UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

REGRESIÓN SIMBÓLICA MEDIANTE REDES NEURONALES

TESIS PARA OPTAR AL GRADO DE MAGÍSTER EN CIENCIAS DE LA INGENIERÍA, MENCIÓN ELÉCTRICA

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO

ROBERTO IGNACIO CHOLAKY MEJÍA

PROFESOR GUÍA:
PABLO ESTÉVEZ VALENCIA

MIEMBROS DE LA COMISIÓN:
RICHARD WEBER HAAS
DORIS SÁEZ HUEICHAPAN

Este trabajo ha sido parcialmente financiado
por ANID-CHILE proyecto FONDECYT 1220829

SANTIAGO DE CHILE
2023

RESUMEN DE LA TESIS PARA OPTAR
AL GRADO DE MAGÍSTER EN CIENCIAS
DE LA INGENIERÍA, MENCIÓN ELÉCTRICA
Y MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL ELÉCTRICO
POR: ROBERTO IGNACIO CHOLAKY MEJÍA
FECHA: 2023
PROF. GUÍA: PABLO ESTÉVEZ VALENCIA

REGRESIÓN SIMBÓLICA MEDIANTE REDES NEURONALES

La Inteligencia Artificial (IA) interpretativa es un enfoque para diseñar modelos que puedan ser interpretados por expertos. Una de sus áreas es la regresión simbólica, cuyo objetivo es obtener una expresión matemáticamente sencilla e interpretable a partir de los datos, que los expertos puedan analizar para comprender mejor el proceso al que se aplica la regresión.

En esta tesis se propone y desarrolla un nuevo modelo de regresión simbólica basado en redes neuronales artificiales. El modelo propuesto presenta un algoritmo de búsqueda estructural que selecciona las funciones que deben componer la expresión y luego ajusta los coeficientes de las expresiones generadas mediante retropropagación de gradientes. El método propuesto se aplica al *benchmark* Nguyen de ecuaciones, y a un modelo de baterías de litio. Entre los principales resultados se destaca la robustez del método para encontrar las estructuras que componen las expresiones buscadas, la capacidad de adaptarse a polinomios de alto orden, y la flexibilidad para ajustar los coeficientes de diferentes tipos de funciones.

Como trabajo futuro se propone la reducción de los tiempos de cómputo, el uso de combinaciones lineales sin puntos flotantes y la modificación de la estructura para permitir el uso de técnicas avanzadas de Deep Learning.

No digas que es imposible, mejor di que aún no lo has hecho.

Agradecimientos

A través de este proceso he pasado por muchas emociones, momentos dulces y momentos amargos, momentos de angustia y de mucha alegría. Finalmente, terminando este proceso me doy cuenta de toda la gente que ha estado ahí para mi cada vez que lo he necesitado y me hace sentir eternamente agradecido por esas personas.

A mi familia por todo el esfuerzo que han puesto para que pueda estudiar y desarrollarme a plenitud. A mi querida madre, que sin importar su enfermedad siempre ha luchado para salir adelante, entregándome su cariño incondicional, su oído para mis problemas, aconsejándome cada vez que sea necesario. Este logro es tanto mío como suyo.
A Florencia, mi compañera de vida, gracias por ser ese bastión que me ha apoyado e impulsado durante todo este tiempo, entregándome cariño y fuerzas en los momentos que más me faltaba. Verdaderamente, sin ti no lo hubiese logrado.
A mi tío Rigoberto por siempre recordarme que lo más importante es terminar la tesis e impulsarme cada vez que estaba desganado. Agradezco cada almuerzo en el Tobu que tuvimos conversando sobre este proceso y lo que se venía.
A Agustina, que me ha criado de pequeño, siempre buscando que no me preocupara de nada más que estudiar y que descansara las horas adecuadas.
Al Profesor Pablo por siempre buscar ayudarme y juntarse conmigo a ver mis miles de ideas distintas.
A mis amigos Arturo, Renato y Nelson, que sin importar el momento me apoyaron durante todo el proceso y creyeron en mí incondicionalmente. Quiero recordar especialmente a Nelson, que a pesar de que no pueda verme titulado sé que le habría encantado estar ahí.
A mi amigo Juan, con el que siempre pude contar, fuiste de gran ayuda durante todo el proceso, haciendo llamadas por zoom para que te contara los problemas en el desarrollo del modelo o discutir ideas nuevas.
A mis amigos Simón, Cristóbal, Ricardo y Alonso, con los que he compartido mi vida universitaria desde el primer año. Han sido parte fundamental del proceso acompañándome durante toda mi formación universitaria y personal.
A todos ellos y muchos más agradezco su apoyo y cercanía para escucharme cada vez que tuve complicaciones o alegrías en este proceso. Fueron parte fundamental de mi vida durante estos años. ¡Infinitas gracias!

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

1. Introducción

1.1. Motivación

El modelamiento de fenómenos naturales ha permitido que la civilización tenga avances en el entendimiento de su entorno y, por ende, el aprovechamiento de éste para su propio beneficio. Basta nombrar momentos históricos como las leyes de Kepler para modelar el movimiento de los planetas, las leyes de Newton que permitieron el desarrollo de la física clásica o las ecuaciones de Schrödinger para el desarrollo de la mecánica cuántica. Todos estos hitos históricos tienen su fundamento en expresiones matemáticas, las cuales permiten modelar el mundo que los rodea o los fenómenos físicos que buscan entender.

