Benemérita Universidad Autónoma de Puebla

Facultad de Ciencias Físico Matemáticas

# ESTIMACIÓN DE LAS TASAS DE INFECCIÓN INTERPOBLACIONAL EN UN MODELO SIR METAPOBLACIONAL

Tesis presentada al

**Colegio de Física**

como requisito parcial para la obtención del grado de

**Licenciado en Física Aplicada**

por

Josué Rodríguez Hernández

Asesorado por

Dr. Jorge Velázquez Castro

Puebla Pue.
Junio de 2023

**Título:** ESTIMACIÓN DE LAS TASAS DE INFECCIÓN INTERPOBLACIONAL EN UN MODELO SIR METAPOBLACIONAL
**Estudiante:** Josué Rodríguez Hernández

COMITÉ

Dr. Enrique Varela Carlos
Presidente

Dra. Beatriz Bonilla Capilla
Secretario

Dr. Carlos Arturo Hernández Gracidas
Vocal

Dr. Jorge Velázquez Castro
Asesor

# Dedicatoria

A mi familia.

# Agradecimientos

Agradezco de corazón a todas aquellas personas que han contribuido tanto a mi desarrollo profesional como a la elaboración de este trabajo. Su apoyo y dedicación han sido fundamentales, y les estaré eternamente agradecido por su invaluable colaboración.

# Índice general

# Resumen

La estimación de parámetros en modelos epidémicos es fundamental para comprender y controlar la propagación de enfermedades, así como para tomar decisiones informadas en la salud pública. Estos modelos son herramientas matemáticas que permiten simular y predecir el comportamiento de una enfermedad en una población. Para lograr predicciones precisas, es necesario estimar parámetros como las tasas de infección, mortalidad y recuperación. Estos parámetros brindan un mejor entendimiento de cómo se propaga la enfermedad y cuánto tiempo puede durar una epidemia. También son utilizados para evaluar futuros escenarios y sus posibles consecuencias, preparando a la población ante futuros brotes. Es por ello que el desarrollo e implementación de técnicas para la estimación de estos parámetros se ha vuelto un aspecto importante en la ciencia, tratando de implementar métodos cada vez más precisos, confiables y aptos para cualquier tipo de brote epidémico.

Estas técnicas de estimación, están expuestas a varias dificultades debidas a problemas diversos como la naturaleza de los datos epidemiológicos, que a menudo son limitados y ruidosos. Por otro lado, la variabilidad espacial y temporal que pueden presentar las epidemias, también dificulta la precisión y confiabilidad de las estimaciones, ya que los patrones de transmisión pueden variar según la ubicación geográfica y el tiempo. Otros factores importantes son la heterogeneidad de la población, la evolución de la enfermedad y los métodos de control implementados. Por estas razones, la elección del método de estimación de parámetros dependerá de la naturaleza del modelo epidémico, la disponibilidad de datos y los objetivos de investigación.

En las siguientes páginas se presenta una nueva propuesta de técnica para la estimación de parámetros en modelos epidémicos. Específicamente, centrada en la estimación de las tasas de infección interpoblacional de un model SIR Metapoblacional mediante el uso de redes neuronales artificiales (RNA). Para ello, se realizan múltiples simulaciones matemáticas del modelo SIR Metapoblacional, con diferentes tasas de infección interpoblacional.

# Capítulo 1

# Introducción a los Modelos Epidémicos

Los modelos epidémicos, son modelos matemáticos que se utilizan para comprender y predecir la propagación de enfermedades infecciosas. Enfermedades transmisibles tales como el sarampión, la influenza y la tuberculosis han sido de presencia constante en la vida humana. Además, tanto las epidemias (brotes repentinos de una enfermedad), como las situaciones endémicas (enfermedades que siempre están presentes) son eventos de preocupación e interés para muchas personas. Un hecho importante es que enfermedades como la malaria, el tifus, el cólera, la esquistosomiasis y la enfermedad del sueño son endémicas en muchas partes del mundo, y tienen un impacto significativo en la economía y la esperanza de vida media en los países afectados.

La idea de criaturas vivientes invisibles como agentes de enfermedades, viene desde los escritos de Aristóteles hasta el desarrollo de la teoría de los gérmenes de las enfermedades en el siglo XIX por parte de científicos como Robert Koch, Joseph Lister y Louis Pasteur. En la actualidad, se conocen el mecanismo de transmisión de infecciones para la mayoría de las enfermedades y cómo es diferente para los agentes virales y los agentes bacterianos. Incluso, existen algunas enfermedades que no se transmiten directamente de humano a humano, sino por vectores, que son agentes (generalmente insectos) que son infectados por humanos y luego transmiten la enfermedad a otros humanos.

El análisis de la dinámica de transmisión de una infección de individuo a individuo en una población, parte de ideas muy similares que surgen en otras formas de transmisión, como las características genéticas y culturales que comparten una misma región. Por otro lado, la delimitación de lo que se entiende por individuo (unidad epidemiológica) también es de relevancia en dicho análisis, ya que, este puede variar dependiendo del tipo de enfermedad y de la población de interés.

Una epidemia es el brote repentino de una enfermedad que afecta a una gran parte de una población antes de desaparecer y potencialmente reaparece en intervalos de varios años.

Las enfermedades transmisibles han tenido un impacto considerable en la historia de la humanidad. Algunos ejemplos relevantes son el aumento de la población en el siglo XVIII, la caída de los imperios por las llamadas plagas Antoninas y el colapso de las civilizaciones Azteca e Inca a manos de la viruela traída por los europeos. Otro hecho importante fue la propagación de la Peste Negra en Europa y su efecto en los desarrollos políticos y económicos en la época medieval. El libro de

W.H. McNeill "Plagues and Peoples"(1976) es una fuente principal sobre la historia de los efectos y la propagación de enfermedades. En dicho texto se valida que los ejemplos dados representan el impacto repentino y dramático que las enfermedades han tenido en las poblaciones humanas a través de la mortalidad inducida por enfermedades.

Usar modelos matemáticos en epidemiología presenta un gran desafío, sin embargo, sus aportaciones han sido de gran valor, específicamente en lo que respecta a comprender la propagación de enfermedades y sugerir estrategias de control. Debido a esto, los experimentos en epidemiología con controles a menudo son difíciles o imposibles de diseñar, además los datos de epidemias pasadas pueden ser incompletos o inexactos. Por otro lado, el modelado matemático puede brindar comprensión de los mecanismos subyacentes que influyen en la propagación de la enfermedad, como el comportamiento umbral que establece que, si el número promedio de infecciones secundarias causadas por un infeccioso promedio es menor que uno, la enfermedad desaparecerá, pero si es mayor que uno habrá una epidemia. También ayuda a estimar la efectividad de las políticas de vacunación y la probabilidad de que una enfermedad pueda ser eliminada o erradicada.

En el modelado matemático de la transmisión de enfermedades, siempre hay una retribución entre modelos simples y detallados. Los modelos simples son útiles para comprender el comportamiento cualitativo general, pero los modelos detallados son necesarios para que los profesionales de la salud pública hagan recomendaciones para situaciones específicas. Las primeras aportaciones a la epidemiología matemática se deben a médicos de la salud pública como Daniel Bernouilli en 1760 y más recientemente a científicos como Sir R.A. Ross, W.H Hamer, A.G. Mckendrick y W.O. Kermack entre 1900 y 1935. Todo lo anterior lleva a la formulación de modelos compartimentales, en donde se divide a la población estudiada en compartimentos y se hacen supuestos sobre la naturaleza y la tasa de tiempo de transferencia de un compartimento a otro. Las tasas de transferencia entre compartimentos se expresan como derivadas con respecto al tiempo del tamaño del compartimento, lo que inicialmente conduce a una formulación con ecuaciones diferenciales. Sin embargo, existen modelos donde se emplean tipos más generales de ecuaciones funcionales, debido al tipo de dependencia en la tasa de transferencia.

## 1.1. El modelo epidémico simple de Kermack-McKendrick

El primer modelado de la epidemiología matemática cuyas predicciones fueron muy similares al comportamiento observado en innumerables epidemias, fue el modelo simple de Kermack-McKendrick en 1927. El cual es un modelo compartimental donde se divide a la población en tres clases etiquetadas por $S$, $I$ y $R$. Denotaremos con $S(t)$ al número de individuos susceptibles que aún no han sido infectados, con $I(t)$ al número de individuos infectados que pueden propagar la enfermedad y con $R(t)$ al número individuos quienes se han recuperado de la infección y ya no pueden propagar la enfermedad. Esta última clase, puede deberse a aislamiento, inmunización, recuperación o muerte.

Las epidemias generalmente suelen deberse a enfermedades que confieren inmunidad contra la reinfección, para describir este tipo de enfermedades se usa el término SIR. Por otro lado, para las enfermedades que no confieren esta inmunidad, se emplea el término SIS. Aunque existen otro tipo de modelos como el SEIR y SEIS que incluyen un período expuesto entre ser infectado y volverse infeccioso.

El punto de partida para el análisis de la propagación de enfermedades transmisibles, serán los modelos compartimentales, donde la variable independiente es el tiempo t y las tasas de transferencia

entre compartimentos se expresan matemáticamente como derivadas. Para estos modelos, se asume que el proceso epidémico es determinista y que el número de miembros en un compartimento es una función diferenciable del tiempo.

El modelo compartimental básico que describe la transmisión de una enfermedad es el modelo epidémico de Kermack-McKendrick, el cual, solo es un caso especial de un modelo más general que se introdujo en una secuencia de tres artículos publicados en 1927, 1932 y 1933 por W.O. Kermack y A.G. McKendrick. El modelo general incluye la dependencia de la edad de infección, y se puede utilizar para proporcionar un enfoque unificado de los modelos epidémicos compartimentales. El caso especial del modelo propuesto por Kermack y McKendrick es

$$\begin{aligned} S' &= -\beta SI, \\ I' &= \beta SI - \alpha I, \\ R' &= \alpha I, \end{aligned} \tag{1.1}$$

Un diagrama de flujo para este modelo se muestra en la figura 1.1.

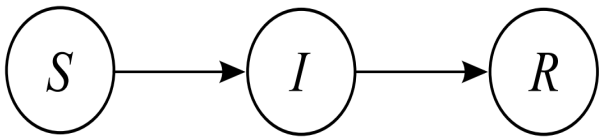



Figura 1.1: Diagrama de flujo para el modelo SIR. Figura tomada de: "Mathematical Models in Population Biology and Epidemiology", Brauer y Castillo. (2011).

Dicho modelo se basa en ciertos supuestos, tales como que un miembro promedio de la población hace contacto suficiente para transmitir la infección a otras personas a una tasa $\beta N$ por unidad de tiempo (índice de acción masiva) y que las personas infectadas abandonan la clase infectiva a una tasa $\alpha I$ por unidad de tiempo. También supone que no hay entrada o salida de personas en la población, y que no hay muertes por enfermedad, por lo tanto, el tamaño total de la población $N$ permanece constante y no se consideran los problemas demográficos.

Algunas de estas suposiciones son irrealistamente simples. Se pueden construir y analizar modelos más generales, sin embargo, modelos exageradamente simples pueden deducir comportamientos cualitativos muy similares a los que resultan de modelos más realistas. En el modelo SIR, $R$ es determinado una vez $S$ e $I$ son conocidas, por lo tanto, se puede eliminar la ecuación para $R$ del modelo y de esta manera se obtiene

$$\begin{aligned} S' &= -\beta SI, \\ I' &= (\beta S - \alpha)I, \end{aligned} \tag{1.2}$$

con las condiciones iniciales

$$S(0) = S_0, \qquad I(0) = I_o, \qquad S_0 + I_0 = N.$$

De este sistema, se puede deducir facilmente que si $S_0 < \alpha/\beta$, $I$ decrece a cero (no epidemia), mientras que si $S_0 > \alpha/\beta$, $I$ incrementará hasta alcanzar un máximo cuando $S = \alpha/\beta$ y después disminuirá hasta cero (epidemia).

A partir de esta última dedución, se encuentra una cantidad umbral, conocida como el **número de reproducción básica**, denotado por el término $\Re_0$, la cual determina si habrá una epidemia. Si $\Re_0 < 1$, la infección desaparece; por otro lado si $\Re_0 > 1$, hay una epidemia.

Otra relación importante que se obtiene a partir del análisis de (1.2), mediante un enfoque un poco distinto al tradicional, donde se hace uso de algunas propiedades de funciones decrecientes suaves no negativas es la **Relación de tamaño final**, la cual esta dada por

$$log\frac{S_0}{S_\infty} = \frac{\beta}{\alpha}[N - S_\infty] = \Re[1 - \frac{S_\infty}{N}]. \tag{1.3}$$

Donde el término $N - S_\infty$ representa el número de miembros de la población que son infectados durante el transcurso de la epidemia. A menudo, también se emplea el término **Tasa de Ataque** $(1 - \frac{S_\infty}{N})$, para describir esta cantidad. Para modelos epidémicos con estructuras compartimentales más complicadas que la del modelo SIR simple, es posible hacer una generalización de la relación de tamaño final (por ejemplo, modelos con períodos expuestos).

Una de las dificultades a las que es necesario enfrentarse al tratar de modelar una enfermedad en particular, es la estimación de parámetros. Por ejemplo, la tasa de contacto $\beta$ puede depender no solo de la enfermedad estudiada, sino también de factores sociales y de comportamiento. Sin embargo, se pueden estimar cantidades como $S_0$ y $S_\infty$ mediante estudios serológicos (mediciones de respuestas inmunitarias en muestras de sangre) antes y después de una epidemia, y a partir de estos datos se pueden estimar otras cantidades como $\Re_0$ usando (1.3). El problema resulta en que esta estimación es retrospectiva, y solo se puede derivar después de que la epidemia haya terminado.

El número máximo de contagios en cualquier instante de tiempo es una cantidad interesante y no difícil de calcular, ya que esta es el número de contagios cuando la derivada de $I$ es cero, lo cual sucede cuando $S = \alpha/\beta$, entonces tenemos que

$$I_{max} = S_0 + I_0 - \frac{\alpha}{\beta}logS_0 - \frac{\alpha}{\beta} + \frac{\alpha}{\beta}log\frac{\alpha}{\beta}, \tag{1.4}$$

### 1.1.1. La Gran Plaga en Eyam

De 1665 a 1666 el pueblo de Eyam (cerca de Sheffield, Inglaterra) sufrió un brote de peste bubónica, donde sobrevivieron solo 83 personas de una población inicial de 350. La enfermedad de Eyam se ha utilizado como caso de estudio para modelar [Raggett (1982)], ya que fueron conservados los registros detallados y, además, se convenció a la comunidad de ponerse en cuarentena. Donde se intenta ajustar el modelo simple SIR (1.2) durante el periodo comprendido entre mediados de Mayo y mediados de Octubre de 1666, midiendo el tiempo en meses y con las condiciones iniciales $S_0 = 254$ y $I_0 = 7$, y una población final de $S_\infty = 83$ (ver figura 1.2).

Con estos valores, se puede calcular la relación de tamaño final, dando como resultado $\beta/\alpha = 6.54x10^{-3}$, $\alpha/\beta = 153$ y como el período infectivo fue de 0.3667 meses, entonces $\alpha = 2.73$ y

| Date (1666) | Susceptibles | Infectives |
|---|---|---|
| July 3/4 | 235 | 14.5 |
| July 19 | 201 | 22 |
| August 3/4 | 153.5 | 29 |
| August 19 | 121 | 21 |
| September 3/4 | 108 | 8 |
| September 19 | 97 | 8 |
| October 4/5 | Unknown | Unknown |
| October 20 | 83 | 0 |

Figura 1.2: Tabla de datos de la plaga de Eyam. Figura tomada de: "Mathematical Models in Population Biology and Epidemiology", Brauer y Castillo. (2011).

$\beta = 0.0178$. Por último, se usa (1.4) para estimar $I_{max} = 30.4$. Con todo lo anterior, se grafican $S$ e $I$ como funciones de t (ver figuras 1.3 y 1.4); y para validar el modelo se grafican los puntos de los datos en la figura 1.2, junto con los valores obtenidos para $S$ e $I$ (ver figura 1.5).

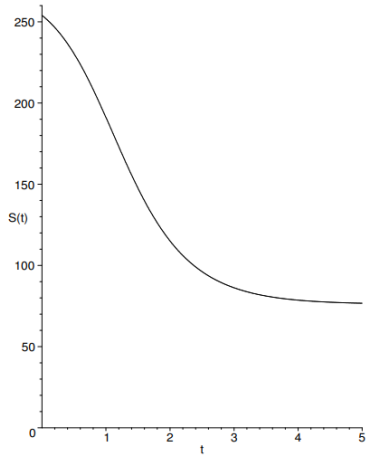



Figura 1.3: S como función de t. Figura tomada de: "Mathematical Models in Population Biology and Epidemiology", Brauer y Castillo. (2011).

Los resultados obtenidos muestran que un modelo tan simple como este, es capaz de dar una mejor estimación de la propagación de la enfermedad en Eyam, que modelos más complicados (modelos estocásticos). Sin embargo, las suposiciones que se hicieron al establecerlo lo hacen un poco real, debido a que la peste bubónica se transmite principalmente por las pulgas de las ratas y no por humanos.

En el pueblo de Eyam, la cuarenta solo aumentó la tasa de infección, debido a que mantuvo en contacto cercano a pulgas, ratas y personas. Es por esto, que las estrategias de control basadas en modelos falsos pueden ser dañinas, además es importante distinguir entre supuestos que simplifican, pero no alteran sustancialmente los efectos previstos, y supuestos erróneos que marcan una diferencia importante.

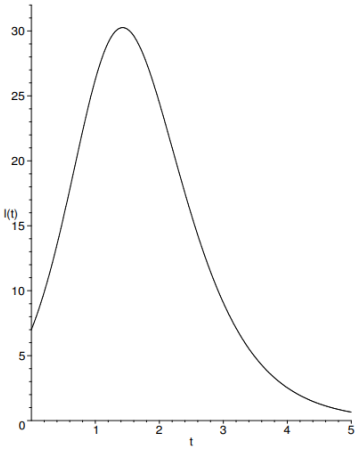



Figura 1.4: I como función de t. Figura tomada de: "Mathematical Models in Population Biology and Epidemiology", Brauer y Castillo. (2011).

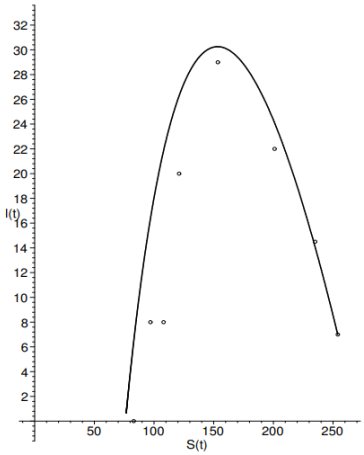



Figura 1.5: El plano S-I, modelo y datos. Figura tomada de: "Mathematical Models in Population Biology and Epidemiology", Brauer y Castillo. (2011).

Para prevenir la aparición de una epidemia en una población, se puede hacer uso del concepto de inmunidad colectiva, la cual consiste en transferir miembros de la clase susceptible a la clase recuperada, para lo cual es necesario reducir el número de reprodución básico hasta por debajo de uno. Por otro lado, la tasa de crecimiento inicial de una epidemia se puede determinar experimentalmente; desafortunadamente, debido a incompletud de los datos y al subregistro de casos, esta estimación puede no ser muy precisa, especialmente para un brote de una enfermedad previamente desconocida.

## 1.2. Modelos epidémicos más detallados

El modelo simple de Kermack-McKendrick tiene algunas deficiencias, como la descripción del comienzo de un brote de una enfermedad. Debido a esto, la formulación de modelos con estructuras compartimentales más complicadas es necesaria. Afortunadamente, las propiedades

básicas establecidas para el modelo simple de Kermack-McKendrick se mantienen también para estos otros modelos. Por lo tanto, se cumplen las siguientes afirmaciones

1. Hay un número básico de reproducción $\Re_0$ tal que si $\Re_0 < 1$, la enfermedad desaparece, mientras que si $\Re_0 > 1$, hay una epidemia.
2. El número de infectados siempre se aproxima a cero y el número de susceptibles siempre se acerca a un límite positivo cuando $t \to \infty$.
3. Existe una relación entre el número básico de reproducción y el tamaño final de la epidemia, cumpliéndose la igualdad si no hay muerte por enfermedad.

### 1.2.1. Perídos Expuestos

Un período de exposición es el tiempo que transcurre desde el momento en que un miembro susceptible es infectado hasta el momento en que desarrolla los síntomas y puede transmitir la infección. Para incorporar un período de exposición $1/\kappa$ al modelo epidémico (1.2), se agrega una clase de exposición $E$ y se toma el tamaño total de la población como $N = S + E + I + R$. El modelo toma la forma

$$\begin{aligned} S' &= -\beta SI, \\ E' &= \beta SI - \kappa E, \\ I' &= (\beta S - \alpha)I, \end{aligned} \tag{1.5}$$

En la figura 1.6, se muestra una diagrama de flujo para este modelo.

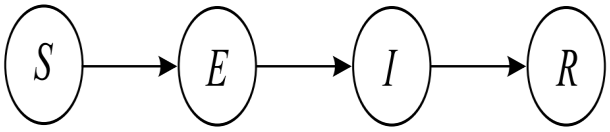



Figura 1.6: Diagrama de flujo del modelo SEIR. Figura tomada de: "Mathematical Models in Population Biology and Epidemiology", Brauer y Castillo. (2011).

En este modelo, se remplaza $I$ por $E + I$; es decir, se considera el número total de miembros infectados, sean o no capaces de transmitir la infección.

