 0, repitiendo a estos últimos tantas veces como cohortes haya con los inicios de programa correspondientes. Luego, entrenar el modelo con todos estos².
3. Una vez que el modelo haya sido entrenado, realizar inferencia sobre todos los individuos no tratados, incluidos aquellos usados en el entrenamiento. Para llevar a cabo esta inferencia, habría que repetir todos estas unidades con los diferentes inicios de programa.
4. Los individuos para los cuales el modelo haya inferido un 1 serán los controles y para los que haya inferido un 0 serán los NiNi. Puede ocurrir que el mismo individuo sirva como control para diferentes cohortes — esto sucedería si un mismo individuo es identificado como 1 pero con diferentes inicios de programa —.

Un detalle importante es que en el paso **2**, lo que muy probablemente ocurra es que algunos de los no tratados que colocamos en el conjunto de entrenamiento — a los cuales etiquetamos con 0 — puedan servir como controles. Es por esto que en el paso **3** habría que incluirlos en el conjunto sobre el cual se realiza la inferencia.

Una alternativa sería, en lugar de entrenar la red con todos los tratados, dividir este grupo en dos, y usar una parte para entrenar la red, y la otra como *proxy* para verificar la correcta identificación de 1s antes de hacer la inferencia final.

Además, una vez realizada la inferencia, se pueden llevar a cabo chequeos manuales para verificar que los controles identificados sean similares a los tratados. Por ejemplo, se podrían hacer pruebas de hipótesis de diferencia de medias entre ambos grupos en cada uno de los n períodos que se hayan tomado. Si las diferencias resultan significativas en la mayoría de los períodos, entonces el grupo de controles seleccionado probablemente no sea adecuado.

¹Como mencionamos anteriormente, según los resultados observados, mientras mayor sea este n , mayores son las probabilidades de identificar buenos controles.

²Lo ideal aquí sería entrenar el modelo tomando los valores de hiperparámetros que resultaron seleccionados como los óptimos con mayor frecuencia, como mostramos recién, según la arquitectura que se esté usando.

6.3. Trabajos futuros

Una limitación de nuestro trabajo es que, al trabajar con datos sintéticos, sabemos de antemano quiénes deberían ser identificados como controles y quiénes como NiNi. Sin embargo, como mencionamos anteriormente, lo que ocurre en la realidad es que solamente se tiene conocimiento sobre quiénes fueron tratados y quiénes no. Por lo tanto, siguiendo la propuesta de aplicación, al entrenar el modelo utilizando algunos no tratados etiquetados como 0, es muy probable que se excluyan posibles controles válidos. En esta línea, se podrían explorar alternativas para evitar el uso de no tratados con etiqueta 0 durante el entrenamiento, y en su lugar generar artificialmente individuos que sean claramente distintos de los tratados.

Otra mejora consiste en el uso de arquitecturas de redes neuronales más recientes como los Transformers, que han demostrado tener un desempeño notable en tareas para las que anteriormente se utilizaban redes recurrentes, como el procesamiento de lenguaje natural.

Asimismo, sería interesante investigar la utilización de técnicas de aprendizaje no supervisado, como el clustering, para evaluar si en los escenarios propuestos aquí, resulta posible separar al grupo de los controles de los NiNi sin etiquetas previas.

Un aporte valioso sería comparar la metodología propuesta con otras técnicas de identificación de grupos de control distintas del PSM, como los controles sintéticos [1], las diferencias en diferencias para múltiples períodos de tiempo [6], y la regresión de dos vías de Mundlak [43].

Por último, incorporar un mayor número de variables sobre cada individuo — tanto temporales como estáticas — permitiría simular escenarios más complejos y evaluar de forma más robusta el desempeño de las arquitecturas propuestas. Además, consideramos que incluir variables exógenas que también varían con el tiempo y conforman el contexto en el que se encuentran los individuos, como la tasa de interés, el índice de inflación, el tipo de cambio de monedas extranjeras, el valor de los bonos, entre otras, podría contribuir a que los modelos hagan una mejor identificación de controles.

Bibliografía

- [1] Alberto Abadie y Jinglong Zhao. *Synthetic Controls for Experimental Design*. 2025. DOI: <http://dx.doi.org/10.48550/arXiv.2108.02196>. URL: <https://arxiv.org/abs/2108.02196>.
- [2] Imtiyaz Ali et al. «Health insurance support on maternal health care: evidence from survey data in India». En: *Journal of Public Health* 45.2 (mar. de 2022), págs. 368-378. DOI: 10.1093/pubmed/fdac025.
- [3] Joshua D. Angrist y Jörn-Steffen Pischke. *Mostly Harmless Econometrics: An Empiricist's Companion*. Princeton University Press, 2009. ISBN: 9780691120348. DOI: <https://doi.org/10.2307/j.ctvc4j72>. URL: <http://www.jstor.org/stable/j.ctvc4j72>.
- [4] Y. Bengio, P. Simard y P. Frasconi. «Learning long-term dependencies with gradient descent is difficult». En: *IEEE Transactions on Neural Networks* 5.2 (1994), págs. 157-166. DOI: 10.1109/72.279181.
- [5] 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).
- [6] Brantly Callaway y Pedro H.C. Sant'Anna. «Difference-in-Differences with multiple time periods». En: *Journal of Econometrics* 225.2 (2021), págs. 200-230. DOI: <https://doi.org/10.1016/j.jeconom.2020.12.001>. URL: <https://www.sciencedirect.com/science/article/pii/S0304407620303948>.
- [7] Centro de Computación de Alto Desempeño - Universidad Nacional de Córdoba. *Computadora Nabucodonosor*. URL: <https://supercomputo.unc.edu.ar/equipamiento/computadora-nabucodonosor/>.
- [8] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow*. 1st. O'Reilly Media, 2017. ISBN: 9781491962299.
- [9] 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>.