En la actualidad, la AI ha permitido el desarrollo de modelos versátiles, los cuales pueden ser usados para resolver una gran cantidad de tareas. Ejemplos de estos modelos son los métodos de reconocimiento facial que se utilizan en empresas como Meta o los asistentes virtuales que ha generado Amazon con el desarrollo de la tecnología Alexa, los cuales permiten ver la versatilidad no solamente en temáticas científicas sino que en la vida cotidiana. Todos estos modelos, tanto de Machine Learning (ML) como de Deep Learning (DL), tienen ventajas y desventajas. Una gran desventaja es su falta de interpretabilidad, es decir, modelos que tienen una alta facilidad para ajustarse a los datos pero que no pueden explicar las razones de su decisión. Esto es particularmente delicado en ámbitos donde las decisiones que debe tomar el modelo son de alto riesgo y, por ende, un error puede tener graves consecuencias.

El "Aprendizaje de Máquinas Interpretativo" es la habilidad de un modelo de ser explicado o presentado de manera comprensible para un humano, de manera tal que los resultados o decisiones tomadas por este puedan ser confiables [1]. De este modo, desarrollar el área del aprendizaje de máquinas interpretativo es una manera de entender como el modelo funciona y toma decisiones, haciendo de este modo que la gente que analiza dichos modelos pueda efectivamente confiar en sus predicciones. Dicho desarrollo permitirá que la AI se vea inmersa aún más en la vida cotidiana y en la industria.

Es en el marco del ML interpretativo donde se encuentra la regresión simbólica o Symbolic Regresión (SR) en inglés, la cual consiste en "encontrar una función en forma simbólica que logre aproximar un conjunto finito de datos muestreados"[2]. De este modo, la SR provee un método para identificar funciones a partir de los datos. En este tipo de modelos el principal desafío es encontrar expresiones que no solamente se ajusten a los datos en el conjunto de entrenamiento, sino que logren generalizar.

El trabajo de Tesis tiene por objetivo generar un modelo de aprendizaje de máquinas interpretativo el cual utilice redes neuronales dedicadas para solucionar problemas de regresión simbólica, donde se pueda encontrar una explicación de la expresión generada simplemente analizando el modelo.

1.2. Trabajos Previos

Los problemas de SR se han abordado mediante diversos métodos, los cuales generalmente dividen el problema principal de encontrar la expresión correcta en dos subproblemas. El primero hace referencia a encontrar los coeficientes adecuados de la expresión, mientras que el segundo aborda las funciones que se requieren para encontrar la solución. A cada uno de estos subproblemas los denominamos como "problema de ajuste de parámetros" y "problema de búsqueda estructural" respectivamente.

Entre los métodos que se utilizan podemos destacar los modelos de algoritmos genéticos (AG) [2]-[6]. Este tipo de métodos hacen uso de poblaciones de individuos que son representados mediante vectores, estos codifican el fenotipo de cada individuo permitiendo el armado de árboles computacionales que lo representan. La debilidad del método sobre el problema de ajuste de parámetros, se debe a que no existe una manera directa de optimizar los parámetros considerando una dirección de mejora sobre la aptitud del individuo a una tarea, es por esto que la gran mayoría de algoritmos evolutivos en SR hace uso de optimizadores no lineales como BFGS o L-BFGS para corregir el valor de los coeficientes de la solución encontrada.

Otra manera de abordar los problemas de SR es mediante aprendizaje reforzado (RL, del inglés Reinforcement Learning) [7]. En el caso específico visto anteriormente los AG intentan resolver el primer subproblema haciendo cambios en la forma de selección, mientras que los modelos que utilizan RL hacen uso de redes neuronales para modelar la distribución de probabilidad de las funciones que componen la solución. Ambos modelos hacen uso de estructuras tipo árbol, se basan en el uso de una población y de optimización no lineal para el ajuste de parámetros.

El modelo secuencia a secuencia (seq2seq) [8],[9] a diferencia de AG y RL no busca la expresión en sí, sino que la estructura de la ecuación sin considerar los coeficientes. A este se le encuentran los coeficientes presentes en la estructura mediante un optimizador no lineal.

Las redes neuronales también se utilizan para resolver problemas de SR [10],[11], en específico se les denomina Equation Learning Networks (EQL) en la literatura. Las redes EQL son redes neuronales que presentan diferentes funciones de activación que se pueden encontrar generalmente en leyes científicas, algunas de estas son por ejemplo: seno, coseno, exponencial, logaritmo, polinomios entre otros.

AIFeynman es una aproximación al problema de la regresión simbólica basado en dividir para conquistar, este método utiliza modelos básicos de regresión simbólica y heurísticas de descomposición para crear un modelo simbólico [12]. De este modo, el algoritmo va intentando distintas técnicas conocidas que simplifiquen el problema de manera tal de poder utilizar estos modelos básicos de regresión simbólica para encontrar dicha solución.

Los métodos mencionados anteriormente serán explicados en detalle en el siguiente capítulo, pero dichas aproximaciones al problema tienen en común que su enfoque principal está en encontrar las estructuras correctas al problema de regresión simbólica. De este modo, el uso de modelos "seq2seq","AG" o de "RL" enfocan su trabajo en muestrear de manera diferente el espacio de posibles estructuras, dejando la optimización de coeficientes a un optimizador no lineal. Para el caso de las redes neuronales, esta solución está enfocada en aplicar retro propagación de gradiente a las soluciones sin abordar el problema de selección de estructuras. De esta manera el estado del arte por lo general define una aproximación del problema donde soluciona concretamente una parte del problema en desmedro de la otra. En contraste, el método propuesto tiene la particularidad de abordar ambas partes del problema no enfocando el diseño del algoritmo en un solo subproblema.