Por otro lado, existen enfermedades donde hay cierta infectividad durante el período de exposición, por lo que se requiere recalcular la tasa de nuevas infecciones, suponiendo que la infectividad se reduce en un factor $\epsilon$ durante el período de exposición.

$$\begin{aligned} S' &= -\beta S(I + \epsilon E), \\ E' &= \beta S(I + \epsilon E) - \kappa E, \\ I' &= \kappa E - \alpha I, \end{aligned} \tag{1.6}$$

Y se toman las condiciones iniciales

$$S(0) = S_0, \qquad E(0) = E_0, \qquad I(0) = I_0.$$

De un análisis similar al que se hizo con (1.2), se obtiene la relación de tamaño final para este modelo.

$$log\frac{S_0}{S_\infty} = \Re_0[1 - \frac{S_\infty}{N}] - \frac{\epsilon\beta I_0}{\kappa},$$

La cual tiene la misma forma que (1.3) cuando $I(0) = 0$.

### 1.2.2. Modelos de tratamiento

La vacunación es una forma de tratamiento que puede proteger contra la infección antes de que comience una epidemia y puede modelarse reduciendo el tamaño total de la población por la fracción de la población que está vacunada.

Para algunas enfermedades, existe un tratamiento para la infección una vez adquirida la enfermedad. El tratamiento se puede modelar asumiendo que una fracción $\gamma$ de los infectados se seleccionan para el tratamiento, lo que reduce la infectividad en una fracción $\delta$, con una tasa de eliminación de la clase tratada $\eta$. Con lo anterior, se formula el modelo SITR, con $T$ para la clase tratada.

$$\begin{aligned} S' &= -\beta S[I + \delta T], \\ I' &= \beta S[I + \delta T] - (\alpha + \gamma)I, \\ T' &= \gamma I - \eta T. \end{aligned} \tag{1.7}$$

En la figura 1.7, se muestra un diagrama de flujo para este modelo.

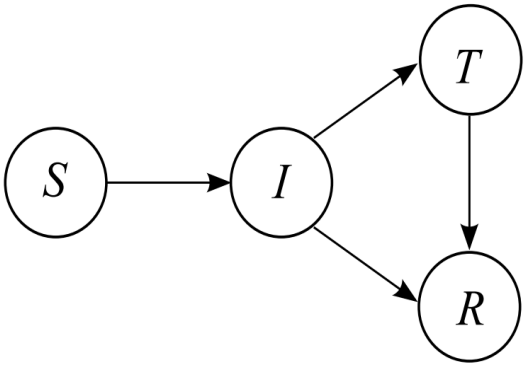



Figura 1.7: Diagrama de flujo del modelo SITR. Figura tomada de: "Mathematical Models in Population Biology and Epidemiology", Brauer y Castillo. (2011).

De una manera muy similar a la usada para el modelo simple (1.2), también es posible calcular el número de reproducción básico y la relación de tamaño final para este modelo. Solo es importante tomar en cuenta que ahora el tiempo de permanencia en el compartimiento infeccioso es $1/(\alpha+\gamma)$, que se trata una fracción $\gamma/(\alpha+\gamma)$ de infectivos, para los cuales el número de nuevas infecciones por unidad de tiempo es $\delta\beta N$ y que el tiempo medio en la clase de tratamiento es $1/\eta$. Entonces, tomando en cuenta todo lo anterior se tiene

$$\Re_0 = \frac{\beta N}{\alpha+\gamma} + \frac{\gamma}{\alpha+\gamma} + \frac{\delta\beta N}{\eta}. \tag{1.8}$$

Y la relación de tamaño final tiene la misma forma que (1.3).

### 1.2.3. Un modelo de cuarentena-aislamiento

Cuando una población se enfrenta a la propagación de una nueva enfermedad para la cual no hay vacuna ni algún tratamiento disponible, las únicas medidas de control posibles son el aislamiento de los casos diagnosticados y la cuarentena de los casos sospechosos.

Para modelar dichas epidemias, es necesario hacer varios supuestos, como que los miembros expuestos se vuelven infecciosos a una tasa $\kappa_E$ y se ponen en cuarentena a una tasa proporcional $\gamma_Q$, donde $Q$ es la clase de miembros en cuarentena. Los infecciosos se diagnostican a una tasa $\gamma_J$ y se aíslan ($J$ denota la clase aislada), pero el aislamiento es imperfecto y pueden trasmitir la enfermedad en un factor $\epsilon_J$. Los miembros infecciosos y aislados abandonan sus respectivas clases a una tasa proporcional a $\alpha_I$ Y $\alpha_J$ respectivamente. Estos supuestos conducen al modelo SEQIJR

$$\begin{aligned}
S' &= -\beta S[\epsilon_E E + \epsilon_E\epsilon_Q Q + I + \epsilon_J J],\\
E' &= \beta S[\epsilon_E E + \epsilon_E\epsilon_Q Q + I + \epsilon_J J] - (\kappa_E + \gamma_Q)E,\\
Q' &= \gamma_Q E - \kappa_j Q,\\
I' &= \kappa_E E - (\alpha_I + \gamma_J)I,\\
J' &= \kappa_Q Q + \gamma_J I - \alpha_J J.
\end{aligned} \tag{1.9}$$

Antes de que se apliquen las medidas de control, se tomarían

$$\gamma_Q = \gamma_J = \kappa_Q = \alpha_J = 0, Q = J = 0,$$

y se tendría el mismo modelo dado en (1.6).

En la figura 1.8, se muestra un diagrama de flujo para este modelo.

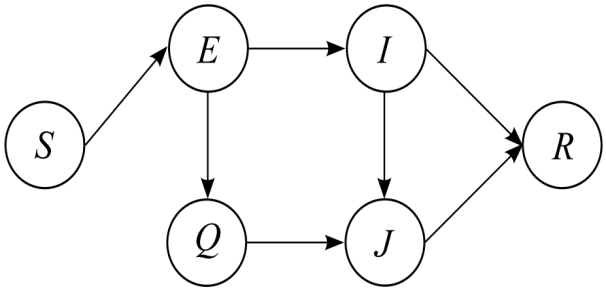



Figura 1.8: Diagrama de flujo del modelo SEQIJR. Figura tomada de: "Mathematical Models in Population Biology and Epidemiology", Brauer y Castillo. (2011).

Cuando se aplican medidas de control en una epidemia, en lugar de usar el número de reproducción básico, se define el *número de reproducción de control* $\Re_c$, el cual describe el comienzo del reconocimiento de la epidemia, en lugar del comienzo de la misma (análogamente a $\Re_0$). Ambas cantidades toman el mismo valor, cuando

$$\gamma_Q = \gamma_J = \kappa_Q = \alpha_J = 0.$$

Se puede calcular $\Re_c$ de la misma que se hizo para calcular $\Re_0$ para (1.6), tomando en cuenta el modelo completo con las clases de cuarentena y aisalamiento. Se obtiene

$$\Re_c = \frac{\epsilon_E \beta N}{D_1} + \frac{\beta N \kappa_E}{D_1 D_2} + \frac{\epsilon_Q \epsilon_E \beta N \gamma_Q}{D_1 \kappa_Q} + \frac{\epsilon_J \beta N \kappa_E \gamma_J}{\alpha_J D_1 D_2} + \frac{\epsilon_J \beta N \gamma_Q}{\alpha_J D_1},$$

donde $D_1 = \gamma_Q + \kappa_E$, $D_2 = \gamma_J + \alpha_I$.

Todos los términos de $\Re_c$ tienen una interpretación epidemiológica. Cada uno describe distintos aspectos del brote de la enfermedad, como la duración media en cada clase, el flujo de individuos y la tasa de contacto entre clases.

Los parámetros $\gamma_Q$ y $\gamma_J$ en el modelo (1.9) se consideran parámetros de control porque pueden ajustarse para manejar la epidemia. Los parámetros $\epsilon_Q$ y $\epsilon_J$ también se consideran medidas de control porque dependen de la rigurosidad de los procesos de cuarentena y aislamiento. Los demás parámetros del modelo son específicos de la enfermedad que se está estudiando y no se pueden cambiar, pero sus mediciones pueden tener un error experimental.

Con un análisis similar al que se ha hecho con los modelos anteriores y considerando la condiciones iniciales

$$S(0) + E(0) = N(0) = N, \qquad Q(0) = I(0) = J(0) = 0,$$

Se obtiene el análogo a la relación de tamaño final derivada del modelo (1.2) para el modelo de tratamiento (1.9)

$$log \frac{S_0}{S_\infty} = \Re_c [1 - \frac{S_\infty}{N}].$$

la cual tiene el mismo comportamiento asintótico que el modelo simple (1.2).

## 1.3. Modelos Epidémicos de Metapoblaciones

Todos los modelos epidémicos que se han visto hasta ahora, asumen que toda la población vive en una misma área y que está bien mezclada; sin embargo, esto no siempre es así. El flujo de individuos a través de distintas locaciones que formen parte de un misma población (heterogeneidad espacial) también afecta la transmisión de la enfermedad.

Para dar cuenta de esto, se han utilizado varios enfoques de modelado, donde se toma en cuenta el espacio y el movimiento de personas. Uno de estos enfoques es el de metapoblaciones, que incorpora al espacio como una variable discreta y típicamente son modelos matemáticos formados de un gran sistema de ecuaciones diferenciales ordinarias (ODE, por sus siglas en inglés).

El concepto de metapoblación se refiere a un grupo de poblaciones de la misma especie que viven en áreas espacialmente aisladas, pero que interactúan entre sí a través de la migración. Estas áreas aisladas se denominan parches, los cuales pueden ser ciudades, países, islas u otras regiones geográficamente autónomas; sin embargo, deben estar conectados a través de la migración. La migración se define como el movimiento físico de individuos de un área a otra, y puede ser de corto o largo plazo. La migración a corto plazo se refiere a las personas que visitan otro lugar por un período corto de tiempo y regresan a su lugar de origen; mientras que, la migración a largo plazo se refiere a las personas que se mudan a otro lugar y se establecen allí. Ambos tipos de movimiento contribuyen a la transmisión de la enfermedad de un parche a otro.

Los modelos epidémicos de metapoblación se componen de n parches y la población de cada parche se divide en las clases habituales de susceptibles, infectivos y otras más. Los tamaños de estas clases varían entre los parches y los individuos de algunas o todas las clases viajan entre estos parches, lo que lleva al movimiento de la enfermedad (ver figura 1.9).

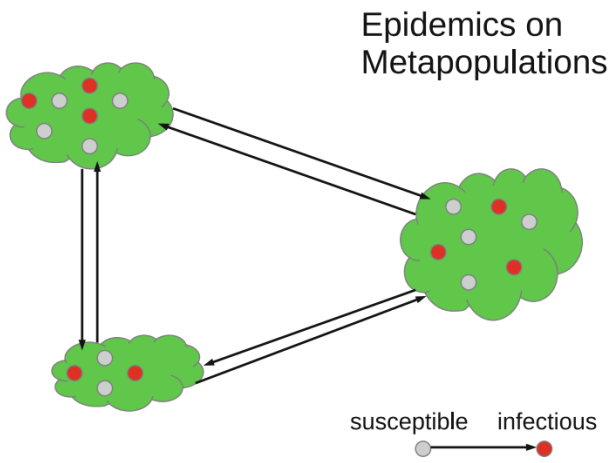



Figura 1.9: Representación esquemática de un modelo epidémico metapoblacional. Figura tomada de: An Introduction to Mathematical Epidemiology", Martcheva, M. (2015).

Históricamente, los modelos epidémicos de metapoblación incorporaron un movimiento a corto plazo entre parches, denominado movimiento lagrangiano, que se modeló utilizando el marco de metapoblación lagrangiano. Más tarde, se desarrollaron modelos eulerianos, que incorporan movimiento a largo plazo explícito y a una determinada velocidad, modelados utilizando el marco de metapoblación euleriano.

### 1.3.1. Modelos epidémicos del movimiento lagrangiano

Para empezar a modelar el movimiento lagrangiano, primero se asume que la población total ocupa $n$ regiones o parches. Sea $N_i$ la población del i-ésimo parche, con $N = N_1 + \cdots + N_n$ siendo el tamaño total de la población. La población en la i-ésima región está afectada por una enfermedad y está compuesta por individuos susceptibles ($S_i$), individuos infectados ($I_i$) e individuos recuperados/inmunes ($R_i$). El tamaño total de la población de cada región está dado por $N_i$, el cual, es la suma de las otras tres categorías. Se supone que los miembros de cada región hacen viajes cortos a otras regiones y regresan a su área de origen, lo que conduce a la propagación de la enfermedad si las personas infectadas de una región la transmiten a las personas susceptibles de otra durante estas visitas. También se supone que la cantidad de tiempo dedicado a visitar otras regiones es la misma para todos los miembros de una región, pero puede diferir según el destino. El modelo no toma en cuenta la migración a largo plazo ni los cambios en el tamaño de la población de cada región, que se supone que permanece constante debido al equilibrio de las tasas de natalidad y mortalidad ($\mu_i N_i$). Lo anterio lleva al modelo

$$\begin{aligned} S' &= \mu_i N_i - S_i \sum_{j=1}^{n} \beta_{ij} I_j - \mu_i S_i, \\ I' &= S_i \sum_{j=1}^{n} \beta_{ij} I_j - (\mu_i + \gamma_i) I_i, i = 1, \ldots, n, \end{aligned} \tag{1.10}$$

donde, una vez más, la ecuación para la clase $R$ es omitida. En este modelo, $\gamma_i$ representa la tasa de recuperación de los individuos infectados en el i-ésimo parche, y $\beta_{ij}$ representa la tasa a la que los individuos infectados del parche j transmiten la enfermedad a los individuos susceptibles en el parche i. Es importante tener en cuenta que asumir un tamaño de la población constante en cada parche significa que la tasa de natalidad es igual a la tasa de mortalidad. Esto se aprecia al obtener una ecuación diferencial para el tamaño de la población i: $N_i' = 0$ después de sumar las ecuaciones para $S_i$, $I_i$ y $R_i$.

Se puede definir el número de reproducción para cada parche, suponiendo que no hay interacción entre parches, es decir, $\beta_{ij} = 0$ para $i \neq j$. Entonces

$$\Re_i = \frac{\beta_{ii} N_i}{\mu_i + \gamma_i}.$$

El parche i se llamará un sumidero si al aislarlo de la metapoblación la enfermedad desaparece ($\Re_i < 1$). Por otro lado, si la enfermedad persiste en el parche i aun si se aísla de la metapoblación, se llamará una fuente ($\Re_i > 1$).

# Capítulo 2

# Introducción a las Redes Neuronales Artifiaciales

A lo largo de la historia, la naturaleza ha sido la fuente de inspiración humana para la creatividad, siendo la idea base de miles de inventos. Las redes neuronales artificiales (RNA), no son la excepción. Inspiradas en las redes neuronales biológicas que se encuentran en el cerebro, son el centro mismo del Deep Learning (aprendizaje profundo), y aunque cada vez difieren más de sus contrapartes biológicas, las RNA siguen siendo herramientas potentes y escalables, ideales para resolver problemas complejos de Machine Learnig (aprendizaje automático) como la clasificación de millones de imágenes, el reconocimiento de voz e incluso son utilizadas en la creación de juegos de video.

## 2.1. De las neuronas biológicas a las artificiales

El concepto de RNA fue introducido por primera vez en 1943 por McCulloch y Pitts, presentando un modelo computacional simplificado de neuronas biológicas que trabajan en conjunto para realizar cálculos complejos usando lógica proposicional. Desde entonces, hasta la década de los 60, las RNA tenían muy buena aceptación científica, incluso se creía que pronto estaríamos conversando con máquinas verdaderamente inteligentes. Desafortunadamente, los avances en esta nueva tecnología empezaron a ser cada vez menores, llevándose el interés y la financiación científica a otra parte. Las RNA se encontraban en una era oscura hasta la década de los 80, cuando nuevas arquitecturas de red y mejores técnicas de entrenamiento fueron desarrolladas. Sin embargo, en la década de 1990, las máquinas de vector de soporte (SVM) fueron preferidas por sus mejores resultados y fundamentos teóricos más sólidos.

Finalmente, ahora se está presenciando una nueva ola de interés en las RNA, la cual se cree que tendrá un impacto mucho más profundo en el mundo. Esto se debe a que hoy en día se cuenta con una gran cantidad de datos disponibles para entrenar redes neuronales, una mayor potencia informática y algoritmos de entrenamiento mejorados. Esto a llevado ha las RNA dentro de un círculo virtuoso de financiación y progreso.

## 2.2. Neuronas Biológicas

Para entender las neuronas artificiales, primero llevemos a cabo un análisis preliminar de las neuronas biológicas (figura 2.1). Una neurona biológica se compone de un cuerpo celular, muchas extensiones ramificadas llamadas dendritas y una extensión muy larga llamado axón. El axón se divide en muchas ramas llamadas telodendrias, que terminan en estructuras minúsculas llamadas terminales sinápticas (sinapsis), las cuales están conectadas a otras neuronas. Las neuronas reciben impulsos eléctricos cortos llamados señales de otras neuronas a través de estas sinapsis, y cuando una neurona recibe una cantidad suficiente de señales de otras neuronas, dispara sus propias señales.

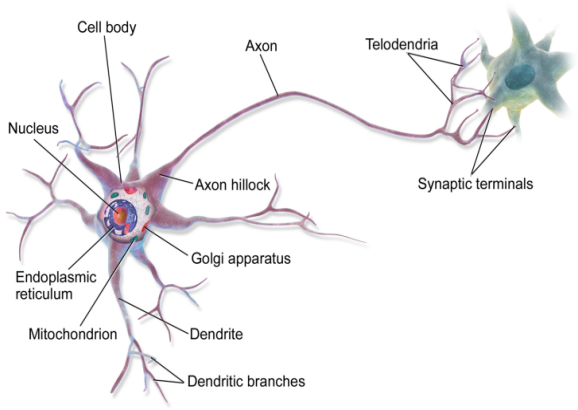



Figura 2.1: Representación esquemática de una neurona. Figura tomada de: "Hands On Machine Learning with Scikit-Learn and TensorFlow", Géron, A. (2017).

Las neuronas están organizadas en una vasta red de miles de millones de neuronas, y cada neurona normalmente está conectada a miles de otras neuronas. La arquitectura de las redes neuronales biológicas sigue siendo objeto de investigación activa, pero se han mapeado algunas partes del cerebro y parece que las neuronas a menudo se organizan en capas consecutivas (ver figura 2.2). Esta organización de neuronas en capas también se utiliza en el diseño de RNA.

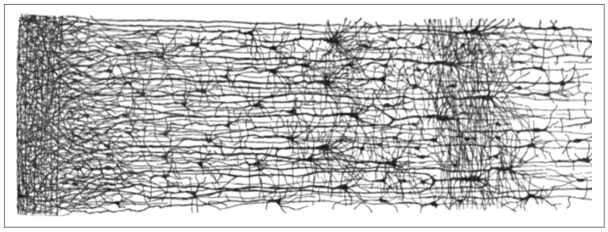


Figura 2.2: Red de neuronas biológicas multicapas. Figura tomada de: "Hands On Machine Learning with Scikit-Learn and TensorFlow", Géron, A. (2017).

## 2.3. El Perceptrón

El modelo simple de neuronas biológicas propuesto por McCulloch y Pitts (neurona artificial) consitia de una o más entradas binarias (on/off) y una salida binaria. Una neurona artificial activa su salida, cuando un cierto número de sus entradas están activas. De esta manera, es posible calcular cualquier proposición lógica a partir de una red de neuronas artificiales.

Por otro lado, en 1957 Frank Rosenblatt inventó una de las arquitecturas más simples de RNAs llamado perceptrón. En un perceptrón, las entradas y salidas son números (en lugar de valores binarios) y cada conexión de entrada está asociada con un peso basado en una cierta unidad de umbral lineal (LTU, ver figura 2.3). La LTU calcula una suma ponderada de sus entradas ($z = w_1x_1 + w_2x_2 + \cdots + w_nX_n = W^t \cdot X$), luego aplica una función de paso a esa suma y genera la salida: $h_w(x) = step(z) = step(W^t \cdot X)$.

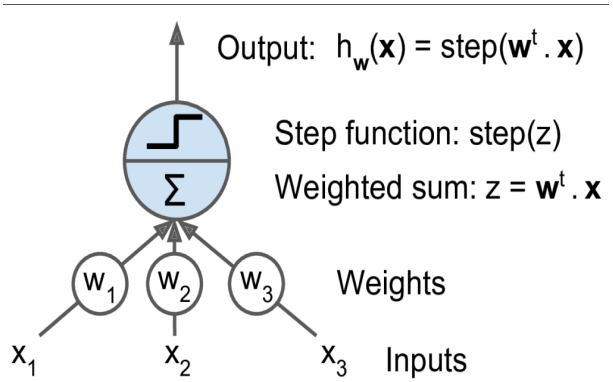



Figura 2.3: Unidad de umbral lineal. Figura tomada de: "Hands On Machine Learning with Scikit-Learn and TensorFlow", Géron, A. (2017).

A continuación, se muestran las dos más comúnes funciones de paso utilizadas.

$$\text{heaviside}(z) = \begin{cases} 0 & \text{si } z < 0 \\ 1 & \text{si } z \geq 0 \end{cases}, \quad \text{sgn}(z) = \begin{cases} -1 & \text{si } z < 0 \\ 0 & \text{si } z = 0 \\ +1 & \text{si } z > 0 \end{cases}.$$

Un perceptrón consta de una sola capa de LTU, donde cada neurona está conectada a todas las entradas. Estas conexiones a menudo se representan mediante neuronas de entrada, que simplemente pasan cualquier entrada que reciben. En la figura 2.4 se muestra un perceptrón de dos entradas y tres salidas, el cual es capaz de clasificar instancias simultánemente en tres clases binarias diferentes (clasificador de salida múltiple).

