Universidad de la República
Facultad de Ingeniería

PredGenIA: Transformers para Predicción Genómica

Memoria de proyecto presentada a la Facultad de Ingeniería de la Universidad de la República por

Graciana Castro[1], Romina Hoffman[2], Mateo Musitelli[1]

en cumplimiento parcial de los requerimientos para la obtención del título de Ingeniero Electricista[1] e Ingeniero en Sistemas de Comunicación[2].

Tutor

María Inés Fariello ..... Universidad de la República
Federico Lecumberry ..... Universidad de la República

Tribunal

Federico La Rocca ..... Universidad de la República
Guillermo Moncecchi ..... Universidad de la República
Camila Simoes ..... Universidad de la República

Montevideo
lunes 6 noviembre, 2023

PredGenIA: Transformers para Predicción Genómica, Graciana Castro[1], Romina Hoffman[2], Mateo Musitelli[1].

Esta tesis fue preparada en LaTeX usando la clase iietesis (v1.1).
Contiene un total de 77 páginas.
Compilada el lunes 6 noviembre, 2023.
http://iie.fing.edu.uy/

"No es el conocimiento, sino el acto de aprendizaje, y no la posesión, sino el acto de llegar allí, que concede el mayor disfrute".
Carl Friedrich Gauss

"Las preguntas más importantes de la vida, de hecho, no son en su mayoría más que problemas de probabilidad".
Pierre Simon Laplace

Esta página ha sido intencionalmente dejada en blanco.

Agradecimientos

Queremos agradecer a todos los que de una manera u otra contribuyeron a que estemos hoy acá. A nuestras familias y amigos por acompañarnos y ser siempre sostén. A Federico Lecumberry y María Inés Fariello por guiarnos, no solo para realizar este proyecto, sino también en los últimos pasos de nuestra formación como ingenieros y primeros pasos en el ambiente académico. A la Universidad de la República y todos sus actores (docentes y compañeros) por brindarnos todos los recursos y posibilidades para nuestra formación.

Al grupo de investigación que desarrolló el *GPTransformer* por brindarnos material que fue guía para nuestro trabajo y pacientemente contestar nuestras dudas.

Esta página ha sido intencionalmente dejada en blanco.

"Para vos Barney, mirá de quien te burlaste"

Esta página ha sido intencionalmente dejada en blanco.

Resumen

Se define el genotipo como la descripción del material físico real conformado por el ADN de un organismo y el fenotipo como cualquier característica observable del organismo (Rival, et al.). La predicción genómica busca predecir un determinado fenotipo de un individuo a partir del genotipo. Para eso, se cuenta con una base de datos genotípicos a los que se les asocia el fenotipo a predecir. Al ser los datos genotípicos una secuencia de letras, se puede tomar cada secuencia como si fuera un enunciado y las bases que lo componen (adenina (A), timina (T), citosina (C) y guanina (G)) las palabras que lo forman.

Debido al reciente auge de las redes neuronales bidireccionales para el trabajo en Procesamiento de Lenguaje Natural ("Natural Languaje Processing", *NLP*), surge la interrogante de si estos algoritmos, como las redes neuronales, redes neuronales recurrentes o *Transformers*, son igualmente eficientes en dominios que comparten similitudes en términos de estructuras de datos.

En este proyecto, se plantea el objetivo de entrenar un modelo para predicción genómica basado en *Transformers*. Se toma como secuencia de entrada el genotipo de individuos de una especie haploide para comparar su desempeño con el de los modelos más utilizados en esta área, haciendo énfasis en comprender el funcionamiento del modelo. ¿Obtiene el modelo mejores resultados que los modelos ya existentes? Además, ¿es capaz de identificar las porciones importantes de esta secuencia, para realizar la predicción deseada?

Para esta investigación se realizó un estudio del algoritmo *Transformers*, su funcionamiento y aplicaciones en el campo del NLP. Comprendido esto, se procedió a realizar el análisis de cómo adaptar un algoritmo de *Transformers* para su funcionamiento con datos genómicos de levadura con el objetivo de predecir el crecimiento de los individuos en distintos ambientes. Se estudió el modelo *GPTransformers*, propuesto por Jubair et al., 2021, en el cual se propone una estructura de *Transformers* basada solamente en el *Encoder*, debido a que para la predicción de un fenotipo es necesario contar con el conocimiento de la estructura local del ADN, la cual es determinada por este módulo. Se realizó el preprocesamiento de la base de datos de levadura, búsqueda de hiperparámetros óptimos y entrenamiento del modelo realizando validación cruzada. Se simularon dos fenotipos (lineal y no-linealmente) a partir de los genotipos que componen la base de datos, con los que se buscó evaluar cómo funciona el modelo con este tipo de datos. Luego se entrenaron modelos para realizar predicciones del crecimiento de levadura en los ambientes *Lactato* y *Lactosa*. También se realizaron predicciones conjuntas (*Multitrait*) para *Lactato* y *Lactosa* a la vez.

Se concluyó en base a los resultados obtenidos, que el algoritmo de *Transformers*, basado en mecanismos de atención, presenta resultados prometedores para el campo de la predicción genómica.

Estructura del documento

Este documento está estructurado de la siguiente manera: comenzamos en el capítulo 1 con una breve introducción al problema abordado y el campo de desarrollo del mismo, destacando sus aspectos fundamentales. En el capítulo 2, presentamos los aspectos principales del campo de la genómica, y los modelos que hoy en día se utilizan generalmente para realizar predicción genómica. En este capítulo también se describen los datos con los que se trabajará. En el capítulo 3, realizamos una descripción de los fundamentos del aprendizaje automático, construyendo la base necesaria para introducir los algoritmos secuenciales, que son desarrollados en el capítulo 4. En este capítulo también se introduce el algoritmo de los *Transformers*, sus componentes, y se ilustra un ejemplo para poder bajar los conceptos e ideas a tierra. En el capítulo 5 se presentan los modelos implementados, y en la sección 5.3 los resultados obtenidos. Se finaliza en el capítulo 6 con las conclusiones y presentando posibles trabajos futuros.

Esta página ha sido intencionalmente dejada en blanco.

Tabla de contenidos

Capítulo 1

Introducción

La predicción genómica consiste en utilizar la información contenida en el genoma de un individuo, o una población, para realizar inferencias sobre fenotipos, por ejemplo rasgos o enfermedades específicas. Se basa en la premisa de que ciertas variaciones en el ADN, como las variaciones de nucleótidos conocidas como Polimorfismos de Nucleótido Simple (SNPs), están asociadas, debido al desequilibrio de ligamiento, a mutaciones responsables de la variación que presentan ciertos rasgos, o presencia o ausencia de enfermedades. En este contexto, mejorar la interpretación y predicción de datos es un desafío constante, debido a las diferencias significativas en los conjuntos de datos, la estructura de la población y el número de muestras. La precisión en la predicción genómica es muy importante para diversos campos, como la medicina, la agricultura, y la biología, ya que puede conducir a tratamientos más eficaces, cultivos más resistentes y una comprensión más profunda del genoma. Además, las predicciones genómicas permiten en algunos casos ahorrar años de investigación y recursos que serían necesarios para obtener datos de manera empírica y realizar las mejoras basado en éstos. Por ejemplo, en el estudio genético de los toros para la producción lechera, es necesario analizar la segunda camada inseminada por un mismo toro, lo cual implica una numerosa cantidad de años, que podría reducirse con la incorporación de técnicas de predicción genómica.

Hasta el momento, los modelos mixtos y las regularizaciones bayesianas han sido ampliamente utilizados, sin que exista un método único que en todos los casos obtenga los mejores resultados. Esto se debe a las variaciones en los SNPs y las características específicas de los datos genómicos. Recientemente, las redes neuronales han ganado atención en la predicción genómica. No obstante, aún no han logrado superar los resultados obtenidos por los modelos lineales, que además ofrecen la ventaja de un menor costo computacional en su ejecución.

Con el objetivo de seguir mejorando los resultados de los modelos lineales y buscando alternativas a las Redes Neuronales, los *Transformers* parecen ser un camino prometedor, ya que han demostrado gran capacidad para capturar relaciones a largo plazo en secuencias. Lograr adaptar y entrenar un modelo que pueda extraer y aprender las dependencias biológicas desde las dependencias entre posiciones de secuencias de datos puede llevar a un gran avance [5]. La capacidad

de los *Transformers* en la tarea de capturar información contextual y modelar dependencias de largo alcance los hace grandes candidatos para esta tarea.

Nos basamos en el modelo *GPTransformer* (Jubair et al.,2021 [16]), propuesto para trabajar con una base de datos de cebada. Se busca predecir la presencia de Deoxinivalenol (DON), que es una micotoxina natural que se presenta en alimentos, y Fusariosis de la Espiga (FHB), que es una enfermedad que afecta al trigo y la cebada[1]. Para esto, implementan un modelo basado en la arquitectura de un *Transformer*, para realizar la predicción de DON y FHB con el mismo modelo, dada la fuerte correlación de estos dos componentes. Se utiliza únicamente el *Encoder* debido a que es en este bloque donde se obtiene la información sobre la secuencia de entrada y la importancia que tiene cada una de las posiciones. El *Decoder* por su parte se basa en los puntajes de atención obtenidos en el *Encoder* para predecir una secuencia, por lo cual no es necesario para esta aplicación, ya que no es lo que se busca.

Además del *Transformer*, con el fin de comparar su desempeño, se entrenaron los siguientes modelos: *Best Linear Unbiased Predictor* (BLUP), *Residual Fully Connected Neural Netowork*(RFCNN), *Linear Regression* (LR) y *Decision Tree* (DT). Los resultados obtenidos se presentan comparando según la métrica del Coeficiente de Correlación de Pearson (PCC) que mide el grado de asociación lineal entre dos variables [6]. Sus conclusiones muestran que, en general, BLUP y *Transformer* obtienen los mejores resultados. Este estudio muestra el potencial del método basado en *Transformers* como alternativa a modelos lineales.

Motivados por los resultados anteriores, nos proponemos entrenar un modelo basado en el propuesto por Jubair et al.,2021 [16], para predecir el crecimiento de levadura en dos ambientes diferentes, Lactato y Lactosa. El primer inconveniente al momento de adaptar este modelo a nuestro conjunto de datos es que la cebada es un organismo diploide, y la levadura es haploide, por lo cual la codificación de los datos será diferente. Se trabaja con este conjunto de datos dado que existe un estudio previo[2] en el cual se compara el desempeño de diferentes modelos (*Bayesian Linear Regression* [10], *Gradient Boosting* [2] [8], *Random Forest* [14], *Ridge Regression* [15], *SVR* [4]) en la predicción del crecimiento de levadura. Se toman estos resultados como punto de partida para evaluar el desempeño de nuestro modelo.

[1]Estos dos componentes afectan directamente a la cebada y a los individuos que la consuman posteriormente, por lo que la presencia de ellos puede tener grandes consecuencias en la producción

[2]Proyecto de grado DNAi [7].

Capítulo 2

Fundamentos Genómicos y Bases de Datos

En este capítulo se definen los conceptos básicos para trabajar con datos genómicos y se presenta el estado del arte para la predicción genómica. También, se realiza una descripción del problema a trabajar, la base de datos utilizada y el preprocesamiento implementado.

2.1. Datos Genómicos 101

Se le llama ADN (Ácido Desoxirribonucleico) a la molécula compleja que se encarga de codificar la información genética para la transmisión de datos hereditarios [3]. Está formada por dos cadenas enrolladas formando una estructura denominada doble hélice. Cada cadena se encuentra formada por una secuencia de nucleótidos, que se componen de grupos de fosfato y azúcares (desoxirribosa), en los que hay enlazadas una de las siguientes cuatro bases nitrogenadas: adenina (A), timina (T), citosina (C) y guanina (G). Las cadenas se unen por enlaces que conectan adenina con timina y citosina con guanina. El orden de estas bases es el que determina toda la información necesaria para el desarrollo e identificación de cada individuo (figura 2.1).

Esta estructura fue descrita por primera vez en 1953 por James Watson y Francis Crick en un articulo publicado en la revista Nature [22]. El descubrimiento de la doble hélice publicado por estos dos científicos fue posible debido a la imagen obtenida por la Doctora Rosalind Franklin, a la que tuvieron acceso presuntamente sin el conocimiento de ella. La conocida como "fotografía 51" fue obtenida utilizando la técnica de difracción de Rayos X en la cual Franklin era experta. Si bien el descubrimiento del ADN data de finales del siglo XIX, no fue hasta los años de Watson y Crick que se pudo saber a ciencia cierta su importancia.

Se define el genoma como el conjunto de todo el ADN de una célula de una especie. Los genes son fracciones de ADN capaces de ser traducidos a una proteína. Estos representan aproximadamente un 30 % del genoma, mientras que al otro 70 % se le denomina porciones intergénicas, ya que se encuentran entre los genes.

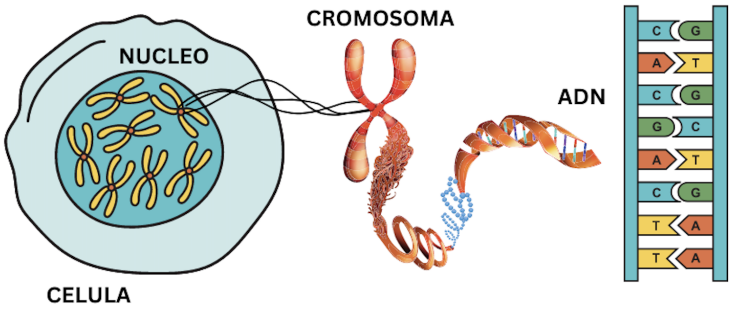


Figura 2.1: Ilustración del ADN de un organismo siendo extraído de un cromosoma perteneciente a un núcleo celular.

El ADN se organiza en cromosomas. En los organismos haploides los cromosomas aparecen organizados individualmente, mientras que en los organismos diploides existen dos cromosomas y por lo tanto cada gen se encuentra duplicado, ocupando el mismo locus [1] en cromosomas homólogos. A cada una de las variantes del gen se las denomina *alelo*.

Dentro del genoma se pueden encontrar los denominados "polimorfismos de nucleótido simples" (SNP - "Single nucleotid polymorphisms") que son una variante genómica en la posición de una base única del ADN. Comúnmente, los SNPs se encuentran en secciones intergénicas, pero cuando se encuentran cercanos al gen o dentro de ellos, pueden generar alteraciones o enfermedades en el individuo. Si bien los SNPs se distribuyen de manera uniforme en el genoma de los indiviudos, muy pocos causan alteraciones en el fenotipo del individuo. Algunos de ellos, pueden no ser responsables de alteraciones, pero al estar ligados físicamente a las causantes a una alteración en el genoma, pueden resultar marcadores de las mismas.

La determinación del orden en que se encuentran las bases en el genoma de un individuo se denomina secuenciación. Por ejemplo, el genoma humano está compuesto por aproximadamente tres mil millones de pares de bases, por lo que la secuenciación completa del mismo es una tarea extremadamente costosa, superando aproximadamente los más de mil dólares por individuo, a una profundidad razonable, y a muy baja profundidad aproximadamente unos cuarenta y cinco dólares.

El genotipado es una estrategia menos costosa mediante la cual se mira una cantidad de *loci* fijos del genoma en los cuales se sabe que hay variación genética. Se codifica el dato para indicar si en esa determinada posición hubo una variación o no. En el caso de especies haploides, suele identificarse con un cero (si esa posición en el genoma es igual a la del genoma de referencia) o un uno (en caso contrario). Al conjunto de información genética de un individuo particular se le denomina genotipo.

El fenotipo está formado por las características medibles de un individuo (tanto

[1] Lugar que ocupa un gen dentro de un cromosoma, del plural *loci*

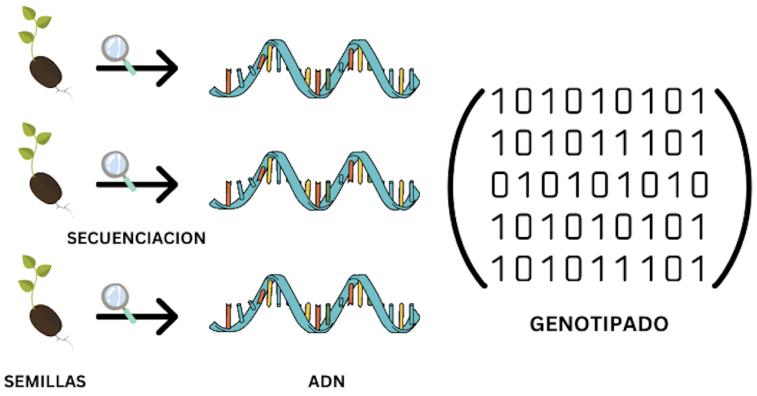


Figura 2.2: Ejemplificación del proceso de genotipado a partir de organismos haploides, semillas.

macroscópicas como microscópicas), que resultan de la interacción de un determinado genotipo con un determinado ambiente. La influencia del segundo implica que si existieran dos individuos con genotipos exactamente iguales (clones) pero criados en distintas condiciones ambientales (como ser el clima o la alimentación), pueden tener distinto fenotipo. El porcentaje de varianza del fenotipo que se debe solamente al genotipo se denomina *heredabilidad*.

2.2. Predicción Genómica

La predicción genómica es una metodología que tiene como objetivo predecir un valor fenotípico (por ejemplo, el crecimiento de un individuo o la producción de leche de una vaca) a partir de datos genéticos. Para eso, se entrena un modelo predictivo en base a un conjunto de datos genotípicos y fenotípicos (a los que se denominan conjunto de entrenamiento). El modelo entrenado es luego utilizado para realizar las predicciones del fenotipo de un conjunto de datos que se encuentra únicamente formado por datos genotípicos.

La principal ventaja de esta técnica radica en que al no ser necesario contar con los datos de fenotipo, se reducen los costos y tiempo de procesado, logrando abarcar mayor cantidad de datos para predecir y características que pueden llegar a ser difíciles de medir. Por ejemplo, en los casos en los que se quieren estudiar datos de fenotipo que requieren del transcurso del tiempo, como ser la reproducción de un animal, ya no será necesario esperar al ciclo completo, que el animal tenga crías, para poder evaluar el desempeño del individuo [18]. De conseguir buenos resultados, se puede ver considerablemente beneficiada la producción agropecuaria, por ejemplo, tanto por la reducción de costos mencionada como por la mejora fenotípica de los nuevos individuos.

Sin embargo, los resultados exitosos se encuentran estrechamente ligados a la calidad de los datos, tanto de los que se desean predecir como del conjunto de entrenamiento. En particular, este segundo debe ser lo suficientemente grande

para que el modelo sea capaz de generalizar lo aprendido para los nuevos datos, cosa que no ocurre si la muestra utilizada para entrenar es muy pequeña o no representativa de la población.

Inicialmente, para la predicción genómica fueron utilizados modelos clásicos como regresiones lineales penalizadas y modelos mixtos. En una siguiente instancia, se comenzó a: (1) incorporar no linealidades, (2) no imponer priors de antemano, (3) evaluar métodos que descubren relaciones entre regiones lejanas de la secuencia, (4) trabajar (en último caso) con modelos Bayesianos [18].

Debido a que los datos genómicos están conformados por largas secuencias de las bases A, C, T y G, cuyo orden conforma un fenotipo específico y por tanto es de alta importancia, se busca implementar métodos de aprendizaje profundo que pueden llevar a mejoras en los resultados. Métodos como las redes neuronales recurrentes (RNN), para el trabajo con datos secuenciales, ordenados y de alta dimensionalidad, están comenzando a implementarse, junto con métodos para enfatizar qué aspectos de la secuencia genómica son de alta importancia, como el *self-attention*, y para el manejo de la memoria, como las neuronas *long-short-term-memory* (LSTM). Adicionalmente, pueden incorporarse otras fuentes de información (como son los datos ambientales) para evaluar si se mejora la predicción. Esto conlleva a que los datos presenten relaciones no lineales que suelen ser problemáticas para los modelos clásicos. Se refuerza entonces la necesidad de trabajar con métodos de aprendizaje profundo que tienen entre sus ventajas que no es necesario tener determinadas todas las relaciones entre los datos, ni tampoco trabajar con un tipo de dato específico [20]. En el presente trabajo, fue implementado el método de los *Transformers*, en cuya implementación están incluidos múltiples *self-attention* y redes neuronales.