1.3. Hipótesis

Es posible generar modelos de regresión simbólica que aprendan tanto la estructura, como los coeficientes para encontrar la función generadora, mediante redes neuronales dedicadas. Una vez, planteada la hipótesis general de la tesis, se desprenden las siguientes hipótesis complementarias:

1. Un modelo de regresión simbólica basado en redes neuronales dedicadas a problemas de SR debiese converger a óptimos cercanos a la solución, de manera eficiente.
2. Una red neuronal con una estructura de funciones correcta debiese aprender más rápido y mejor que una que no la tiene.
3. Los modelos que se entrenen mediante retro-propagación de gradiente debiesen generar mejores puntos de partida para el posterior uso de optimizadores no lineales en caso de tener la estructura correcta.

1.4. Objetivos

1.4.1. Objetivo General

Proponer y desarrollar un método de regresión simbólica basado en redes neuronales dedicadas que permita aprender tanto la estructura de la solución como sus coeficientes.

1.4.2. Objetivos Específicos

Los objetivos específicos del trabajo de tesis son los siguientes:

- Diseñar, implementar y evaluar un modelo de red neuronal dedicada y su correspondiente método de entrenamiento que puedan resolver problemas de regresión simbólica de hasta 3 variables.
- Comparar el modelo propuesto con el estado del arte en regresión simbólica, en una base de datos sintéticas de regresión simbólica que tengan función generadora conocida, donde los resultados serán evaluados mediante las métricas R2, NRMSE y tiempo de cómputo.
- Aplicar el modelo propuesto a una base de datos de baterías de lítio.

1.5. Estructura de la Tesis

El presente trabajo de tesis está organizado en 5 capítulos. En el capítulo 2, *Marco Teórico*, se presentan el marco teórico utilizado a través del trabajo de tesis, además de los métodos que se han utilizado anteriormente para resolver el problema de regresión simbólica. En el capítulo 3, *Metodología*, se describe el modelo desarrollado en esta tesis. En el capítulo 4, *Resultados y Análisis*, se muestran los resultados obtenidos junto con su análisis respectivo para los problemas resueltos. Finalmente, en el capítulo 5, *Conclusiones*, se hace un resumen de los resultados encontrados, los análisis realizados, su impacto en el área de la regresión simbólica, además de proponer trabajos futuros que podrían mejorar el presente trabajo.

2. Marco Teórico

Este capítulo tiene como objetivo explicar a cabalidad el problema de la regresión simbólica. Para esto primero se explicará qué es la regresión simbólica y se diferenciará de otros métodos de regresión los cuales utilizan modelos de cajas negras. En la sección 2.2 se explica detalladamente las distintas técnicas que existen en la actualidad para abordar el problema de la regresión simbólica, además de explica las fortalezas y debilidades de cada método. Posteriormente, se explican algunos términos generales que se utilizan a través de la tesis como son las métricas comunes utilizadas para los problemas de regresión simbólica y la regularización utilizada en modelos de "Deep Learning" Finalmente se concluye con una discusión sobre el capítulo indicando qué modelos y qué partes de los modelos sirvieron de inspiración para generar el modelo de regresión simbólica propuesto.

2.1. Regresión Simbólica

La regresión simbólica (SR) consiste en "encontrar una función en forma simbólica que logre aproximar un conjunto finito de datos muestreados"[2]. De este modo, La SR provee un método para identificar funciones a partir de los datos. El principal desafío es encontrar expresiones que se ajusten a los datos en el conjunto de entrenamiento, y que además logren generalizar bien ante nuevos datos. La teoría se basa en la suposición de que existe una función generadora del conjunto de datos la cual al recuperarse mediante métodos de regresión simbólica permita encontrar una expresión que transforme de manera correcta las entradas del modelo a la salida que se está buscando. Dicha expresión debiese generalizar los resultados incluso fuera del conjunto de entrenamiento.

Los problemas de regresión simbólica en la actualidad constan de dos partes que los diversos modelos de la literatura tratan de abordar. El primero hace relación con el ajuste de coeficientes y lo denominaremos problema de ajuste de parámetros, mientras que el segundo hace relación con la estructura de la solución y lo denominaremos problemas estructurales. Todo modelo en la literatura trata de abordar estos problemas usualmente con énfasis mayor en alguno de estos problemas.

Para ilustrar un problema de regresión simbólica, supongamos que existe un sistema S que genera salidas (y) que se rigen por la siguiente ley numérica desconocida:

$$y = exp\left(-\frac{(x-\mu)^2}{\sqrt{2\pi\sigma^2}}\right), \tag{2.1}$$

Se desea modelar y para así poder entender la relación existente entre la entrada x y la salida y. Un modelo de regresión se entrenaría con un conjunto de datos (x, y) donde x es la entrada al sistema e y la respuesta para dicho valor de x. Una vez terminado el entrenamiento

dicho modelo estará ajustado a los datos y por ende al recibir x_i como entrada, esta retorne el valor y_i correspondiente. En este caso la representación del modelo sería la siguiente:

$$\hat{y} = f(x; \theta), \tag{2.2}$$

donde θ representa los parámetros del modelo entrenado y f es la función que representa el modelo, dicha función es desconocida y solo se puede acceder a ella mediante el uso del modelo. Notemos como en este caso el modelo entrenado no permitiría entender que existe una relación entre la entrada x y las variables internas del sistema como son μ y σ.