El entrenamiento de un perceptrón se realiza mediante una variante del aprendizaje hebbiano, inspirado en la idea de que las células que disparan juntas se conectan entre sí, la cual simplemente sigue la regla de que el peso de la conexión entre dos neuronas aumenta siempre que tengan la

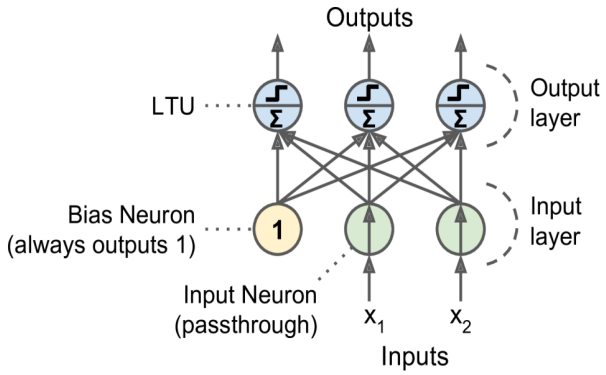



Figura 2.4: Diagrama del perceptrón. Figura tomada de: "Hands On Machine Learning with Scikit-Learn and TensorFlow", Géron, A. (2017).

misma salida. El perceptrón recibe una instancia de entrenamiento a la vez y sus predicciones se comparan con los resultados esperados. Por cada neurona de salida que produce una predicción incorrecta, se refuerzan los pesos de conexión de las entradas que contribuyen a la predicción correcta. En la ecuación 2.1 se muestra matemáticamente esta regla.

$$w_{i,j} = w_{i,j} + \eta(\hat{y_j} - y_j)x_i, \tag{2.1}$$

donde

- $w_{i,j}$ es el peso de conexión que hay entre la i-ésima neurona de entrada y la j-ésima neurona de salida.
- $x_i$ es el i-ésimo valor de entrada de la instancia de entrenamiento actual.
- $\hat{y_j}$ es la salida en la j-ésima neurona de salida para la instancia de entrenamiento actual.
- $y_j$ es la salida objetivo de la j-ésima neurona de salida para la instancia de entrenamiento actual.
- $\eta$ es la tasa de aprendizaje.

El límite de decisión de cada neurona de salida en un perceptrón es lineal, lo que limita su capacidad para aprender patrones complejos. Esta limitación la comparte con otros modelos de clasificación lineal como la regresión logística. Sin embargo, si las instancias de entrenamiento son linealmente separables, se demostró a través del teorema de convergencia de perceptrón que el algoritmo encontrará una solución.

Las debilidades de los perceptrones se destacaron en la monografía titulada "Perceptrones"de Marvin Minsky y Seymour Papert en 1969. Mostraron que los perceptrones no podían resolver ciertos problemas triviales, como el problema de clasificación XOR (ver figura 2.5, lado izquierdo). Esto llevó a una disminución en el interés por las redes neuronales en ese momento. Sin embargo, más tarde se descubrió que al apilar varios perceptrones, un Perceptrón Multicapa (MLP, figura 2.5, lado derecho) podría superar estas limitaciones. Los MLP son capaces de resolver problemas complejos, incluido el problema XOR, mediante la introducción de capas ocultas entre las capas de entrada y salida.

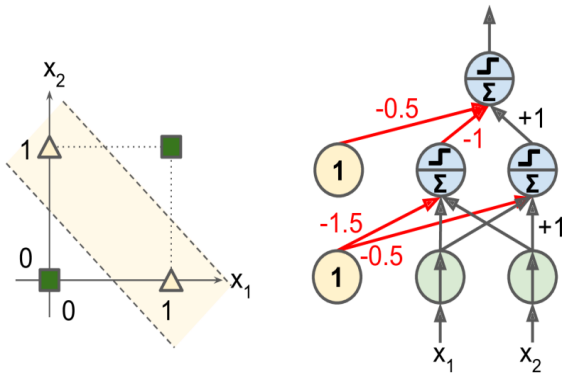



Figura 2.5: Problema de clasificación XOR y un MLP que lo resuelve. Figura tomada de: "Hands On Machine Learning with Scikit-Learn and TensorFlow", Géron, A. (2017).

## 2.4. Perceptrón multicapa y el algoritmo Backpropagation

Un MLP consta de diferentes capas que trabajan en conjunto. Comienza con una capa de entrada (capa de paso), seguida de una o más capas ocultas compuestas por LTU y termina con una capa de salida que también esta compuesta por LTU (ver figura 2.6). Cada capa, excepto la capa de salida, incluye una neurona de sesgo (bias neuron) y está conectada en su totalidad a la siguiente capa. Cuando una RNA tiene dos o más capas ocultas, se denomina red neuronal profunda (DNN, por sus siglas en inglés).

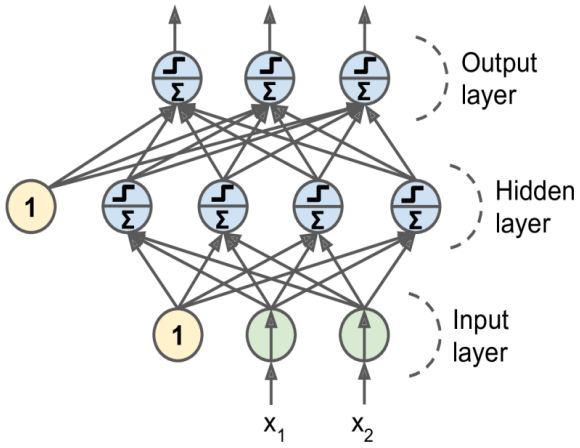



Figura 2.6: Diagrama de un perceptrón multicapa. Figura tomada de: "Hands On Machine Learning with Scikit-Learn and TensorFlow", Géron, A. (2017).

Los investigadores lucharon por encontrar una forma de entrenar a los MLP hasta 1986, año en que se introdujo el algoritmo de entrenamiento de Backpropagation por D. E. Rumelhart et al. Este algoritmo, similar a Gradient Descent, implica un paso hacia adelante para calcular las salidas de las neuronas, medir el error de salida y luego propagar el error hacia atrás a través de las capas para calcular las contribuciones de error de cada neurona.

El algoritmo Backpropagation realiza un paso inverso para medir el gradiente de error en todos los pesos de conexión de la red. De esta manera, propaga eficientemente el gradiente de error hacia

atrás, de ahí el nombre del algoritmo. El paso final implica el uso de Gradient Descent para ajustar los pesos de conexión en función de los gradientes de error. Por lo que, en resumen, el algoritmo de Backpropagation hace predicciones para cada instancia de entrenamiento, mide el error, calcula las contribuciones de error en un paso inverso a través de las capas y luego ajusta los pesos de conexión para minimizar el error usando Gradient Descent.

Por otro lado, para hacer que todo esto funcione, los autores reemplazaron la función de paso en la arquitectura del MLP con la función logística (función sigmoide), $\sigma(z) = 1/(1 + exp(-z))$ para permitir que Gradient Descent pueda avanzar en cada paso y no tenga problemas con los segmentos planos de la función de paso, en donde no hay gradiente. Esto se debe a que la función sigmoide tiene una derivada distinta de cero en todas partes, a diferencia de la función escalonada.

El algoritmo de Backpropagation también se puede utilizar con otras funciones de activación. Dos populares son la función tangente hiperbólica, $tanh(z) = 2\sigma(2z) - 1$ y la función ReLU (unidad lineal rectificada), $ReLU(z) = max(0, z)$. La función de tangente hiperbólica produce valores de salida que van de -1 a 1 y puede ayudar a acelerar la convergencia. La función ReLU es rápida de calcular y funciona bien en la práctica, aunque no es diferenciable en z = 0. En la figura 2.7, se representan estas dos de funciones de activación junto con sus derivadas.

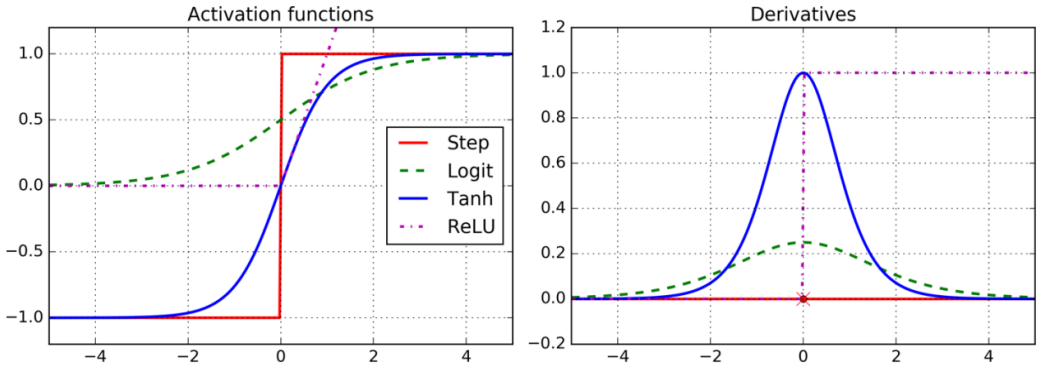



Figura 2.7: Funciones de activación y sus derivadas. Figura tomada de: "Hands On Machine Learning with Scikit-Learn and TensorFlow", Géron, A. (2017).

La arquitectura MLP se usa a menudo para la clasificación, con cada neurona de salida correspondiente a una clase binaria diferente (por ejemplo, perros/gatos, correos deseados/no deseados, etc). En clases exclusivas (por ejemplo, clasificación de imágenes de dígitos), la capa de salida se modifica utilizando ahora la función softmax (ver figura 2.8), donde la salida de cada neurona se interpreta como la probabilidad estimada para cada clase. Es importante destacar que la señal se mueve en una sola dirección, desde las entradas hacia las salidas. Este tipo de arquitectura es un ejemplo de una red neuronal secuencial (FN).

Es importante tener mencionar que la función de activación sigmoide se usaba comúnmente en RNA debido a su parecido con las neuronas biológicas, pero la función de activación ReLU generalmente funciona mejor en la práctica.

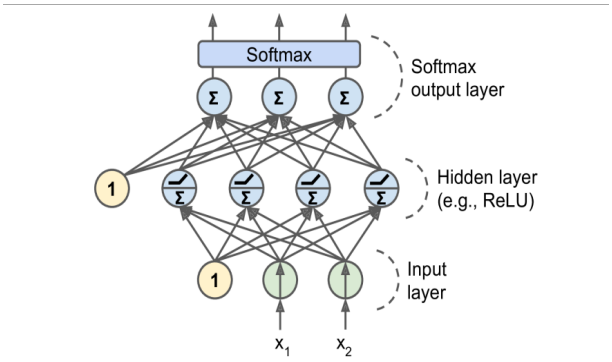



Figura 2.8: Un MLP moderno para la clasificación que incluya ReLU y softmax. Figura tomada de: "Hands On Machine Learning with Scikit-Learn and TensorFlow", Géron, A. (2017).

## 2.5. Hiperparámetros de una red neuronal

La flexibilidad de las redes neuronales, las lleva a la gran desventaja de tener muchos hiperparámetros para ajustar. Algunos de estos pueden ser la topología de la red (cómo se interconectan las neuronas),la cantidad de capas ocultas, la cantidad de neuronas por capa, el tipo de función de activación, el método de inicialización de pesos entre otros varios. Esto hace que encontrar la mejor combinación de hiperparámetros sea todo un desafío.

Para darle una solución a esto la búsqueda en cuadrícula con validación cruzada es un enfoque común para encontrar los hiperparámetros adecuados, pero requiere mucho tiempo. Por otro lado, una mejor opción puede ser la búsqueda aleatoria o emplear herramientas especializadas como Oscar, que se pueden utilizar para una optimización de hiperparámetros más rápida. Sin embargo, también es útil tener una comprensión de los valores adecuados para cada hiperparámetro, lo cual permite limitar el rango de búsqueda. Empecemos analizando la cantidad de capas ocultas.

### 2.5.1. Número de capas ocultas

El número de capas ocultas es un hiperparámetro importante. Mientras que una sola capa oculta puede modelar funciones complejas con suficientes neuronas, las redes profundas con múltiples capas ocultas tienen una mayor eficiencia de parámetros y pueden modelar funciones complejas con menos neuronas que las redes superficiales. Esto a su vez, hace que sea posible entrenarlas en muy poco tiempo.

Las redes neuronales profundas aprovechan las estructuras jerárquicas de los datos. Las capas inferiores modelan estructuras de bajo nivel, las capas intermedias las combinan y las capas superiores modelan estructuras de alto nivel. Esta arquitectura jerárquica ayuda con la convergencia y la generalización a nuevos conjuntos de datos. Una herramienta muy poderosa para ayudar en la generalización de la red, es la reutilización de capas inferiores de una red preentrenada, ya que esto puede acelerar el entrenamiento de nuevas redes para tareas similares. Esto permite que la red se concentre en aprender estructuras de nivel superior en lugar de comenzar desde cero.

Por lo general, comenzar con una o dos capas ocultas puede funcionar bien para muchos problemas. A medida que aumenta la complejidad de la tarea, el número de capas ocultas se puede

aumentar gradualmente hasta que se produzca un sobreajuste en los datos de entrenamiento. Tareas complejas como la clasificación de imágenes o el reconocimiento de voz pueden requerir redes con docenas de capas y una cantidad significativa de datos de entrenamiento.

### 2.5.2. Número de neuronas por capa oculta

Evidentemente, el número de neuronas en las capas de entrada y salida siempre será un valor fijo, puesto que depende únicamente del tipo de entrada y salida que requiera la tarea o problema en el que se éste trabajando. Por otro lado, el número de neuronas por capa oculta puede seguir una estructura similar a la de un embudo, disminuyendo en número de una capa a la siguiente. Esto se debe a que múltipes características de menor nivel pueden combinarse en un número reducido de características de nivel superior. Sin embargo, hoy en día es preferible optar por usar el mismo tamaño para todas las capas ocultas, y así tener un solo hiperparámetro que ajustar, en lugar de uno por cada capa. No obstante, aumentar el número de capas tiende a proporcionar más beneficios que aumentar el número de neuronas por capa.

### 2.5.3. Funciones de activación

La función de activación de ReLU, se usa comúnmente en las capas ocultas, debido a su eficiencia computacional y resistencia a quedarse atascada en mesetas (regiones planas), ya que no se satura con valores de entrada grandes, a diferencia de las funciones sigmoide o tangente hiperbólica que se saturan en 1. Para la capa de salida, La función softmax es adecuada cuando se está trabajando con tareas de clasificación, mientras que si se trata de un problema de regresión, no es necesario poner una función de activación.

Estos son los tres hiperpárametros más simples que se pueden ajustar al momento de entrenar una red neuronal. Sin embargo, existen muchos otros más hiperparámetros o técnicas que se pueden emplear para facilitar el entrenamiento de una RNA, principalmente cuando se trata de redes profundas. A continuación, se describen dos técnicas importantes.

## 2.6. Programación de la tasa de aprendizaje

Establecer una tasa de aprendizaje óptima para la tarea con la que se éste trabajando, puede ser todo un desafío. Si se toma demasiado alta puede hacer que el entrenamiento diverja, mientras que establecerla demasiado baja conduce a una convergencia lenta. Una tasa de aprendizaje ligeramente alta puede hacer un progreso rápido inicialmente, pero no lograr establecerse en torno al óptimo. Sin embargo, generalmente se opta por encontrar soluciones subóptimas para la tasa de aprendizaje.

Encontrar una buena tasa de aprendizaje puede implicar entrenar la red varias veces con diferentes tasas de aprendizaje y comparar las curvas de aprendizaje como se muestra en la figura 2.9. La tasa de aprendizaje ideal logra un aprendizaje rápido y converge a una buena solución.

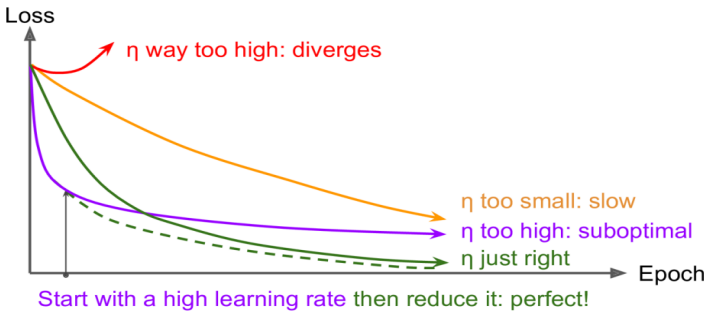



Figura 2.9: Curvas de aprendizaje para varias tasas de aprendizaje. Figura tomada de: "Hands On Machine Learning with Scikit-Learn and TensorFlow", Géron, A. (2017).

Por otro lado, usar una tasa de aprendizaje constante no siempre es óptimo. Es posible llegar a una buena solución más rápido comenzando con una tasa de aprendizaje alta y reduciéndola cuando el progreso se ralentiza. Las diferentes estrategias para reducir la tasa de aprendizaje durante el entrenamiento se conocen como programas de aprendizaje. Las más comunes son:

- **Tasa de aprendizaje constante por partes predeterminada**. Esta implica establecer diferentes tasas de aprendizaje en puntos específicos del entrenamiento. Requiere un ajuste manual para encontrar las tasas de aprendizaje correctas y cuándo cambiarlas.

- **Programación del rendimiento.** Aquí, se mide el error de validación periódicamente y se reduce la tasa de aprendizaje en un factor $\lambda$ cuando el error deja de mejorar, de forma similar a la detención temprana.

- **Programación exponencial.** Esta establece la tasa de aprendizaje en función del número de iteraciones. La tasa de aprendizaje disminuye exponencialmente, normalmente por un factor de 10 cada determinados pasos. Requiere ajustar la tasa de aprendizaje inicial y la tasa de caída exponencial.

- **Programación de energía.** Esta establece la tasa de aprendizaje mediante una función de energía. Esto hace que la tasa de aprendizaje disminuya más lentamente en comparación con la programación exponencial.

Un estudio que comparó la eficacia de algunos de los programas de aprendizaje más utilizados al entrenar redes neuronales profundas en reconocimiento de voz encontró que tanto la programación de desempeño como la programación exponencial funcionaron bien. Sin embargo, se favoreció la programación exponencial debido a su simplicidad, facilidad de ajuste y convergencia ligeramente más rápida a la solución óptima.

Sin embargo, los algoritmos de optimización como AdaGrad, RMSProp y Adam ajustan automáticamente la tasa de aprendizaje durante el entrenamiento, lo que hace innecesaria la programación de tasas de aprendizaje adicionales. Para otros algoritmos de optimización, el uso de la programación exponencial o la programación del rendimiento puede acelerar significativamente la convergencia.

## 2.7. Técnicas de Regularización

Las redes neuronales profundas generalmente suelen tener millones de parámetros, lo cual hace que tengan una gran flexibilidad para adaptarse a distintos conjuntos de datos complejos. No obstante, tienden a sobreajustarse al conjunto de entrenamiento con gran facilidad. Para solucionar este problema, se emplean distintas técnicas de regularización, a continuación se presentan algunas de ellas.

### 2.7.1. Detención anticipada

La detención anticipada es una técnica de regularización común en la que el entrenamiento se interrumpe cuando el rendimiento del modelo en el conjunto de validación comienza a disminuir. Esta técnica es muy fácil de implementar al entrenar una RNA y además, se ha observado que proporciona mejores resultados cuando se combina con otras técnicas de regularización.

### 2.7.2. Regularizaciones $l_1$ y $l_2$

La regularización $l_1$, agrega un término a la función de costo del modelo que es proporcional a la suma de los valores absolutos de los coeficientes del modelo. Esto tiene el efecto de penalizar los coeficientes más pequeños y, en algunos casos, establecer algunos de ellos en cero. En consecuencia, $l_1$ tiende a producir modelos más dispersos y puede ayudar en la selección automática de características.

La regularización $l_2$, agrega un término a la función de costo del modelo que es proporcional a la suma de los cuadrados de los coeficientes del modelo. Al igual que $l_1$, $l_2$ penaliza los coeficientes más grandes, pero a diferencia de $l_1$, no establece los coeficientes en cero de forma automática. En su lugar, $l_2$ tiende a empujar los coeficientes hacia valores más pequeños y distribuirlos de manera más uniforme, reduciendo la complejidad del modelo y evitando el sobreajuste.

En resumen, la regularización $l_1$ y $l_2$ son similares a los modelos lineales y se utilizan para restringir los pesos de conexión de las redes neuronales.

### 2.7.3. Dropout

Dropout, es probablemente la técnica de regularización más popular al entrenar RNAs. Su funcionamiento es bastante simple, pues consiste solo en descarta neuronas aleatoriamente (excluyendo las neuronas de salida) durante cada paso de entrenamiento. Esto obliga a la red a ser más robusta y menos dependiente de neuronas específicas, lo que lleva a una mejor generalización. Se ha demostrado que el uso de esta técnica mejora ligeramente la precisión de las redes neuronales.

En otras palabras, Dropout genera una red neuronal única en cada paso de entrenamiento, creando un conjunto de diversos modelos. Este conjunto contribuye a mejorar el rendimiento y la capacidad de generalización de la red. Por otro lado, durante el entrenamiento, para compensar la deserción de neuronas, los pesos de conexión de entrada deben multiplicarse por la probabilidad de mantenimiento (1 - tasa de deserción) o dividir la salida de la neurona por la probabilidad de mantenimiento.

Existe una variante de Dropout, llamada Dropconnect, donde las conexiones individuales se eliminan aleatoriamente en lugar de neuronas completas. Sin embargo, Dropout en general ha mostrado dar mejores resultados que Dropconnect.

## 2.8. Redes Neuronales Convolucionales

Es momento de hablar de una de las arquitecturas de RNA más impresionantes y poderosas en la actualidad. Para ello, primero hablemos de un dato histórico interesante. Mientras que, nosotros los humanos podemos realizar tareas como detectar un cachorro en una imagen o reconocer tareas habladas sin esfuerzo alguno, las computadoras lucharon con tales tareas hasta hace poco. Esto se debe a que la percepción ocurre en gran medida fuera de nuestra conciencia, dentro de módulos sensoriales especializados en nuestros cerebros. Comprender estos módulos sensoriales es clave para comprender la percepción.