2.3. Descripción del problema

Se cuenta con una base de datos con información del crecimiento de levadura para cuarenta y ocho ambientes diferentes. El problema abordado es el de la predicción del crecimiento de la levadura en cada uno de los ambientes antedichos. Puntualmente, se trabajó con los ambientes Lactato y Lactosa. El crecimiento de la levadura, es un fenotipo que se cuantifica numéricamente, teniendo para cada genotipado de levadura su correspondiente fenotipo de crecimiento, como se ilustra en la figura 2.3.

Además, con el objetivo de analizar si el modelo implementado "presta atención" adecuadamente dentro del genotipo, se simulan datos fenotípicos a partir de transformaciones lineales y no lineales de los datos de levadura, para comparar si las posiciones a las que el modelo "presta atención" coinciden con las utilizadas para la simulación. Para profundizar en la comprensión del problema, en las siguientes secciones se presenta un breve análisis de los datos fenotípicos de levadura y de los obtenidos a partir de las perturbaciones en el genotipo.

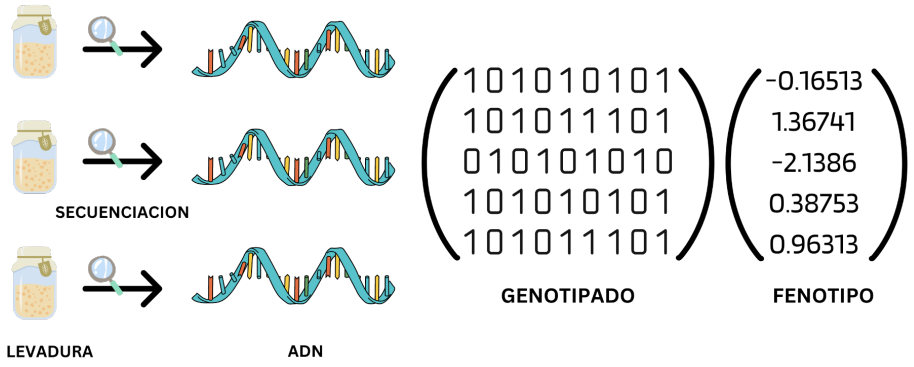


Figura 2.3: Representación de los datos de entrada a nuestro problema.

2.4. Estado del Arte

En el campo de la predicción genómica no existe ningún método que supere a todos los demás para todas las especies, rasgos y poblaciones, ya que la arquitectura genética de la característica a predecir presenta diferencias significativas entre los conjuntos de datos. Además varía también la estructura de la población y el número de muestras, que son factores que afectan directamente el desempeño de los modelos. La mayoría de los modelos utilizados en este ámbito son modelos lineales mixtos y regularizaciones bayesianas. Las redes neuronales han ganado atención recientemente, sin embargo no han obtenido mejores resultados que los modelos lineales, los cuales además requieren menor costo computacional para su ejecución. Por otro lado, los modelos de aprendizaje automático más populares para esta aplicación son: *Random Forest*, que combina la salida de múltiples árboles de decisión, y máquinas de vectores de soporte (*Support Vector Machines* o SVM por sus siglas en inglés).

Modelos Lineales Mixtos

Los Modelos Lineales Mixtos (o LMM por sus siglas en inglés, *Linear Mixed Models*) se utilizan para analizar datos que presentan tanto efectos fijos como aleatorios. La ecuación que los caracteriza es:

$$\mathbf{y} = \mathbf{X}\beta + \mathbf{Z}\mathbf{b} + \epsilon, \tag{2.1}$$

donde:

- $\mathbf{y}$ es el vector $(n \times 1)$ de respuesta aleatoria.
- $\mathbf{X}$ es la matriz $(n \times p)$ de efectos fijos, con la información de los genotipos de n individuos, donde p es el número de SNPs.
- $\beta = (\beta_0, \beta_1, ..., \beta_{p-1})^T$ es el vector de coeficientes de efectos fijos.

- $\mathbf{Z}$ es la matriz de efectos aleatorios, de dimensión $n \times q$.
- $\mathbf{b}$ es el vector $(q \times 1)$ de efectos aleatorios, con distribución $\mathcal{N}(0, D)$, donde D es una matriz de varianza-covarianza de efectos aleatorios $(q \times q)$.
- ϵ es el vector de errores aleatorios con distribución $\mathcal{N}(0, R)$, donde $\mathbf{R}$ es una matriz de varianza-covarianza.

En genómica, $\mathbf{b}$ suele incluir los efectos genotípicos y los efectos de interacción genotipo-ambiente, mientras que $\mathbf{X}$ puede contener información sobre las covariables del entorno. Se observa que bajo este modelo, E($\mathbf{y}$)=$\mathbf{X}\beta$, y la matriz de varianza-covarianza del vector de respuesta $\mathbf{y}$ es var($\mathbf{y}$)=ZDZ^T+$\mathbf{R}$. Se define la varianza marginal de $\mathbf{y}$ como: $\mathbf{V}$=$\mathbf{Z}^T\mathbf{DZ}$+$\mathbf{R}$

Una solución para estimar los parámetros β y $\mathbf{b}$ propuesta por Henderson consiste en resolver la siguiente ecuación (MME, *Mixed Model Equation*):

$$\begin{pmatrix} X^T R^{-1} X & X^T R^{-1} Z \\ Z^T R^{-1} X & Z^T R^{-1} X Z + \mathbf{D}^{-1} \end{pmatrix} \begin{pmatrix} \hat{\beta} \\ \hat{b}^* \end{pmatrix} = \begin{pmatrix} X^T R^{-1} y \\ Z^T R^{-1} y \end{pmatrix}, \tag{2.2}$$

en donde la solución obtenida para β es el mejor estimador lineal insesgado (BLUE, *Best Linear Unbiased Estimate*) de efectos fijos y la obtenida para $\mathbf{b}$ es el mejor predictor lineal insesgado (BLUP, *Best Linear Unbiased Prediction*) de efectos aleatorios.

Si $\mathbf{D}$ y $\mathbf{R}$ son conocidas, el mejor predictor lineal insesgado de los efectos aleatorios $\mathbf{b}$ está dado por: $\hat{b}^* = \mathbf{DZ^T V^{-1}}(\mathbf{y} - \mathbf{X}\hat{\beta})$, donde $\hat{\beta} = (\mathbf{X}^T\mathbf{V^{-1}X})^{-1}\mathbf{X}^T\mathbf{V^{-1}y}$ es el estimador generalizado de mínimos cuadrados de β.

Regresión Lineal Bayesiana

En un modelo de regresión lineal bayesiana para predicción genómica, se establece una relación entre los marcadores genéticos y el fenotipo de interés utilizando un enfoque de regresión lineal. Es decir, se asume que los efectos genéticos de los marcadores son lineales, lo que significa que se supone que los cambios en los marcadores genéticos se traducen en cambios proporcionales en el fenotipo. La diferencia con otros modelos de regresión lineal es que en lugar de obtener un único valor de estimaciones para los coeficientes de regresión, en el enfoque bayesiano se obtiene una distribución de probabilidad, como se presenta en la siguiente ecuación:

$$\beta = argmax_\beta p(\beta|\mathbf{x}, \mathbf{y}). \tag{2.3}$$

Esta distribución "a *posteriori*" puede combinarse con nuevos datos observados para hacer predicciones sobre el fenotipo de individuos no muestreados en la población de estudio.

La ventaja de utilizar un enfoque bayesiano es que permite incorporar información previa sobre los efectos genéticos de los marcadores, lo que puede mejorar la precisión de las predicciones.

Los modelos *Bayes A*, *Bayes B*, *Bayes C* y *Bayes Lasso* son variantes de la regresión lineal bayesiana que se utilizan en la predicción genómica. Estos modelos permiten la selección de variables y tienen en cuenta la posibilidad de que algunos marcadores no estén relacionados con el fenotipo. En el modelo *Bayes A* se asume que todos los marcadores contribuyen de manera significativa a la variación del fenotipo. Cada marcador tiene un coeficiente de regresión asociado que se estima a partir de los datos. Sin embargo, el modelo no tiene en cuenta la posibilidad de que algunos marcadores no estén relacionados con el fenotipo y, por lo tanto, puede incluir marcadores irrelevantes en el modelo. A diferencia del modelo *Bayes A*, el modelo *Bayes B* utiliza una distribución mixta para modelar el efecto de cada marcador, lo que implica que algunos marcadores tendrán un efecto nulo o cercano a cero. La distribución mixta permite estimar qué marcadores son relevantes y cuáles no, proporcionando así una forma de selección de variables en el modelo.

2.5. Multi-trait

Aunque la predicción genómica es un enfoque muy prometedor en la rama de la genética, aumentar el *accuracy* de las predicciones genómicas de los diversos modelos sigue siendo un desafío. Modelos multifenotípicos, esto es, que predicen múltiples fenotipos a la vez, han demostrado resultados prometedores al ser evaluados según el artículo *"Multi-trait multi-environment genomic prediction of agronomic traits in advanced breeding lines of winter wheat"* [11].

Un acercamiento comunmente utilizado en la genética multivariada es la selección indexada, la cual asigna diferentes pesos a cada característica, basándose en su importancia económica. Sin embargo, la selección indexada clásica, solo optimiza la ganancia genética en la próxima generación, y requiere de experimentación para encontrar los pesos que conllevan a la obtención de los resultados esperados, según el artículo *"Multi-trait genomic selection methods for crop improvement"* [19].

2.6. Descripción de los Datos

La base de datos de levadura cuenta con información del crecimiento de 1008 cepas de levadura en cuarenta y ocho ambientes diferentes. Cada cepa contiene información de 11623 SNPs, codificados con valores cero o uno de acuerdo a si el individuo presenta en esa posición de su genotipo una variación. Asociado a cada individuo, se encuentra el valor de fenotipo que cuantifica su crecimiento en ese ambiente. En la figura 2.4 se presentan los histogramas de distribución de los fenotipos para los ambientes trabajados, Lactato y Lactosa. Se observa en la figura 2.5 la gráfica de dispersión de Lactato y Lactosa, donde se puede observar una fuerte relación lineal. Se realizó el cálculo de la correlación entre ellos cuyo resultado fue 0.8.

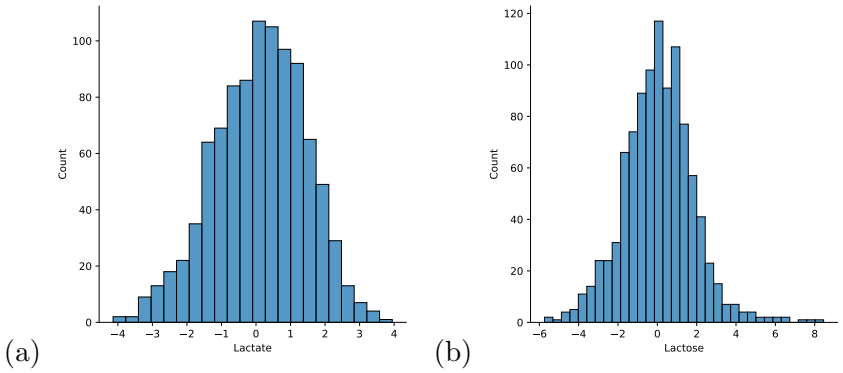


Figura 2.4: Histograma de distribución de los datos de (a) Lactato, y (b) Lactosa.

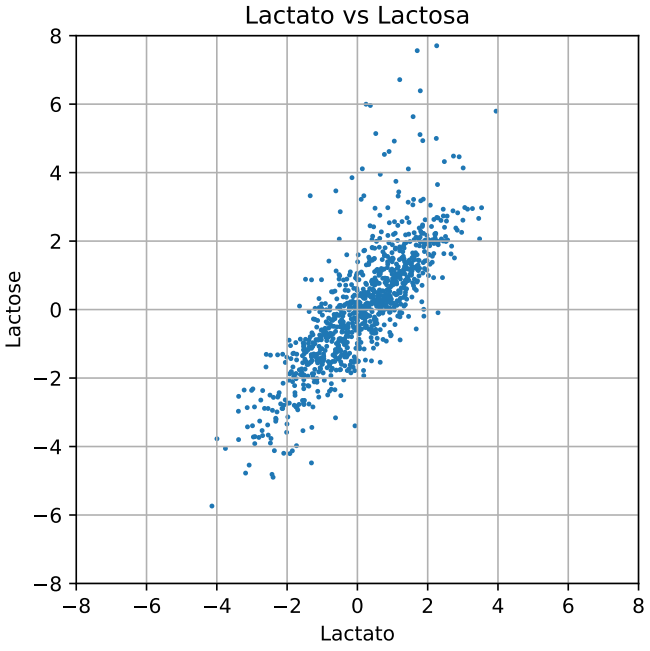


Figura 2.5: Gráfica de dispersión de Lactato vs Lactosa.

Datos simulados

Con el objetivo de realizar un análisis del funcionamiento del modelo, se realizaron dos simulaciones de fenotipos a partir del genotipo original, una con relaciones lineales y otra con relaciones no lineales entre las posiciones perturbadas. El objetivo es saber qué posiciones del genoma afectan el fenotipo creado, para después poder evaluar la interpretabilidad del modelo.

Para obtener el primer conjunto de datos simulados, el procedimiento consistió en:

1. Elegir doscientas posiciones aleatorias del genotipo.
2. Multiplicarlas por constantes de diferente magnitud.
3. Sumar todo el genotipo perturbado para obtener un número real asociado a cada individuo, que será el fenotipo.
4. Normalizar el vector de resultados.
5. Sumar un ruido de varianza $\sigma^2 = 0{,}1$, que representa el 10 % de la varianza total.

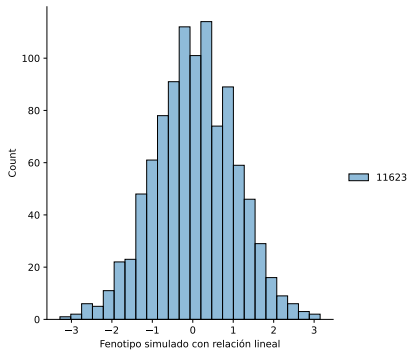
Para el segundo se tomó el mismo genotipo perturbado y se realizaron productos entre distintas posiciones para que resulte la combinación no lineal buscada. Luego, se continuó con el procedimiento llevado a cabo para la simulación ya descrita. La distribución de los datos resultantes se puede ver en la figura 2.6.

La simulación de datos permite saber qué posiciones influyen sobre el fenotipo a predecir, por lo que puede realizarse el análisis de si el modelo logra identificar correctamente las zonas a las que se les debe "prestar más atención" al momento de realizar predicciones y a cuales no.

2.6.1. Preprocesamiento

El preprocesamiento de los datos consistió en dos etapas:

(a)



(b)

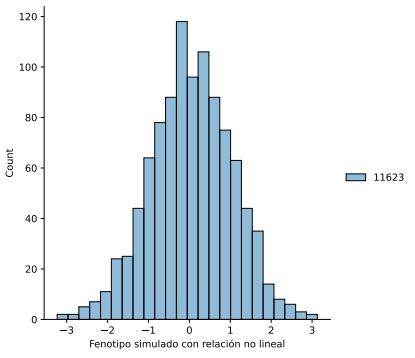


Figura 2.6: Histogramas de distribución de los datos para cada simulación, con (a) perturbaciones lineales, y con (b) perturbaciones no lineales.

Ambiente	Muestras
Lactato	973
Lactosa	1004
Combinación	973
Lineal	1008
No lineal	1008

Tabla 2.1: Cantidad de muestras con información de fenotipo para cada ambiente (base de datos de Levadura) y simulaciones.

1. Descartar datos de individuos para los que no se tenía la información del fenotipo.

2. Definir conjuntos de entrenamiento, validación y test.

En la tabla 2.2 se presenta la cantidad de muestras que resultaron para cada ambiente luego de descartar los datos faltantes.

Para ambos ambientes se realizó la misma división en conjuntos de entrenamiento, validación y test. Se tomó el 20 % para test y se dividió el 80 % restante en entrenamiento y validación, tomando el 80 % y el 20 % para cada conjunto respectivamente. Los conjuntos de validación y de entrenamiento se utilizaron en la búsqueda de hiperparámetros que optimicen los resultados. Sin embargo, al momento de entrenar, se utilizaron los conjuntos de entrenamiento y validación separados en cinco folds para realizar validación cruzada.

Ambiente	Entrenamiento	Validación	Test
Lactato	623	155	195
Lactosa	643	160	201

Tabla 2.2: Tamaño de los conjuntos de entrenamiento, validación y test para los diferentes ambientes de la base de datos de levadura.

Capítulo 3

Aprendizaje Automático 101

En el presente capítulo se presentan las herramientas esenciales para trabajar con modelos de aprendizaje automático, particularmente en el enfoque supervisado.

Los algoritmos de aprendizaje supervisado tienen el objetivo de predecir una variable en función de otras variables, en general más fáciles de medir. Éstos se dividen en dos grandes clases, que dependen de la naturaleza de la variable que se busca predecir: algoritmos de *regresión* y de *clasificación*.

Algoritmos de regresión

Un algoritmo de regresión, es aquél que busca predecir los valores numéricos de una variable continua $\mathbf{y} = (y_0, \cdots, y_{n-1})$ (variable objetivo) en función de n variables $(\mathbf{x}_0, \cdots, \mathbf{x}_{n-1})$, que se mapean a $\mathbf{y}$ mediante una función matemática $g(\cdot)$ que se busca aprender. Las ordenadas $\tilde{y}_i = g(\mathbf{x}_i) \sim y_i$ son pertenecientes a un conjunto numérico contínuo. En la figura 3.1(a) se presenta un ejemplo de regresión, donde se destaca que la función $g(\cdot)$ busca aproximar de la mejor manera a todos los puntos (x_i, y_i). Más adelante se presenta el fenómeno de *sobreajuste*, donde se explica por qué la función $g(\cdot)$ no debe verificar a todos los $\mathbf{x}_i$. En el presente proyecto, los algoritmos implementados son del tipo de regresión, ya que se predice un valor numérico (del fenotipo).

Algoritmos de clasificación

A diferencia de los algoritmos de regresión, un algoritmo de clasificación es aquél que busca predecir una variable discreta y, que toma valores del tipo: {sí, no}, {verde, rojo, amarillo}, etc. Es decir, aquellos que predicen la pertenencia de un elemento a una determinada clase. Se aprende una función $g(\cdot)$ tal que sus ordenadas $\tilde{y}_i = g(x_i)$ son el atributo ordinal antedicho. En la figura 3.1(b) se presenta un ejemplo de clasificación, en el cual se visualiza que el objetivo de $g(\cdot)$ es segmentar correctamente los elementos $\mathbf{x}_i \in \mathbb{R}^2$ en cada una de las clases a las que pertenecen.

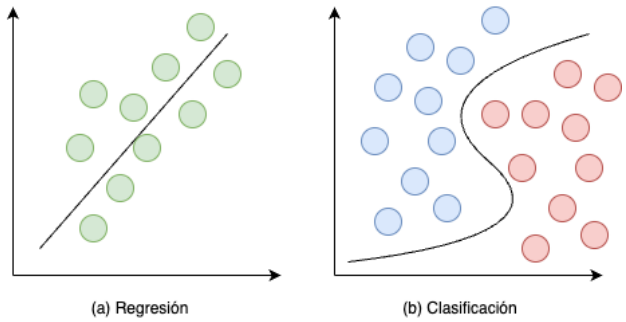


Figura 3.1: Ejemplo de Regresión (a), donde se busca predecir y_i a partir de $x_i \in \mathbb{R}$, y Clasificación (b), donde se busca predecir a qué clase pertenece cada $x_i \in \mathbb{R}^2$

.

3.1. El modelo de aprendizaje automático

En la figura 3.2 se ilustra el diagrama de un modelo de aprendizaje de aprendizaje automático, tomado del libro *"Learning From Data"* [1]. De forma sintética, en cada uno de los recuadros del diagrama se destaca que existe una función desconocida $f(\cdot)$ que se busca aproximar empleando un determinado algoritmo $\mathcal{A}$, obtenido de un conjunto de hipótesis $\mathcal{H}$, y aplicado sobre un conjunto de datos de entrenamiento $\{(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_n, y_n)\}$. El algoritmo $\mathcal{A}$ ejecuta una secuencia de pasos determinados sobre cada uno de los puntos $(\mathbf{x}_i, y_i)$ del conjunto de entrenamiento, obteniendo para cada instante una medida del error, indicada en rojo en el diagrama. La función $e(\cdot)$ que cuantifica el error, es denominada *función de costo* y es una función que se busca **minimizar**: a menor resultado de esta, mejor el resultado final. El resultado final del algoritmo, arroja una función que en el diagrama se ejemplifica como $g(\cdot)$. Es mediante ésta función, que se puede calcular el error en cada paso del entrenamiento, realizando la comparación entre $y_i = f(\mathbf{x}_i)$ e $\tilde{y}_i = g(\mathbf{x}_i)$. Finalizado el entrenamiento, se puede verificar mediante las denominadas *funciones de ganancia*, qué tan óptimos son los resultados que obtiene la función encontrada $g(\cdot)$, y qué tanto se parecen estos a los valores $y_i = f(x_i)$. En contraposición a las funciones de costo, una función de ganancia es una función que se busca **maximizar**, a mayor resultado de ésta, mejor el resultado final.