Ahora, si se entrenara un modelo de regresión simbólica, el modelo también sería entrenado con el mismo conjunto de datos (x, y) pero una vez entrenado el modelo se tendría tanto un modelo que se ajuste a los datos, como también una representación escrita de la expresión que relacione x con y. En este caso dicho modelo podría entregar una expresión escrita de la siguiente forma:

$$\hat{y} = exp\left(-\frac{(x - 0.997)^2}{2.5}\right), \tag{2.3}$$

en este caso, asumiendo que los valores reales del sistema para μ y σ son ambos 1, el modelo entregaría una representación donde $\mu = 0.997$ y $\sqrt{2\pi\sigma} = 2.5$ mostrando de este modo que los valores de y decrecen de manera exponencial al aumentar x y que dicha expresión depende de ciertas constantes. Notemos como toda esta información se pierde al simplemente utilizar un modelo de regresión.

En este caso el problema de ajuste de parámetros sería poder aproximar los coeficientes del problema, mientras que el problema estructural haría referencia a encontrar las funciones que se deben ocupar (en este caso exponencial, división, potencia y resta) y el orden en que se deben unir.

2.1.1. Aprendizaje de Máquinas Interpretativo

El "Aprendizaje de Máquinas Interpretativo" es la habilidad de un modelo de ser explicado o presentado de manera comprensible para un humano, de manera tal que los resultados o decisiones tomadas por este puedan ser confiables [1]. Este es distinto al "Aprendizaje de Máquinas explicativo" el cual es otra rama del ML, que busca interpretar mediante otro modelo las predicciones realizados por un modelo de caja negra que resuelve el problema. Una de las razones de porque es importante hacer la distinción entre ambos métodos, es que el ML explicativo es un método que trata de entregar información de cómo el modelo funciona y toma decisiones sin entregar detalles del cómputo que este realiza, mientras que el ML interpretativo restringe su estructura para que la interpretación del modelo pueda ser útil para el usuario que ya tiene cierto conocimiento del área, o para obedecer conocimientos estructurales del problema que se conocen de antemano [1].

La SR se encuentra en el marco del ML interpretativo, ya que su finalidad al igual que el ML interpretativo es encontrar una representación del modelo que permita reemplazar a un modelo de caja negra. Esto se ilustra en la figura 2.1, donde existe un sistema que genera salidas y nos gustaría entender como funciona, para reducir costos de funcionamiento o mejorar su desempeño. Un primer enfoque es aproximarlo con un modelo de caja negra, pero el problema de dichos modelos de caja negra es que no permiten entender lo que considera el modelo para generar el cómputo. Por otro lado están los modelos interpretativos, cuya estructura está diseñada para ser interpretado y entender el resultado al que llega dicho modelo. De este modo, soluciones que utilicen modelos de ML interpretativo tienen la ventaja

de tener a su disposición una expresión analítica o estructuras explicativas, que es justamente lo que hoy en día no se obtiene con el uso de modelos de caja negra.

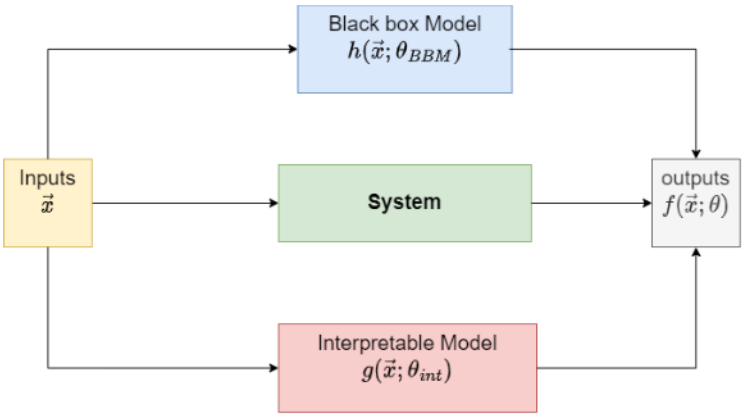


Figura 2.1: Modelo interpretable en comparación a modelo de caja negra, ambos presentan un buen ajuste al sistema que se busca representar pero al contrario de los modelos de caja negra, el modelo interpretable permite que sus salidas sean interpretadas por un experto, para asegurar que dicho resultado sea correcto.

2.2. Modelos de Regresión Simbólica

2.2.1. Programas Computacionales Tipo Árbol

Los programas computacionales están entre las estructuras más complejas creadas por ser humano [2], estos incluyen programas que tienen la capacidad de resolver problemas sin necesidad de ser programados para hacerlo. Esto quiere decir que no se debe indicar la forma, el tamaño o la complejidad estructural de dicho programa, sino que este debe emerger del proceso de solución del problema. La razón de porque interesan los programas computacionales se debe a su capacidad para realizar:

1. Operaciones de un modo jerárquico.
2. Cómputos condicionados a resultados intermedios.
3. Iteraciones y recursiones.
4. Realizar cómputos de variables de diferente tipo
5. Generar subprogramas que pueden ser reutilizados.

El objetivo de los métodos que utilizan programas computacionales es aprovechar la flexibilidad de estos para resolver la tarea. Esto genera la necesidad de buscar dentro del espacio de programas computacionales el programa deseado. Es importante remarcar que dicho espacio según [2] es un espacio muy amplio como para buscar de manera aleatoria por lo que se requieren métodos que permitan realizar búsquedas de manera adaptativa e inteligente.