Las redes neuronales convolucionales (CNN), fueron inspiradas en la corteza visual del cerebro y se han utilizado para el reconocimiento de imágenes desde la década de 1980. Con los avances en el poder computacional, los datos de entrenamiento disponibles y las técnicas de entrenamiento de redes neuronales profundas, las CNN han logrado un rendimiento sobrehumano en tareas visuales complejas. Impulsan aplicaciones como la búsqueda de imágenes, los vehículos autónomos y la clasificación de videos. Además, no solo son buenas en tareas visuales, también son buenas en el reconocimiento de voz y en el análisis de series de tiempo.

Una razón importante por la cual optar por una CNN, en lugar de una red neuronal profunda regular con capas completamente conectadas para el reconocimiento de imágenes, es debido a la gran cantidad de parámetros que se requieren para imágenes grandes. Las CNN abordan este problema mediante el uso de capas parcialmente conectadas, lo que reduce la cantidad de conexiones y mejora la eficiencia del modelo.

### 2.8.1. Capa Convolucional

La pieza clave de una CNN, radica en la capa convolucional, la cual consta de neuronas que no están conectadas a cada píxel de la imagen de entrada (como lo estarían con capas planas), sino solo a los píxeles dentro de sus campos receptivos como se muestra en la figura 2.10. Esta estructura jerárquica permite que la red se centre en características de bajo nivel en las primeras capas y las reúna en características de nivel superior en capas posteriores, lo que hace que las CNN sean efectivas para el reconocimiento de imágenes.

Cada neurona de una capa está conectada a una región específica de la capa anterior, determinada por la altura $f_h$ y el ancho $f_w$ del campo receptivo (ver figura 2.11). El relleno con ceros (zero padding) se usa comúnmente para mantener la misma altura y ancho entre las capas, y como su nombre lo dice, implica agregar ceros alrededor de las entradas. Otro parámetro importante en una capa convolucional es el paso (stride), que es la distancia entre dos campos receptivos consecutivos. Esta distancia, se puede ajustar para conectar una capa de entrada más grande con una capa más pequeña. Las neuronas de la capa superior están conectadas a regiones específicas de la capa anterior en función de los valores del paso.

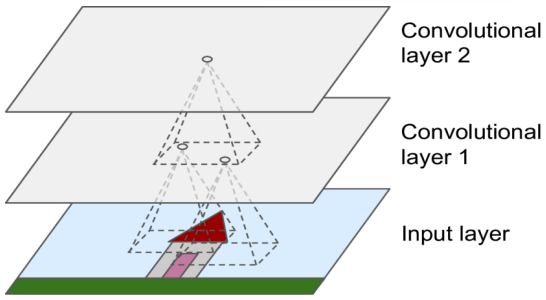



Figura 2.10: Capas CNN con campos receptivos locales rectangulares. Figura tomada de: "Hands On Machine Learning with Scikit-Learn and TensorFlow", Géron, A. (2017).

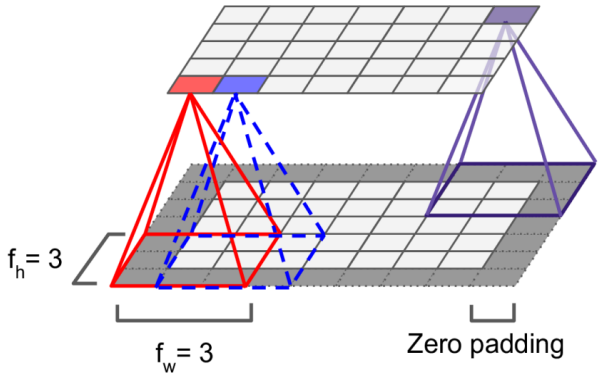



Figura 2.11: Conexiones entre capas y padding cero. Figura tomada de: "Hands On Machine Learning with Scikit-Learn and TensorFlow", Géron, A. (2017).

### 2.8.2. Filtros

Los pesos de las neuronas en una CNN se pueden representar como filtros o núcleos de convolución. Los filtros son pequeñas imágenes que coinciden con el tamaño del campo receptivo, los cuales resaltan patrones específicos en la imagen de entrada. Por ejemplo, un filtro de línea vertical enfatiza las líneas verticales, mientras que un filtro de línea horizontal enfatiza las líneas horizontales (ver figura 2.12). Las neuronas que usan estos filtros ignoran otras partes de su campo receptivo.

Cuando todas las neuronas de una capa usan el mismo filtro, la capa genera un mapa de características que resalta las áreas en la imagen similares al filtro, como se muestra en la figura 2.12. La CNN aprende a combinar estos filtros en patrones más complejos durante el entrenamiento. Por ejemplo, un patrón cruzado puede activar neuronas que responden a filtros tanto verticales como horizontales, lo que indica la presencia de un patrón en forma de cruz en una imagen.

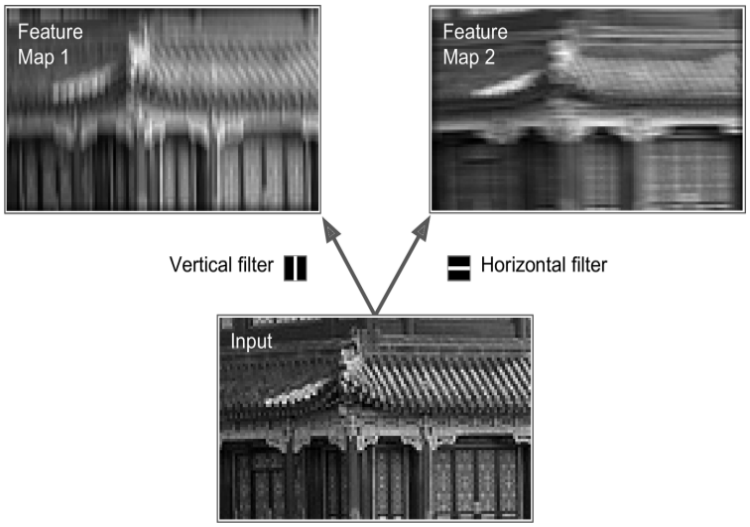



Figura 2.12: Aplicando dos filtros diferentes para obtener dos mapas de características. Figura tomada de: "Hands On Machine Learning with Scikit-Learn and TensorFlow", Géron, A. (2017).

### 2.8.3. Múltiples mapas de características

En realidad, una capa convolucional se compone de varios mapas de características del mismo tamaño representados en 3D, en lugar de una fina capa 2D (ver figura 2.13). Cada mapa de características consta de neuronas que comparten los mismos parámetros (pesos y términos de sesgo), mientras que diferentes mapas de características pueden tener parámetros diferentes. El campo receptivo de una neurona se extiende a través de todos los mapas de características de las capas anteriores, lo que permite que la capa convolucional aplique simultáneamente múltiples filtros para detectar varias características en las entradas.

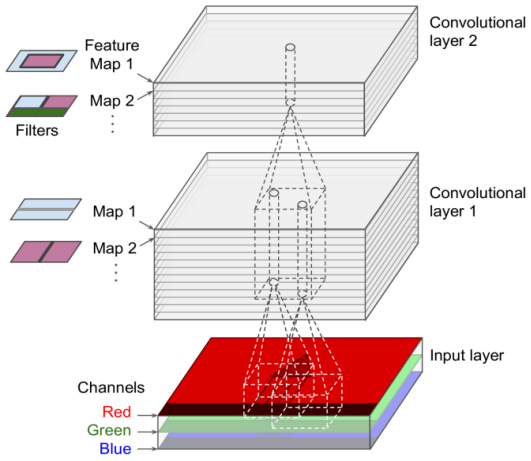



Figura 2.13: Capa de convolución con múltiples mapas de característica e imágenes con tres canales. Figura tomada de: "Hands On Machine Learning with Scikit-Learn and TensorFlow", Géron, A. (2017).

Compartir parámetros entre neuronas en un mapa de características reduce significativamente la cantidad de parámetros en el modelo. Además, una vez que la CNN aprende a reconocer un patrón en una región, puede reconocerlo en cualquier otra región. A diferencia de una red neuronal profunda regular, que solo puede reconocer un patrón en la región específica que lo aprendió.

Por otro lado, las imágenes de entrada, también se suelen componer de varias subcapas, donde cada una de ellas representa un canal de color. Generalmente, las imágenes están en formato RGB, por lo que tienen 3 canales (rojo, verde y azul). Sin embargo, las imágenes en escala de grises tienen un solo canal, mientras que otras imágenes, como las imágenes satelitales que capturan diferentes frecuencias de luz, pueden tener una mayor cantidad de canales.

Finalmente, la conexión entre una neurona ubicada en el mapa de características $k$ de la capa convolucional $l$ y las salidas de las neuronas en la capa anterior $l-1$, está dada por la ecuación 2.2, la cual es un poco confusa por todos los índices que tiene, pero lo único que hace es calcular la suma ponderada de todas las entradas, más el término de sesgo.

$$z_{i,j,k} = b_k + \sum_{u=1}^{f_h} \sum_{v=1}^{f_w} \sum_{k'=1}^{f'_n} (x_{i',j',k'} \cdot w_{u,v,k',k}) \quad \text{con} \begin{cases} i' = u \cdot s_h + f_h - 1 \\ j' = v \cdot s_w + f_w - 1 \end{cases} \tag{2.2}$$

- $z_{i,j,k}$ es la salida de la neurona localizada en la fila $i$, columna $j$ en el mapa de características $k$ de la capa convolucional actual (capa $l$).
- $s_h$ y $s_w$ son los pasos verticales y horizontales (tamaño del stride), $f_h$ y $f_w$ son la altura y el ancho del campo receptivo, y $f_{n'}$ es el número total de mapas de características de la capa anterior (capa $l-1$).
- $x_{i',j',k'}$ es la salida de la neurona localizada en la capa $l-1$, fila $i'$, columna $j'$, mapa de características $k'$ (o canal $k'$ si la capa anterior es la capa de entrada).
- $b_k$ es el término de sesgo para el mapa de características $k$ (en la capa $l$).
- $w_{u,v,k',k}$ es el peso de conexión entre cualquier neurona en el mapa de características $k$ de la capa $l$ y su entrada ubicada en la fila $u$, columna $v$ (relativo al campo receptivo de la neurona) y el mapa de características $k'$

### 2.8.4. Capa de Agrupación (Pooling Layer)

Las capas de agrupación tienen como objetivo reducir los mapas de características o los canales de la imagen (para el caso de la capa de entrada), lo que reduce la carga computacional, el uso de memoria y la cantidad de parámetros. Esto ayuda a evitar el sobreajuste y permite que la red tolere ligeros cambios en la imagen (invariancia de ubicación).

La conexión entre una neurona en una capa de agrupación y las neuronas de la capa anterior, es muy similar a las descritas para capas convolucionales, es decir, se conecta a un número limitado de neuronas en la capa anterior, dentro de un pequeño campo receptivo rectangular. Es necesario definir el tamaño del paso y el tipo de relleno de la capa de agrupación. No obstante, las neuronas agrupadas no tienen pesos; agregan las entradas usando una función de agregación como el máximo o la media.

En la figura 2.14, se ilustra una capa de agrupación máxima (maxpooling), que comunmente es el tipo más usado de estas capas. En este ejemplo, se utiliza un núcleo de agrupación de 2x2, un paso de 2 y ningún tipo de relleno. Solo el valor de entrada máximo dentro de cada kernel se pasa a la siguiente capa, mientras que las otras entradas se descartan. Esto da como resultado una reducción de la muestra de la salida por un factor de 2 en ambas dimensiones, reduciendo efectivamente el área de la imagen cuatro veces y por lo tanto eliminando el 75 % de los valores de entrada.

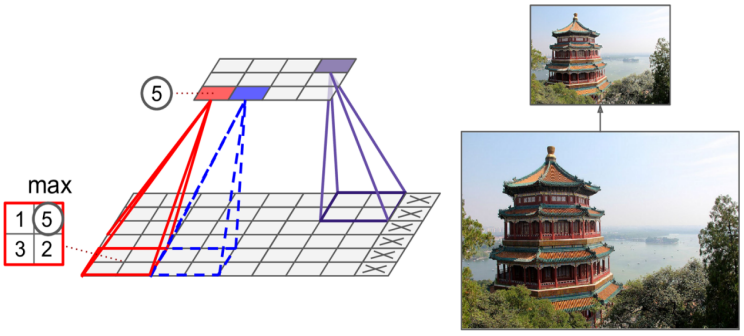



Figura 2.14: capa maxpooling (núcleo de agrupación 2x2, paso 2, sin relleno). Figura tomada de: "Hands On Machine Learning with Scikit-Learn and TensorFlow", Géron, A. (2017).

Las capas de agrupación normalmente funcionan de forma independiente en cada canal de entrada, manteniendo la misma profundidad en la salida que en la entrada. Alternativamente, la agrupación se puede realizar sobre la dimensión de profundidad, reduciendo el número de canales mientras se mantienen las dimensiones espaciales (alto y ancho) sin cambios.

## 2.9. Redes Neuronales Residuales (ResNet)

Finalemente, pero no menos importante, es turno de hablar de las redes neuronales residuales (ResNet). Presentadas por primera vez por Kaimig He y su equipo en 2015, las ResNet son una de las arquitecturas de RNA más populares en el campo del aprendizaje profundo. De hecho, un modelo realizado con esta nueva arquitectura de red neuronal fue capaz de ganar el primer lugar en el desafío ILSVRC ImageNet de ese mismo año.

La innovación clave de las ResNet es el uso de conexiones de salto, también conocidas como conexiones de acceso directo o simplemente conexiones residuales. En estas conexiones residuales a la entrada que alimenta a una capa también se le agrega (es decir, suma) la salida de una capa ubicada un poco antes de ella. Lo cual hace, que en lugar de esperar que cada capa aprenda directamente la representación deseada, las capas aprendan las diferencias entre la representación deseada y la representación actual.

Cuando se agrega una conexión de residual, la red modela la función f(x) = h(x) - x, en lugar de solo la función h(x), donde h(x) es la función objetivo que se quiere modelar. Esto recibe el nombre de aprendizaje residual (ver figura 2.15). Este enfoque acelera el entrenamiento, especialmente cuando la función objetivo es similar a la función de identidad.

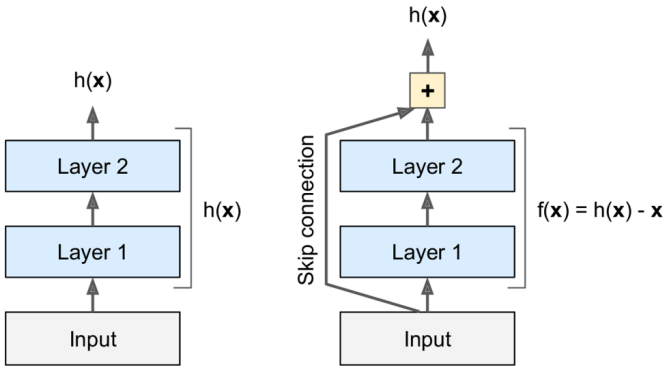



Figura 2.15: Aprendizaje residual. Figura tomada de: "Hands On Machine Learning with Scikit-Learn and TensorFlow", Géron, A. (2017).

Emplear muchas conexiones residuales permite que la señal se propague a través de la red, incluso si algunas capas aún no han comenzado a aprender. Esto permite que la red progrese al transferir fácilmente la señal a través de ella sin llegar a ser atenuada. Las ResNet profundas, están construidas como una pila de unidades residuales, donde cada unidad es una pequeña red neuronal con una conexión de salto. En la figura 2.16, se muestra una comparación entre una red neuronal secuencial y una red residual (ambas redes son profundas)

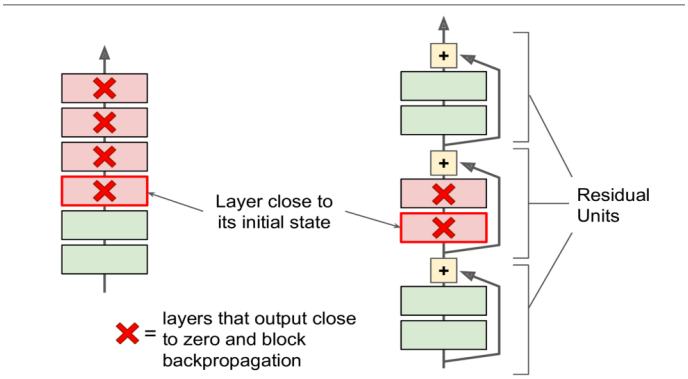



Figura 2.16: Red neuronal secuencial profunda (lado izquierdo) comparada con una red residual (lado derecho). Figura tomada de: "Hands On Machine Learning with Scikit-Learn and TensorFlow", Géron, A. (2017).

La arquitectura de ResNet es simple y se le puede implementar tanto a una red densa (Perceptrón multicapa), como a una red convolucional. Actualmente, las ResNet se considera la arquitectura más potente y sencilla. Sin embargo, también vale la pena explorar otras arquitecturas como VGGNet e Inception-v4, con un rendimiento competitivo en el desafío ILSVRC.

# Capítulo 3

# Metodología

Esta sección contiene la metodología que se siguió para la estimación de las tasas de infección interpoblacional de un modelo SIR metapoblacional mediante el uso de redes neuronales. Incluye el cómo se realizaron las simulación númericas del modelo SIR metapoblacional para la enfermedad elegida, la realización de la base de datos y la configuración y entrenamiento de cada uno de los diferentes tipos de redes neuronales artificiales (RNA) con los que se trabajó.

Durante todo el desarrollo de este trabajo, se creó un repositorio de **GitHub** [11], el cual cuenta con tres carpetas principales, cada una enfocada a una tarea o problema en específico, que se fueron presentando conforme a la marcha de este proyecto. A continuación, se describe la función de cada una de ellas.

## 3.1. Modelos Epidémicos

Esta carpeta contiene los scripts de python diseñados para la simulación matemática de algunos modelos epidémicos, los cuales fueron el punto de partida de este trabajo. Se inició con el modelo epidémico simple de Kermack-McKendrick, es decir, el modelo SIR visto en la sección 1.1. La simulación se hizo mediante el uso de la función **Odeint** de la biblioteca **scipy** de python. Los parámetros que se eligieron fueron los reportados en el ejemplo "La Gran Plaga en Eyam"(sección 1.1), de esta forma, se pudo comprobar la confiabilidad de la simulación, ya que se obtuvo el mismo comportamiento que el reportado en la literatura (figura 3.1).

Una vez lograda la simulación para el modelo SIR, lo que siguió ahora fue simular un modelo epidémico más complicado. Se eligió un modelo SEAIRD (Susceptibles, Expuestos, Asintomáticos, Infectados, Recuperados y Muertos), el cual, se tomó del artículo "Estrategias de movilidad basadas en la teoría de percolación para evitar la diseminación de enfermedades: COVID-19" [6], de modo que, los parámetros utilizados en esta simulación fueron los reportados en dicho artículo. Nuevamente se hizo uso de la mencionada biblioteca de python, por lo que el código es esencialmente el mismo utilizado para el modelo SIR, solo que se hicieron ligeras modificaciones respecto al sistema de ecuaciones diferenciales ordinarias correspondiente para este nuevo modelo. En la figura 3.2, se muestra el resultado de la simulación matemática obtenida.

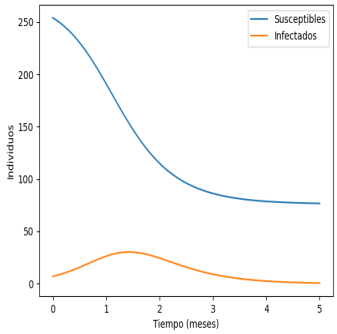



Figura 3.1: Simulación matemática del modelo SIR para "La Gran Plaga en Eyam"

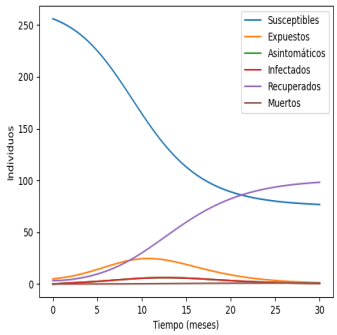



Figura 3.2: Simulación matemática del modelo SEAIRD para Covid 19.

A partir de estas dos simulaciones, se logró apreciar que se puede simular cualquier tipo de modelo epidémico de esta manera, solo hay que tener cuidado al declarar el sistema de ecuaciones correspondiente al modelo de interés. Ahora, es momento de realizar este mismo procedimiento con el modelo epidémico utilizado para esta tesis, es decir, el modelo epidémico SIR Metapoblacional (ver sección 1.3).