Se presentan a continuación algunos ejemplos de funciones de costo, como el *error cuadrático medio* en la ecuación 3.1 y el *error medio absoluto:* en la ecuación 3.2, para un algoritmo de regresión:

$$\mathrm{MSE}(\mathbf{x}) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \tilde{y})^2, \tag{3.1}$$

$$\mathrm{MAE}(\mathbf{x}) = \frac{1}{n}\sum_{i=0}^{n-1}|y_i - \tilde{y}|, \tag{3.2}$$

en donde $\tilde{y}_i$ son las predicciones realizadas por el modelo de aprendizaje automático. Particularmente, el *error cuadrático medio* (MSE) es una medida de qué tan

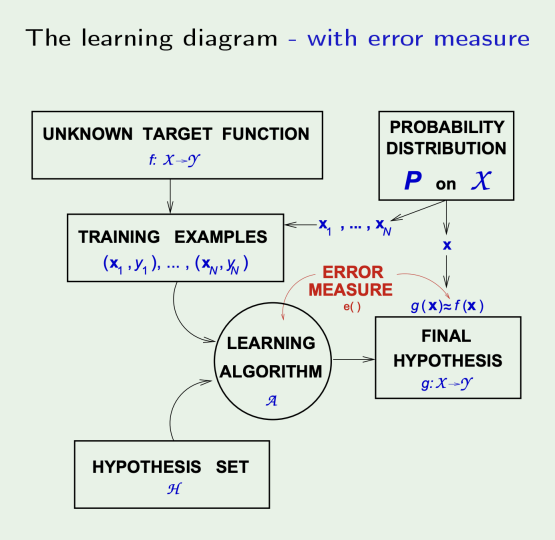


Figura 3.2: Diagrama de un modelo de aprendizaje automático tomado del libro *"Learning From Data"* [1].

acertado está siendo el modelo de aprendizaje automático en términos de predecir los valores $\tilde{y}_i = g(\mathbf{x}_i)$. Es utilizada en casos donde se busca tener sensibilidad alta a valores *outliers*, debido a ser elevada al cuadrado. Los outliers son predicciones que están muy por fuera de lo que es esperado, generando una performance altamente negativa en el desempeño del modelo entrenado

Se destacan a continuación algunos ejemplos de funciones de ganancia, como el *precision* en la ecuación (3.3) y el *recall* en la ecuación (3.4), para un algoritmo de clasificación, en el que se busca predecir la pertenencia a una clase:

$$P = \frac{TP}{TP + FP}, \tag{3.3}$$

$$R = \frac{TP}{TP + FN}, \tag{3.4}$$

donde TP (*true positive*) y TN (*true negative*) son las predicciones de pertenecer y de no pertenecer a la clase positiva, respectivamente, que fueron acertadas. Por el contrario, FP (*false positive*) y FN (*false negative*) son las predicciones de pertenecer y de no pertenecer a la clase positiva, respectivamente, que no fueron acertadas. Es importante resaltar que se cuantifican los resultados respecto a una clase, en este caso la denominada clase positiva. En la figura 3.3, se ejemplifica gráficamente la idea de pertenencia y no pertenencia, así como la predicción correcta e incorrecta.

Para los problemas de regresión, un ejemplo de función de ganancia es la Correlación de Pearson (PCC), $r(\mathbf{x}, \mathbf{y})$, cuyo resultado es un coeficiente que mide la dependencia lineal entre las variables $\mathbf{x}$ e $\mathbf{y}$, y cuyo valor pertenece al intervalo $[-1, +1]$. Cuanto más cercano a los extremos del intervalo se encuentre r, mayor será la dependencia lineal entre las variables, mientras que cuanto más cercano al

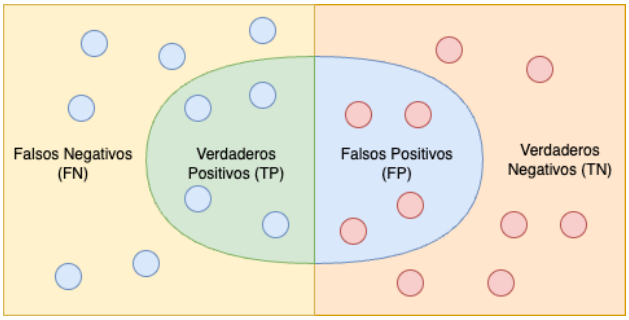


Figura 3.3: Ejemplificación gráfica de precision y recall. Los puntos azules son las muestras pertenecientes a la clase positiva, mientras que los puntos rojos pertenecen a la clase negativa. La zona dentro de la curva es la zona de positivos mientras que por fuera está la zona de negativos.

medio, 0, menor será la misma. Su expresión matemática es la siguiente:

$$r(\mathbf{x},\mathbf{y}) = \frac{\operatorname{cov}(\mathbf{x},\mathbf{y})}{\sigma_{\mathbf{x}}\sigma_{\mathbf{y}}} = \frac{\sum_{i=0}^{n-1}(x_i - m_{\mathbf{x}})(y_i - m_{\mathbf{y}})}{\sqrt{\sum_{i=0}^{n-1}(x_i - m_{\mathbf{x}})^2 \sum_{i=0}^{n-1}(y_i - m_{\mathbf{y}})^2}}, \tag{3.5}$$

en donde $\sigma_{\mathbf{x}}$ y $\sigma_{\mathbf{y}}$ son las varianzas de $\mathbf{x}$ e $\mathbf{y}$, respectivamente, y $m_{\mathbf{x}}$ y $m_{\mathbf{y}}$ son las medias de $\mathbf{x}$ e $\mathbf{y}$, respectivamente. Esta métrica también se puede utilizar en algoritmos de clasificación.

3.2. Algoritmos de Aprendizaje Automático

3.2.1. El Perceptrón

El perceptrón es uno de los modelos de aprendizaje automático más sencillo, y constituye la base de algoritmos más sofisticados, como las redes neuronales. Es un algoritmo de clasificación en dos clases, en el que el espacio de entrada es $\mathcal{X} = \mathbb{R}^d$, y el espacio de salida es $\mathcal{Y} = \{1, -1\}$. El valor que toma y_i, determinará a qué clase pertenece cada instancia $\mathbf{x}_i$, por ejemplo, puede representar las clases *sí* y *no*. El perceptrón se implementa mediante una función $h(\cdot) \in \mathcal{H}$, que asigna a cada una de las coordenadas x_i de un vector $\mathbf{x} \in \mathbb{R}^d$ un peso w_i, que reflejan la importancia que tiene la coordenada i-ésima en la decisión de la clase a la que pertenece $\mathbf{x}$. Todas las coordenadas con sus correspondientes pesos son combinadas de la siguiente manera:

$$s(x) = \sum_{i=0}^{d-1} w_i x_i. \tag{3.6}$$

Luego se comparan contra un umbral b: si la combinación (3.6) es mayor a b la clase será 1 y de ser menor a b, la clase será -1. Esto se puede expresar como se muestra a continuación:

$$\left.\begin{array}{l} \sum_{i=0}^{d-1} w_i x_i > b \Rightarrow +1 \\ \sum_{i=0}^{d-1} w_i x_i < b \Rightarrow -1 \end{array}\right\} \Rightarrow h(\mathbf{x}) = \operatorname{sign}\left(\left(\sum_{i=0}^{d-1} w_i x_i\right) - b\right). \tag{3.7}$$

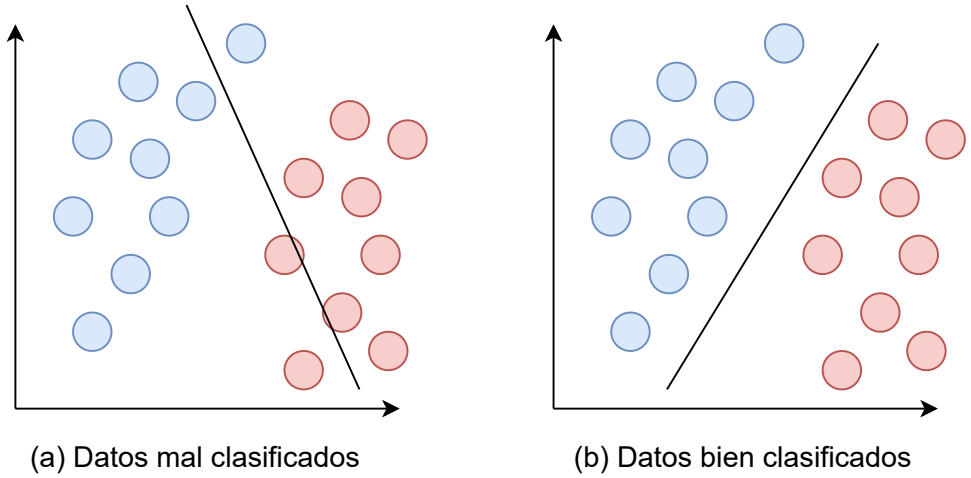


Figura 3.4: Visualización de dos modelos distintos de perceptrón sobre un mismo conjunto de datos. El modelo (a) clasifica mal algunos datos mientras que el modelo (b) clasifica a todos los datos correctamente.

Durante el proceso de entrenamiento, se buscan los pesos $w_0, \cdots, w_{d-1}$ y el valor b, denominado *sesgo*, que obtienen los mejores resultados. Es decir, que logran clasificar de la mejor manera a los datos. Visualmente, esto implica separar los datos con categoría 1 de aquellos con categoría -1, como se ilustra en la imagen 3.4. Se puede visualizar en el caso (a) que existen datos clasificados incorrectamente, mientras que en el (b) todos fueron clasificados correctamente. Además, se destaca que en este ejemplo los datos son **linealmente separables**: existe una función lineal que separa correctamente a los datos. En el caso (b) se ilustra la función encontrada por el algoritmo.

La ecuación (3.7) se puede reescribir de manera vectorial considerando los vectores $\mathbf{w} = (b, w_0, \cdots, w_{d-1}) \in \{1\} \times \mathbb{R}^d$, y $\mathbf{x} = (1, x_0, \cdots, x_{d-1}) \in \{1\} \times \mathbb{R}^d$, obteniendo:

$$h(\mathbf{x}) = \text{sign}(\mathbf{w}^T\mathbf{x}). \tag{3.8}$$

Gráficamente, el perceptrón es diagramado según ilustra la imagen 3.5, para el caso de $d = 2$. Dicho diagrama es el que, se verá posteriormente, escala al diagrama de las redes neuronales. El aprendizaje, es decir la búsqueda del mejor vector $\mathbf{w} = (b, w_0, \cdots, w_{d-1})$, define la función $g(\cdot) \in \mathcal{H}$, la hipótesis final.

3.2.2. El Perceptrón Multicapa

En la figura 3.4 se ilustra que los datos deben ser linealmente separables para que pueda encontrarse la función $g(\cdot)$ que separa correctamente a los datos. Cuando los datos no son linealmente separables, como es el ejemplo de la figura 3.6, otro enfoque es necesario. El enfoque no dista de lo implementado por el perceptrón: aquí se aproxima el resultado final $g(\cdot)$ por medio de dos perceptrones (figura 3.7). Se puede demostrar que funciones complejas, necesarias para clasificar datos que no son linealmente separables, se pueden obtener como combinaciones

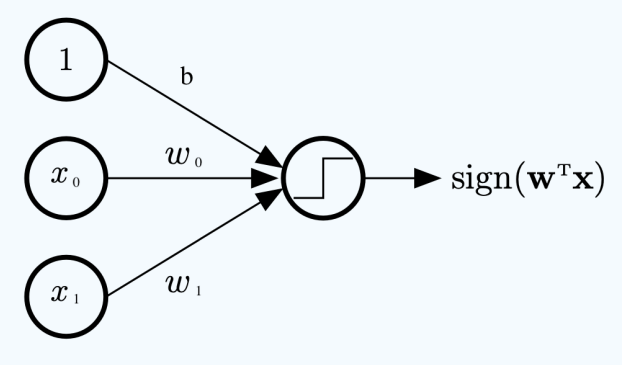


Figura 3.5: Representación gráfica del perceptrón para el caso en que $d = 2$. Las entradas son los círculos de la izquierda, denominados nodos, que encierran a los valores 1, x_0 y x_1, mientras que los pesos son representados sobre las flechas, b, w_0 y w_1. Todas las flechas confluyen en un nodo, en donde se realiza la suma $w_1x_1 + w_0x_0 + b$ y luego se aplica la función signo, representada por el escalón dentro del círculo que representa al nodo. El nodo de la derecha es denominado como capa de salida, mientras que los tres nodos de la izquierda son denominados como capa de entrada. Imagen tomada del libro *"Learning From Data"* [1].

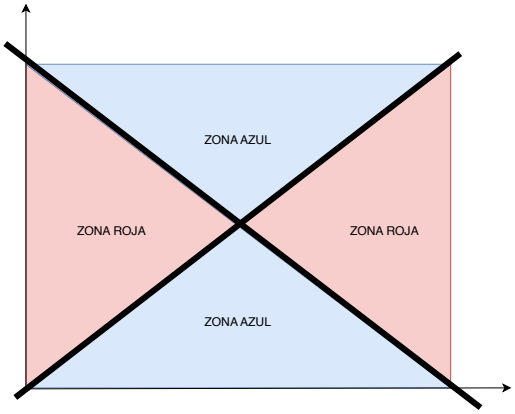


Figura 3.6: Ejemplo de datos que no son linealmente separables. Aquí el perceptrón como fue introducido no presenta una solución al problema, no puede separar las zonas rojas de las azules.

lineales de perceptrones. Para el caso de la figura 3.6, cuya solución son dos perceptrones combinados, se procede a computar primero cada uno de los perceptrones, y a continuación realizar la combinación de estos. Se ilustra en la figura 3.8 la representación gráfica de lo anterior. Esta arquitectura se denomina *perceptrón multicapa*, debido a que aparecen varias capas consecutivas, denominadas *capas ocultas*, además de las anteriormente introducidas capas de entrada y salida. Entre la capa de entrada y la primera capa oculta se computan los dos perceptrones, y entre las restantes capas y la capa de salida se computa la combinación lineal entre estos perceptrones. Se destaca que las capas fluyen hacia adelante, sin haber saltos de flechas hacia atras o esquivando capas.

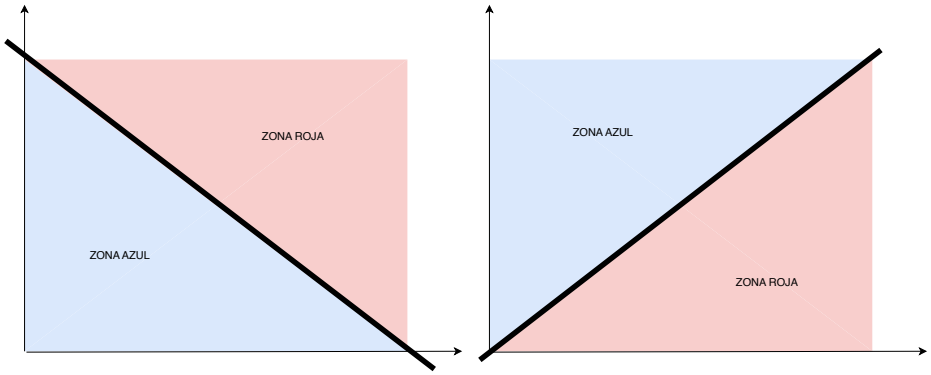


Figura 3.7: Solución al problema de la figura 3.6 por medio de la generación de dos perceptrones, ahora si cada uno siendo solución al problema, pues logran separar zonas rojas de azules.

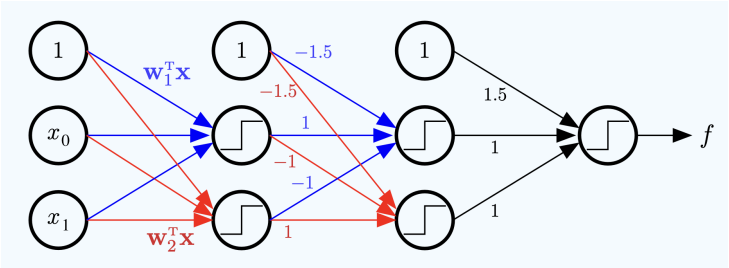


Figura 3.8: Representación gráfica de la combinación de perceptrones. Cada uno de los perceptrones está representado por una de las dos capas ocultas, y es solución a uno de los problemas de la imagen 3.7. Imagen tomada del libro *"Learning From Data"* [1].

3.2.3. Redes Neuronales

Continuando con el grado de complejidad de las funciones $f(\cdot)$ que se busca aproximar, las redes neuronales son una generalización del perceptrón multicapa, donde ahora las funciones de activación no siempre serán la función *signo*, sino que podrán ser otras funciones, a las que denominaremos $\theta(\cdot)$. Un caso usual es tomar $\theta(\cdot) = \tanh(\cdot)$, función diferenciable que genera una aproximación mas suave que la función signo, obteniendo así resultados más robustos en muchos casos de aplicación. Además, ahora los nodos serán denominados *neuronas*. El diagrama básico de una red neuronal se muestra en la figura 3.9. Nuevamente, se resalta el flujo hacia adelante de una capa a la otra, y la conexión entre todas las neuronas, sin esquivarse entre sí. Es por ello que este tipo de red neuronal es denominado *hacia adelante* y *totalmente conectado*. En inglés esto se traduce como *feed-forward-fully-connected* (abreviado como *FFN*).

A lo largo del entrenamiento, cada una de las neuronas de la capa l, adquiere un peso $w_{ij}^{(l)}$ específico, donde el subíndice i, j refiere al peso que va desde la neurona i de la capa $l-1$ hasta la neurona j de la capa l. El vector de entrada $\mathbf{x}^{(0)}$, con el supraíndice indicando la capa en cuestión, atraviesa cada una de las l capas,

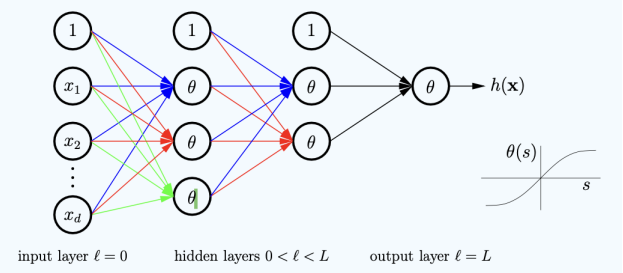


Figura 3.9: Arquitectura de una red neuronal profunda. Imagen tomada del libro *"Learning From Data"* [1].

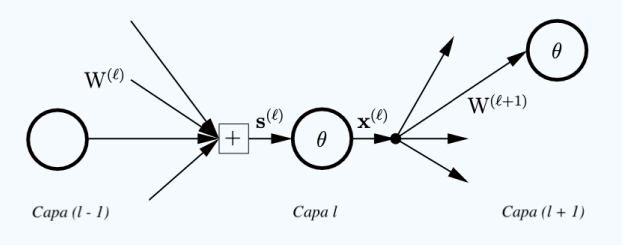


Figura 3.10: Esquema de propagación hacia adelante en una neurona l. Imagen tomada del libro *"Learning From Data"* [1]

.

siguiendo una propagación hacia adelante como se ilustra a continuación[1]:

$$\mathbf{x} = \mathbf{x}^{(0)} \xrightarrow{\mathbf{W}^{(1)}} \mathbf{s}^{(1)} \xrightarrow{\theta} \mathbf{x}^{(1)} \xrightarrow{\mathbf{W}^{(2)}} \mathbf{s}^{(2)} \xrightarrow{\theta} \mathbf{x}^{(2)} \cdots \xrightarrow{\mathbf{W}^{(L)}} \mathbf{s}^{(L)} \xrightarrow{\theta} \mathbf{x}^{(L)} = g(\mathbf{x}), \tag{3.9}$$

Las salidas de cada una de las capas, $\mathbf{x}^{(l)}$, se obtienen de aplicar la función de activación $\theta(\cdot)$ de cada capa al producto de la matriz de pesos $\mathbf{W}^{(l)}$ de la capa l, multiplicado por el vector $\mathbf{x}^{(l-1)}$, e incorporando el 1 correspondiente al *bias*, como fue mencionado en la sección 3.2.2:

$$\mathbf{x}^{(l)} = \left[1, \theta(\mathbf{s}^{(l)})\right]^T, \text{ donde: } \mathbf{s}^{(l)} = (\mathbf{W}^{(l)})^T \mathbf{x}^{(l-1)}. \tag{3.10}$$

Se puede visualizar lo establecido en la equación (3.10) en la figura 3.10.