En el caso específico de la regresión simbólica, el programa computacional se representa mediante árboles. La dificultad está en encontrar la estructura de la expresión buscada así

como sus coeficientes. De este modo, el árbol se compone de un modelo que toma como entradas las variables independientes del problema y retorna las variables dependientes del mismo como salida. Formalmente el problema de regresión simbólica expuesto para un programa computacional se puede definir de la siguiente manera:

El conjunto de posibles estructuras para un árbol está compuesto por todas las posibles composiciones de funciones que se pueden generar de manera recursiva de un conjunto de $F = f_1, f_2, f_3, ..., f_{Nfunc}$ y el conjunto de nodos terminales $T = a_1, a_2, ..., a_{NT}$. Cada función en el conjunto de funciones toma un número específico de entradas $z(f_i)$.

Los nodos terminales en estos casos son típicamente las variables independientes del problema o algunos valores constantes, dependiendo del caso que se busque resolver. Además, toda función del conjunto de funciones debe poder recibir en su dominio cualquier salida de las otras funciones del conjunto. En el caso específico de la regresión simbólica, las funciones o operadores en el conjunto F generan salidas numéricas que pueden sufrir de problemas de indefinición donde podemos destacar errores de dominio para el caso de las funciones (logaritmo evaluado en cero, raíz cuadrada evaluada en números negativos) o errores por definición de operadores como es el caso de la división por cero. Es por esto que las funciones que presente el conjunto F deben estar corregidas para así evitar restricciones complejas sobre la sintaxis estructural del programa computacional que puedan complicar la obtención de un candidato óptimo.

2.2.2. Algoritmos Evolutivos

En la naturaleza, los procesos evolutivos ocurren cuando las siguientes cuatro condiciones se satisfacen:

1. Una entidad tiene la habilidad de reproducirse.
2. Existe una población de entidades que se pueden reproducir.
3. Existe una variedad entre estas entidades que pueden reproducirse.
4. Alguna diferencia en la habilidad de supervivencia se asocia a dicha variedad.

Dicha variedad se manifiesta como una diferencia en los cromosomas de las entidades de la población, la que se ve reflejado en diferencias tanto estructurales como de comportamiento. Estas variaciones en el largo plazo se reflejarán en una mayor o menor tasa de supervivencia, estos conceptos se denominan “Supervivencia del más apto” y “selección natural” habiendo sido descritos por Charles Darwin en el libro “El origen de las especies” (1859).

De este modo, a medida que pasa el tiempo, la población pasa a contener una mayor cantidad de individuos que se desenvuelven mejor en su entorno debido a la prevalencia de cromosomas que generan estructuras y comportamientos mejor adaptados. Los cuales, por ende, tienden a reproducirse y sobrevivir más. Este es el efecto de la selección natural a través de las generaciones de una población, que incentiva nuevas estructuras a medida que estas sean más aptas para la supervivencia. Los algoritmos evolutivos, simulan los procesos y operaciones naturales que ocurren en los cromosomas para transformar un conjunto (población) de objetos matemáticos individuales, cada uno con un valor numérico que representa su aptitud para la tarea que se busca resolver, en una nueva población [2].

En la práctica se utiliza una población de individuos que son representados mediante vectores, estos codifican el genotipo de cada individuo permitiendo el armado de los objetos

matemáticos que lo representan.[1] La evolución propiamente tal se inspira en la posibilidad que tienen los individuos de mutar y reproducirse, esto se logra con dos operaciones denominadas mutación y entrecruzamiento. Por lo tanto al inicio se genera una población con condiciones bases conocidas, luego a ésta se le calcula su aptitud (*fitness*) para la tarea en cuestión, una vez realizado este proceso un subconjunto de los mejores individuos es seleccionado [2] para luego proceder a la reproducción de estos y su posible mutación. Una vez terminado este proceso a estos nuevos individuos se le denomina segunda generación y el proceso itera por una cantidad de generaciones definida con anterioridad.

2.2.2.1. Algoritmos Genéticos (AG) y Regresión Simbólica

Para el problema de la regresión simbólica el método más común de decodificación y representación de un genotipo utilizando programas computacionales de tipo árbol [2]. Los árboles computacionales se generan con nodos terminales y nodos no terminales. En los nodos terminales se encuentran las variables del problema además de constantes para la generación de expresiones. Por el otro lado, en los nodos no terminales se encuentran las funciones que utilizan los nodos terminales y subproductos de los nodos no terminales inferiores para generar una expresión. Esto se puede apreciar en la figura 2.2, donde se tiene una representación del genotipo a la izquierda, su representación fenotípica en el centro de la imagen y la salida que genera al momento de evaluarse. Por último la evaluación de dichos árboles se hace desde los nodos terminales hacia la raíz del árbol.

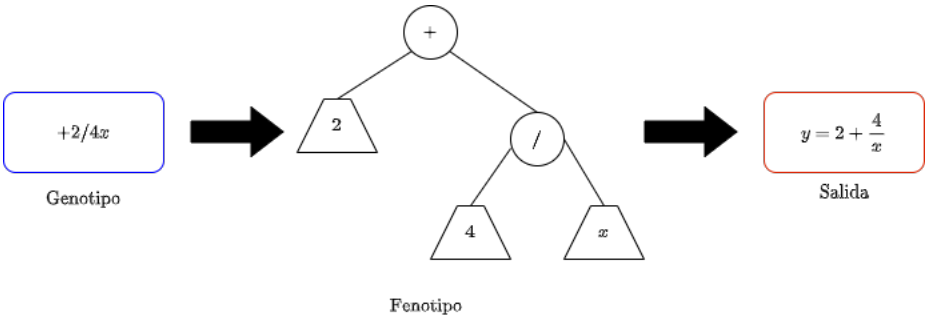


Figura 2.2: La figura muestra como el genotipo del individuo (+2/43) puede ser representado como un programa computacional de tipo árbol, este se encuentra representado en la figura por un grafo acíclico. el cual puede ser evaluado en un computador.