En principio, se siguió el mismo razonamiento que el empleado anteriormente, sin embargo, hay que tener mucho cuidado al definir este modelo, ya que, como se vio en la sección 1.3, en este tipo de modelos sí se toma en cuenta el flujo de individuos a través de distintas locaciones que formen parte de una misma población, ya que incorporan al espacio como una variable discreta, mediante un gran sistema de ecuaciones diferenciales ordinarias (EDO). Una vez teniendo claro las principales diferencias entre el modelo SIR normal y el metapoblacional, se decidió solo considerar dos poblaciones o parches por simplicidad; en consecuencia el sistema toma la forma mostrada en la ecuación 3.1. Además, se eligió la **Peste Negra**, como enfermedad específica a modelar, ya que la bibliografía que existe acerca de esta enfermedad es extensa y frecuentemente se usa como caso de estudio para modelar. Por lo tanto, la elección de los parámetros utilizados en la simulación fue hecha respecto al rango de valores reportados para esta enfermedad (ver tabla 3.1).

$$\begin{aligned} S_1' &= \mu_1 N_1 - S_1 \sum_{j=1}^{2} \beta_{1j} I_j - \mu_1 S_1, \\ S_2' &= \mu_2 N_2 - S_2 \sum_{j=1}^{2} \beta_{2j} I_j - \mu_1 S_2, \\ I_1' &= S_1 \sum_{j=1}^{2} \beta_{1j} I_j - (\mu_1 + \gamma_1) I_1, \\ I_2' &= S_2 \sum_{j=1}^{2} \beta_{2j} I_j - (\mu_2 + \gamma_2) I_2. \end{aligned} \tag{3.1}$$

| Parámetro | Significado | Valor |
|---|---|---|
| $\beta$ | Matriz de las tasas de infección | $\begin{bmatrix} 0.001 & 0.001 \\ 0.002 & 0.002 \end{bmatrix}$ |
| $\mu$ | Vector de las tasas de mortalidad | $(0.00011, 0.00013)$ |
| $\gamma$ | Vector de las tasas de recuperación | $(1/1.3, 1/1.4)$ |
| $N$ | Vector de los parches o subpoblaciones | $(600, 400)$ |
| $\Re_1$ | Número de reprodución básica para el parche 1 | 0.7798 |
| $\Re_2$ | Número de reprodución básica para el parche 2 | 1.1197 |

Tabla 3.1: Descripción de los parámetros empleados para la simulación matemática del modelo SIR Metapoblacional

Nuevamente, la simulación se hizo mediante el uso de la función **Odeint**, la cual, fue capaz de resolver el sistema de EDO que definen a este modelo (3.1) sin problema alguno, solo pasándole las condiciones iniciales y parámetros adecuados. Por otro lado, como el número de reproducción básico para el i-ésimo parche $\Re_i$ depende del tamaño de la población de ese parche, para este tipo de modelos se consideró una metapoblación $N$ de 1000 habitantes, conformada por dos subpoblaciones $n_1$ y $n_2$ de 600 y 400 individuos respectivamente. Con todo lo anterior en mente, se eligieron las siguientes condiciones iniciales:

$$S(0) = (593, 393), \quad I(0) = (7, 7), \qquad S_0 + I_0 = N,$$

donde $S(0)$ e $I(0)$ son los vectores de individuos susceptibles e infectados respectivamente, al tiempo $t = 0$ y siendo $N$ el vector de las poblaciones. La simulación se realiza durante el intervalo de tiempo $0 \leq t \leq 20$, midiendo al tiempo en meses. En la figura 3.3, se observan las gráficas obtenidas para $S$ e $I$ para cada subpoblación, en las cuales se aprecia un comportamiento muy similar al visto en el ejemplo "La Gran Plaga en Eyam"(sección 1.1), lo que era de esperarse, puesto que se trata de la misma enfermedad.

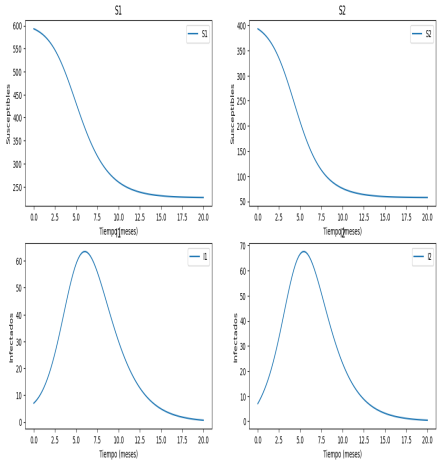



Figura 3.3: Simulación matemática del modelo SIR Metapoblación.

## 3.2. Base de Datos

En todo problema de aprendizaje automático, es necesario contar con una base de datos relacionada al contexto o problema que se está abordando. La elección de esta base de datos debe hacerse cuidadosamente, prestando atención a los elementos que la conforman, ya que esto determinará los resultados que nuestro modelo de aprendizaje automático pueda brindarnos, así como su complejidad. Si nuestro modelo recibe información precisa y adecuada sobre lo que queremos predecir, será más fácil que identifique patrones y realice extrapolaciones, obteniendo mejores resultados. Con todo lo anterior en mente, se creó la carpeta titulada "Base de Datos", donde se almacenaron todos los scripts de Python y demás archivos que se utilizaron durante la elaboración de la base de datos que se empleó en este trabajo.

Primero, hay que recordar que el objetivo principal de esta tesis es hacer una estimación de las tasas de infección interpoblacional de un modelo SIR metapoblacional; es decir, a partir de un cierto modelo SIR metapoblacional conocido, queremos predecir dichas tasas de infección mediante un modelo de aprendizaje automático. Estos modelos requieren principalmente dos cosas: datos de entrada y datos de salida. Es fácil ver que los datos de salida que necesitamos son estas tasas de infección interpoblacional pero, ¿qué datos de entrada debemos elegir? Para responder esta pregunta, necesitamos analizar la información que tenemos acerca de los modelo epidémicos.

Recordemos que según la epidemiologia matemática, vista en el capítulo 1, cualquier epidemia puede ser descrita mediante un sistema de EDO y, por lo tanto, la solución de este sistema describe la evolución de la epidemia conforme el paso del tiempo. Sin embargo. en la realidad, cuando ocurre una epidemia, lo único que podemos medir o reportar en el transcurso de esta son cantidades tales como:

- Número de casos. La cantidad de personas que han sido diagnosticadas con la enfermedad en cuestión durante un período de tiempo determinado.
- Número de hospitalizaciones. La cantidad de personas que han requerido hospitalización debido a la enfermedad.

- Número de muertes. La cantidad de personas que han fallecido a causa de la enfermedad.
- Tasa de incidencia. La cantidad de nuevos casos de la enfermedad que se presentan en una población durante un período de tiempo determinado.
- Tasa de mortalidad. La cantidad de muertes por la enfermedad en una población durante un período de tiempo determinado.

Debido a esto, deberíamos de ser capaces de llevar a cabo el análisis correspondiente solo haciendo uso de cantidades como estas, ya que son con las que realmente se cuenta cuando ocurre una epidemia. Para nuestro análisis, nos centraremos en el número de casos, es decir, la cantidad de individuos que han sido infectados y, por lo tanto, forman parte de la clase infectados $I$. En teoría, esta cantidad debería estar fuertemente relacionada con las tasas de infección interpoblacional, por lo que supondremos que existe una correlación entre ambos y, así, deducir las tasas de infección a partir de ella.

Se optó por simplificar el problema mediante una simulación matemática de una epidemia, en lugar de utilizar datos reales. El modelo SIR metapoblacional fue elegido por su simplicidad, ya que es el modelo más sencillo para abordar esta situación. De esta forma, fue como se eligieron los elementos que conforman nuestra base de datos, siendo los datos de entrada las gráficas de individuos infectados con respecto al tiempo obtenidas en la simulación del modelo SIR metapoblacional realizada en la sección anterior, y la salida las tasas de infección $\beta_{ij}$ asociadas a ese modelo, almacenadas todas en una matriz de 2x2 (matriz de las tasas de infección).

Por último, es necesario realizar más simulaciones del modelo SIR metapoblacional con parámetros distintos, puesto que ajustar un modelo de aprendizaje automático que solo haga esta estimación para la simulación específica realizada en la sección anterior, no sería aprendizaje automático, ni tampoco lo que realmente se está buscando. Lo que se busca es que, a partir de las curvas de infección (gráficas de individuos infectados con respecto al tiempo) obtenidas de una simulación cualquiera del modelo SIR metapoblacional, se pueda obtener una muy buena estimación de las $\beta_{ij}$ correspondientes a dicha simulación. Nuevamente, como en todo problema de aprendizaje automático, entre mayor sea la cantidad de datos que tengamos (en este caso, simulaciones), la probabilidad de éxito será mayor. Sin embargo, es evidente que realizar multiples simulaciones sería tardado aun con el código realizado anteriormente. Para solucionar esto, se creó un nuevo objeto de python llamado **Class SIR Metapoblacional**.

Este nuevo objeto de python, se crea a partir de una matriz de tasas de infección y dentro de su constructor **init** declara los parámetros del modelo, que incluyen las tasas de infección $\beta_{ij}$, la tasa de mortalidad $\mu$, la tasa de recuperación $\gamma$ y el tamaño de la población $n$ para cada subpoblación. Luego, se define la simulación del modelo mediante la función interna **Simulate**, la cual, sigue la misma estructura que la de los códigos empleados en la sección anterior, tomando como argumentos las condiciones iniciales $S_0$ e $I_0$ (vectores de susceptibles e infectados iniciales para cada subpoblación) y el tiempo de simulación $t$. Los parámetros que se utilizan al crear uno de estos objetos, son los mismo que los reportados en la tabla 3.1, excepto por las tasas de infección $\beta_{ij}$, ya que lo que se busca es hacer distintas simulaciones del modelo SIR metapoblacional, variando solo estas tasas y dejando fijos los demás parámetros.

Una vez hecho esto, para lograr una comparación en distintos escenarios, se crearon distintas matrices de infección donde se fueron cambiando las tasas de infección. Se inició con los valores $\beta_{11} = \beta_{21} = 0.001$, $\beta_{22} = \beta_{12} = 0.002$ y se fueron incrementando en $\Delta\beta_1 = 0.0001$, para $i = j$ y $\Delta\beta_2 = 0.00005$, para $i \neq j$, hasta llegar a un valor $\beta_{ij} \leq 0.1$ (los valores iniciales de $\beta_{ij}$, no formaron parte de la base de datos). Después, se creó un objeto **Class SIR Metapoblacional** para cada

una de las matrices de infección y se realizó una simulación de la epidemia para cada uno de estos objetos durante el intervalo de tiempo $0 \leq t \leq 20$ y condiciones iniciales:

$$S(0) = (593, 393), I(0) = (7, 7), S_0 + I_0 = N$$

En los archivos **'Is.npy'** y **'betas.npy'**, se guardaron las curvas de infección obtenidas en cada simulación y las matrices de infección asociadas a dichas simulaciones respectivamente. Ambos se guardaron como objetos numpy de python, para después poder acceder a ellos. En total, se cuenta con 980 datos de entrada y salida, conformados por matrices de 2x100 para los datos de entrada (cada columna corresponde a la curva de infectados correspondiente a una subpoblación) y por las matrices de las tasas de infección para los datos de salida.

## 3.3. RNA

Una vez teniendo nuestra base de datos lista, el siguiente paso en los problemas de aprendizaje automático es seleccionar el modelo que mejor se ajuste o describa dicha base de datos. Para los objetivos de este trabajo, se decidió trabajar únicamente con Redes Neuronales Artificiales (RNA). No obstante, en la literatura existen diversas arquitecturas de RNA, por lo que se eligieron solo tres diferentes tipos de redes: Densas, Convolucionales y Residuales. Cada una de ellas se probó con la base de datos obtenida en la sección anterior y se compararon las precisiones obtenidas para determinar cuál describía mejor los datos de acuerdo a ciertos criterios.

A continuación, se muestra el análisis que se llevó a cabo con cada tipo de red.

### 3.3.1. Red Densa

Se inició con una Red Densa, ya que como se vio en el capítulo 2, es el tipo de red más simple que existe. Sin embargo, una vez que se cargó la base de datos, fue necesario hacerle cierto tratamiento a los datos antes de empezar con las pruebas, dependiendo del modelo que se vaya a emplear. El primer paso consistió en dividir nuestra base de datos en datos de entrenamiento y datos de prueba, lo cual se hizo utilizando la biblioteca **sklearn**, tomando el 20 % de los datos como datos de prueba (cantidad estándar que se emplea en aprendizaje automático). Una ventaja de utilizar **sklearn** es que no solo divide nuestros datos de manera rápida y sencilla, sino que también los barajea, evitando así el sobreajuste del modelo debido a la manera en que están ordenados. Después, para este tipo de red se tuvieron que aplanar tanto los datos de entrada como los de salida, convirtiéndolos ahora en matrices de 1x200 y 1x4 respectivamente. Por lo tanto, ahora se tiene un vector de curvas de infección, donde los primeros 100 elementos corresponden al primer parche y los restantes al segundo.

Ahora, como estamos empleando redes neuronales, es importante normalizar nuestros datos de entrada para mejorar la eficiencia del entrenamiento, prevenir el sobreajuste del modelo y mejorar el rendimiento general de la red neuronal. La normalización se refiere al proceso de ajustar los datos de entrada, de modo que tengan una distribución uniforme. Esto puede ayudar a garantizar que los datos de entrada se encuentren en el mismo rango y que la magnitud de los valores no afecte el resultado final.

Para hacer esto, una primera opción podría ser dividir nuestros datos de entrada entre el valor máximo global, en este caso 600, que es el tamaño de la población más grande y, por lo tanto, las curvas de infección no pueden exceder este valor. Sin embargo, esto lograría efectivamente que se encuentren entre 0 y 1, pero no en el mismo rango de magnitud, debido a que se observó que el máximo de la curva de infección depende de las tasas de infección. Cuando estas tasas son lo suficientemente pequeñas, el máximo de la curva no llega ni cerca de 600, teniendo datos con órdenes de magnitud considerablemente distintos, lo cual puede afectar el resultado final. Una posible alternativa para solucionar este problema sería dividir cada curva de infectados entre su máximo local en lugar de utilizar el máximo global, sin embargo, esta solución no es tan trivial de ejecutar. Afortunadamente, **sklearn** también nos proporciona una solución para esto mediante el objeto **StandardScaler**. Este objeto estandariza los datos restando la media y dividiéndolos por la desviación estándar de cada característica, de esta manera, los valores normalizados estarán siempre centrados en cero y tendrán una desviación estándar igual a 1.

Después de darle el tratamiento necesario a los datos, se dio inicio al entrenamiento de la red neuronal. Por experiencia previa con problemas de aprendizaje automático, se eligió una red densa conformada de solo dos capas ocultas y una capa de salida. Cada capa oculta esta conformada por 512 neuronas y una función de activación **relu**, mientras que la capa de salida tiene solo 4 neuronas (igual a la cantidad de tasas de infección que queremos predecir) y una función de activación **sigmoid**. En la figura 3.4 se muestra la estructura de este modelo.

| | Name | Type | # Parameters | Output Shape |
|---|---|---|---|---|
| ● | dense_input | InputLayer | 0 | ,200 |
| ● | dense | Dense | 102912 | None, 512 |
| ● | dense_1 | Dense | 262656 | None, 512 |
| ● | dense_2 | Dense | 2052 | None, 4 |

Figura 3.4: Resumen del modelo de la primera red densa entrenada. Gráfica obtenida del software Weights & Biases.

Se utilizó la biblioteca **TensorFlow** para implementar la red neuronal y se configuró con los siguientes hiperparámetros:

- Función de costo: Error cuadrático medio **mse**
- Métrica: Error porcentual absoluto medio **mape**
- Optimizador: **RMSprop**
- Tasa de aprendizaje: $\eta = 2.5$

Debido a que se trata de un problema de regresión, se eligieron la función de costo **mse** y la métrica **mape**. Esta métrica se utilizó para medir la precisión del modelo durante el entrenamiento, ya que mide el tamaño del error absoluto en términos porcentuales, por lo que, un valor de **mape** del 10 % significa que, en promedio, las predicciones o estimaciones de nuestro modelo tienen el 10 % de error en comparación con los valores verdaderos. El optimizador **RMSprop**, se eligió porque ha demostrado ser más efectivo en problemas de aprendizaje profundo. Finalmente, se tomó un

valor inicial $\eta = 2.5$ para poder identificar el umbral de la tasa de aprendizaje y a partir de ahí ir reduciendo en órdenes de 10, hasta encontrar la tasa de aprendizaje adecuada para esta tarea.

Este modelo fue entrenado durante un total de 30 épocas con un minibatch de 20 y se utilizó el callback **Wandb** [12] para guardar los datos del entrenamiento. Se realizaron seis corridas diferentes, cada una con valores de $\eta$ distintos, donde se identificó que el valor $\eta = 0.025$ es el umbral de la tasa de aprendizaje, ya que a partir de este valor, la función de costo comienza a disminuir, es decir, el modelo comienza a aprender. En la figura 3.5, se muestra la gráfica de la función de costo para las diferentes tasas de aprendizaje empleadas, en la cual se observa una curva bien comportada (suave) en $\eta = 2.5 \times 10^{-5}$.

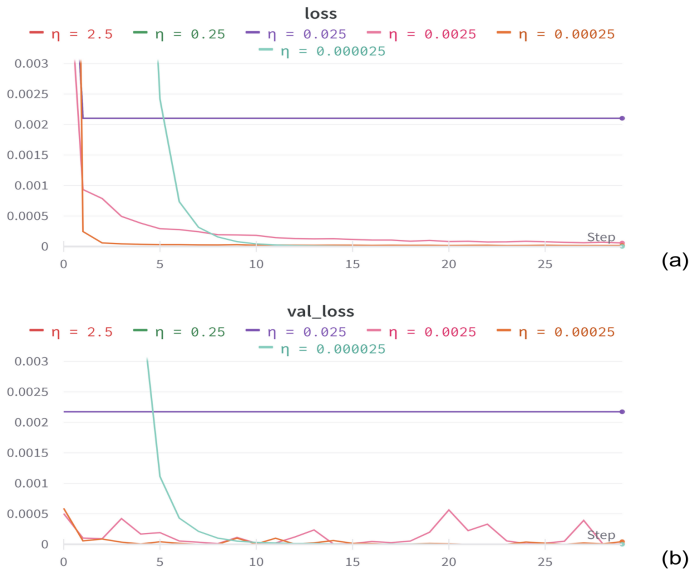



Figura 3.5: Gráfica de la función de costo vs. épocas del modelo para los diferentes valores de $\eta$. (a) Con respecto a los datos de entrenamiento. (b) Con respecto a los datos de prueba (validación). Gráficas obtenidas del software Weights & Biases

Como se mencionó anteriormente, se utilizó la métrica **mape** para medir la precisión del modelo, por lo que es relevante observar la evolución de este valor durante el entrenamiento. Aunque en la figura 3.5 se muestra una disminución considerable de la función de costo durante el entrenamiento, esto no es suficiente para afirmar qué valor de $\eta$ proporciona mejores resultados para el modelo. Esto se debe, en parte, a la magnitud de los datos de salida, puesto que al estar en un rango de $0.1 - 0.001$, una diferencia de 0.0005 entre las predicciones y los valores reales aún es significativa. Al graficar esta métrica contra las épocas para diferentes valores de la tasa de aprendizaje, se observa un comportamiento similar al obtenido para la función de costo, siendo nuevamente $\eta = 2.5 \times 10^{-5}$ el que presenta un mejor desempeño (ver figura 3.6).

Una vez identificado el valor $\eta = 2.5 \times 10^{-5}$ como el óptimo de la tasa de aprendizaje, se fija este hiperparámeto y se comienzan a hacer más pruebas ajustando los demás hiperparámetros, como el número de épocas, el tamaño del minibatch, el número de capas ocultas, la cantidad de neuronas en cada capa, etc. Después de esto, se comienzan a aplicar técnicas de regularización y optimización al modelo, con el fin de mejorar su precisión.

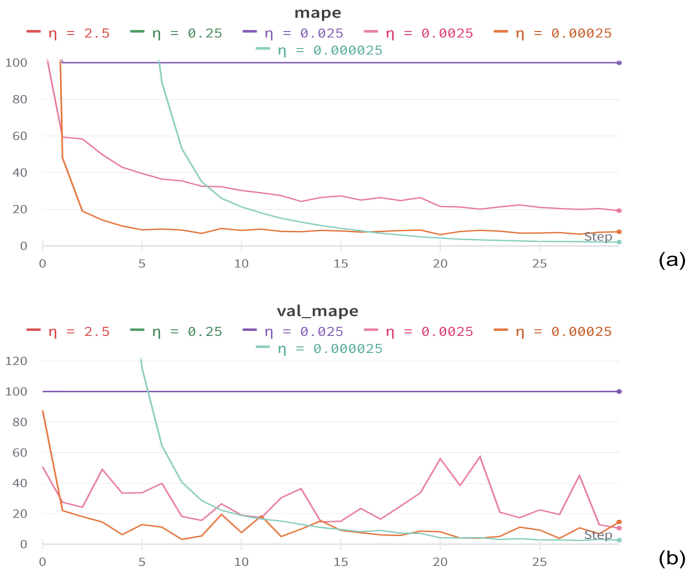



Figura 3.6: Gráfica del valor de mape vs. épocas del modelo para los diferentes valores de $\eta$. (a) Con respecto a los datos de entrenamiento. (b) Con respecto a los datos de prueba (validación). Gráficas obtenidas del software Weights & Biases.

### 3.3.2. Red Convolucional

Aunque, generalmente, las redes neuronales convolucionales (CNN) se utilizan solo para el procesamiento de imágenes, se decidió probar con este tipo de redes, ya que a diferencia de las redes neuronales densas, que procesan los datos de entrada de una forma secuencial y plana, las CNN los pueden procesar en 1D, 2D o 3D, lo que les permite capturar características espaciales y temporales en los datos de entrada (ver sección 2.8). Lo anterior, podría ser útil en nuestro análisis, debido a que nuestros datos de entrada no son más que series de tiempo. Además, como estamos trabajando con un modelo epidemiológico metapoblacional, la parte espacial también es relevante y al aplanar nuestros datos, parte de esta información se puede perder. Algunas de las ventajas que se esperan tener al usar este tipo de redes, en comparación con una densa son:

1. Captura de características locales: las redes convolucionales son especialmente útiles para identificar patrones o características locales en una secuencia. Al utilizar convoluciones, se pueden identificar patrones más fácilmente y con mayor precisión que en una red densa.

2. Menor cantidad de parámetros: las redes convolucionales utilizan menos parámetros que las redes densas, lo que hace que sean más eficientes computacionalmente y requieran menos datos para entrenarse.

3. Reducción del sobreajuste: la arquitectura de las redes convolucionales permite una mayor generalización y reducción del sobreajuste. Esto se debe a que las capas de convolución actúan como extractores de características, lo que permite una mejor generalización de los datos.

### Red Convolucional 1D

El primer tipo de CNN con que se decidió trabajar, fue con una convolucional de una dimensión (conv1D). Las redes conv1D realizan una convolución a lo largo de una secuencia unidimensional, como una señal de audio o una serie temporal. Por lo tanto, el tratamiento que se le dio a nuestros datos antes de alimentar a esta red, fue exactamente el mismo que el realizado en la sección 3.3.1.