3.3. Regularización

En el proceso de entrenamiento, es mandatorio realizar una división del conjunto de datos en subconjuntos de entrenamiento y test, respectivamente: X_{train} y X_{test}. El conjunto de entrenamiento X_{train} es el utilizado, como indica su nombre, a lo largo del proceso de entrenamiento del algoritmo, hasta llegar a un modelo final. Este modelo final será probado sobre un conjunto de datos que no ha sido visto o estudiado aún, el conjunto X_{test}. El objetivo de lo anterior es evitar el fenómeno

[1](tomado de "*Learning from Data*" [1])

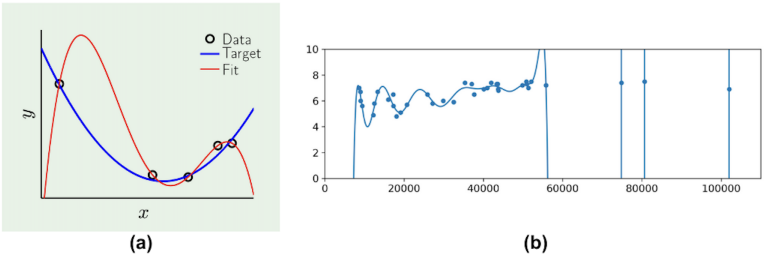


Figura 3.11: Ejemplos de sobreajuste. En (a) se ilustran las muestras, la función real $f(\cdot)$ y la hipótesis final $g(\cdot)$. Se visualiza que debido al ruido presente, la función $g(\cdot)$ verifica todos los puntos (x_i, y_i), debido al sobreajuste, pero no se aproxima a la función real $f(\cdot)$, resultando una función más compleja y que evidentemente dista de la original. En (b) no se ilustra la función $f(\cdot)$, pero sí se muestra que la hipótesis final $g(\cdot)$ verifica a todos los puntos (x_i, y_i), resultando en una función polinómica con un grado demasiado alto. Imágenes extraídas de [1] y [9], respectivamente.

de *sobreajuste* del algoritmo a los datos de entrenamiento. El sobreajuste implica que se está aprendiendo de los datos de entrenamiento más de lo que se puede, esto implica que se está *memorizando* al conjunto de entrenamiento. La *regularización* tiene como objetivo combatir el fenómeno del sobreajuste. Se evidencia el sobreajuste cuando el algoritmo obtiene óptimos resultados de desempeño sobre el conjunto de entrenamiento, pero esto no se ve reflejado en el conjunto de test. Como se observa en la figura 3.11 (a), donde se busca ajustar un polinomio, se visualiza el sobreajuste del modelo obtenido (rojo en la figura) en contraposición con la función que se pretende predecir (la azul). El modelo obtenido efectivamente verifica todos los puntos del conjunto de entrenamiento (*data*), pero en el afán de cumplir esto aumenta su complejidad y obtiene en consecuencia un error demasiado grande. Un ejemplo similar de sobreajuste polinomial se puede visualizar en la figura 3.11 (b).

Durante el proceso de entrenamiento, es usual generar una nueva división sobre X_{train} y extraer un conjunto que se denomina de *validación*, X_{val}. El objeto de esto está en poder replicar el enfoque que se aplica con el conjunto X_{test}, pero en la etapa de entrenamiento del modelo. Esto permite al algoritmo poder evitar el sobreajuste con mayor anticipación y aplicando técnicas de solución más robustas. Algunas de estas técnicas aplican restricciones al algoritmo durante su entrenamiento, para evitar ese aprendizaje excesivo del conjunto de entrenamiento. A continuación, serán mencionadas las técnicas de regularización empleadas en este proyecto.

3.3.1. Dropout

Dropout es una de las técnicas más populares para el trabajo con redes neuronales. Se emplea cuando una probabilidad p determina si cada una de las neuronas pertenecientes a la red neuronal será dejada de lado (*dropped out*) durante una época específica del entrenamiento del algoritmo. El parámetro p es denominado

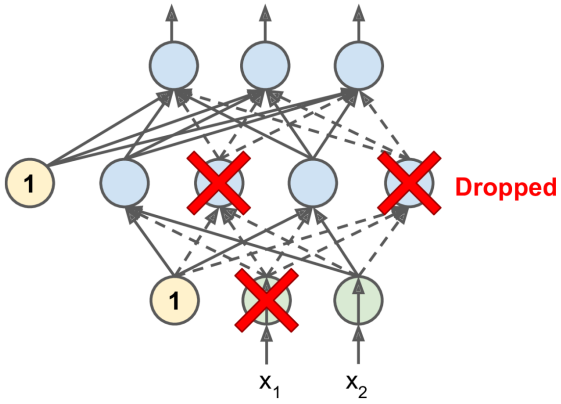


Figura 3.12: Ejemplificación del *dropout*, en rojo se identifican las neuronas que fueron descartadas para una determinada época del entrenamiento. Imagen tomada del libro *"Hands on Machine Learning"* [9]

ratio de dropout, y la técnica se ilustra en la figura 3.12. Que una neurona no se tenga en cuenta durante una época específica del entrenamiento, se obtiene asignando para dicha época un peso $w_{ij}^{(l)} = 0$.

3.3.2. Early stopping

La técnica *early stopping*, que significa *detener antes de tiempo*, aplica para algoritmos que se ejecutan durante múltiples épocas, como las redes neuronales mencionadas en la subsección 3.3.1. Su uso es sencillo: se registran las métricas de desempeño a lo largo de las épocas de entrenamiento del algoritmo, y se establece un parámetro que es la cantidad de épocas máximas sin que la métrica mejore. Superada esta cantidad de épocas, el algoritmo finaliza el entrenamiento.

Capítulo 4

Modelos Secuenciales

En este capítulo se describen los modelos secuenciales, en particular las RNN, con sus limitaciones que dan origen a las neuronas LSTM, y finalmente los *Transformers* que utilizan la estructura *Encoder-Decoder* y un novedoso mecanismo de *attention*, más conocido como *Self-Attention*, presentado por Vaswani et al.,2017 [21].

4.1. Redes Neuronales Recurrentes (RNN)

Una red neuronal recurrente es similar a una *feed-forward neural network*, ver sección 3.2.3, con la diferencia de que es capaz de utilizar información de eventos anteriores para determinar la salida en el instante actual, por lo cual se dice que tiene "memoria". Esto se puede ilustrar considerando únicamente una neurona que en un instante de tiempo discreto n recibe su entrada $\mathbf{x}[n]$, así como su salida en el instante de tiempo anterior $\mathbf{y}[n-1]$. Repitiendo esta idea para instantes de tiempo posteriores, sucede lo que se denomina "desenrollar la red en el tiempo", como se puede ver en la figura 4.1. Se destaca que en la figura, la representación desenrollada de la neurona es una representación temporal de la misma neurona para cada instante de tiempo pasado. Cada una de las capas, tiene dos matrices de pesos, una para el vector de entrada, $\mathbf{x}[n]$, y otra para la salida de la capa en el instante de tiempo anterior, $\mathbf{y}[n-1]]$, $\mathbf{P}_x$ y $\mathbf{P}_y$, respectivamente. Consecuentemente, la salida en el instante de tiempo n se calcula siguiendo el modelo usual de las redes neuronales:

$$\mathbf{y}[n] = \phi\left(\mathbf{P}_x^T\mathbf{x}[n] + \mathbf{P}_y^T\mathbf{y}[n-1] + \mathbf{b}\right), \tag{4.1}$$

donde $\phi(\cdot)$ es la función de activación y $\mathbf{b}$ es el vector de sesgo o *bias*. La ecuación 4.1 es la reformulación de la ecuación (3.10) para RNN.

El estado de una neurona en el instante de tiempo n es denominado $\mathbf{u}[n]$, y es función de la entrada a la neurona en dicho instante de tiempo, y del estado en el instante de tiempo anterior, es decir: $\mathbf{u}[n] = f(\mathbf{u}[n-1], \mathbf{x}[n])$. De la misma forma, la salida de la neurona $\mathbf{y}[n]$, será también función del estado anterior $\mathbf{u}[n-1]$ y de la entrada actual $\mathbf{x}[n]$.

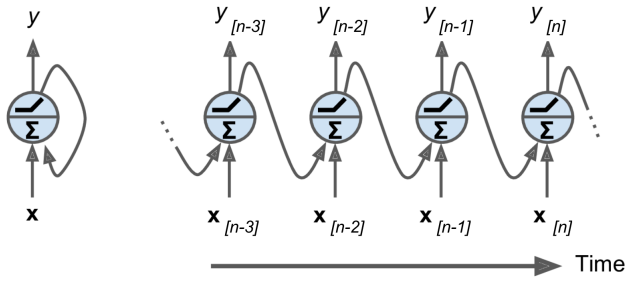


Figura 4.1: Neurona recurrente a la izquierda y desenrollado de la neurona en el tiempo a la derecha. Imagen tomad del libro *"Hands on Machine Learning"* [9]

Existen diferentes clasificaciones de las RNN de acuerdo a cómo es su entrada y su salida, en particular si la entrada y/o la salida es una secuencia de vectores o un único vector. Si tanto la entrada como la salida son secuencias de vectores, se clasificará como una *seq-to-seq network* (figura 4.2 (a)), utilizada usualmente para la predicción de series temporales. Si la entrada es un vector y la salida una secuencia de vectores, entonces es una *vec-to-seq network* (figura 4.2 (b)). Si se obtiene un único resultado, y no una secuencia, se denomina *seq-to-vec network*, donde en realidad se obtiene nuevamente una secuencia a la salida, pero se ignoran todas las salidas excepto la última, como se puede ver en la figura 4.2 (c). Por último en la figura 4.2 (d) se presenta la estructura de tipo *Encoder-Decoder*, en la cual el *Encoder* es una estructura *sequence-to-vector* y el *Decoder* una estructura *vector-to-secuence*. Esta clase de modelos en dos pasos, ha demostrado presentar ventajas asociadas a la separación de tareas, como por ejemplo en el procesamiento del lenguaje natural (NLP), que se mencionará posteriormente. De todo lo anterior, se observa que en el trabajo de las RNN con series temporales, no se necesita de entradas con dimensiones predeterminadas, sino que las mismas pueden ser variables.

La limitación principal de las *RNN* es que conforme el tiempo avanza, su memoria se va perdiendo debido a las sucesivas transformaciones sobre los datos. En el caso de procesamiento de lenguaje natural sucede que a medida que se avanza en el procesamiento de un texto, va olvidando las palabras del principio. Aquí se menciona como tiempo, pero cabe destacar que en lugar de tiempo n, se puede tratar de una posición n en un vector, como es el caso en aplicaciones dentro de la genómica, por ejemplo, o en el caso del procesamiento del lenguaje natural, la posición n será una palabra en un enunciado. La solución a esta problemática son las neuronas *long-term-memory*, de las cuales la clase más utilizada son las *long-short-term-memory* (*LSTM*).

4.1.1. LSTM

En la figura 4.3 se presenta la arquitectura de una celda LSTM, donde los vectores de entrada $\mathbf{x}(t)$, y $\mathbf{u}[n-1]$ (que guarda la información del estado anterior), son direccionados hacia cuatro capas totalmente conectadas (FFN), las cuales se

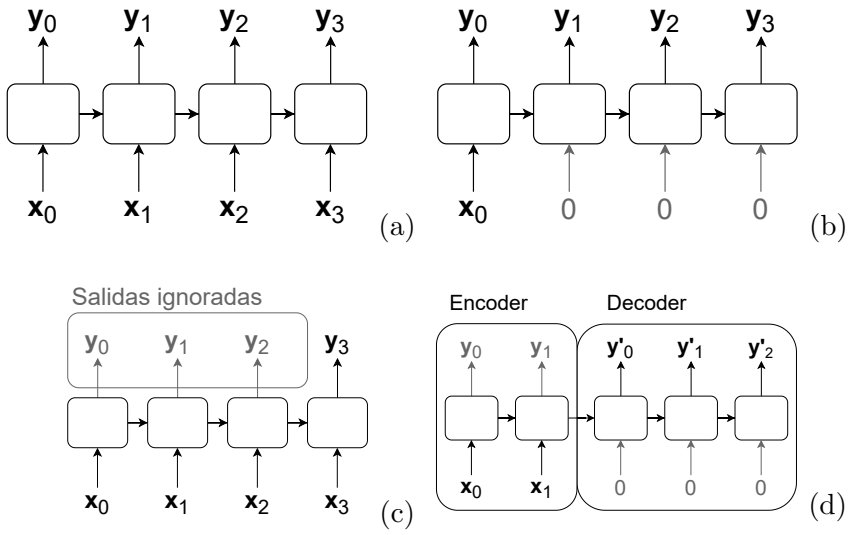


Figura 4.2: Tipos de RNN: *sequence-to-sequence* (a), *vector-to-sequence* (b), *sequence-to-vector* (c), y *Encoder-Decoder* (d).

conectan a tres compuertas: *Forget gate*, *Input gate* y *Output gate*. La salida de las FFN se determina en función del vector de entrada $\mathbf{x}[n)]$ y el vector de corto plazo del estado anterior $\mathbf{u}[n-1]$. Luego pasa por una función de activación, logística o tangente hiperbólica. La principal diferencia con una neurona simple, es que en este tipo de neuronas la salida devuelve además de $\mathbf{u}[n]$ que contiene información de memoria a corto plazo, un vector asociado a la memoria a largo plazo llamado $\mathbf{l}[n]$.

A continuación se describe cada una de las compuertas mencionadas anteriormente, destacando que el detalle de las ecuaciones se encuentra en el anexo A:

- *Forget Gate*: controla qué partes del vector de largo plazo del estado anterior ($\mathbf{l}[n-1]$) deben ser borradas, de ahí su nombre *Forget* (olvidar). Esto se controla por medio del vector $\mathbf{r}[n]$.
- *Input Gate*: controla qué partes de $\mathbf{s}[n]$ (la salida de la FFN con función de activación tangente hiperbólica) deben ser agregadas al vector de largo plazo del estado actual ($\mathbf{l}[n]$). Esto se controla por medio del vector $\mathbf{i}[n]$.
- *Output Gate*: controla qué partes del vector de largo plazo deben ser leídas y enviadas al vector de corto plazo del estado actual ($\mathbf{u}[n]$). Esto se controla por medio del vector $\mathbf{o}[n]$.

El vector de estado de memoria de largo plazo del instante de tiempo anterior, $\mathbf{l}[n-1]$, ingresa a la neurona, donde por medio de una compuerta (*forget gate*) elimina parte de su memoria, para luego incorporar memoria nueva mediante la operación de la suma (*addition*), memoria seleccionada por medio de otra compuerta (*input gate*). El resultado de estas operaciones es el vector de estado de

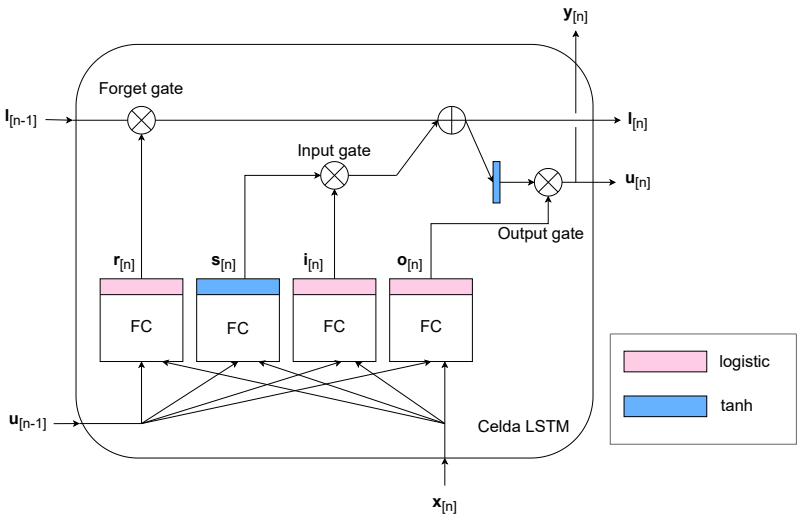


Figura 4.3: Arquitectura de una celda LSTM, compuertas: *Forget gate*, *Input gate* y *Output gate*, controladas por la salida de tres capas *FFN* con función de activación logística (*logistic*) y una cuarta *FFN* con función de activación tangente hiperbólica (*tanh*). (Imagen basada en el libro *Hands on Machine Learning* [9]).

memoria de largo plazo $\mathbf{l}[n]$, el cual se copia para una de las salidas de la neurona, así como también hacia una tercer compuerta (*output gate*), atravesando una función *tanh* para adicionar parte de la información al vector $\mathbf{u}[n]$. Además se copia la información del estado actual $\mathbf{u}$[n] a la salida $\mathbf{y}$[n].

En resumen, una neurona LSTM es capaz de aprender a reconocer una entrada importante, que es el rol de la compuerta de *input*, almacenar información de esta entrada en el vector de estado de memoria de largo plazo por el tiempo que sea necesario, que es el rol de la compuerta *output*, y eliminar esta información cuando sea necesario por medio de la compuerta *forget*, lo cual mejora la memoria de la RNN.

Si bien la implementación de neuronas LSTM mejoró la performance de las RNN, por Vaswani et al.,2017 [21] plantearon una nueva arquitectura llamada *Transformers* de tipo *Encoder-Decoder*, basada en mecanismos de atención, que presentó mejores resultados con un menor tiempo de entrenamiento, superando ampliamente a los mejores modelos conocidos hasta el momento.

4.2. Transformers

El *Transformer* es un modelo presentado en 2017 [21] para realizar tareas de Procesamiento de Lenguaje Natural (PLN), principalmente traducciones de texto inglés a francés e inglés a alemán. Presenta una estructura *Encoder-Decoder* basada en el mecanismo de *attention* (en español: atención) que tiene como principal característica el carecer de capas recurrentes. El mecanismo de atención logra establecer dependencias globales entre secuencias de datos, logrando identificar que

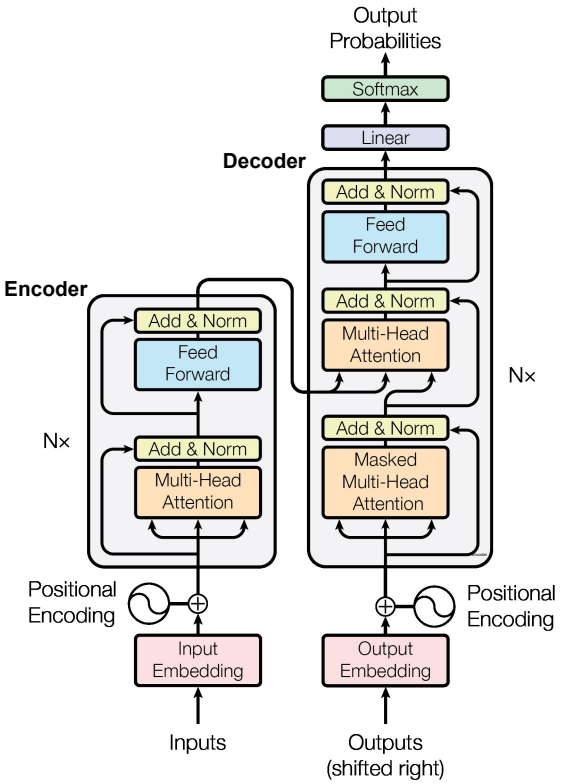


Figura 4.4: Arquitectura del modelo Transformer presentada en el paper "Attention Is All You Need" [21].

posiciones de una secuencia tienen estrecha relación sobre determinada posición de la otra y de forma más general, que secuencias son importantes sobre otras. En particular, se utilizará el mecanismo *Self-Attention* que logra relacionar diferentes posiciones de una mismas secuencia [21].