Como se mencionó anteriormente, la inicialización de la primera generación de individuos es aleatoria, En la práctica esto permite hacer pruebas sobre el espacio de búsqueda el cual queda definido por el largo del cromosoma, las funciones que se utilizan para la aproximación y además del tamaño de los valores constantes que se utilizarán. Todos estos son definidos con anterioridad y específicamente para el problema.

Una vez inicializada la población, se procede a evaluar los individuos generados, estos se seleccionan mediante algún método de selección que se define con anterioridad, algunos ejemplos son la selección por torneo o la selección por ranking [2]. La idea detrás de los métodos de selección radica principalmente en la búsqueda de soluciones diversas que permita

[1] Los vectores en su representación codificada se denominan genotipos mientras que los objetos matemáticos ya decodificados tienen la denominación de fenotipos.

[2] A este proceso se le denomina selección y se explicará en profundidad más adelante.

recorrer de manera adecuada el espacio de búsqueda para así encontrar óptimos globales en las soluciones y no óptimos locales.

Terminada la selección se procede a la reproducción de los individuos seleccionados mediante recombinación, en este se eligen dos individuos diferentes, se define al azar un punto del largo del cromosoma[3] y se reemplazarán de manera inversa cada tramo desde los extremos hasta el punto de recombinación haciendo luego el cambio respectivo con los otros individuos. Esto se ilustra en la figura 2.3. Este cambio de "material genético" tiene la finalidad de mover el espacio de búsqueda en el que se estaba con la generación anterior a otro que haya sido inexplorado.

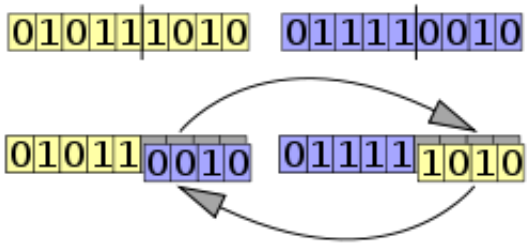


Figura 2.3: recombinación en algoritmos genéticos, el punto desde donde se corta el cromosoma se denomina punto de recombinación y este punto marcará el intercambio de material genético con el otro individuo.

Una vez que se han reproducido los individuos seleccionados de la población, se procede a definir si estos realizarán una mutación de sus genes. En general, en este tipo de algoritmos se define una probabilidad de mutación de un gen aleatorio, lo que permite variar el espacio de búsqueda ya definido por la reproducción y hacerle pequeñas variaciones y por ende salir de mínimos locales que se puedan generar en el proceso. Por lo general, la forma de realizar este proceso es cambiando aleatoriamente un gen por otro valor aleatorio. Posterior a este proceso se debe iterar los pasos anteriormente expuestos sobre una cantidad predefinida de generaciones.

El *fitness* utilizado es otra diferencia importante entre diversos métodos de algoritmos genéticos. Métodos como [3]-[6] utilizan funciones multiobjetivo que buscan optimizar más de una condición, forzando de este modo la convergencia a soluciones buscadas. En [3] se indica que en regresión simbólica generalmente hay una o dos funciones objetivo. Esta condición más los posibles cambios en los pasos anteriormente mencionados, generan diversos métodos que se encuentran en la literatura. Métodos como [3]-[6] cambian las funciones objetivo y además el método de selección para obtener soluciones más variadas.

Un punto relevante de los algoritmos evolutivos es que los procesos presentan una componente aleatoria que no asegura una convergencia hacia los óptimos globales, normalmente el espacio de búsqueda es muy grande y dicha búsqueda debe hacerse no solamente sobre la estructura de la solución, sino que además sobre los parámetros de la misma [2], [7]. Esto hace que los algoritmos evolutivos tarden más tiempo en encontrar las soluciones y que existan casos en que estructuras correctas no logren ser óptimas ya que sus coeficientes están mal inicializadas, su actualización es aleatoria y su mejora no está dirigida hacia un óptimo necesariamente. Es por esto que los AG se basan en el uso de grandes números de individuos para así poder recorrer de mejor manera el espacio y por ende encontrar mejores soluciones.

[3] Este punto se denomina punto de recombinación

También existen métodos para reducir el espacio de búsqueda que han sido utilizadas en [3],[13],[14] donde se utilizan restricciones sobre la solución encontrada las cuales definen si un individuo es o no válido para el problema a resolver.

Otra forma de restringir el espacio de soluciones es la usada por los métodos de evolución gramatical [15], [16], [17], los cuales definen reglas gramaticales o estructurales para decodificar cada secuencia. Una de las ventajas de este método es que los nodos no terminales generan una gramática que a su vez genera más nodos terminales y no terminales, mientras que los nodos terminales son fijos y por ende no alargan la expresión. Esto permite que una solución no se vea limitada por el tamaño de los cromosomas sino que se basa en el ajuste a la función objetivo.
Existen métodos complementarios que buscan optimizar las coeficientes que tienen los individuos de AG, el método más común es mediante escalamiento lineal [18], [19], cambios de subárboles mediante retro propagación semántica [20] o optimización de coeficientes mediante gradiente descendente [21].