La red convolucional seleccionada consta de una capa convolucional 1D con 10 filtros. La ventana de atención de la capa convolucional es de 1x10 y se aplica una operación de maxpooling de 2x1. La función de activación utilizada en esta capa es **relu**. A continuación, se añade una capa densa de 100 neuronas, también activadas por **relu**. La capa de salida mantiene la configuración de la red anterior (sección 3.3.1) . En la figura 3.7, se muestra el resumen de este modelo.

| | Name | Type | # Parameters | Output Shape |
|---|---|---|---|---|
| • | conv1d_input | InputLayer | 0 | ,200,1 |
| • | conv1d | Conv1D | 110 | None, 191, 10 |
| • | activation | Activation | 0 | None, 191, 10 |
| • | max_pooling1d | MaxPooling1D | 0 | None, 95, 10 |
| • | flatten | Flatten | 0 | None, 950 |
| • | dense | Dense | 95100 | None, 100 |
| • | activation_1 | Activation | 0 | None, 100 |
| • | dense_1 | Dense | 404 | None, 4 |
| • | activation_2 | Activation | 0 | None, 4 |

Figura 3.7: Resumen del modelo de la primera red convolucional 1D entrenada. Gráfica obtenida del software Weights & Biases.

En cuanto a la configuración del modelo, nuevamente se empleó la biblioteca **TensorFlow**. Se utilizaron los mismos hiperparámetros que con la red anterior: función de costo **mse**, métrica **mape** y optimizador **rmsprop**. Sin embargo, se seleccionó un valor inicial para la tasa de aprendizaje $\eta = 2.5 \times 10^{-5}$, ya que fue el que mostró mejores resultados en la sección anterior.

El entrenamiento de este modelo se llevó a cabo durante 30 épocas con un minibatch de 20, y se continuó utilizando el callback **Wandb** para almacenar los datos del entrenamiento. Después de finalizar el entrenamiento, se observó que el valor inicial de $\eta$ elegido resultó ser demasiado pequeño para este modelo. Por lo que se realizan más iteraciones, incrementando en un factor de dos el valor de $\eta$ en cada iteración, hasta encontrar el valor adecuado de $\eta$ para este modelo. En las figuras 3.8 y 3.9, se muestra el comportamiento de la función de costo y de la métrica mape, para los diferentes valores de $\eta$ utilizados.

Para este modelo, fue más complicado elegir un valor de $\eta$ óptimo, ya que aun con la información obtenida en las figuras 3.8 y 3.9, todas las curvas tienen un comportamiento similar, por lo que no es tan claro cual puede dar mejores resultados al entrenarla por más épocas. Debido a esto, se optó por elegir los dos valores de la tasa de aprendizaje que tuvieron un valor de la métrica mape más bajo, los cuales fueron $\eta_1 = 0.0001$ y $\eta_2 = 0.0002$. Una vez fijados estos dos valores, se comenzaron a hacer más pruebas, ajustando los demás hiperparámetros, tomando en cuenta que como estamos trabajando con un red convolucional, los más relevante son: el número de filtros, el tamaño de la ventana de atención y del maxpooling.

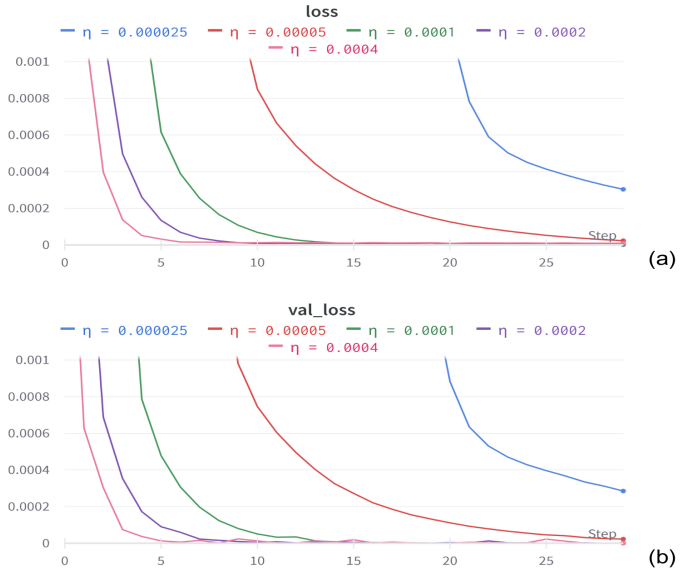



Figura 3.8: Gráfica de la función de costo vs. épocas del modelo para los diferentes valores de $\eta$. (a) Con respecto a los datos de entrenamiento. (b) Con respecto a los datos de prueba (validación). Gráficas obtenidas del software Weights & Biases

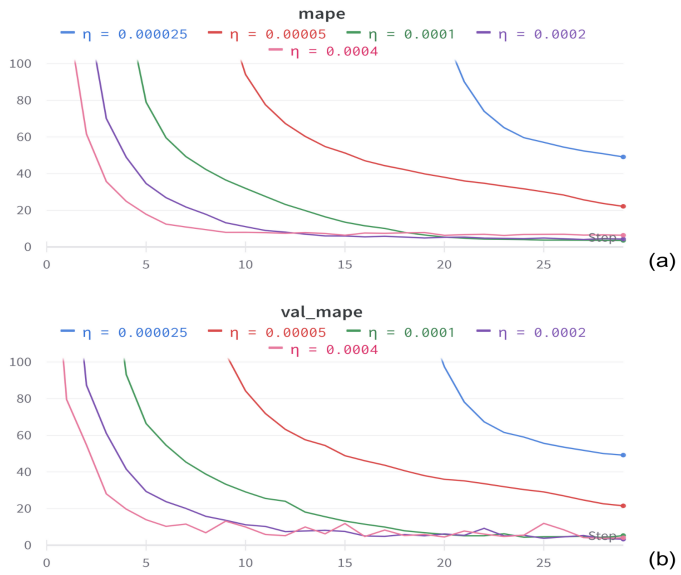



Figura 3.9: Gráfica del valor de mape vs. épocas del modelo para los diferentes valores de $\eta$. (a) Con respecto a los datos de entrenamiento. (b) Con respecto a los datos de prueba (validación). Gráficas obtenidas del software Weights & Biases.

### Red Convolucional 2D

El segundo y último tipo de CNN con el que se trabajó fue una convolucional de dos dimensiones (conv2D). Con este tipo de red, se busca conservar la información espacial de nuestros datos de entrada (de la metapoblación), ya que se reciben en forma de una matriz bidimensional. Por esta razón, esta vez los datos de entrada no fueron aplanados, conservando su forma orginal, es decir, se tiene una matriz de 2x100, donde cada columna representa la evolución de los individuos infectados en cada parche o subpoblación. El tratamiento de los datos fue el mismo que se realizó en los casos anteriores, excepto por el aplanado de los datos de entrada.

Para este modelo, se seleccionó una red convolucional compuesta de una capa convolucional de 2D con 10 filtros y una ventana de atención de 2x2, a la que se le aplica una función de activación **Relu** y un maxpooling de 2x1. Posteriormente, se añadió una capa densa de 100 neuronas, activadas nuevamente por **relu**. Por último, se incluyó la capa de salida, que fue la misma que se empleó en las redes descritas anteriormente. Un resumen visual de este modelo se muestra en la figura 3.10.

| Name | Type | # Parameters | Output Shape |
|---|---|---|---|
| conv2d_input | InputLayer | 0 | ,100,2,1 |
| conv2d | Conv2D | 50 | None, 99, 1, 10 |
| activation | Activation | 0 | None, 99, 1, 10 |
| max_pooling2d | MaxPooling2D | 0 | None, 49, 1, 10 |
| flatten | Flatten | 0 | None, 490 |
| dense | Dense | 49100 | None, 100 |
| activation_1 | Activation | 0 | None, 100 |
| dense_1 | Dense | 404 | None, 4 |
| activation_2 | Activation | 0 | None, 4 |

Figura 3.10: Resumen del modelo de la primera red convolucional 2D entrenada. Gráfica obtenida del software Weights & Biases.

En cuanto a la configuración del modelo, se mantuvo exactamente igual que la utilizada para la red conv1D, incluyendo los mismos hiperparámetros. Por lo tanto, se usó un valor inicial $\eta = 0.0001$, ya que fue uno de los valores que mejores resultados mostró con la red conv1D. Mantener esta configuración facilitó el entrenamiento de esta red, que se llevó a cabo de la misma manera descrita para la red conv1D. Al finalizar el entrenamiento, se observó que este valor de $\eta$ elegido también se ajustó muy bien a este modelo.

Antes de hacer más pruebas ajustando los demás hiperpárametros, se decidió probar con tasas de aprendizaje diferentes, para ver si se podía mejorar la precisión obtenida anteriormente. Los valores elegidos, fueron $\eta = 0.0002$ y $\eta = 0.0004$. Estas pruebas se hicieron aumentando a 50 el número de épocas, ya que después de haber entrenado la primera red, se observó que todavía era posible reducir considerablemente la función de costo. Además, debido a la disminución de parámetros en este modelo, el tiempo de entrenamiento fue muy corto, lo que permitió realizar más pruebas en menos tiempo. El comportamiento obtenido de la función de costo y de la métrica mape con estos valores de $\eta$, se muestra en la figura 3.11, donde se representa el comportamiento del modelo con respecto a los datos de prueba y de entrenamiento mediante líneas continuas y discontinuas, respectivamente.

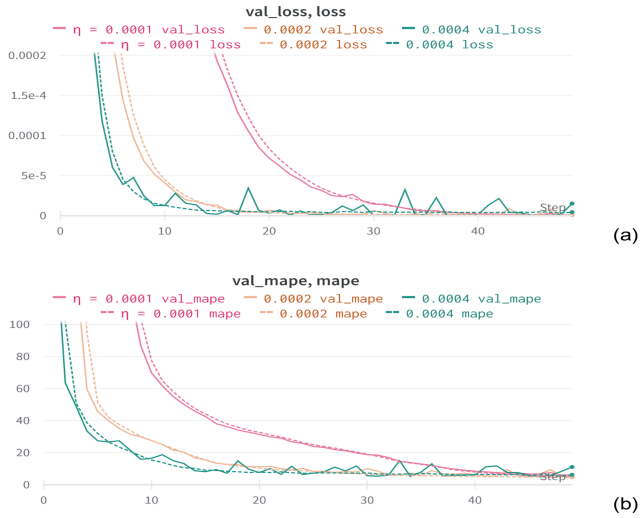



Figura 3.11: (a) Gráfica de la función de costo vs. épocas del modelo para los diferentes valores de $\eta$ con respecto a los datos de prueba (validación) y entrenamiento.(b) Gráfica del valor de mape vs. épocas del modelo para los diferentes valores de $\eta$ con respecto a los datos de prueba (validación) y entrenamiento. Gráficas obtenidas del software Weights & Biases.

Finalmente, se realizaron más pruebas variando los demás hiperparámetros, prestando especial atención a valores como, el número de filtros, el tamaño de la ventana de atención y del maxpooling.

### 3.3.3. Red Residual

Después de realizar varias pruebas con diferentes configuraciones de redes neuronales densas y convolucionales, se observaron las principales diferencias y similitudes entre ellas. Sin embargo, se encontró un problema común al intentar entrenar redes más profundas. Ambos tipos de redes mostraban un buen rendimiento con modelos simples, pero al probar con modelos más complejos, el aprendizaje comenzaba a estancarse. Esto se asocia principalmente con problemas como el gradiente inestable o la saturación de las neuronas. Para abordar este problema de estancamiento en el entrenamiento, se optó por utilizar redes neuronales residuales (ResNets, ver sección 2.9).

El problema radica en que hasta ahora se ha trabajado con redes secuenciales, donde la información fluye de manera secuencial de una capa a la siguiente. Cada capa toma como entrada la salida de la capa anterior y aprende nuevas representaciones a medida que se profundiza en la red. En contraste, en las ResNets, las conexiones residuales permiten que la información fluya directamente a través de saltos, evitando la necesidad de que cada capa aprenda todas las representaciones desde cero. Esto facilita el entrenamiento de redes más profundas y ayuda a prevenir el problema de degradación del rendimiento.

Por lo tanto, ahora se realizarán pruebas con redes densas y convolucionales más profundas utilizando modelos residuales. Al hacer uso de esta representación, se espera obtener algunas ventajas, como las siguientes:

1. Capacidad para entrenar redes más profundas: Las ResNets permiten entrenar redes mucho

más profundas sin enfrentar problemas de estancamiento en el entrenamiento.

2. Mejora en la precisión y eficiencia del entrenamiento: Al evitar la degradación del rendimiento, las ResNets pueden alcanzar una mayor precisión en tareas de aprendizaje profundo y pueden entrenarse de manera más eficiente con menos iteraciones o épocas.

3. Mayor capacidad de representación: Las ResNets permiten que las capas aprendan representaciones más ricas y complejas de los datos. Al agregar conexiones de salto, la red tiene la opción de utilizar la información original sin procesar en lugar de depender únicamente de las transformaciones aprendidas.

### Red Residual Densa

El primer tipo de ResNets con el que se trabajó fue una densa, ya que, como se mencionó anteriormente, es el tipo de red más simple, lo que facilitó su adaptación a esta nueva arquitectura de red neuronal. Además, el tratamiento de los datos fue el mismo que se realizó con la red densa secuencial (ver sección 3.3.1).

Se optó por utilizar una red residual compuesta por 6 capas ocultas, lo que significa que es tres veces más profunda que la red utilizada en la sección 3.3.1. Cada capa oculta está formada por 512 neuronas, todas activadas por la función **relu**. La capa de salida fue la misma utilizada en todas las configuraciones anteriores. En la figura 3.12, se muestra una representación visual de este modelo, donde se aprecia que la conexión residual se hace cada dos capas.

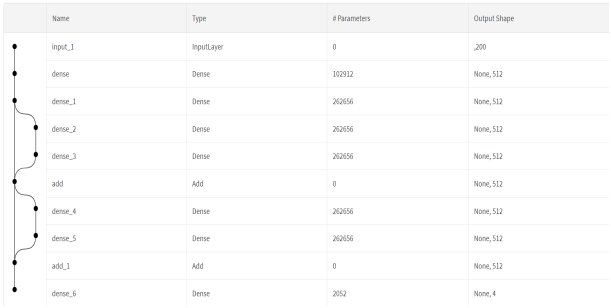



Figura 3.12: Resumen del modelo de la primera red residual densa entrenada. Gráfica obtenida del software Weights & Biases.

Dado que esta sigue siendo una red densa, se configuró con los mismos hiperparámetros que se utilizaron para entrenar la red densa descrita en la sección 3.3.1. La tasa de aprendizaje utilizada fue $\eta = 2.5 \times 10^{-5}$, que fue el valor óptimo observado para este tipo de redes. Esta configuración permitió nuevamente un entrenamiento más fácil de este tipo de red, que se entrenó durante un total de 50 épocas con un minibatch de 20.

Después de entrenar el modelo mostrado en la figura 3.12, se realizaron más pruebas en las que solo se varió el número de capas ocultas y la cantidad de neuronas en cada una de ellas. Los demás hiperparámetros se mantuvieron fijos, por lo que en esta ocasión no se realizaron pruebas con diferentes valores para $\eta$. Se llevaron a cabo un total de 15 pruebas, cada una con una configuración distinta. Todas se entrenaron durante 50 épocas utilizando un minibatch de 20, una vez más. Durante el entrenamiento de estas configuraciones, se observó que algunas de ellas comenzaban a sobreajustarse a los datos de entrenamiento, por lo que se aplicaron diferentes métodos de regularización, como Dropout, regularización L2 y BatchNormalization. En la tabla 3.2 se describen

las 6 más representativas configuraciones de las 15 pruebas que se realizaron. En las figuras 3.13 y 3.14 se muestra el comportamiento de la función de costo y la métrica mape para cada una de ellas.

| Nombre | Número de capas ocultas | Número de neuronas en las capas ocultas | Tipo de regularización empleada | Observaciones |
|---|---|---|---|---|
| Prueba 1 | 6 | 512, 800 y 1000 | L2 | Solo las capas 3 y 5 son de 800 y 1000 neuronas respectivamente. |
| Prueba 2 | 6 | 512, 800 y 1000 | Dropout | Se aplica un Dropout de 0.2 después de cada dos capas ocultas |
| Prueba 3 | 6 | 512, 800 y 1000 | Dropout | Solo se aplica un Dropout de 0.2 después de la sexta capa oculta |
| Prueba 4 | 6 | 512 | Dropout y BatchNormalization | El BatchNormalization es aplicado en cada una de las capas ocultas antes de ser activadas. El Dropout se aplica igual que en la prueba 3 |
| Prueba 5 | 10 | 512 | Sin regularización | Ninguna |
| Prueba 6 | 4 | 512 | Sin regularización | Ninguna |

Tabla 3.2: Descripción de las 6 más representativas configuraciones se hicieron usando redes neuronales residuales densas.

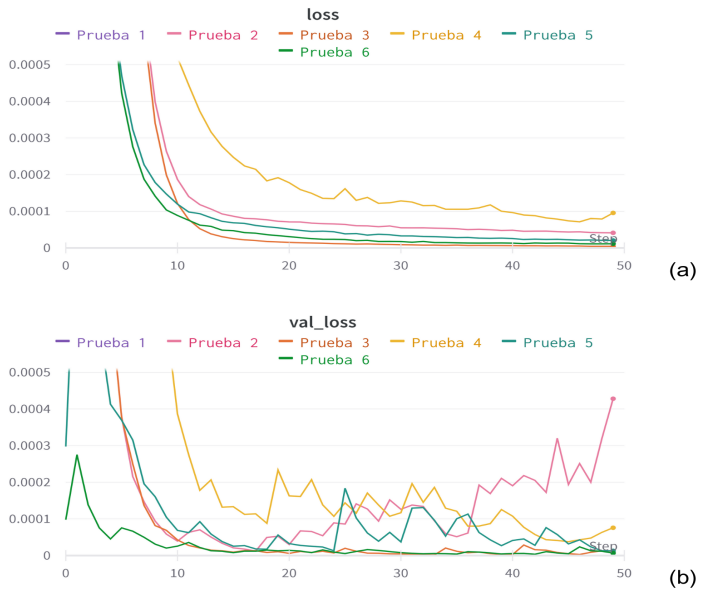



Figura 3.13: Gráfica de la función de costo vs. épocas del modelo para las diferentes pruebas realizadas. (a) Con respecto a los datos de entrenamiento. (b) Con respecto a los datos de prueba (validación). Gráficas obtenidas del software Weights & Biases

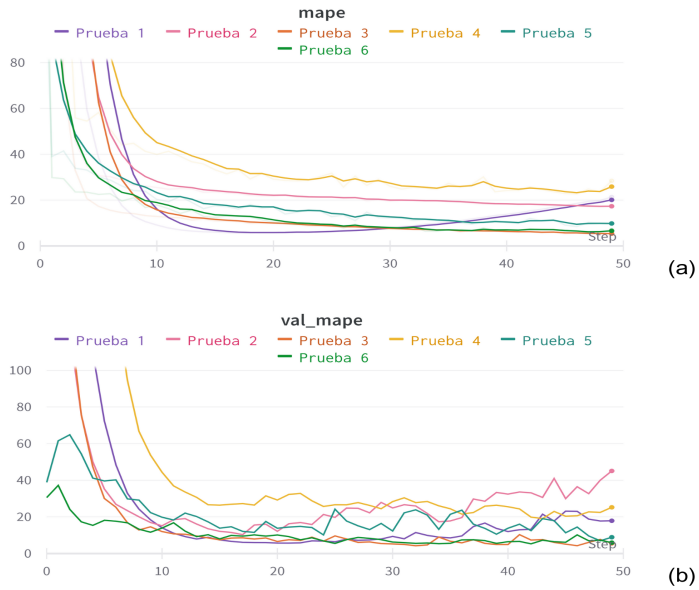



Figura 3.14: Gráfica del valor de mape vs. épocas del modelo para las diferentes pruebas realizadas.(a) Con respecto a los datos de entrenamiento. (b) Con respecto a los datos de prueba (validación). Gráficas obtenidas del software Weights & Biases.

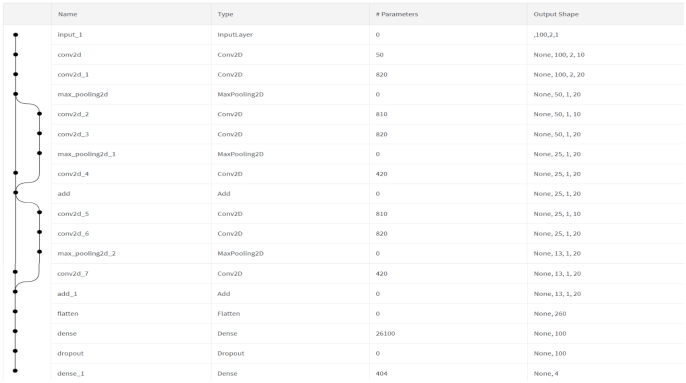
### Red Residual Conv2D

Finalmente, el último tipo de red neuronal con el que se trabajó fue una Red Residual Convolucional 2D. Este modelo se dejó para el final debido a su mayor complejidad en comparación con los modelos anteriores. La transición de una red secuencial convolucional 2D (sección 3.3.2) a una residual no fue una tarea trivial, ya que las redes residuales en general tiene una arquitectura más compleja, debido, principalmente, a sus conexiones residuales. Sin embargo, la red convolucional 2D vista en la sección 3.3.2 sirvió como base para este nuevo modelo, por lo que el tratamiento de los datos fue el mismo que se realizó para ese tipo de red.

La red elegida consta de un total de 9 capas ocultas, 8 convolucionales de 2D y una densa. Al usar conexiones residuales, es común que el tamaño de las salidas de las capas de convolución no coincida con el tamaño de las entradas. Por esta razón, es importante ahora considerar el parámetro **padding** en las capas convolucionales. Este parámetro determina cómo se maneja el borde de los filtros durante la convolución. Por lo tanto, para mantener las dimensiones entre las capas y asegurar la compatibilidad entre ellas, se usó un **padding - same** en cada capa convolucional.