Sea $\mathbf{X} = [\mathbf{x}_0, \cdots, \mathbf{x}_{p-1}]$, la matriz de los vectores columna $\mathbf{x}_i$ ordenados, donde $\mathbf{x}_i \in \mathbb{R}^{d_\mathcal{M}}$, el objetivo del *Encoder* es mapearlos a una secuencia continua representada por $\mathbf{Z} = [\mathbf{z}_0, \cdots, \mathbf{z}_{q-1}]$ con $\mathbf{z}_i \in \mathbb{R}^{d_\mathcal{M}}$, donde $d_\mathcal{M}$ es un hiperparámetro del modelo que define la dimensión en la que se representan los datos para entrar al *Encoder*. El *Decoder* tiene como función generar una salida de símbolos $\mathbf{Y} = [\mathbf{y}_0, \cdots, \mathbf{y}_{m-1}]$ donde $\mathbf{y}_i \in \mathbb{R}^{d_\mathcal{M}}$, un elemento a la vez.

En la imagen 4.4 se presenta el modelo completo del *Transformer* con el detalle de los bloques que lo componen. A la izquierda se encuentra la primera etapa del modelo compuesta por la capa de *Embeddings*, la etapa de *Positional Encoding* y el *Encoder*. A la derecha se encuentra el módulo *Decoder*, al cual le entran las salidas previas del *Transformer* y la salida del *Encoder*. Nuevamente, previo a entrar al *Decoder* se encuentran los módulos de *Embeddings* y *Positional Encoding*.

A continuación se detallará el funcionamiento de cada uno de los bloques que conforman el *Transformer* tal como fue presentado por Vaswani et al.,2017.

4.2.1. *Embeddings* y *Positional Encoder*

Previo a entrar al *Encoder* y *Decoder*, las secuencias de entrada pasan por el bloque de *Embeddings*. La función de este es separar las oraciones en palabras o pedazos de ellas (*tokens*) que tienen asignados un índice en una tabla de posibles entradas (por ejemplo: palabras de un idioma si la entrada es un texto o todas combinaciones de largo seis formadas por unos y ceros, como se verá en el ejemplo de la sección 4.3). Cada uno de los índices es mapeado a un vector, denominado *embedding*, de dimensión $d_{\mathcal{M}}$. A la salida de este módulo, cada palabra quedará representada por un vector de números que guarda información de la dos palabras en el contexto.

El *Positional Encoding* consta de sumar una constante en cada una de las posiciones de la secuencia de salida del *embedding* que depende de esta posición (i) y la posición que ocupaba el elemento original en la secuencia de entrada (pos). Este módulo responde a que, por como está estructurado el modelo (sin recurrencias ni convoluciones), la información posicional se pierde. Se codifica de acuerdo a las siguientes ecuaciones:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{\mathcal{M}}}}\right) \tag{4.2}$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{\mathcal{M}}}}\right) \tag{4.3}$$

En este punto, la secuencia de palabras se encuentra representada como una matriz, donde cada columna está asociada a una palabra y es un vector de tamaño $d_{\mathcal{M}}$ al que se le sumó una constante de acuerdo a su posición en el texto. Esta matriz es la que entra en el *Encoder* o *Decoder*. Un ejemplo de ella se muestra a la izquierda en la figura 4.5

4.2.2. Mecanismo de Atención

Los módulos de atención son la parte principal de los *Transformers*, utilizados para trazar dependencias globales entre secuencias. Tiene como una de sus principales ventajas que logra trabajar con secuencias largas y puntos distantes entre sí. En el *Transformer* se presenta en tres variantes:

- Atención cruzada (*Cross-Attention*): calcula las dependencias entre dos secuencias distintas. En el modelo, se utiliza para trazar las dependencias entre la secuencia que entra directo al *Decoder* y la que sale del *Encoder*, como se puede ver en el segundo bloque de atención que tiene el *Decoder* en la figura 4.4.

- Auto-atención (*Self-Attention*): traza las dependencias entre una secuencia con sí misma. Se observa en el *Encoder* como las tres entradas al módulo provienen de la misma secuencia.

- Atención enmascarada (*Masked-Attention*): traza dependencias entre una secuencia con si misma, enmascarando las posiciones siguientes a la actual

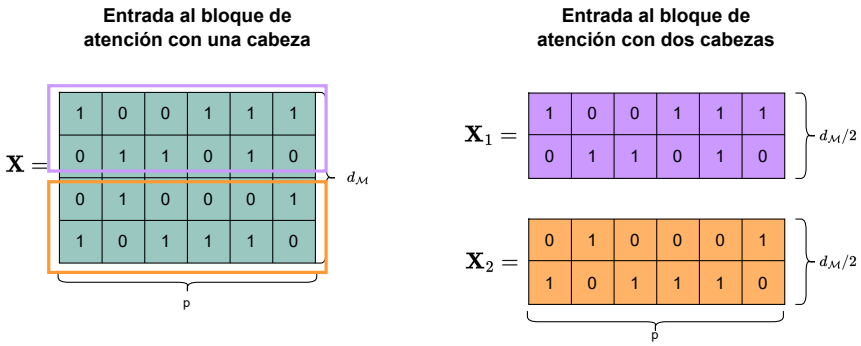


Figura 4.5: A la izquierda, matriz de entrada al *Self-Attention* con una sola cabeza. A la derecha, como se separan las matrices para ser entradas del *Self-Attention* con dos cabezas.

para evitar la no causalidad. La prediccion actual se realiza con la información de posiciones pasadas pero no de posiciones futuras.

Los bloques de atención utilizados en el modelo (figura 4.4) en todos los casos se definen como *Multi-Head Attention*. Se define una cabeza como un único bloque de atención. En el modelo *Multi-Head* se tienen h cabezas que trabajan de forma paralela. La entrada a cada bloque, que de haber una única cabeza sería una matriz de dimensión $(d_{\mathcal{M}}, p)$, pasa a ser una matriz de dimensión $(d_{\mathcal{M}}/h, p)$ donde cada una es una submatriz de la principal. Lo descrito anteriormente se presenta en la figura 4.5, donde a la izquierda se puede ver la matriz que entraría al bloque de atención (después de los módulos *Positional Encoding* asociados tanto el *Encoder* como al *Decoder*), en caso de tener una única cabeza y a la derecha las dos submatrices si se tuvieran dos cabezas.

Cada una de las cabezas trabaja con su propia matriz de pesos y a la salida se concatenan las matrices resultantes, volviendo a tener una matriz de dimensión $(d_{\mathcal{M}}, p)$.

La descripción que se presenta a continuación considera una única cabeza en la entrada a cada bloque de atención.

¿Cómo funciona cada bloque de atención?

A continuación se presenta el paso a paso del sistema a nivel matemático:

$$1)\ \mathbf{Q} = \mathbf{X}^T \times \mathbf{P_q},\ \mathbf{K} = \mathbf{M}^T \times \mathbf{P_k},\ \mathbf{V} = \mathbf{M}^T \times \mathbf{P_v} \tag{4.4}$$

$$2)\ \mathbf{L} = \mathbf{Q} \times \mathbf{K}^T, \text{ donde: } \mathbf{L} \in \mathbb{R}^{q \times m} \tag{4.5}$$

$$3)\ \mathbf{W} = \text{Softmax}\left(\frac{\mathbf{L}}{\sqrt{k}}\right), \text{ donde: } \mathbf{W} \in \mathbb{R}^{q \times m} \tag{4.6}$$

$$4)\ \mathbf{O} = \mathbf{W} \times \mathbf{V}, \text{ donde: } \mathbf{O} \in \mathbb{R}^{q \times k} \tag{4.7}$$

$$5)\ \mathbf{Y} = \mathbf{O} \times \mathbf{P_{OUT}}, \text{ donde: } \mathbf{Y} \in \mathbb{R}^{q \times d_{\mathcal{M}}} \tag{4.8}$$

donde $\mathbf{X}$ y $\mathbf{Z}$ con dimensiones $d_{\mathcal{M}} \times p$ y $d_{\mathcal{M}} \times m$ son las matrices de entrada (en la figura 4.4 $\mathbf{M}$ es la salida del encoder que entra al *Multi-Head Attention* y $\mathbf{X}$ es

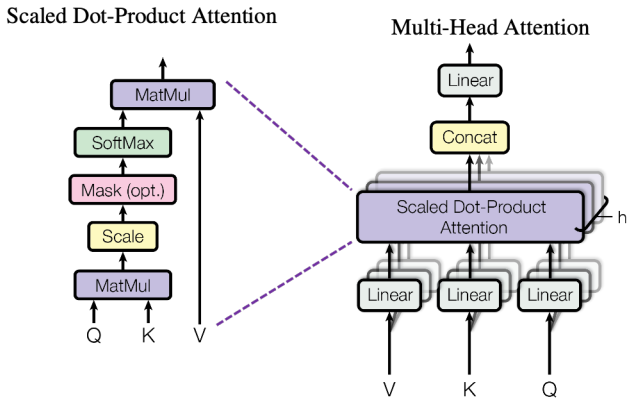


Figura 4.6: Arquitectura del módulo *Multi-Head Attention* con el detalle de las operaciones que se realizan adentro del mismo [21]

la salida del módulo de atención anterior). Las matrices $\mathbf{P}_Q$, $\mathbf{P}_K$, y $\mathbf{P}_V$ son matrices de pesos que al iniciar el entrenamiento se inicializan aleatoriamente. Tienen dimensiones $q \times k$, $m \times k$ y $m \times k$ respectivamente, donde k es un hiperparámetro.

Un ejemplo de *Self-Attention* se muestra en la figura 4.8. El resultado del producto entre $\mathbf{Q}$ y $\mathbf{K}$ es un "puntaje" (o *score*) de similitud entre los vectores de las mismas. Cuanto más alta sea la entrada, más "atención" se le debe prestar a esa posición de la secuencia, mientras que las posiciones que influyen menos tienen valores cercanos a cero. El producto por la matriz $\mathbf{V}$ se realiza para seleccionar la información relevante entre las secuencias de entrada.

Como fue mencionado previamente, el módulo *Self-Attention* es matemáticamente igual al *Cross-Attention*, con la diferencia de que calcula las dependencias entre una secuencia con sí misma, por lo que la entrada es una misma matriz $\mathbf{X}$. En la figura 4.6 se muestra el diagrama de bloques para el módulo *Multi-Head Attention* con el detalle de las operaciones que se realizan. En el apéndice B se muestra un ejemplo numérico de las operaciones que realiza este bloque.

4.2.3. *Feed-Forward* y *Add & Norm*

El otro módulo importante que compone tanto al *Encoder* como al *Decoder* es una red neuronal *Feed-Forward* definida en la sección 3.2.3. Esta capa realiza dos transformaciones lineales con una activación ReLU y se aplica a cada una de las posiciones que componen la entrada, como se resume en la siguiente ecuación:

$$\mathrm{FFN}(\mathbf{X}) = \mathrm{ReLU}(\mathbf{w}_1\mathbf{X} + \mathbf{b}_1)\mathbf{w}_2 + \mathbf{b}_2, \tag{4.9}$$

donde $\mathbf{X}$ es la entrada a la red *FFN* y $\mathbf{w}_i$ y $\mathbf{b}_i$ son vectores conformables. Esta capa se utiliza para tener representaciones más abstractas y de mayor nivel.

A la salida del bloque de atención y también del bloque *FFN* se encuentra una capa de normalización denominada *Add & Norm*. Ésta, es una capa recurrente que suma la entrada y la salida del bloque previo. Una vez sumados, normaliza el

resultado, es decir, hace que las distintas entradas de un mismo vector sumadas den uno.

4.2.4. Capa de salida

En última instancia, a la salida del *Decoder*, se encuentra la capa de salida compuesta por una capa lineal y una *Softmax*. La capa lineal es una red *feed-forward* que actúa como clasificador, con dimensión igual a la cantidad de clases de salida. Luego se pasa a la *Softmax* que devuelve la probabilidad de que pertenezca a una clase u otra. A continuación se toma el índice de la entrada con el valor máximo de probabilidad a posteriori y se elige como salida la palabra asociada a ese índice.

4.2.5. Modelo Completo

El *Transformer* presenta N módulos *Encoder* dispuestos en serie. El texto a traducir es la entrada al primer *Encoder* y la salida de este es la entrada al siguiente. De esta forma, se encuentran los N módulos concatenados y la salida del último es la que pasa a la etapa de decodificación. Cada *Encoder* tiene un módulo *Multi-Head Self-Attention*, seguido de una capa *Add & Norm*, luego una FFN y finalmente otra capa *Add & Norm*. Se cuenta también con N módulos *Decoder* dispuestos de igual forma que los *Encoder*, donde la entrada del bloque N es la salida del bloque $N-1$. Al primer *Decoder* entran las predicciones previas ya realizadas por el *Transformer*. Además, a todos los *Decoder* les entra la salida del último *Encoder*. Cada bloque *Decoder* presenta un módulo *Masked Multi-Head Attention*, otro *Multi-Head Attention* al que entra la salida del último *Encoder* y una FFN. A la salida de los tres bloques se encuentra nuevamente una capa *Add & Norm*. Finalmente, luego del *Decoder*, hay una capa lineal seguida de una *Softmax* encargadas de realizar las predicciones eligiendo de las posibles salidas, la que tiene mayor probabilidad.

Para una mejor comprensión de la arquitectura del *Transformer* se presenta en la siguiente sección un ejemplo que ilustra con números cada una de las operaciones que realiza el *Transformer*.

4.3. Desglosando el *Transformer* con un ejemplo

En esta sección se presenta un ejemplo tomando como entrada del *Transformer* una secuencia de unos y ceros. Se busca analizar como se comporta el modelo a medida que el vector va avanzando por los distintos módulos. Se realizó el análisis con este tipo de vector debido a que en la presente investigación los datos de entrada tienen esta estructura (secuencia de unos y ceros) pero de mayor dimensión.

Sea $\mathbf{x}$ el vector de dimensión $p = 6$ que se considera como una única palabra. Este vector pasa por el módulo *Embeddings*, considerado de dimensión $d_{\mathcal{M}} = 4$ para el ejemplo. A la salida se tiene una matriz $\mathbf{IE}$ de dimensión $(4, 6)$ que pasa por el *Positional Encoder*, sumando en todas las entradas una misma constante

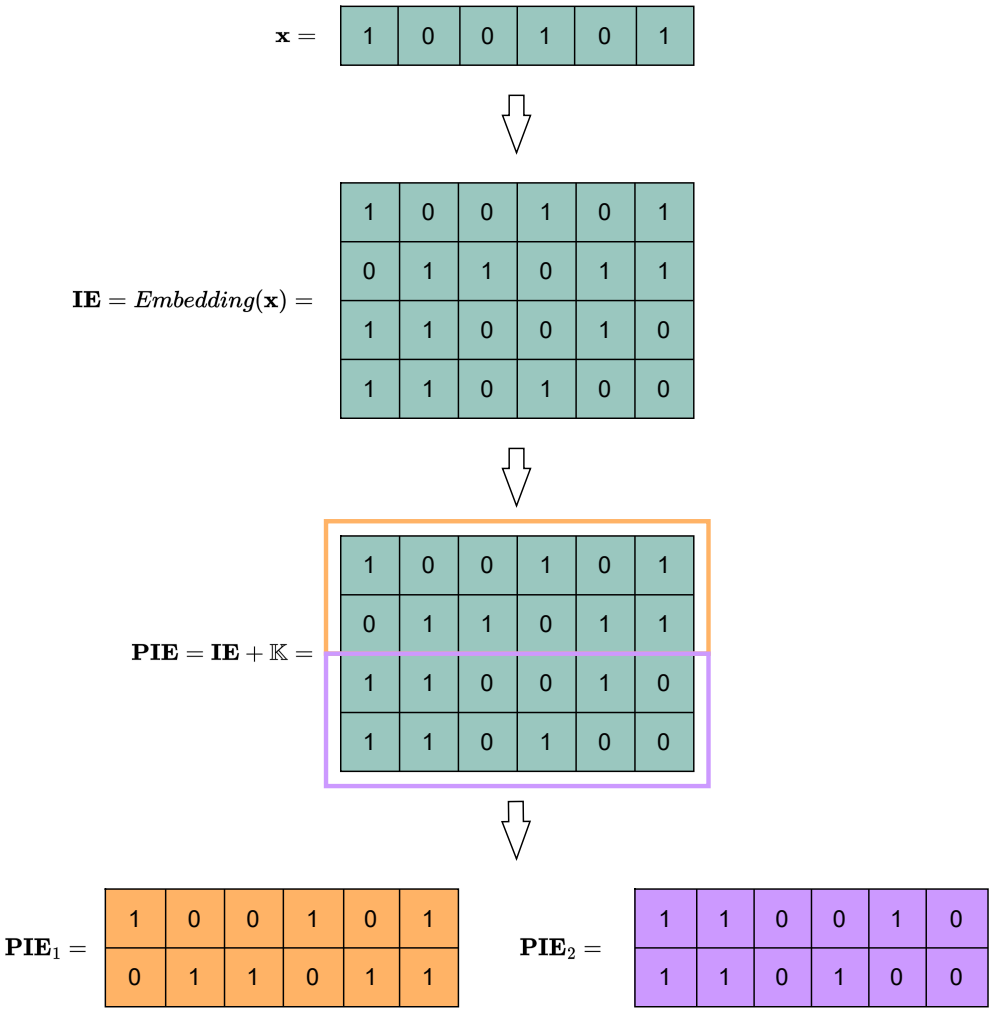


Figura 4.7: Definición del vector de entrada **x**, los resultados que se obtienen al pasar por la capa de *Embeddings* (matriz **IE**) y *Positional encoding* (matriz **PIE**) y la separación en dos matrices para el *Multi-Head Attention*

$\mathbb{K}$ (ya que es una única palabra). Para el ejemplo, esta entrada se toma como la palabra de posición cero, por lo que la constante que suma el *Positional Encoder* de acuerdo a la ecuación (4.2) es $\mathbb{K} = 0$.

Se considera un modelo donde los módulos de atención tienen dos cabezas, por lo que $h = 2$. La matriz de entrada al *Encoder* es de dimensión $(4, 6)$ por lo que, para definir la matriz que entra a cada cabeza, se la separa en dos matrices de dimensión $(2, 6)$, como se describió previamente en la sección 4.2.2. Se definen entonces, las matrices $\mathbf{PIE_1}$ y $\mathbf{PIE_2}$ que entraran a la primera y segunda cabeza del *Multi-Head Attention*, respectivamente.

En la imagen 4.7 se muestran los pasos descritos previamente, desde que se define el vector de entrada hasta que sale del *positional encoder* pronto para entrar al *Encoder*.