Por último, revisando el desarrollo del estado del arte de los AG, se puede indicar que el problema de los tiempos de resolución, el espacio de búsqueda, la inicialización y mejora de coeficientes son todavía un desafío que se busca resolver.

2.2.3. Equation Learning Networks (EQL)

2.2.3.1. Redes Neuronales (MLP)

Las redes neuronales profundas son un modelo de *Deep Learning* el cual tiene como finalidad aproximar una función f^*. En estos modelos la información fluye en una sola dirección desde la entrada x hasta la salida intermedia que define f para así obtener una salida deseada y. Estos modelos tienen la denominación de redes ya que generalmente se representan como una composición de varias funciones, una manera de representar una red neuronal es mediante un gráfico direccionado acíclico que muestra como las funciones están compuestas entre ellas.

El entrenamiento de las redes neuronales tiene el objetivo de que el modelo $f(x)$ se ajuste a la función deseada $f^*(x)$. Para esto, los datos de entrenamiento nos entregan una aproximación ruidosa de la función deseada en distintos puntos del dominio de entrenamiento. El conjunto de entrenamiento está compuesto de entradas x y sus correspondientes salidas $y \approx f^*(x)$. Esto permite que los ejemplos de entrenamiento especifiquen que debe hacer la capa de salida en cada punto x para obtener un valor cercano a y. De este modo, el algoritmo de entrenamiento es el responsable de encontrar la manera de usar esas capas para así obtener la salida buscada. Estos modelos introdujeron el concepto de capas ocultas, donde se utilizan funciones de activación no lineales para calcular la salida de dicha capa. Esto permite que las capas siguientes no puedan ser fácilmente condensadas a una sola capa ya que si no existiesen dichas funciones de activación o estas fueran funciones lineales todas las capas podrían ser reemplazadas por una sola capa.

Dichas redes neuronales se entrenan mediante el método de retró propagación del gradiente, que hace uso de la regla de la cadena para calcular el gradiente de toda la red. Esto es posible gracias a que las conexiones que tienen dichas redes son en cadena, por lo que solo basta la regla de la cadena para calcular los gradientes de la función de pérdida con respecto a todos los parámetros. El método del gradiente descendente busca minimizar el valor de la

función de pérdida, pero no garantiza la convergencia a un óptimo global, sino que más bien esta convergencia depende de la convexidad de la función de costos además de la inicialización de los parámetros de la red [22]. Los métodos de gradiente descendente en redes neuronales hacen uso de particiones de los datos de entrenamiento en conjuntos llamados *batches*, estos permiten que los parámetros del modelo se actualicen una mayor cantidad de veces utilizando solamente un número predefinido de datos para su actualización.
Existen varios métodos de optimización de parámetros de redes neuronales tales como Stochastic Gradient Descent (SGD), Adam, RMSprop, los que hacen uso de medias móviles sobre los gradientes calculados para que así un mal muestreo de los datos de entrenamiento no genere efectos tan grandes sobre la dirección de mejora de los parámetros del modelo.

2.2.3.2. Equation Learning Networks

Las *Equation Learning Networks*(EQL) [23] son redes neuronales que presentan diferentes funciones de activación que se pueden encontrar generalmente en leyes científicas, como por ejemplo: seno, coseno, exponencial, logaritmo, polinomios entre otros. La única restricción que existe sobre dichas funciones es que sean continuas en todo el dominio y en caso de no serlo que se puedan modificar para evitar dichas discontinuidades. Estas redes permiten realizar procesos de regresión sobre los datos que a la vez entregan una representación simbólica. Al final del entrenamiento se puede hacer un análisis sobre la red para obtener la función que representa los datos de entrenamiento.

La idea basal de este tipo de redes se encuentra en que la estructura de una ecuación se genera en cada capa, donde la separación se genera del uso de una función de activación. Por ejemplo, $sin(x^2)$ se debería generar con 2 capas, una que aplique una elevación al cuadrado y la segunda que aplique una función seno. En la figura 2.4 se ilustra una muestra gráfica de dicho modelo.

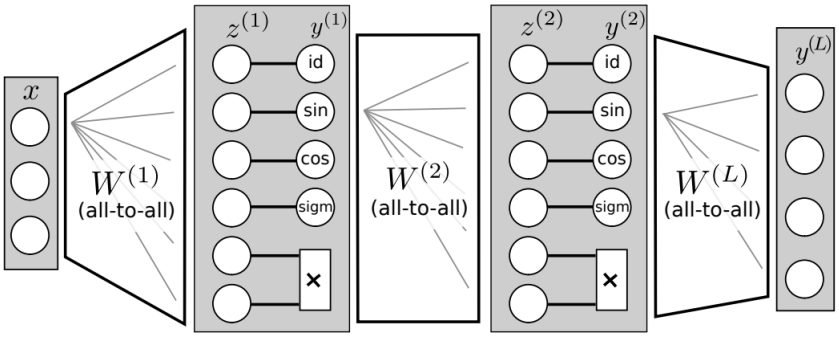


Figura 2.4: Estructura de red EQL que muestra como los pesos de la red neuronal se utilizan para generar salidas de una capa oculta ($z^{(i)}$). A estas se les aplica la función ($y^{(i)}$) que se indica en la imagen logrando encontrar una expresión matemática que mapee la entrada a la salida. Tomada de [23]