- [10] David Augusto Giuliadori. *Estadística Aplicada - Principales Conceptos Teóricos y Problemas Resueltos - Aplicaciones en Python*. 2022.
- [11] Ian Goodfellow, Yoshua Bengio y Aaron Courville. *Deep Learning*. <http://www.deeplearningbook.org>. MIT Press, 2016.
- [12] Carolyn Heinrich, Alessandro Maffioli y Gonzalo Vázquez. *A Primer for Applying Propensity-Score Matching*. 2011. DOI: 10.18235/0008567. URL: <https://doi.org/10.18235/0008567>.
- [13] Sepp Hochreiter y Jürgen Schmidhuber. «Long Short-Term Memory». En: *Neural Computation* 9 (nov. de 1997), págs. 1735-1780. DOI: 10.1162/neco.1997.9.8.1735.
- [14] Sergey Ioffe y Christian Szegedy. «Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift». En: *CoRR* abs/1502.03167 (2015). URL: <http://arxiv.org/abs/1502.03167>.
- [15] *Jupyter*. URL: <https://docs.jupyter.org/en/latest/>.
- [16] 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.
- [17] Diederik P. Kingma y Jimmy Ba. «Adam: A Method for Stochastic Optimization». En: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. 2015. URL: <http://arxiv.org/abs/1412.6980>.
- [18] Sebastian Martinez, Sophie Naudeau y Vitor Azevedo Pereira Pontual. «The promise of preschool in Africa : a randomized impact evaluation of early childhood development in rural Mozambique (English)». En: (2012). URL: <http://documents.worldbank.org/curated/en/819111468191961257>.
- [19] Warren McCulloch y Walter Pitts. «A logical calculus of the ideas immanent in nervous activity». En: *The bulletin of mathematical biophysics* (1943). URL: <https://doi.org/10.1007/BF02478259>.
- [20] Tom M. Mitchell. *Machine Learning*. McGraw-hill, 1997. ISBN: 0070428077.
- [21] *MLFlow*. URL: <https://mlflow.org/docs/latest/index.html>.
- [22] Izaak Neutelings. *Neural Networks with TikZ*. URL: https://tikz.net/neural_networks/.
- [23] Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015. URL: <http://neuralnetworksanddeeplearning.com/>.
- [24] *NumPy*. URL: <https://numpy.org/doc/stable/>.

- [25] Christopher Olah. *Understanding LSTM Networks*. 2015. URL: <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>.
- [26] *Optuna*. URL: <https://optuna.readthedocs.io/en/stable/index.html>.
- [27] *pandas*. URL: <https://pandas.pydata.org/docs/>.
- [28] Simon J.D. Prince. *Understanding Deep Learning*. The MIT Press, 2024. URL: <http://udlbook.com>.
- [29] *Programa Jornada Extendida y su impacto en el clima escolar*. Informe de evaluación de impacto. Publicado en la Biblioteca Digital de la UEICEE. Unidad de Evaluación Integral de la Calidad y Equidad Educativa (UEICEE), Ministerio de Educación, Gobierno de la Ciudad de Buenos Aires, 2023. URL: <https://bde-ueicee.bue.edu.ar/documentos/708-programa-jornada-extendida-y-su-impacto-en-el-clima-escolar>.
- [30] *Python*. URL: <https://docs.python.org/3/>.
- [31] *PyTorch*. URL: <https://pytorch.org/docs/stable/index.html>.
- [32] PAUL R. ROSENBAUM y DONALD B. RUBIN. «The central role of the propensity score in observational studies for causal effects». En: *Biometrika* 70.1 (1983), págs. 41-55. ISSN: 0006-3444. DOI: 10.1093/biomet/70.1.41. URL: <https://doi.org/10.1093/biomet/70.1.41>.
- [33] 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.
- [34] Donald Rubin. «ESTIMATING CAUSAL EFFECTS OF TREATMENTS IN EXPERIMENTAL AND OBSERVATIONAL STUDIES». En: *Journal of Educational 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>.
- [35] 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>.
- [36] Stuart J. Russell y Peter Norvig. *Artificial Intelligence: A Modern Approach*. 4, Global Edition. Pearson Education, 2020. ISBN: 9781292401133.
- [37] *Scikit-learn*. URL: https://scikit-learn.org/stable/user_guide.html.
- [38] Denis Smirnov y Engelbert Mephu Nguifo. «Time Series Classification with Recurrent Neural Networks». En: sep. de 2018.
- [39] *Stata*. URL: <https://www.stata.com/features/documentation/>.
- [40] *UNC Supercómputo (CCAD)*. URL: <https://supercomputo.unc.edu.ar>.