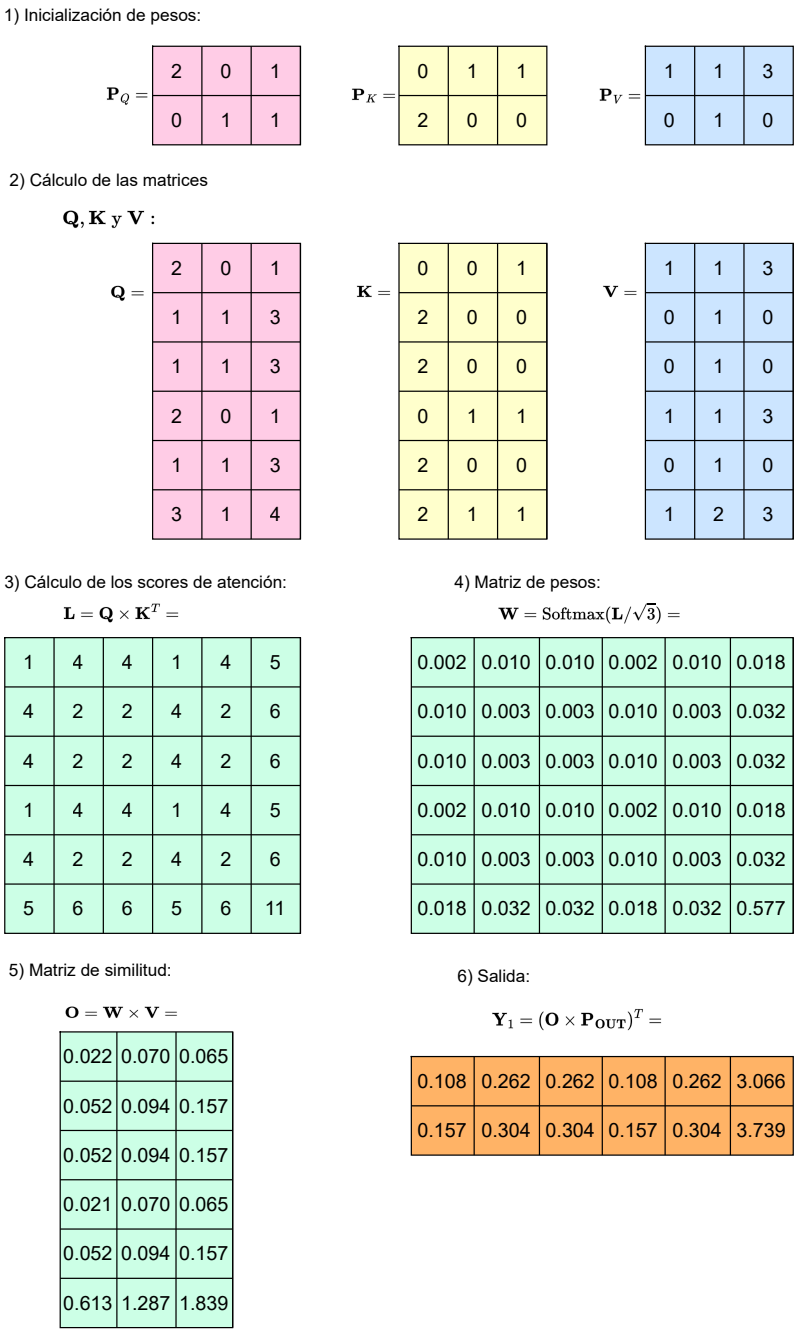


Figura 4.8: Paso a paso de las operaciones realizadas dentro de una cabeza del *Self-Attention*

Se detalla el paso a paso de las operaciones realizadas dentro de la primera cabeza, siendo análogas para la otra de la cual se expresará directamente el resultado.

Dentro de la cabeza del *Self-Attention* se inicializan las matrices de pesos $\mathbf{P_Q}$, $\mathbf{P_K}$ y $\mathbf{P_V}$ que deberan ser conformables con $\mathbf{PIE}_1^T$ de dimensión $(6, 2)$. Se definen

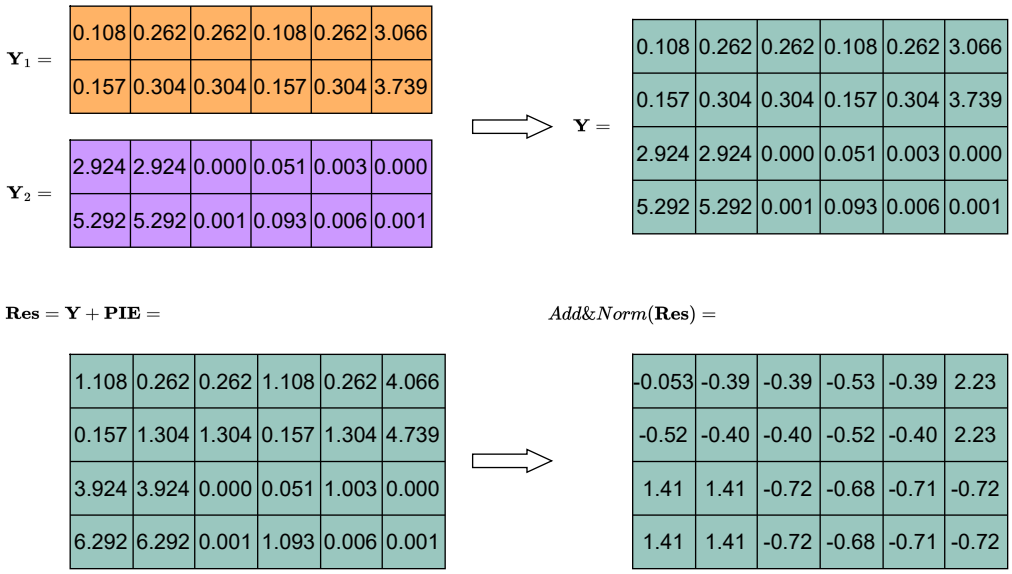


Figura 4.9: Concatenación de la salida de las dos cabezas del *Multi-Head Attention* con la posterior conexión residual y capa de normalización.

entonces, matrices de dimensión $(2, k)$, donde k es el hiperparámetro definido en la sección 4.2.2, tomando $k = 3$. La figura 4.8 muestra el cálculo numérico realizado para obtener $\mathbf{Y}_1$, la matriz de salida de la primera cabeza.

Sea $\mathbf{Y}_2$ la salida de la segunda cabeza del *Multi-Head Attention*, la matriz $\mathbf{Y}$ será la concatenación de $\mathbf{Y}_1$ e $\mathbf{Y}_2$. Una vez concatenadas, la salida se suma a la entrada y pasa por la capa de normalización, como se muestra en la figura 4.9.

La matriz normalizada es la que luego entra a la FFN, para luego pasar una vez más por un bloque residual y de normalización. Es necesario que la salida del *Encoder* tenga igual dimensión que la entrada, ya que los N encoders dispuestos en serie tienen la misma estructura y la salida de uno es la entrada del siguiente. La dimensión de salida de la FFN se define teniendo esta consideración.

La matriz del último *Encoder* tiene la información de qué posiciones de las secuencias de entrada son más importantes para una correcta predicción y es la información que luego entra al *Decoder* junto con las predicciones ya realizadas. Tomando para el ejemplo que la predicción previa fue un único vector de dimensión $p = 6$ que se denomina $\mathbf{y}_{prev}$, se pasa por el módulo de *Output Embeddings* y *Positional Encoding*. Nuevamente se considera que el módulo *Multi-Head* tiene $h = 2$ cabezas, por lo que la matriz de entrada se separa en dos, tal como se muestra en la figura 4.10.

El primer módulo *Attention* del *Decoder* se denomina *Masked Multi-Head Attention*. Calcula el puntaje de atención entre las predicciones ya realizadas y la palabra actual a predecir, pero debe evitarse que tenga acceso a la información de posiciones futuras. Para eso se crea una máscara, como se muestra en la figura 4.11, que se aplica previo a calcular el *Softmax* dentro del bloque. A partir de las predicciones previas y los pesos $\mathbf{P}_Q$, $\mathbf{P}_K$ y $\mathbf{P}_V$ se definen las matrices $\mathbf{Q}$, $\mathbf{K}$ y $\mathbf{V}$

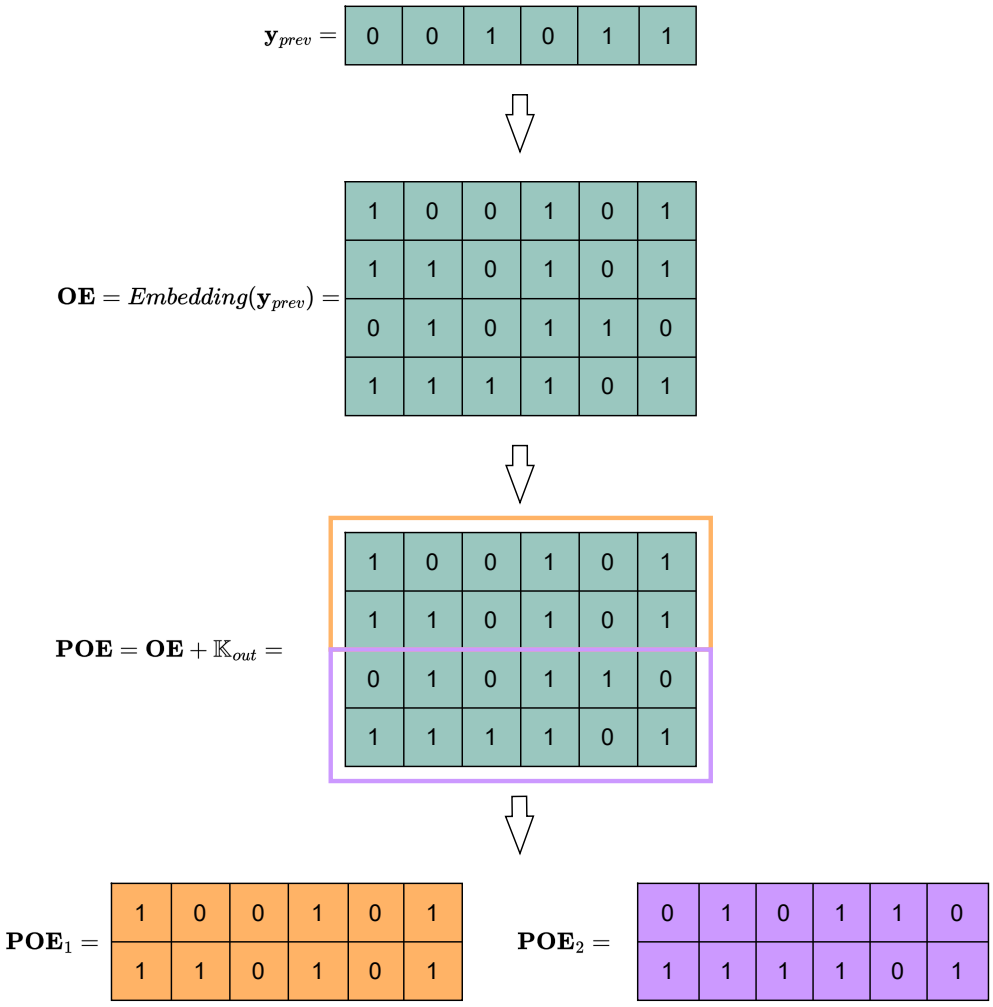


Figura 4.10: Vector de entrada al *Decoder* (predicción previa) que pasa por el módulo de *Output Embeddings*, *Positional Encoding* y se separa en dos para entrar a cada una de las cabezas del *Self-Attention*

.

tal como se procedió con el módulo de atención del *Encoder*. A la salida de este bloque se tiene una matriz de dimensión $(4, 6)$, al igual que en la entrada, que pasa por la capa *Add & Norm*.

La siguiente instancia es el módulo *Multi-Head Attention*, que por primera vez en el modelo calcula atención cruzada. Toma la salida del módulo previo del *Decoder* para la matriz $\mathbf{Q}$ y la salida del bloque de *Encoders* para definir $\mathbf{K}$ y $\mathbf{V}$. Una vez obtenidas las matrices, se procede de igual manera que la ilustrada en la figura 4.8. A la salida, se tienen los puntajes de atención para la secuencia, teniendo en cuenta la información obtenida por el *Encoder* de qué parte de la secuencia "es importante" para cada posición. Luego, nuevamente hay un bloque *Add & Norm*, una FFN y otro *Add & Norm* a la salida de esta, como fue descrito previamente.

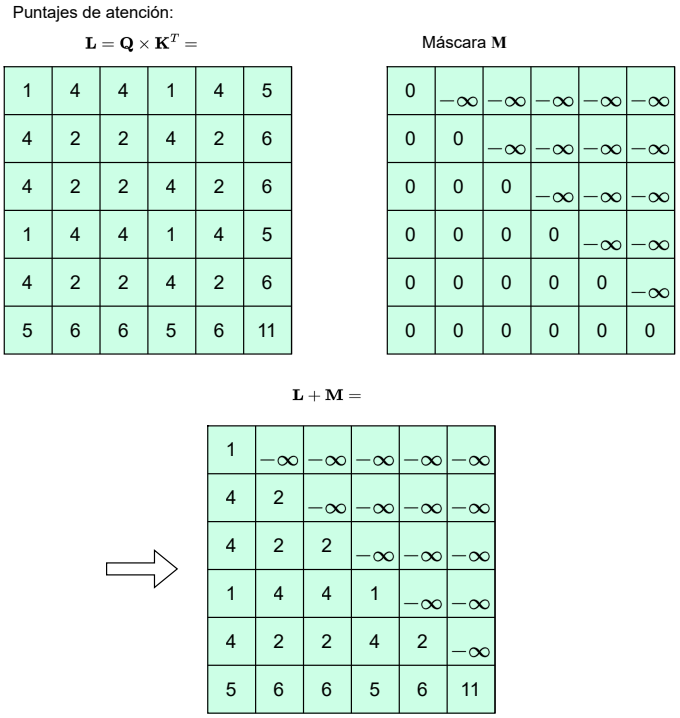


Figura 4.11: Ejemplo de suma de la matriz de *scores* **L** con una máscara que anula las posiciones futuras.

La parte del *Decoder* está compuesta por N bloques dispuestos nuevamente en serie, la salida de uno es la entrada del siguiente. Todos reciben como entrada la salida del último *Encoder* directo a la etapa de *attention*, luego del *Masked-Attention*.

Una vez finalizada la etapa del *Decoder* se encuentra la capa de salida. En este caso, donde se busca predecir secuencias de dimensión $p = 6$ con valores cero o uno, se tienen 2^6 opciones de salida. Se tendrán entonces 2^6 probabilidades, donde se tomará como mejor predicción aquella que sea máxima.

El *Transfomer* es un modelo de aprendizaje automático supervisado, donde se evalúa respecto a las salidas verdaderas qué tan buenas fueron las predicciones realizadas. En la etapa de entrenamiento, la salida predicha es la que ingresa nuevamente al *Decoder* para calcular las dependencias que tiene la nueva palabra a predecir con las predicciones ya realizadas. En cada época se utiliza una función de costo para analizar qué tan buena fue la predicción. Con esta información, se actualizan los pesos de las matrices $\mathbf{P}_Q$, $\mathbf{P}_K$ y $\mathbf{P}_V$, y los pesos de las redes neuronales de los bloques, para mejorar el modelo y obtener mejores resultados.

Es importante destacar que en un modelo *Encoder-Decoder*, la salida a predecir es una secuencia, donde el *Encoder* se encarga de obtener la información que brinda la entrada y el *Decoder* aquella que brindan las predicciones previas. En el ejemplo, tanto la entrada como la salida, son vectores de dimensión $p = 6$. Sin embargo, se puede trabajar únicamente con una porción del modelo, dependiendo de la

tarea que se quiere realizar. Si lo que se busca predecir es cierta característica a partir de los datos de entrada, puede realizarse únicamente con el *Encoder* y una correspondiende capa de salida que devuelva las predicciones deseadas, que es el caso con el que se trabaja en este proyecto.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 5

Modelos Implementados y Resultados

En este capítulo se presenta una descripción del modelo implementado y los experimentos realizados para la predicción del crecimiento de levadura en distintos ambientes. Luego se presentan los resultados obtenidos para cada implementación. En primera instancia, se entrenó un modelo para predecir los fenotipos simulados para confirmar si el *Transformer* puede adaptarse al dominio de la genómica y, en particular, al conjunto de datos que se utiliza en este proyecto. Luego se realizaron entrenamientos para la predicción del crecimiento en Lactato y Lactosa por separado. Por último, se entrenó un modelo para que realice la predicción *Multi-Trait* del fenotipo en los dos ambientes.

En Jubair et al., 2021 [16] se presenta un modelo basado en el *Encoder* donde se toman como entradas secuencias genotípicas de cebada, para predecir la presencia de DON y FHB. Al ser la cebada una especie diploide, en cada posición del genotipo se deben indicar dos datos que refieren a si ambos alelos se condicen con el alelo de referencia, solo uno de ellos o ninguno. Esta base de datos es codificada previo a entrar al modelo con dos métodos: (1) codificación categórica con valores (-1, 0 ,1) y (2) codificación Hardy-Weinberg de acuerdo a la frecuencia de aparición.

Para el presente trabajo se tiene una base de datos ya codificada categóricamente, como se describe en la sección 2.6, por lo que no se realiza ninguna etapa previa de codificación.

5.1. Modelo

El modelo implementado presenta en primera instancia una capa lineal funcionando como capa de *Embeddings*. Tiene como dimensión de entrada la cantidad de SNPs (p) que hay por individuo y como salida el hiperparámetro `embed_dim`, definido como $d_{\mathcal{M}}$ en la sección 4.2. Cada una de las posiciones que conforman el genotipo del individuo es representada por un vector de dimensión `embed_dim`, por lo que al entrar al *Encoder* cada individuo se encuentra representado por una matriz de dimensión $(embed\_dim, p)$.

No se utiliza explícitamente un módulo de *Positional Encoding* debido a que cada posición tiene una representación distinta a la salida de la capa lineal, con-

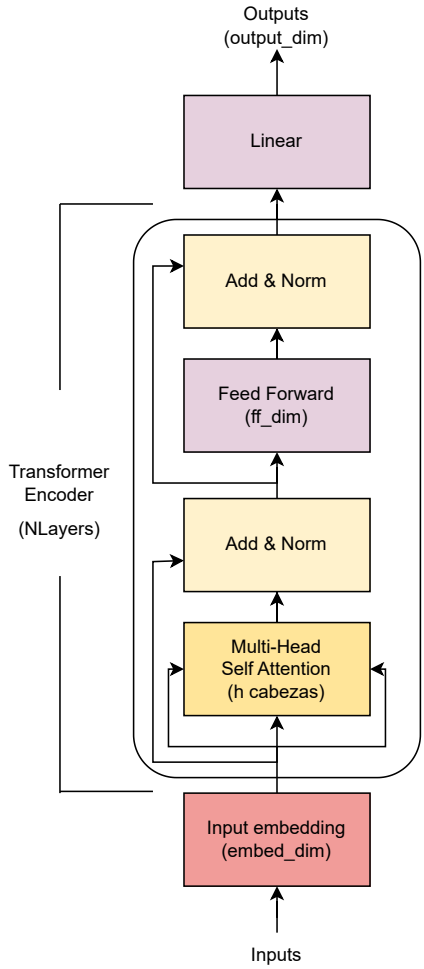


Figura 5.1: Modelo entrenado con la base de datos de Levadura para la predicción del crecimiento en distintos ambientes.

servando así la información sobre la posición. La cantidad de *Encoders* del modelo se define con el hiperparámetro `NLayers`. La estructura, en este caso, es igual a la estructura del *Encoder* presentado para el *Transformer* en la sección 4.2: bloque *Multi-Head Self Attention* formado por `h` cabezas, una FFN de dimensión `ff_dim` y dos capas *Add & Norm* a la salida de cada uno de los módulos mencionados previamente. Tanto `h` como `ff_dim` son hiperparámetros del modelo. Finalmente, el modelo presenta una capa lineal encargada de realizar la predicción de fenotipo para cada uno de los individuos. La dimensión de salida, `output_dim`, será definida acorde a la cantidad de fenotipos que se desee predecir con un mismo modelo (`output_dim` $= 1$ para *Single-Trait* y `output_dim` $= 2$ para *Multi-Trait*). En la figura 5.1 se muestra el diagrama del modelo implementado.

Para implementar el modelo se utlizaron módulos de *Pytorch*. En particular, para el *Encoder* se utilizó la clase `TransformerEncoder` y para la capa de *Embeddings* y FFN de salida se utilizó el módulo `nn.Linear`.

5.1.1. Búsqueda de Hiperparámetros y Entrenamiento

El entrenamiento del modelo se dividió en dos etapas: primero se realizó una búsqueda de hiperparámetros óptimos para luego entrenar el modelo. Todos los experimentos fueron realizados en `ClusetrUY` utilizando una GPU de 40 GB.

Para la búsqueda de hiperparámetros se definieron posibles valores para el *learning rate*, `h`, `ff_dim`, `embed_dim` y `dropout` (utilizado para la regularización), donde en cada entrenamiento se tomó una combinación distinta de ellos. Finalmente, se tomó la combinación con la que se obtuvo el mejor valor de PCC. En primera instancia se realizaron pruebas para considerar como hiperparámetro el número de *Encoders* pero debido a limitaciones computacionales se eligió entrenar el modelo con dos. La implementación se realizó utilizando la biblioteca `Optuna`. Las distribuciones y resultados obtenidos se muestran en el anexo C.

Se dividió el conjunto de datos disponibles en cinco *folds* para poder realizar el entrenamiento con una posterior validación. Se tomaron *batches* de ocho individuos donde cada uno es una secuencia de largo 11623 SNPs.