Las ocho capas ocultas convolucionales, están distribuidas en 3 bloques conformados de dos capas cada uno, una de 10 y otra de 20 filtros, ambas capas con una ventana de atención de 2x2 y activadas por la función **relu**. Al final de cada bloque, se aplica un maxpooling de 2x2. Debido a esto, en el segundo y tercer bloque se agrega otra capa convolucional de 20 filtros con una ventana de atención de 1x1, cuya función es solo reescalar las capas después de aplicar el maxpooling para que vuelvan a coincidir y se pueda aplicar la conexión residual, la cual se realiza después de cada bloque. Al terminar estos bloques, se incluye la capa densa compuesta de 100 neuronas activadas por **relu**, y finalmente se añade la capa de salida que, fue la misma que se ha utilizado anteriormente. En la figura 3.15, se muestra la estructura de este modelo, en la cual se aprecia que se está aplicando un Dropout justo antes de la capa de salida, a pesar de ser la primera prueba. Esto es debido a que en

este tipo de redes es muy común el sobreajuste.



| Name | Type | # Parameters | Output Shape |
|---|---|---|---|
| input_1 | InputLayer | 0 | ,100,2,1 |
| conv2d | Conv2D | 50 | None, 100, 2, 10 |
| conv2d_1 | Conv2D | 820 | None, 100, 2, 20 |
| max_pooling2d | MaxPooling2D | 0 | None, 50, 1, 20 |
| conv2d_2 | Conv2D | 810 | None, 50, 1, 10 |
| conv2d_3 | Conv2D | 820 | None, 50, 1, 20 |
| max_pooling2d_1 | MaxPooling2D | 0 | None, 25, 1, 20 |
| conv2d_4 | Conv2D | 420 | None, 25, 1, 20 |
| add | Add | 0 | None, 25, 1, 20 |
| conv2d_5 | Conv2D | 810 | None, 25, 1, 10 |
| conv2d_6 | Conv2D | 820 | None, 25, 1, 20 |
| max_pooling2d_2 | MaxPooling2D | 0 | None, 13, 1, 20 |
| conv2d_7 | Conv2D | 420 | None, 13, 1, 20 |
| add_1 | Add | 0 | None, 13, 1, 20 |
| flatten | Flatten | 0 | None, 260 |
| dense | Dense | 26100 | None, 100 |
| dropout | Dropout | 0 | None, 100 |
| dense_1 | Dense | 404 | None, 4 |

Figura 3.15: Resumen del modelo de la primera red residual convolucional entrenada. Gráfica obtenida del software Weights & Biases.

En cuanto a la configuración de la red, se utilizaron los mismo hiperparámetros que se emplearon para entrenar la red convolucional 2D descrita en la sección 3.3.2, ya que, como se mencionó, esta sirvió como base para este modelo. Respecto a la tasa de aprendizaje, como en el caso de la red entrenada en la sección 3.3.2, no fue tan clara la elección de un valor óptimo, por lo que se llevaron a cabo diferentes pruebas utilizando distintos valores de $\eta$ que mostraron un mejor comportamiento con las diferentes redes previas. Los valores seleccionados fueron: $\eta_1 = 0.0001$, $\eta_2 = 0.0002$, $\eta_3 = 0.0005$, $\eta_4 = 2.5 \times 10^{-5}$, $\eta_5 = 5 \times 10^{-5}$. Se realizó una prueba con cada uno de estos valores, las cuales se entrenaron durante 50 épocas con un minibatch de 20. En las figuras 3.16 y 3.17, se muestra el comportamiento de la función de costo y de la métrica mape, para los diferentes valores de $\eta$ elegidos.

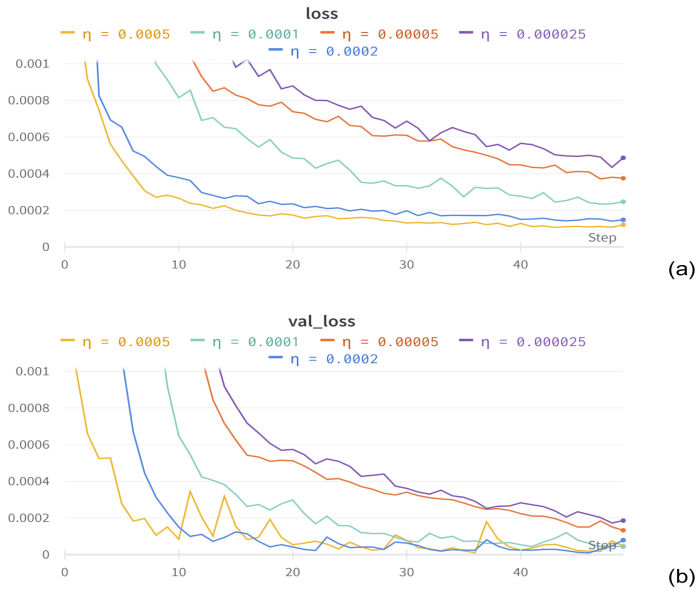



Figura 3.16: Gráfica de la función de costo vs. épocas del modelo para los diferentes valores de $\eta$. (a) Con respecto a los datos de entrenamiento. (b) Con respecto a los datos de prueba (validación). Gráficas obtenidas del software Weights & Biases

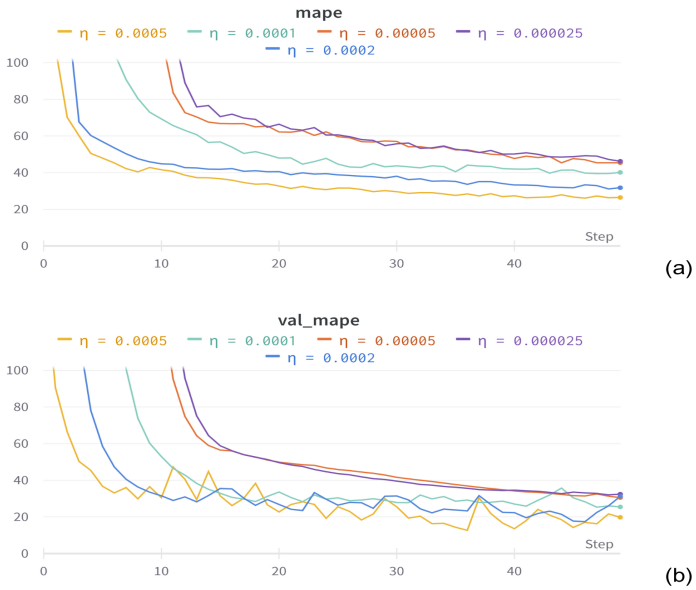



Figura 3.17: Gráfica del valor de mape vs. épocas del modelo para los diferentes valores de $\eta$.(a) Con respecto a los datos de entrenamiento. (b) Con respecto a los datos de prueba (validación). Gráficas obtenidas del software Weights & Biases.

Una vez más, la elección de un valor óptimo para la tasa de aprendizaje, a partir de la información obtenida en las figuras 3.16 y 3.17, no resulta evidente. Todas las curvas exhiben un comportamiento similar y alcanzan casi la misma presición, es decir, el valor final de la métrica mape al que cada una llega se encuentra dentro de un mismo intervalo. Por otro lado, debido a la gran variedad de configuraciones que se pueden entrenar al trabajar con redes residuales, donde es posible jugar con parámetros como la profundida y el número de canales o filtros por capa, sin enfrentar problemas de estancamiento en el entrenamiento, se decidió realizar más pruebas con cada uno de los 5 valores de $\eta$ elegidos. Durante estas pruebas, también se fueron variando los demás hiperparámetros, principalmente la profundidad de la red, el número de filtros en las capas convolucionales y el número de neuronas en la capa densa. Cabe resaltar que en todas las pruebas realizadas se utilizó un Dropout de 0.2 justo antes de la capa de salida, tal como se mostró en la figura 3.15. Se optó por el Dropout, ya que fue el método de regularización que mejores resultados mostró con la red anterior.

# Capítulo 4

# Análisis y Resultados

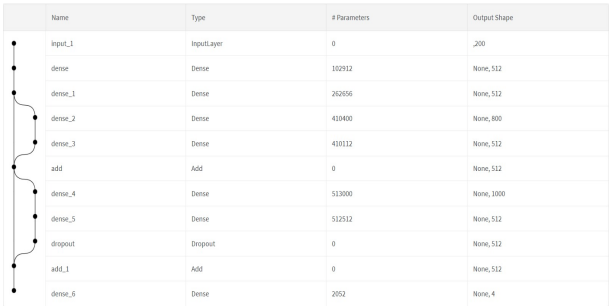
Este capítulo aborda el análisis llevado a cabo con cada tipo de red neuronal entrenada en el capítulo anterior. Se describe la configuración de cada tipo de RNA que obtuvo mejores resultados, es decir, que describió mejor la base de datos realizada en la sección 3.2. Además, se realiza la validación de cada uno de estos modelos y se comparan los resultados obtenidos. Por último, se selecciona el modelo que no solo describe mejor nuestra base de datos, sino que también la generaliza.

## 4.1. Comparación de los mejores modelos elegidos

Después de realizar múltiples pruebas con diferentes configuraciones de cada una de las redes descritas en la sección 3.3, se seleccionó una por cada tipo de red, considerando una para cada tipo de red convolucional (Redes Conv1D y Conv2D) y una para cada tipo de red residual (Redes residuales Densa y Conv2D), lo que suma un total de 5 modelos diferentes. La configuración elegida para cada tipo de red fue aquella que obtuvo el valor más bajo de la métrica mape en los datos de prueba. En la tabla 4.1 se enumeran estos modelos, proporcionando una breve descripción de su configuración y los resultados obtenidos. Es importante mencionar que cada uno de estos modelos se entrenó durante un total de 50 épocas con un minibatch de 20.

| Tipo de RNA | Tasa de aprendizaje | mse | mape | Resumen del modelo |
|---|---|---|---|---|
| Red Densa | $2.5 \times 10^{-5}$ | $3.472 \times 10^{-7}$ | 1.595 | Figura 3.4 |
| Red Conv1D | $2 \times 10^{-4}$ | $4.80 \times 10^{-6}$ | 4.595 | Figura 3.7 |
| Red Conv2D | $2 \times 10^{-4}$ | $3.899 \times 10^{-7}$ | 4.553 | Figura 3.10 |
| ResNet Conv2D | 0.0005 | $3.90 \times 10^{-5}$ | 12.748 | Figura 3.15 |
| ResNet Densa | $2.5 \times 10^{-5}$ | $9.356 \times 10^{-7}$ | 3.858 | Figura 4.1 |

Tabla 4.1: Descripción de la mejor configuración obtenida para cada una de las diferentes RNA seleccionadas. El valor mostrado de mse y mape es con respecto a los datos de prueba.



| | Name | Type | # Parameters | Output Shape |
|---|---|---|---|---|
| | input_1 | InputLayer | 0 | ,200 |
| | dense | Dense | 102912 | None, 512 |
| | dense_1 | Dense | 262656 | None, 512 |
| | dense_2 | Dense | 410400 | None, 800 |
| | dense_3 | Dense | 410112 | None, 512 |
| | add | Add | 0 | None, 512 |
| | dense_4 | Dense | 513000 | None, 1000 |
| | dense_5 | Dense | 512512 | None, 512 |
| | dropout | Dropout | 0 | None, 512 |
| | add_1 | Add | 0 | None, 512 |
| | dense_6 | Dense | 2052 | None, 4 |

Figura 4.1: Resumen del modelo de la mejor ResNet Densa entrenada. Gráfica obtenida del software Weights & Biases.

Para finalizar esta breve comparación entre los mejores modelos seleccionados para cada tipo de red neuronal utilizada, se consideró importante visualizar el rendimiento de cada una de estas configuraciones durante el entrenamiento. Con este fin, en la figura 4.2 se muestra el comportamiento de la métrica mape con respecto a los datos de prueba para cada uno de los modelos presentados en la tabla 4.1. En la gráfica, se observa que solo la ResNet Conv2D tiene un comportamiento ligeramente diferente al resto, mientras que los modelos restantes muestran un comportamiento bastante similar entre sí. Además, se puede apreciar que el valor final de mape al que llegan se encuentra dentro de un mismo intervalo, lo que sugiere que todavía es posible mejorar dicho valor, quizás al entrenar durante más épocas.

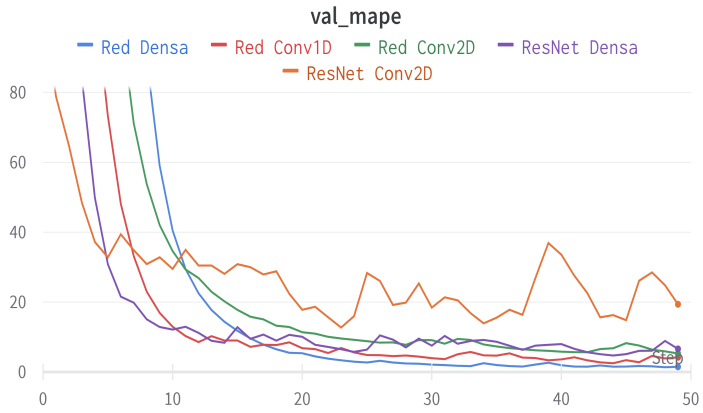



Figura 4.2: Gráfica del valor de mape vs. épocas del modelo con respecto a los datos de prueba para cada uno de los 5 modelos elegidos. Gráfica obtenida del software Weights & Biases.

## 4.2. Validación de los modelos elegidos

Una vez terminado el análisis y la comparación de la estructura y el comportamiento de los 5 modelos mostrados en la tabla 4.1, aún no es posible determinar de manera definitiva cuál es el mejor entre ellos. Una opción podría ser seleccionar el modelo que alcanzó el valor más bajo de la métrica mape durante el entrenamiento, tal como se hizo para elegir cada uno de ellos. Sin embargo, este criterio no es suficiente para afirmar que tendrá un mejor rendimiento en la práctica al

enfrentarse con datos completamente nuevos, es decir, generalice mejor. Para abordar esta situación, se procedió a realizar la validación de cada uno de ellos.

La validación se llevó a cabo en tres etapas importantes. En primer lugar, se evaluaron individualmente cada uno de los modelos seleccionados utilizando un subconjunto de los datos de prueba. A continuación, se utilizó un conjunto de datos dentro del mismo rango de valores que los datos de prueba, pero que no formaron parte de estos, con el objetivo de determinar si los modelos seleccionados eran capaces de realizar predicciones precisas dentro de ese rango (intrapolación). Por último, se empleó un conjunto de datos fuera de ese rango para evaluar si los modelos podían predecir o estimar valores más allá del rango de los datos con los que fueron entrenados (extrapolación).

A continuación, se describe cada una de estas etapas.

### 4.2.1. Validación con datos de prueba

El conjunto de datos empleado para esta validación, está conformado por solo cinco instancias del conjunto de prueba. Estas instancias ya son conocidas por cada uno de los modelos, por lo que se espera obtener resultados al menos similares a los reportados en la tabla 4.1. Las cinco instancias elegidas fueron:

- $\beta_1 = \begin{bmatrix} 0.003 & 0.002 \\ 0.003 & 0.004 \end{bmatrix}$
- $\beta_2 = \begin{bmatrix} 0.091 & 0.046 \\ 0.047 & 0.092 \end{bmatrix}$
- $\beta_3 = \begin{bmatrix} 0.059 & 0.03 \\ 0.031 & 0.06 \end{bmatrix}$
- $\beta_4 = \begin{bmatrix} 0.0095 & 0.00525 \\ 0.00625 & 0.0105 \end{bmatrix}$
- $\beta_5 = \begin{bmatrix} 0.0084 & 0.0047 \\ 0.0057 & 0.0094 \end{bmatrix}$

Las curvas de infección asociadas a cada una de estas matrices, también fueron tomadas de los datos de prueba. Dichas curvas, tuvieron que ser normalizadas utilizando la función **StandardScaler**, tal como se describió en la sección 3.3, para asegurar que tuvieran la misma escala que los datos de entrada utilizados para entrenar cada uno de los 5 modelos elegidos y evitar problemas de escala durante la validación.

Una vez listo este subconjunto de los datos de prueba, se procedió a evaluar cada uno de los modelos descritos en la tabla 4.1. En las tablas 4.2 y 4.3, se muestra el valor la métrica mape a la que llegó cada modelo durante la validación y las predicciones que obtuvieron.

| Tipo de RNA | Valor de mape obtenido en el entrenamiento | Valor de mape obtenido en la 1ra validación |
|---|---|---|
| Red Densa | **1.595** | **1.361** |
| Red Conv1D | 4.595 | 5.084 |
| Red Conv2D | 4.553 | 15.254 |
| ResNet Conv2D | 12.748 | 11.254 |
| ResNet Densa | 3.858 | 3.258 |

Tabla 4.2: Comparación de los valores de mape obtenidos en el entrenamiento y la validación realizada con cinco elemento del conjunto de prueba utilizados durante el entrenamiento de cada modelo..

| **Valor real** | **Estimación Red Densa** | **Estimación Red Conv1D** |
|---|---|---|
| [0.003 0.002<br>0.003 0.004] | [0.00297 0.002<br>0.00298 0.00408] | [0.0032 0.00165<br>0.00288 0.00446] |
| [0.091 0.046<br>0.047 0.092] | [0.09174 0.04619<br>0.04727 0.093] | [0.08686 0.04454<br>0.04539 0.08631] |
| [0.059 0.03<br>0.031 0.06] | [0.05987 0.0302<br>0.03135 0.06109] | [0.05684 0.02916<br>0.02994 0.05666] |
| [0.0095 0.00525<br>0.00625 0.0105] | [0.00951 0.00496<br>0.00626 0.01081] | [0.00989 0.00509<br>0.00626 0.01124] |
| [0.0084 0.0047<br>0.0057 0.0094] | [0.0084 0.0045<br>0.0056 0.00945] | [0.00847 0.00424<br>0.00575 0.00972] |
| **Estimación Red Conv2D** | **Estimación ResNet Conv2D** | **Estimación ResNet Densa** |
| [0.00076 0.00045<br>0.0015 0.0027] | [0.00288 0.00135<br>0.00151 0.00436] | [0.00291 0.00173<br>0.00314 0.00371] |
| [0.09122 0.04599<br>0.04697 0.09231] | [0.10254 0.05291<br>0.05384 0.10324] | [0.09248 0.04746<br>0.04840 0.09209] |
| [0.05937 0.03027<br>0.03117 0.06057] | [0.06607 0.03242<br>0.0337 0.06735] | [0.0601 0.03094<br>0.03199 0.0601] |
| [0.00956 0.00523<br>0.00651 0.0107] | [0.00988 0.00539<br>0.00599 0.01102] | [0.00955 0.00532<br>0.00637 0.00995] |
| [0.00729 0.00374<br>0.00486 0.00832] | [0.00881 0.00492<br>0.00548 0.00989] | [0.00831 0.00458<br>0.00564 0.00871] |

Tabla 4.3: Descripción de las estimaciones obtenidas de la validación realizada con cinco elemento del conjunto de prueba utilizados durante el entrenamiento de cada modelo. Los elementos de cada matriz fueron redondeados a 5 cífras.

Por último, para tener una idea más visual de las predicciones realizadas por los modelos, se calcularon las curvas de infección correspondientes a la predicción hecha por cada uno de ellos para la matriz de infección $\beta_1$ y se graficaron junto con las curvas de infección correspondientes al valor real de $\beta_1$ para cada subpoblación (ver figura 4.3). La matriz de infección $\beta_1$ se eligió de forma totalmente arbitraria en comparación con las demás.

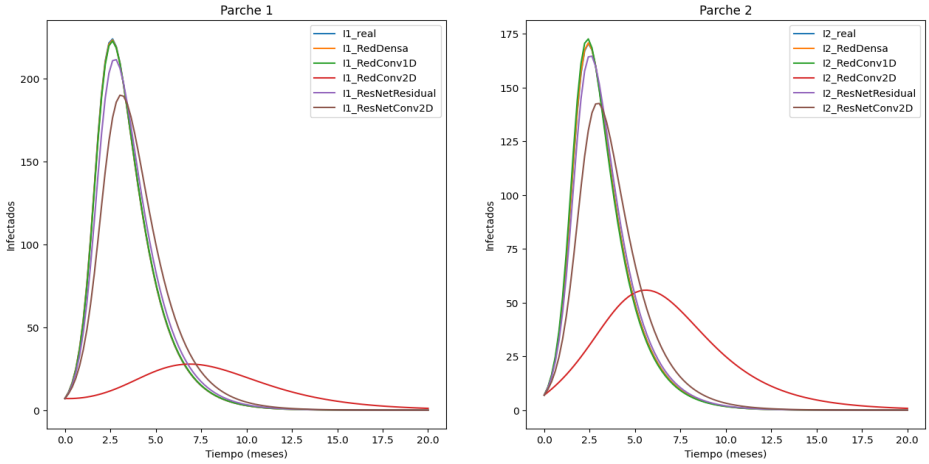



Figura 4.3: Gráficas de las predicciones hechas por cada modelo elegido vs el valor real. Tomando como $\beta_1$ el valor esperado (real).

Los resultados mostrados en las tablas 4.2 y 4.3 son muy cercanos a los que cada modelo obtuvo al finalizar el entrenamiento. Todas las estimaciones de las tasas de infección están dentro de un rango aceptable en comparación con el valor esperado, y además concuerdan con el valor de la métrica mape que cada modelo obtuvo. Las curvas graficadas en la figura 4.3 respaldan esta información, ya que todas las curvas de infección generadas a partir de las predicciones mostradas en la tabla 4.3 tienden a ajustarse al comportamiento esperado. La Red Densa y la Red Conv1D son las que mejor se ajustan a las curvas esperadas. Para todas las tasas de infección restantes, las predicciones fueron igualmente buenas y, en algunos casos, mejores. Esto confirma que los cinco modelos elegidos describen adecuadamente los datos de prueba.