- [41] 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>.
- [42] Howard White. «Theory-based impact evaluation: principles and practice». En: *Journal of development effectiveness* 1.3 (2009), págs. 271-284.
- [43] Jeffrey Wooldridge. «Two-Way Fixed Effects, the Two-Way Mundlak Regression, and Difference-in-Differences Estimators». En: *SSRN Electronic Journal* (ene. de 2021). DOI: <https://doi.org/10.2139/ssrn.3906345>.
- [44] Jeffrey M. Wooldridge. *Introducción a la econometría: un enfoque moderno*. Ed. por Cengage Learning. Cengage Learning, 2010.

Anexo

Resultados de pruebas de hipótesis

Como dijimos, utilizamos pruebas de hipótesis³ de diferencia de medias para verificar si las diferencias observadas en el puntaje F_1 entre cada arquitectura y el PSM son estadísticamente significativas.

Contamos con un modelo A y un modelo B, evaluados independientemente sobre el conjunto de test de cada una de 100 simulaciones. Ante este escenario, aplicamos la prueba t de Student para muestras independientes con varianza poblacional desconocida. El estadístico de prueba sigue una distribución t de Student con $n_A + n_B - 2$ grados de libertad, donde n_A y n_B son los tamaños muestrales de cada modelo (ambos iguales a 100 en este caso).

Concretamente, si denotamos con $\mu_{F_1}^A$ y $\mu_{F_1}^B$ a la media poblacional del puntaje F_1 obtenido con los modelos A y B respectivamente, entonces las hipótesis nula (H_0) y alternativa (H_1) son:

- $H_0: \mu_{F_1}^A = \mu_{F_1}^B$
- $H_1: \mu_{F_1}^A \neq \mu_{F_1}^B$

En nuestro caso, tomamos un nivel de significancia $\alpha = 0.05$. De esta forma, si el p -valor obtenido en la prueba es menor a 0.05 o si el valor absoluto del estadístico observado t_{obs} es mayor a⁴ 1.972, se rechaza la hipótesis nula, concluyendo que la diferencia observada en el desempeño de los modelos es estadísticamente significativa.

En la Tabla 6.1, se presentan los p -valores y los estadísticos observados en los diferentes experimentos al comparar las muestras de F_1 obtenidas con cada red frente a las obtenidas con el PSM. En todos los casos, se rechaza la hipótesis nula, lo que permite concluir que las medias poblacionales son significativamente diferentes.

³Para una explicación más detallada sobre pruebas de hipótesis, se puede consultar el Capítulo 3 de [10].

⁴Este valor se obtuvo utilizando la función `scipy.stats.t.ppf`.

Experimento	Arquitectura	t_{obs}	p -valor
1	LSTM	-48.69896	3.66136×10^{-112}
	Convolutacional	-71.20134	4.91162×10^{-143}
	LSTM + Conv	-73.90965	3.95276×10^{-146}
2	LSTM	-86.86190	1.33819×10^{-159}
	Convolutacional	-100.94082	3.12119×10^{-172}
	LSTM + Conv	-97.97543	1.01923×10^{-169}
3	LSTM	-118.25474	1.26872×10^{-185}
	Convolutacional	-131.15133	2.06197×10^{-194}
	LSTM + Conv	-139.31947	1.49411×10^{-199}
4	LSTM	-31.31390	1.25482×10^{-78}
	Convolutacional	-53.40744	1.53692×10^{-119}
	LSTM + Conv	-55.37216	1.87972×10^{-122}
5	LSTM	-63.45210	1.53054×10^{-133}
	Convolutacional	-78.34732	5.59729×10^{-151}
	LSTM + Conv	-84.41686	3.29239×10^{-157}
6	LSTM	-83.30212	4.26036×10^{-156}
	Convolutacional	-111.66130	9.20778×10^{-181}
	LSTM + Conv	-101.00631	2.75173×10^{-172}
7	LSTM	-102.06262	3.64567×10^{-173}
	Convolutacional	-111.91466	5.91960×10^{-181}
	LSTM + Conv	-110.07937	1.48510×10^{-179}

Tabla 6.1: Estadísticos observados y p -valores en las pruebas de diferencia de medias para el puntaje F_1 , comparando cada red con el PSM en cada experimento. La prueba de hipótesis realizada fue la t de Student para muestras independientes, asumiendo varianzas poblacionales iguales. En todos los casos, el p -valor es menor al nivel de significancia $\alpha = 0.05$, por lo que se rechaza la hipótesis nula que establece que las medias poblacionales del puntaje F_1 en los dos modelos comparados en cada prueba son iguales.