Como función de pérdida se utilizó MSE (3.1), mientras que como función de ganancia se utilizó el PCC (3.5). Esto implicó que para cada época, si bien se actualizaron los parámetros considerando el MSE, una vez hecha la validación, el mejor modelo fue elegido según la época que tuviera mayor PCC. Además, cabe destacar que el MSE de entrenamiento se computa en la propagación hacia adelante de la red (*feed-forward*) y el MSE de validación luego del *backpropagation*, por lo cual los resultados en validación serán mejores que en entrenamiento. El optimizador utilizado fue Adam [17]. Los parámetros del modelo fueron inicializados utilizando el método `Xavier Uniform` [12].

El entrenamiento se realizó durante un máximo de mil épocas. Como método de regularización se utilizó *Early stopping* (sección 3.3.2) deteniéndolo si luego de treinta y cinco épocas consecutivas no se presentaban mejoras en el PCC de validación. Además, se utilizaron etapas de *Dropout* (sección 3.3.1) tanto en la red neuronal dentro del *Encoder* como en la que se encuentra a la salida, ambas con igual *ratio de dropout*.

5.2. Experimentos Realizados

Se realizó la búsqueda de hiperparámetros y posterior entrenamiento de cinco modelos: (1) predicción de fenotipo simulado con relaciones lineales, (2) predicción de fenotipo simulado con relaciones no lineales, datos de crecimiento de levadura en (3) Lactato y (4) Lactosa y por último, (5) un modelo *multi-trait* con el objetivo de predecir Lactato y Lactosa con un único modelo. La elección de estos dos fenotipos para realizar *multi-trait* estuvo basada en que la correlación entre ellos es alta y ambos presentan distribuciones similares, como se puede ver en los histogramas de la figura 2.4, por lo que se quiso investigar si las predicciones se podían ver afectadas de manera positiva en caso de hacerlas juntas.

Para el caso de los fenotipos simulados, se calculó y graficó con un mapa de calor la salida del *Self-Attention*. Como se define en la sección 4.2.2 y se muestra

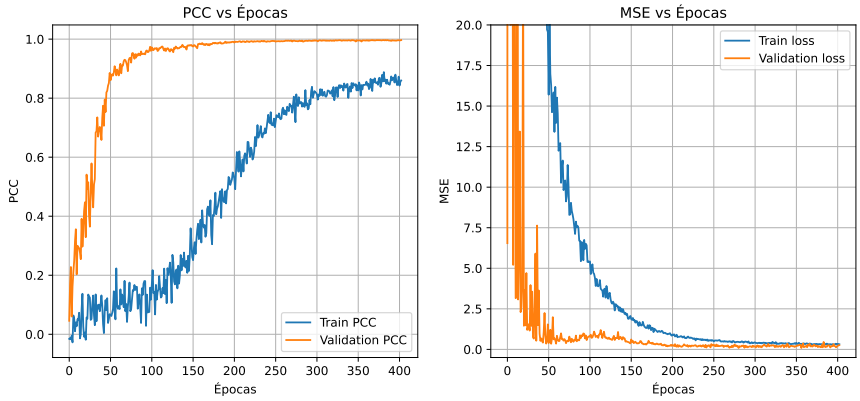


Figura 5.2: Curvas de aprendizaje para los datos simulados con perturbaciones lineales. A la izquierda PCC, y a la derecha MSE. La curva naranja representa los resultados en validación, mientras que la azul los datos de entrenamiento.

luego en el ejemplo de la sección 4.3, la salida del *Self-Attention* es una matriz que tiene en cada posicion (i, j) qué importancia tiene ese elemento para el modelo. Con la visualización se busca observar que posiciones tienen mayor influencia en las predicciones de los distintos fenotipos. Al implementar la simulación del fenotipo, se guardó la información de cuales fueron las posiciones perturbadas, por lo que se puede realizar una comparación de las posiciones que realmente "importaron" frente a las que el modelo logró identificar como "importantes".

Para los modelos *Single-trait* el mejor modelo fue elegido tomando en cuenta los resultados obtenidos de PCC en validación, tomando como mejor época aquella que presenta PCC más alto. En el caso del modelo *Multi-trait*, se obtiene un valor de PCC para cada uno de los fenotipos, por lo que el mejor modelo es elegido a partir del promedio de estos dos valores.

5.3. Resultados

5.3.1. Fenotipos simulados

En esta sección se realiza el análisis de los resultados obtenidos para los fenotipos simulados. En la figura 5.2 se presentan los resultados de PCC y MSE durante el entrenamiento. Se observa que ambos mejoran a medida que avanzan las épocas, PCC tiende a uno, mientras que MSE tiende a cero. Se destaca que los resultados en el conjunto de validación son mejores que en el de entrenamiento, para ambas métricas. Esto se debe a que se realiza *dropout* en entrenamiento pero no en validación. Comparando las curvas de PCC y MSE se observa que a medida que el MSE baja, también sube el PCC, lo cual coincide con lo esperado. En las primeras cien épocas el PCC presenta un crecimiento lento (se mantiene entre 0 y 0.2) que coincide con valores muy altos de MSE, y a medida que este decrece el PCC aumenta.

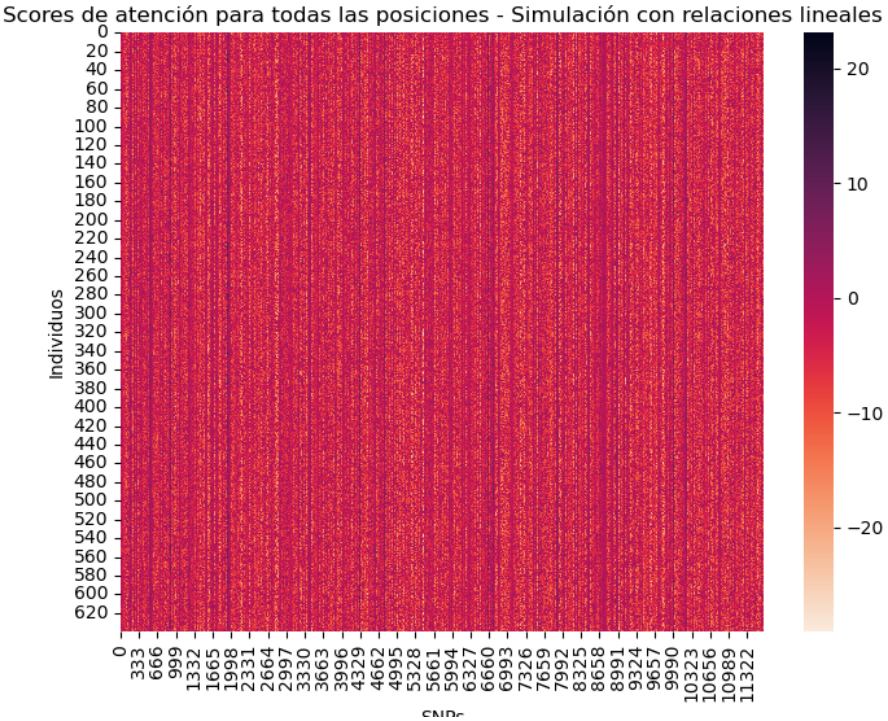


Figura 5.3: Mapa de calor con puntajes de atención para las diferentes posiciones del genotipo, simulado con relaciones lineales.

En la figura 5.3 se presenta el mapa de calor con los puntajes de atención del modelo para cada posición del conjunto total de datos de entrenamiento, donde se muestran en el eje vertical los individuos y en el eje horizontal los SNPs. Se observan líneas verticales lo cual implica que para todos los individuos, el modelo "presta atención" a las mismas posiciones. En la figura 5.4 se muestran cuatro zonas de la matriz de la figura 5.3, donde se visualiza un rango de sesenta SNPs para todos los individuos. En celeste se marcaron las posiciones, dentro del rango, que se perturbaron para la simulación del fenotipo. En todos los casos se pueden observar regiones de mayor puntaje, que implica mayor "atención", en la región donde se encuentra la posición perturbada. Sin embargo, también se presentan zonas de alto puntaje por fuera de las posiciones perturbadas. Esto tiene relación con el desequilibrio de ligamiento (LD por su sigla en ingles, "Linkage Desequilibrium") [1], donde aquellas posiciones que no fueron perturbadas pero presentan LD con posiciones que sí, también influyen sobre el fenotipo. Puntajes altos en posiciones distintas a las posiciones perturbadas también se condicen con los resultados obtenidos, donde las predicciones presentan errores respecto a los datos reales.

En la figura 5.5 se presentan las predicciones en test, las cuales se asemejan

[1]Propiedad que presentan algunos genes de no segregar de forma independiente

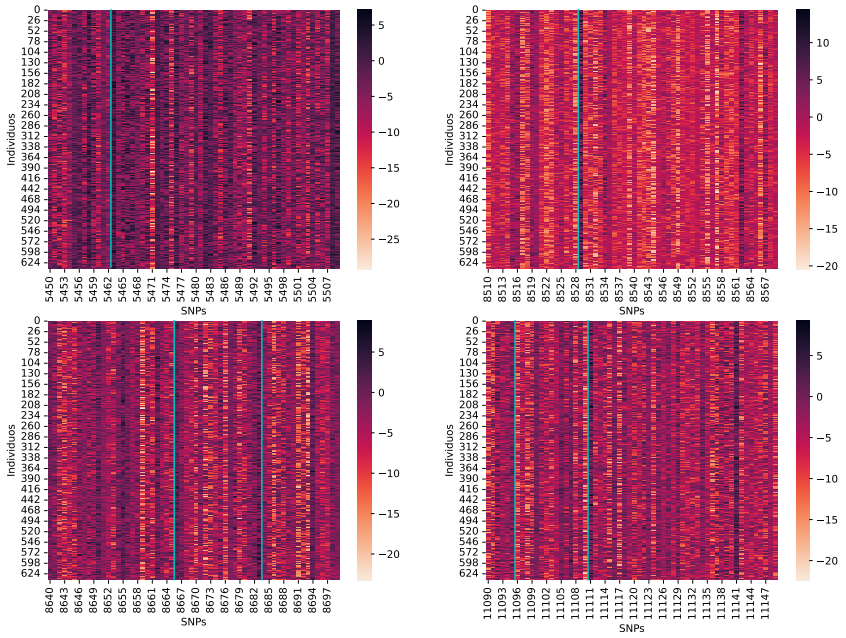


Figura 5.4: Mapa de calor para cuatro rangos de setenta valores donde se puede observar detalladamente qué ocurre en cada posición e incluyendo posiciones perturbadas. La línea vertical celeste marca la posición utilizada para la simulación del fenotipo.

al fenotipo verdadero. Se puede ver que el PCC obtenido es de 0,993, que indica una relación lineal fuerte y el MSE es 0,231, un valor pequeño, implicando que la relación lineal sea próxima a la de la recta $y = x$.

En la figura 5.6 se presentan las curvas de PCC y MSE para el conjunto de datos simulados con perturbaciones no lineales. En entrenamiento se presenta un comportamiento similar al descrito para el caso anterior, el PCC crece y el MSE decrece a medida que avanzan las épocas. Sin embargo, esto no ocurre para los datos de validación. La función que se busca aproximar es una que genera datos con comportamiento no lineal, ergo más compleja que la anterior, que solamente generaba datos con comportamiento lineal. Esto implica que los resultados obtenidos distaran más del fenotipo real debido a la mayor complejidad de éste segundo modelo. Las grandes variaciones que presenta la curva del MSE en validación responden también a la presencia de *outliers* que tienen gran impacto sobre esta métrica.

En concordancia con los resultados expuestos anteriormente, para el caso de las simulaciones con datos no lineales se obtienen resultados en test que presentan una relación lineal fuerte con un valor de PCC de 0,933. El MSE obtenido, 0,855, responde principalmente a la cantidad de predicciones en el intervalo $(-4, -2)$ que no son valores que tome el fenotipo verdadero. Sin embargo, nuevamente puede observarse que las predicciones son alrededor de la recta $y = x$. Lo anterior se

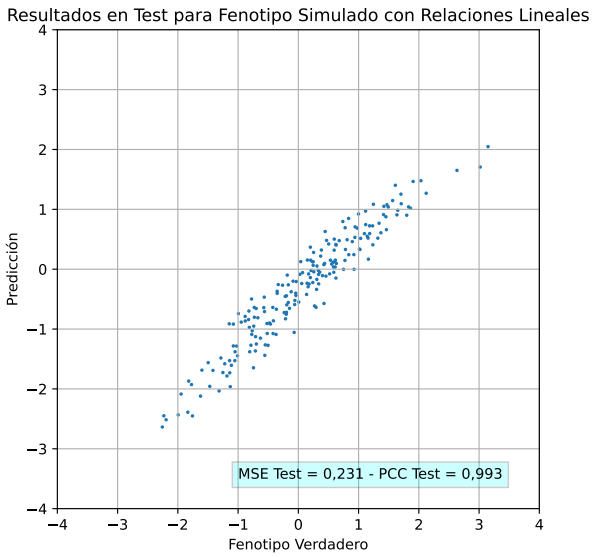


Figura 5.5: Gráfica de fenotipo verdadero comparado con su predicción para los datos obtenidos de la simulación con perturbaciones lineales.

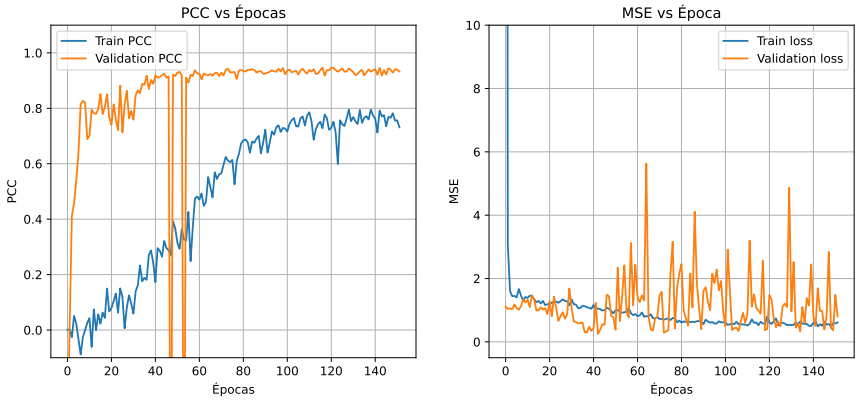


Figura 5.6: Curvas de aprendizaje para los datos simulados con perturbaciones no lineales. A la izquierda PCC, y a la derecha MSE. La curva naranja representa los resultados en validación, mientras que la azul los datos de entrenamiento.

ilustra en la figura 5.7.

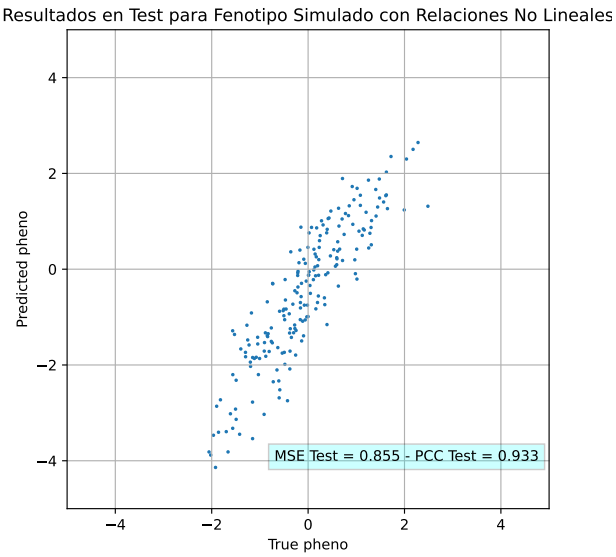


Figura 5.7: Gráfica de fenotipo verdadero comparado con su predicción, para los datos obtenidos de la simulación con perturbaciones no lineales.

5.3.2. Fenotipos reales

En esta sección se presentan los resultados en entrenamiento, validación y test, para los datos de levadura en los ambientes: Lactato y Lactosa, descriptos en la sección 2.6. En la figura 5.8 se presentan las gráficas de variación del PCC y el MSE en función de las épocas para Lactato. Se observa que a medida que avanzan las épocas, ambos resultados mejoran, el PCC crece, mientras que el MSE decrece, como es esperado. Al igual que ocurría en el caso de los datos simulados, los resultados en validación son mejores que en entrenamiento. Además, en la gráfica del MSE se observa que el modelo se sobreajusta a los datos de entrenamiento, ya que a partir de la época 250 aproximadamente, el MSE de validación comienza a crecer, mientras que el de entrenamiento decrece.

En el caso de Lactosa el comportamiento del MSE y el PCC es similar al de Lactato. Se puede ver que existe un sesgo en el MSE, pero no se presenta sobreajuste, como se puede ver en la figura 5.9. Se observa que, a pesar de que no se presenta una mejora por más de 35 épocas en el PCC en validación (por lo cual el entrenamiento se detiene), el PCC en entrenamiento podría haber alcanzado un valor mayor, ya que presenta un comportamiento creciente hasta esta época. Lo mismo podría haberse manifestado en el MSE de entrenamiento, con respecto al decrecimiento.

En la figura 5.10 se presentan las curvas de PCC y MSE para cada fenotipo con el modelo *multi-trait*. Los resultados son similares a los obtenidos al entrenar el modelo con un solo fenotipo, a medida que el PCC crece, el MSE decrece. Sin embargo, el MSE presenta un sesgo más pequeño y a diferencia de lo que ocurría en el caso de Lactato, el modelo no se sobreajusta.

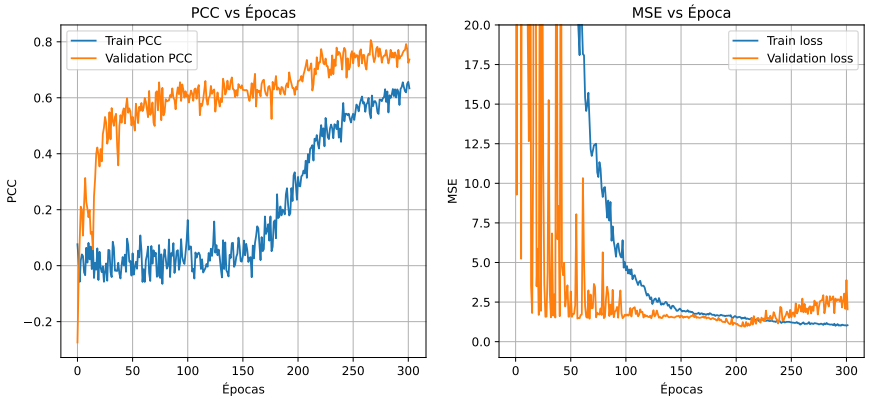


Figura 5.8: Curvas de aprendizaje para el entrenamiento con datos de crecimiento en Lactato. A la izquierda de la curva de PCC, a la derecha la curva de MSE. En azul se presentan las curvas de entrenamiento y en naranja las de validación.

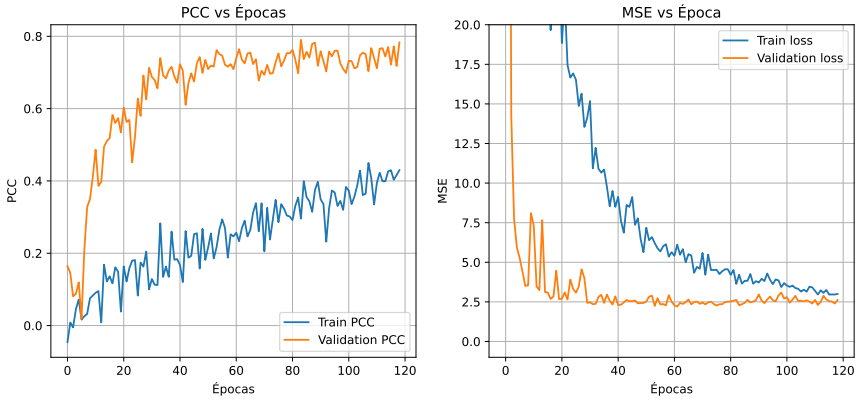


Figura 5.9: Curvas de aprendizaje con datos de crecimiento en Lactosa. A la izquierda de la curva de PCC, a la derecha la curva de MSE. En azul se presentan las curvas de entrenamiento y en naranja las de validación

En todos los modelos se observa que a medida que las épocas avanzan tanto el PCC como el MSE mejoran, esto muestra que el modelo aprende de los datos. Sin embargo, en todos los casos se observan aspectos a mejorar, para trabajos a futuro.

En la figura 5.11 se presentan los resultados obtenidos en test para los datos reales. Se observa que en ambos casos los resultados obtenidos con *multi-trait* fueron mejores que entrenando el modelo con un solo fenotipo. Disminuyó considerablemente el MSE, mientras que el PCC aumentó de manera moderada. Para todos los casos la relación no es fuertemente lineal, debido a que el PCC no es tan alto como en los casos de simulaciones.