### 4.2.2. Validación con datos en el dominio de entrenamiento

Para conseguir un conjunto de datos tal que las matrices que lo conformen no formen parte del conjunto de prueba original, pero se encuentren dentro del mismo rango de valores, a cada una de las entradas de las matrices de infección utilizadas en la evaluación anterior, se le sumó un valor de $\Delta\beta = 0.0005$. De esta manera, el conjunto de datos resultante se considera completamente nuevo para los cinco modelos. A continuación se muestran las cinco matrices de infección resultantes:

- $\beta_1' = \begin{bmatrix} 0.0035 & 0.0025 \\ 0.0035 & 0.0045 \end{bmatrix}$
- $\beta_2' = \begin{bmatrix} 0.0915 & 0.0465 \\ 0.0475 & 0.0925 \end{bmatrix}$
- $\beta_3' = \begin{bmatrix} 0.0595 & 0.0305 \\ 0.0315 & 0.0605 \end{bmatrix}$
- $\beta_4' = \begin{bmatrix} 0.001 & 0.0053 \\ 0.0063 & 0.011 \end{bmatrix}$

- $\beta_5' = \begin{bmatrix} 0.0089 & 0.0052 \\ 0.0062 & 0.0099 \end{bmatrix}$

Las curvas de infección asociadas a cada una de estas matrices se obtuvieron utilizando el objeto **Class SIR Metapoblacional**, de la misma manera que se hizo en la sección 3.2. Luego, se normalizaron del mismo modo como se hizo en la validación anterior para después evaluar nuevamente cada uno de los modelos seleccionados utilizando este nuevo conjunto de datos. En la tabla 4.4, se muestran los resultados obtenidos para las predicciones de las matrices de infección. Con los valores obtenidos, se procedió a realizar nuevamente la gráfica de las curvas de infección predichas junto con las curvas esperadas para cada subpoblación. En esta ocasión, se tomó como referencia la matriz de infección $\beta_5'$, la cual, nuevamente se eligió de forma totalmente arbitraria en comparación con las demás (ver figuras 4.4 y 4.5).

| **Valor real** | **Estimación Red Densa** | **Estimación Red Conv1D** |
|---|---|---|
| $\begin{bmatrix} 0.0035 & 0.0025 \\ 0.0035 & 0.0045 \end{bmatrix}$ | $\begin{bmatrix} 0.00369 & 0.00217 \\ 0.00323 & 0.00467 \end{bmatrix}$ | $\begin{bmatrix} 0.00428 & 0.00254 \\ 0.00347 & 0.0055 \end{bmatrix}$ |
| $\begin{bmatrix} 0.0915 & 0.0465 \\ 0.0475 & 0.0925 \end{bmatrix}$ | $\begin{bmatrix} 0.09235 & 0.04648 \\ 0.04757 & 0.09361 \end{bmatrix}$ | $\begin{bmatrix} 0.08741 & 0.04483 \\ 0.04568 & 0.08685 \end{bmatrix}$ |
| $\begin{bmatrix} 0.0595 & 0.0305 \\ 0.0315 & 0.0605 \end{bmatrix}$ | $\begin{bmatrix} 0.06054 & 0.03055 \\ 0.03169 & 0.06177 \end{bmatrix}$ | $\begin{bmatrix} 0.0575 & 0.0295 \\ 0.03028 & 0.05731 \end{bmatrix}$ |
| $\begin{bmatrix} 0.001 & 0.0053 \\ 0.0063 & 0.011 \end{bmatrix}$ | $\begin{bmatrix} 0.01239 & 0.00702 \\ 0.00981 & 0.01398 \end{bmatrix}$ | $\begin{bmatrix} 0.00648 & 0.00425 \\ 0.00502 & 0.00838 \end{bmatrix}$ |
| $\begin{bmatrix} 0.0089 & 0.0052 \\ 0.0062 & 0.0099 \end{bmatrix}$ | $\begin{bmatrix} 0.00911 & 0.00477 \\ 0.00607 & 0.0104 \end{bmatrix}$ | $\begin{bmatrix} 0.00931 & 0.00474 \\ 0.00601 & 0.01057 \end{bmatrix}$ |
| **Estimación Red Conv2D** | **Estimación ResNet Conv2D** | **Estimación ResNet Densa** |
| $\begin{bmatrix} 0.00281 & 0.00204 \\ 0.0033 & 0.00518 \end{bmatrix}$ | $\begin{bmatrix} 0.00387 & 0.00233 \\ 0.00337 & 0.00426 \end{bmatrix}$ | $\begin{bmatrix} 0.00474 & 0.00276 \\ 0.00274 & 0.00636 \end{bmatrix}$ |
| $\begin{bmatrix} 0.09182 & 0.04628 \\ 0.04725 & 0.0929 \end{bmatrix}$ | $\begin{bmatrix} 0.09303 & 0.04776 \\ 0.04868 & 0.09267 \end{bmatrix}$ | $\begin{bmatrix} 0.10306 & 0.05321 \\ 0.05413 & 0.10374 \end{bmatrix}$ |
| $\begin{bmatrix} 0.06003 & 0.0306 \\ 0.0315 & 0.06124 \end{bmatrix}$ | $\begin{bmatrix} 0.06078 & 0.03128 \\ 0.03234 & 0.06077 \end{bmatrix}$ | $\begin{bmatrix} 0.06707 & 0.03296 \\ 0.03424 & 0.06833 \end{bmatrix}$ |
| $\begin{bmatrix} 0.00709 & 0.00519 \\ 0.00707 & 0.00845 \end{bmatrix}$ | $\begin{bmatrix} 0.01168 & 0.00475 \\ 0.01117 & 0.00932 \end{bmatrix}$ | $\begin{bmatrix} 0.00742 & 0.00397 \\ 0.00453 & 0.00829 \end{bmatrix}$ |
| $\begin{bmatrix} 0.00862 & 0.00459 \\ 0.00582 & 0.00977 \end{bmatrix}$ | $\begin{bmatrix} 0.00909 & 0.00505 \\ 0.00609 & 0.00944 \end{bmatrix}$ | $\begin{bmatrix} 0.00939 & 0.00518 \\ 0.00576 & 0.0105 \end{bmatrix}$ |

Tabla 4.4: Descripción de las estimaciones obtenidas de la segunda validación realizada. Los elementos de cada matriz fueron redondeados a 5 cífras.

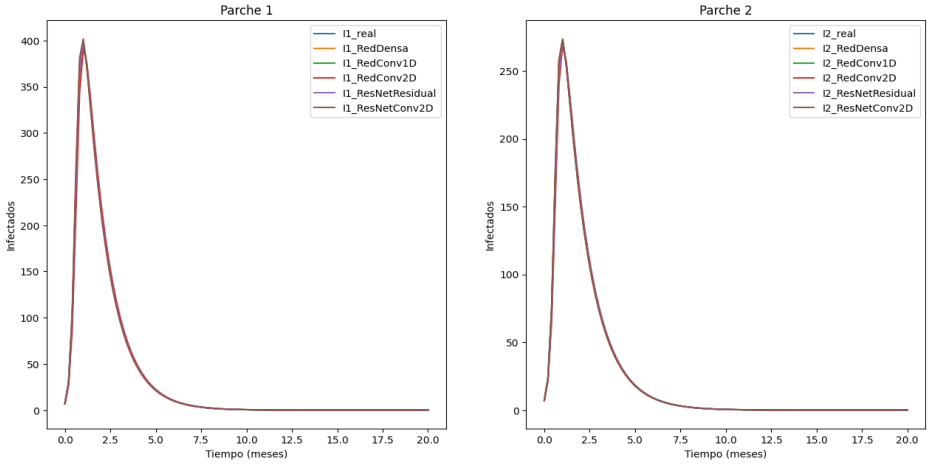



Figura 4.4: Gráficas de las predicciones hechas por cada modelo elegido vs el valor real. Tomando como $\beta_5'$ el valor esperado (real).

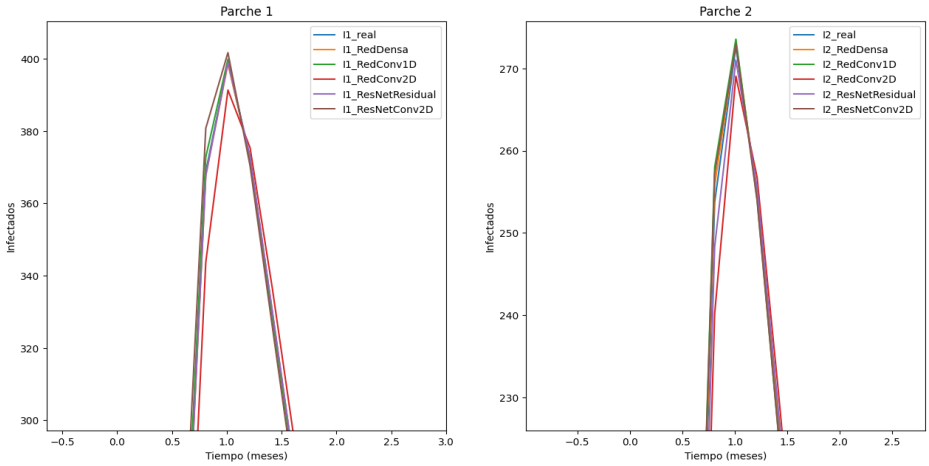



Figura 4.5: Un acercamiento a las gráficas de las predicciones hechas por cada modelo elegido vs el valor real. Tomando como $\beta_5'$ el valor esperado (real).

En la tabla 4.5, se compararon los valores de la métrica mape a los que cada modelo llegó con respecto al conjunto de datos utilizado en la sección anterior y este nuevo conjunto. En dicha tabla, se aprecia una diferencia considerable entre las estimaciones hechas con este nuevo conjunto de datos y las obtenidas utilizando elementos del conjunto de prueba original. Sin embargo, las curvas de infección representadas en las figuras 4.4 y 4.5 brindan suficientes razones para afirmar que, a pesar de que las estimaciones de las matrices de infección no se encuentran dentro del rango esperado, son capaces de describir el comportamiento de las curvas de infección esperadas, con un margen de error bajo.

| Tipo de RNA | Valor de mape obtenido en la 1ra validación | Valor de mape obtenido en la 2da validación |
|---|---|---|
| Red Densa | **1.361** | 65.480 |
| Red Conv1D | 5.084 | **35.786** |
| Red Conv2D | 15.254 | 36.581 |
| ResNet Densa | 3.258 | 61.208 |
| ResNet Conv2D | 11.254 | 47.173 |

Tabla 4.5: Comparación de los valores de mape obtenidos en las dos validaciones anteriores.

### 4.2.3. Validación con datos fuera del dominio de entrenamiento

Para generar un conjunto de datos fuera del dominio de entrenamiento se decidió partir de la matriz de infección utilizada para simulación matemática del modelo SIR Metapoblacional realizada en la sección 3.1 (ver tabla 3.1). Es importante destacar que dicha matriz no forma parte de la base de datos realizada en la sección 3.2. Las demás matrices que conformaron este nuevo conjunto de datos se obtuvieron mediante rotaciones cíclicas de las entradas $\beta_{ij}$ de esta matriz, a excepción de la última, donde solo se incrementó en 0.001 cada una de sus entradas. A continuación se presentan las 5 matrices resultantes:

- $\beta_1'' = \begin{bmatrix} 0.001 & 0.001 \\ 0.002 & 0.002 \end{bmatrix}$
- $\beta_2'' = \begin{bmatrix} 0.002 & 0.001 \\ 0.001 & 0.002 \end{bmatrix}$
- $\beta_3'' = \begin{bmatrix} 0.002 & 0.002 \\ 0.001 & 0.001 \end{bmatrix}$
- $\beta_4'' = \begin{bmatrix} 0.001 & 0.002 \\ 0.002 & 0.001 \end{bmatrix}$
- $\beta_5'' = \begin{bmatrix} 0.002 & 0.002 \\ 0.003 & 0.003 \end{bmatrix}$

Se utilizó la misma metodología empleada en la evaluación anterior para evaluar cada uno de los modelos seleccionados y para generar las gráficas de las curvas de infección predichas. En la tabla y figura 4.6 se presentan los resultados obtenidos. Es importante mencionar, que esta vez, las predicciones se hicieron tomando como referencia a la matriz de infección $\beta_5''$ debido a que fue la que obtuvo la predicciones más razonables según los resultados mostrados en la tabla 4.6.

| **Valor real** | **Estimación Red Densa** | **Estimación Red Conv1D** |
|---|---|---|
| 0.001 0.001<br>0.002 0.002 | 0.00101 0.00002<br>0.00034 0.00067 | 0. 0.00022<br>0.00019 0. |
| 0.002 0.001<br>0.001 0.002 | 0.00194 0.00058<br>0.01638 0.0151 | 0.00001 0.00046<br>0.00055 0.00005 |
| 0.002 0.002<br>0.001 0.001 | 0.00629 0.00145<br>0.07195 0.03181 | 0.00009 0.00343<br>0.0019 0.00177 |
| 0.001 0.002<br>0.002 0.001 | 0.00142 0.00041<br>0.00455 0.0066 | 0.00001 0.00029<br>0.00039 0. |
| 0.002 0.002<br>0.003 0.003 | 0.0022 0.00167<br>0.00261 0.00327 | 0.00233 0.00162<br>0.00256 0.00429 |
| **Estimación Red Conv2D** | **Estimación ResNet Conv2D** | **Estimación ResNet Densa** |
| 0.00012 0.00079<br>0.00027 0.01808 | 0. 0.<br>0. 0. | 0. 0.<br>0. 0. |
| 0. 0.<br>0. 0. | 0. 0.<br>0. 0. | 0. 0.<br>0. 0. |
| 0. 0.<br>0. 0. | 0. 0.<br>0. 0. | 0. 0.<br>0. 0. |
| 0. 0.<br>0. 0. | 0. 0.<br>0. 0. | 0. 0.<br>0. 0. |
| 0.00006 0.00003<br>0.00072 0.00045 | 0.00253 0.00084<br>0.00248 0.00354 | 0.00098 0.00019<br>0.00031 0.00161 |

Tabla 4.6: Comparación de los valores de mape obtenidos en las dos validaciones anteriores.

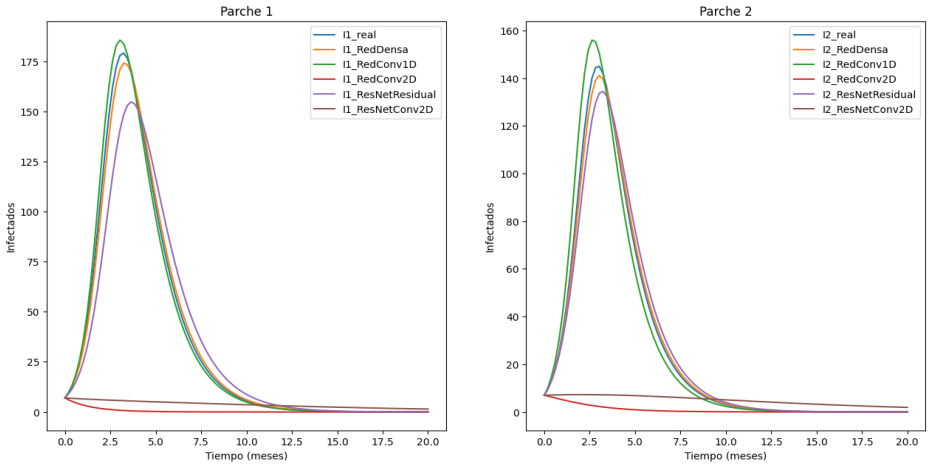



Figura 4.6: Gráficas de las predicciones hechas por cada modelo elegido vs el valor real. Tomando como $\beta_5$ el valor esperado (real).

Finalmente, para hacer una comparación de las estimaciones obtenidas con este conjunto de datos y con el conjunto empleado en la primera validación, se calcula la diferencia que hay entre los valores de mape obtenidos en cada caso (ver tabla 4.7). Los resultados obtenidos en la tabla 4.7, muestran que ninguno de los cinco modelos seleccionados es capaz de hacer estimaciones en regiones

donde no fue entrenado. No obstante, con respecto a las curvas de infección, en la figura 4.6 se puede observar que, a pesar de que las predicciones de las matrices de infección no fueron buenas, la mayoría de las curvas generadas con esas predicciones tienden a acercarse razonablemente a las curvas esperadas, es decir, describen de manera considerable la evolución esperada de los individuos infectados y, por lo tanto, también la evolución de la epidemia.

La Red Densa destaca nuevamente en la predicción, mostrando la curva que mejor se ajusta a la esperada para cada subpoblación. Sin embargo, aún no tenemos argumentos suficientes para elegir este modelo por encima de los demás, ya que este fue el único caso en el que tanto la Red Densa como los demás modelos pudieron dar resultados razonables. Para las demás tasas de infección, todos mostraron comportamientos totalmente distintos a los esperados.

| Tipo de RNA | Valor de mape obtenido en la 1ra validación | Valor de mape obtenido en la 3ra validación | Diferencia |
|---|---|---|---|
| Red Densa | **1.361** | 688.107 | 686.746 |
| Red Conv1D | 5.084 | **72.842** | 67.758 |
| Red Conv2D | 15.254 | 127.809 | 112.555 |
| ResNet Densa | 3.258 | 85.991 | 82.733 |
| ResNet Conv2D | 11.254 | 93.863 | 82.609 |

Tabla 4.7: Comparación de los valores de mape obtenidos en las dos validaciones anteriores.

## 4.3. Elección del *mejor modelo*

En cuanto a la elección del *mejor modelo* entre los cinco seleccionados, a pesar de que las estimaciones de las matrices de infección de la Red Densa fueron considerablemente mejores en la mayoría de los casos, las gráficas de las curvas de infección generadas con las estimaciones de cada uno de los modelos muestran que casi todos tienen un buen desempeño para describir las curvas de infección. Debido a que se trata de un problema lo suficientemente sencillo para que varias arquitecturas puedan tener una buena eficiencia. Sin embargo, estos pueden llegar a ser útiles para ajustar parámetros de epidemias reales de forma eficiente.

# Capítulo 5

# Discusión y Conclusiones

Los resultados obtenidos del modelo propuesto para la estimación de las tasas de infección interpoblacionales en un modelo SIR Metapoblacional plantean varios puntos interesantes en relación a la estimación de parámetros en modelos epidémicos metapoblacionales. En general, se concluye que el uso de redes neuronales artificiales (RNA) puede ser una herramienta bastante útil y poderosa para comprender y controlar la propagación de enfermedades. Aunque en este proyecto nos centramos exclusivamente en las tasas de infección, consideramos que un enfoque similar podría aplicarse a la estimación de cualquier otro parámetro de interés en este contexto.

En cuanto a la elección del tipo de red neuronal para abordar este tipo de tareas, en todas las pruebas que se hicieron, se encontró que casi todas las redes con las que se trabajó tienen un buen desempeño. Por lo que, es útil probar con diversas arquitecturas de redes, ya que al hacerlo se incrementa la probabilidad de que al menos una de ellas converja hacia la solución deseada.

Otro punto importante por mencionar, es que, aunque las estimaciones obtenidas difieran significativamente de los valores reales, esto no implica que no sean capaces de describir de manera confiable el comportamiento esperado de una epidemia. De hecho, sorprendentemente pueden lograrlo. Además, se encontró que la precisión de las estimaciones está limitada por el rango de valores con los que se entrena al modelo y esta baja considerablemente al tratar con datos desconocidos. Específicamente en el caso de las tasas de infección, esto se debe a que la cantidad de matrices de infección posibles dentro de un rango de valores dado es extremadamente amplia, lo que limita la capacidad de generalización del modelo.

Por último, es importante destacar que este modelo se encuentra en una etapa temprana de desarrollo. Por simplicidad, se redujo el problema de estimación de parámetros a su forma más básica. Se trabajó con simulaciones matemáticas en lugar de datos reales, se seleccionó el modelo epidémico metapoblacional más simple y se trabajó con solo dos subpoblaciones. Por lo tanto, no podemos afirmar con certeza cómo funcionaría el modelo con un mayor número de subpoblaciones o con modelos epidémicos más complejos, ni tampoco cómo se comportaría con datos reales de una epidemia. No obstante, consideramos que este enfoque proporciona un punto de partida sólido para abordar problemas de estimación en este contexto, ya que los resultados obtenidos demuestran su eficacia en la obtención de parámetros de un modelo epidémico metapoblacional.

# Bibliografía


[1] Brauer, F., & Castillo-Chavez, C. (2011). Mathematical Models in Population Biology and Epidemiology. Springer.

[2] Martcheva, M. (2015). An Introduction to Mathematical Epidemiology. Springer.

[3] Diekmann, O., Heesterbeek, J. A., & Metz, J. A. (2012). Mathematical Epidemiology of Infectious Diseases: Model Building, Analysis and Interpretation. John Wiley & Sons.

[4] Allen, L. J. S. (2017). An Introduction to Mathematical Biology. Pearson.

[5] Kiss, I. Z., Miller, J. C., & Simon, P. L. (2017). Mathematics of Epidemics on Networks: From Exact to Approximate Models. Physica D: Nonlinear Phenomena, 241(18), 1842-1863.

[6] Rosales Herrera, D., Ramírez, J. E., Velázquez Castro, J., Diaz, B., Martínez, M. I., Vázquez Juárez, P., & Fernández Téllez, A. (2022). Estrategias de movilidad basadas en la teoría de percolación para evitar la diseminación de enfermedades: COVID-19. Revista Mexicana de Física, 68(011701), 1-12.

[7] Géron, A. (2017). Hands-On Machine Learning with Scikit-Learn and TensorFlow: Techniques and Tools to Build Learning Machines. O'Reilly Media.

[8] Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT Press.

[9] Aggarwal, C. C. (2018). Neural Networks and Deep Learning: A Textbook. Springer.

[10] Rashid, T. (2018). Make Your Own Neural Network. CreateSpace Independent Publishing Platform.

[11] Rodríguez Hernández, J. (2023). Tesis-Codigos. Recuperado de https://github.com/Eusoj25/Tesis-Codigos.git

[12] Weight & Biases. (s.f.). Recuperado de https://wandb.ai/site.