Con el fin de comparar los resultados obtenidos con otros modelos que han sido entrenados para este conjunto de datos, se realiza el cálculo de la métrica R^2

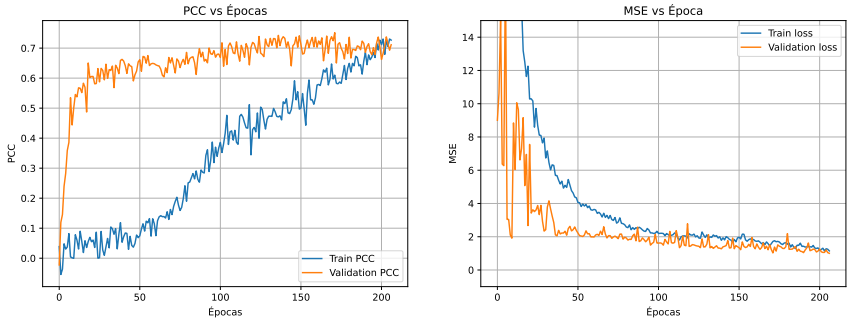


Figura 5.10: Curvas de aprendizaje para Lactato y Lactosa con multitrait. La curva de PCC mostrada es el promedio de los PCC obtenidos para los dos fenotipos, tanto en entrenamiento como en validación.

Ambiente	Grinberg	GBM	One trait	Multitrait
Lactato	0.568	0.830	-0.543	0.504
Lactosa	0.582	0.860	0.349	0.553

Tabla 5.1: Resultados de la métrica R^2 para los fenotipos Lactato y Lactosa de levadura, comparados según predicciones *One trait* y *Multitrait*, comparados con los resultados de la misma métrica expuestos por Grinberg et al., 2020 [13] Elenter et al., 2021 [7].

(coeficiente de determinación). Esta métrica es utilizada para la evaluación de qué tan bien un modelo ha performado sobre un conjunto de datos, siendo su mejor resultado uno y decreciendo hacia cero conforme el modelo decrece en performance. El valor cero para la métrica R^2 indica que las predicciones del modelo son como las de un modelo aleatorio, mientras que si el resultado está por fuera del intervalo $[0, 1]$, entonces el modelo ha obtenido peores resultados que el caso donde se predice de manera aleatoria. Los resultados de la métrica R^2, obtenidos para las predicciones de fenotipo de Lactato y Lactosa *One trait* y *Multitrait*, se presentan en la tabla 5.1, junto con otros resultados expuestos por Elenter et al [7], comparados para los mismos fenotipos predichos por otros modelos. Se observa de la tabla que los resultados para los modelos *Multi-trait* superan considerablemente a los modelos *Single-trait*, lo cual ya había sido evidenciado anteriormente. Por otro lado, si bien estos resultados no logran alcanzar los de GBM expuestos en la tabla, sí logran alcanzar en orden de magnitud a los de Grinberg et al., 2020 [13], confirmando la robustez del algoritmo implementado. En último lugar, se destaca que los malos resultados para Lactato están alineados con el hecho de que fue el modelo que presentó sobreajuste y obtuvo los peores resultados.

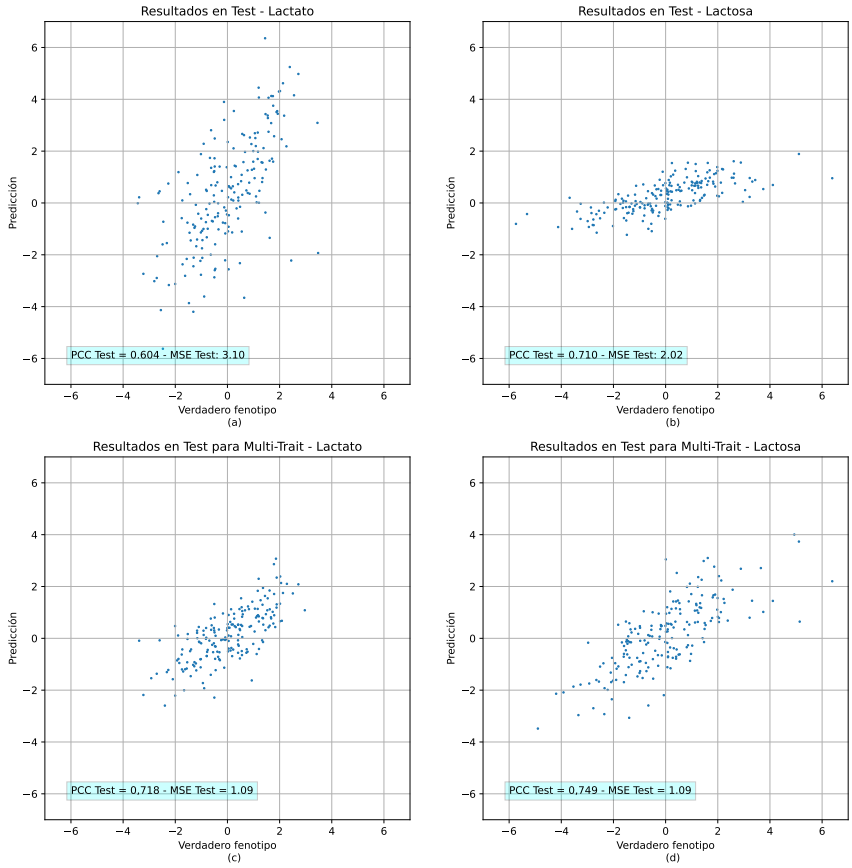


Figura 5.11: Gráfica de fenotipo verdadero comparado con su predicción, para los datos reales. (a) Lactato, (b) Lactosa , (c) Lactato Multitrait , (d) Lactosa Multitrait.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 6

Conclusiones

El presente proyecto constó de una primera parte de estudio de los fundamentos e implementación de *Transformers* que superó ampliamente la expectativa inicial de tiempos. Se realizó un primer estudio del algoritmo de los *Transformers*, comprendiendo su aplicación en el campo del lenguaje natural y posteriormente su adaptabilidad al campo de la genómica. Esto generó que al momento de comenzar con la implementación del código y la adaptación del conjunto de datos al modelo a implementar, el tiempo disponible resultó escaso. Solicitando una prórroga de la entrega final se logró compensar la escasez de tiempo, pero dada la magnitud del proyecto y lo prometedor del mismo, más tiempo sigue siendo necesario. Los resultados obtenidos son prometedores, y permiten afirmar que es factible obtener resultados satisfactorios al utilizar modelos basados en *Self-Attention* a secuencias de datos genómicos. Hay, sin embargo, modificaciones, validaciones y nuevas simulaciones en las que se desearía ahondar, entre ellas:

1. Nuevas búsquedas de hiperparámetros, más exhaustivas.
2. Balance de elección de mejores modelos según PCC y MSE.
3. Investigar mejores métodos de inicialización de los parámetros.
4. Repetir el proceso en nuevas bases de datos.

El primer punto está directamente relacionado con el poder de cómputo con que se contó. La máxima memoria en GPU accesible fue de $40GB$, y con ésta fueron realizadas las búsquedas de hiperparámetros con *Optuna*. Para todas las búsquedas realizadas, los mejores parámetros obtenidos siempre fueron los máximos de los intervalos estudiados, indicando que deben estudiarse intervalos de hiperparámetros mayores. Esto no pudo realizarse, dado que en todos los casos se alcanzó una saturación de la memoria. El segundo punto del listado está directamente relacionado con los resultados obtenidos para Lactato, en los cuales se observa un claro sobreajuste en la curva de entrenamiento con la métrica del MSE. Esto está relacionado a que la condición de parada y la elección de la mejor época fue realizada en base al coeficiente PCC, pero debería haberse encontrado la manera de balancear e incluir los resultados para cada época de la métrica MSE. Si bien fue probado obtener

la mejor época según la métrica MSE y los resultados no fueron mejores, otras alternativas podrían haber sido contempladas, que combinaran ambos resultados de PCC y MSE.

6.1. Trabajo futuro

Para trabajos futuros, es de alta importancia analizar nuevos métodos de inicialización de los parámetros, debido a que no fue un campo ahondado en el presente proyecto. En la sección 5.3 se puede observar para todas las curvas de entrenamiento de MSE, que la métrica inicia con valores muy altos. Estos resultados implican que el modelo al inicio, con sus parámetros inciales, es muy malo, y se tarda bastantes épocas en alcanzar resultados de la métrica MSE óptimos. Investigar e implementar nuevos métodos de inicialización de los parámetros es una alternativa que busca combatir este alto error inicial durante el entrenamiento.

Otra posible vía de estudio es analizar si agregando algún método de *Positional Encoding* puede mejorar las predicciones. En este proyecto se tomaron las secuencias de genotipo como una única palabra. Sin embargo, nos podemos cuestionar si dividirlo en vectores de dimensiones más pequeñas y agregando información de la posición que cada uno de esos vectores tiene en la secuencia completa puede llevar a mejores resultados.

En siguiente lugar, buscando seguir afirmando que los métodos basados en *Self-Attention* arrojan resultados satisfactorios al ser aplicados a secuencias de datos genómicos, se destaca la importancia de trabajar en nuevas bases de datos, idealmente para otros organismos haploides, así como también diploides.

Apéndice A

Ecuaciones de LSTM

Se muestran a continuación las ecuaciones que justifican el funcionamiento de las neuronas LSTM:

Control de la *input gate*:

$$i(t) = \sigma\left(\mathbf{W}_{xi}^T\mathbf{x}(t) + \mathbf{W}_{hi}^T\mathbf{h}(t-1) + \mathbf{b}_i\right)$$

Control de la *forget gate*:

$$f(t) = \sigma\left(\mathbf{W}_{xf}^T\mathbf{x}(t) + \mathbf{W}_{hf}^T\mathbf{h}(t-1) + \mathbf{b}_f\right)$$

Control de la *output gate*:

$$o(t) = \sigma\left(\mathbf{W}_{xo}^T\mathbf{x}(t) + \mathbf{W}_{ho}^T\mathbf{h}(t-1) + \mathbf{b}_o\right)$$

Salida de la principal capa FC (que combina $\mathbf{x}(t)$ y $\mathbf{h}(t-1)$):

$$g(t) = \tanh\left(\mathbf{W}_{xg}^T\mathbf{x}(t) + \mathbf{W}_{hg}^T\mathbf{h}(t-1) + \mathbf{b}_g\right)$$

Vector de estado de memoria de largo plazo:

$$\mathbf{c}(t) = f(t) \otimes \mathbf{c}(t-1) + i(t) \otimes g(t)$$

Vector de estado de memoria de corto plazo y salida de la neurona:

$$\mathbf{y}(t) = \mathbf{h}(t) = o(t) \otimes \tanh(c(t))$$

Se detallan las componentes de las operaciones anteriores, que aún no fueron mencionadas:

- W_{xi}, W_{xf},W_{xo} y W_{xg}: son las matrices de los pesos de cada una de las cuatro capas FC que recibe al vector de entrada $\mathbf{x}(t)$.
- W_{hi}, W_{hf},W_{ho} y W_{hg}: son las matrices de los pesos de cada una de las cuatro capas FC que recibe al vector de estado de memoria de corto plazo del instante anterior $\mathbf{h}(t-1)$.

- $\mathbf{b}_i$, $\mathbf{b}_f$, $\mathbf{b}_o$ y $\mathbf{b}_g$: son los términos de *bias* de cada una de las capas.

Apéndice B

Ejemplo de self-attention

Se muestra un ejemplo numérico de cálculo de *Self-attention* donde se puede ver el proceso entero desde que se tiene la entrada hasta la salida del módulo de atención descrito en la sección **??**. Se consideran las siguientes entradas:

- $I_1 = [1010]$
- $I_2 = [0202]$
- $I_3 = [1111]$

que escritas en forma matricial forman la matriz $\mathcal{X}$:

$$X^T = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 2 & 1 \\ 1 & 0 & 1 \\ 0 & 2 & 1 \end{bmatrix} \Rightarrow X^T = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 2 \\ 1 & 1 & 1 & 1 \end{bmatrix} \tag{B.1}$$

A continuación se definen las matrices de pesos P_q, P_k y P_v, que se usarán para obtener los valores de las matrices $\mathcal{Q}$, $\mathcal{K}$ y $\mathcal{V}$, tal como fueron definidas**??**. Recordar que las matrices de pesos son aprendidas por el algoritmo. El valor de $\mathcal{Q}$ será el resultado de multiplicar la entrada $\mathcal{X}$ por la matriz de pesos P_q. Análogamente se obtienen los valores de *key* y *value*.

$$P_q = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \tag{B.2}$$

$$P_k = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \end{bmatrix} \tag{B.3}$$

$$P_v = \begin{bmatrix} 0 & 2 & 0 \\ 0 & 3 & 0 \\ 1 & 0 & 3 \\ 1 & 1 & 0 \end{bmatrix} \tag{B.4}$$

Haciendo cuentas se obtienen las matrices *key*, *query* y *value*, donde la fila i de cada matriz corresponde al valor de *key*, *query* y *value* de la entrada i.

$$Q = \begin{bmatrix} 1 & 0 & 2 \\ 2 & 2 & 2 \\ 2 & 1 & 3 \end{bmatrix} \tag{B.5}$$

$$K = \begin{bmatrix} 0 & 1 & 1 \\ 4 & 4 & 0 \\ 2 & 3 & 1 \end{bmatrix} \tag{B.6}$$

$$V = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 8 & 0 \\ 2 & 6 & 3 \end{bmatrix} \tag{B.7}$$

El paso siguiente es el cálculo de los *attention scores* (matriz $\mathcal{L}$):

$$L = Q \times K^T = \begin{bmatrix} 1 & 0 & 2 \\ 2 & 2 & 2 \\ 2 & 1 & 3 \end{bmatrix} \times \begin{bmatrix} 0 & 4 & 2 \\ 1 & 4 & 3 \\ 1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 4 & 4 \\ 4 & 16 & 12 \\ 4 & 12 & 10 \end{bmatrix} \tag{B.8}$$

El siguiente paso es aplicar la función *Softmax* a la matriz $\mathcal{L}$, obteniendo scores normalizados.

$$W = softmax(L) = softmax\left(\begin{bmatrix} 2 & 4 & 4 \\ 4 & 16 & 12 \\ 4 & 12 & 10 \end{bmatrix}\right) = \begin{bmatrix} 0{,}0 & 0{,}5 & 0{,}5 \\ 0{,}0 & 0{,}98 & 0{,}02 \\ 0{,}0 & 0{,}88 & 0{,}12 \end{bmatrix} \tag{B.9}$$

Multiplicando la matriz $\mathcal{W}$ por la matriz $\mathcal{V}$ se llega a la matriz $\mathcal{O}$.

$$O = W \times V = \begin{bmatrix} 0{,}0 & 0{,}5 & 0{,}5 \\ 0{,}0 & 0{,}98 & 0{,}02 \\ 0{,}0 & 0{,}88 & 0{,}12 \end{bmatrix} \times \begin{bmatrix} 1 & 2 & 3 \\ 2 & 8 & 0 \\ 2 & 6 & 3 \end{bmatrix} = \begin{bmatrix} 2{,}00 & 7{,}00 & 1{,}50 \\ 2{,}00 & 7{,}96 & 0{,}06 \\ 2{,}00 & 7{,}76 & 0{,}36 \end{bmatrix} \tag{B.10}$$

El último paso es multiplicar la matriz obtenida por una matriz de pesos P_0 de dimensión tal que lleve a la matriz $\mathcal{O}$ a las mismas dimensiones que la entrada.

Apéndice C

Búsqueda de Hiperparámetros

La búsque de hiperparámetros fue realizada utilizando Optuna. Las distribuciones para cada hiperparámetro se muestran a continuación:

- *Learning rate*: *float* tomados dentro del intervalo $(1e^{-6}, 1e^{-4})$.
- h: 2, 3 o 4.
- *embed_dim*: números enteros del intervalo $(h, h*2)$ con paso h. Este intervalo fue definido de forma tal que siempre fuera divisible entre el número de cabezas para una correcta implementación del *Multi-Head Attention*.
- ff_dim: 4, 5 o 6.
- *Dropout*: *float* tomados del intervalo $(0, 1)$ con paso 0,1.

Es importante destacar que para todos los casos se intentó considerar intervalos más grandes pero las limitaciones computacionales llevaron a reducir las opciones.

A continuación, se muestra la tabla C.1 con los valores de los hiperparámetros utlizados para el entrenamiento de cada modelo.

-	*Learning Rate*	h	*embed_dim*	ff_dim	*Dropout*
Sim lineal	$8.608e^{-5}$	4	8	6	0.3
Sim no lineal	$2.017e^{-3}$	4	8	6	0.5
Lactato	$6.040e^{-4}$	4	8	4	0.6
Lactosa	$1.729e^{-3}$	2	4	5	0.2
Multi-trait Lactato-Lactosa	$4.389e^{-}4$	4	8	6	0,5

Tabla C.1: Resultados obtenidos con *Optuna* para la búsqueda de hiperparámetros de los cinco modelos implementados.

Esta página ha sido intencionalmente dejada en blanco.

Referencias

[1] Yaser S Abu-Mostafa, Malik Magdon-Ismail, and Hsuan-Tien Lin. *Learning from data*, volume 4. AMLBook New York, 2012.

[2] Leo Breiman. Arcing the edge. Technical report, Citeseer, 1997.

[3] T Britannica. Editors of encyclopaedia. *Argon. Encyclopedia Britannica*, 2020.

[4] Alexey Ya Chervonenkis. Early history of support vector machines. *Empirical Inference: Festschrift in Honor of Vladimir N. Vapnik*, pages 13–20, 2013.

[5] Jim Clauwaert, Gerben Menschaert, and Willem Waegeman. Explainability in transformer models for functional genomics. *Briefings in bioinformatics*, 22(5):1–11, 2021.

[6] Jorge Dagnino. Coeficiente de correlación lineal de pearson. *Chil Anest*, 43(1):150–153, 2014.

[7] Juan Elenter, Guillermo Etchebarne, and Ignacio Hounie. Dnai: Machine learning for genome enabled prediction of complex traits in agriculture. Master's thesis, 2021.

[8] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[9] Aurélien Géron. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems.* .ºReilly Media, Inc.", 2019.

[10] Daniel Gianola. Priors in whole-genome regression: the bayesian alphabet returns. *Genetics*, 194(3):573–596, 2013.

[11] Harsimardeep S Gill, Jyotirmoy Halder, Jinfeng Zhang, Navreet K Brar, Teerath S Rai, Cody Hall, Amy Bernardo, Paul St Amand, Guihua Bai, Eric Olson, et al. Multi-trait multi-environment genomic prediction of agronomic traits in advanced breeding lines of winter wheat. *Frontiers in Plant Science*, 12:709545, 2021.

[12] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[13] Nastasiya F Grinberg, Oghenejokpeme I Orhobor, and Ross D King. An evaluation of machine-learning for predicting phenotype: studies in yeast, rice, and wheat. *Machine Learning*, 109:251–277, 2020.

[14] Tin Kam Ho. Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, volume 1, pages 278–282. IEEE, 1995.

[15] Arthur E Hoerl and Robert W Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

[16] Sheikh Jubair, James R Tucker, Nathan Henderson, Colin W Hiebert, Ana Badea, Michael Domaratzki, and WG Fernando. Gptransformer: A transformer-based deep learning method for predicting fusarium related traits in barley. *Frontiers in plant science*, 12:2984, 2021.

[17] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[18] Osval Antonio Montesinos López, Abelardo Montesinos López, and José Crossa. *Multivariate Statistical Machine Learning Methods for Genomic Prediction*. Springer, 2022.

[19] Saba Moeinizade, Aaron Kusmec, Guiping Hu, Lizhi Wang, and Patrick S Schnable. Multi-trait genomic selection methods for crop improvement. *Genetics*, 215(4):931–945, 2020.

[20] Håkon Måløy, Susanne Windju, Stein Bergersen, Muath Alsheikh, and Keith L. Downing. Multimodal performers for genomic selection and crop yield prediction. *Smart Agricultural Technology*, 1:100017, 2021.

[21] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[22] James D Watson and Francis HC Crick. Molecular structure of nucleic acids: a structure for deoxyribose nucleic acid. *Nature*, 171(4356):737–738, 1953.

Índice de figuras