De este modo, podemos observar que en primera instancia las funciones que utiliza la red expuesta en la figura 2.4 son todas continuas y distintas. Además, salvo la función sigmoide todas son usadas en expresiones o fórmulas científicas. Para generar funciones más complejas que tengan un mayor número de composición de funciones es necesario utilizar más capas para así generar dichas expresiones. El trabajo que se ha realizado sobre las redes EQL no es muy extenso y la mayoría de los cambios que se le han hecho están basados en [10], el cual es la continuación de [23], y agregan a la capa de salida la posibilidad de poder generar

divisiones. Otro trabajo que avanzó en la línea de mejorar el modelo inicial de red EQL es [11] el cual cambia dicha red agregando una función de x^2 que permite generar las mismas funciones pero con una menor cantidad de capas (aunque en este caso sin la posibilidad de realizar divisiones). También entrena dicha red con una regularización $L_{0.5}$ (presentada en dicho trabajo), la cual busca asemejarse a la regularización L_0 de una manera continua.

2.2.4. Modelos Secuencia a Secuencia

Los modelos secuencia a secuencia (seq2seq) propuestos en [24] surgen para dar solución a problemas donde los datos de entrada son una secuencia de valores mientras que los datos de salida son otra secuencia de valores. En esta tarea la salida deseada no necesariamente se encuentra en el mismo espacio que la entrada del modelo, haciendo que los modelos tengan que generar secuencias de salidas de largos diferentes a la entrada. De este modo, en [24] se presenta un modelo tipo *encoder-decoder*, que primero utiliza una red neuronal recurrente para generar una representación del espacio al que se busca llevar la secuencia de entrada x, y luego mediante otra red recurrente generar la secuencia de salidas que representan la salida deseada (y) dada la entrada x. Este tipo de modelos son muy utilizados en tareas de procesamiento de lenguaje natural, donde a partir de una secuencia inicial que represente una frase, se busca llegar a otra secuencia de salida que represente una frase de respuesta o una traducción a otro idioma. La figura 2.5 muestra una representación gráfica de como se utiliza el modelo seq2seq sobre dos frases diferentes. En la esta se puede observar como se tienen dos modelos uno que realiza una codificación sobre la entrada, que en este caso es $ABC < EOS >$, donde $< EOS >$ es un caracter especial que representa el final de la secuencia, este modelo a la salida genera un estado que representa a toda la secuencia de entrada y una salida final. Esto representa el trabajo del *encoder*. Dicho estado se entrega al decodificador como estado inicial junto con la última salida como. Esto genera una salida que en la figura podemos ver representada como X y esta se vuelve a ingresar como entrada al modelo decodificador, esto se itera hasta que el segundo modelo genere un símbolo de fin de secuencia.

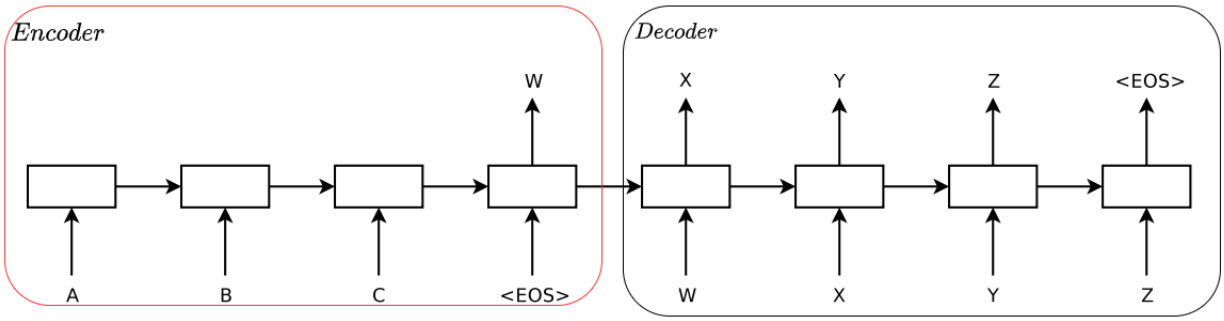


Figura 2.5: Modelo seq2seq, es un modelo al que se le entrega una secuencia de entradas y por cada caracter el modelo generará una salida (flecha hacia arriba en la parte superior de los rectángulos) y un estado (flecha hacia la derecha). El *encoder* representa la parte del modelo que recibe una entrada ABC<EOS> y genera un estado que representa dicha secuencia. Por el otro lado el *decoder* recibe la última salida del *encoder* (W) y genera una nueva salida y estado utilizando la salida anterior hasta que se genere un caracter <EOS> que representa el final de la secuencia. Imagen extraída de [24].

2.2.4.1. Redes Recurrentes

Las redes neuronales recurrentes (RNN_s) sirven para procesar datos secuenciales. Dada una secuencia de entradas $(x_1, ..., x_T)$, la red genera una secuencia de salidas $(y_1, ..., y_T)$ [24]. Parte importante de dicha red es que se comparten pesos a través de distintas partes del modelo, esto permite que la red pueda ser aplicada a datos de largos distintos sin que el largo de la secuencia sea una variable a definir del problema. Una de las principales diferencias con las redes neuronales convencionales es que si se quisiera hacer uso de estas para una secuencia de datos, entonces se tendría que definir un largo de secuencia y además definir un peso respectivo para cada característica de la entrada, haciendo que el modelo tenga que aprender a reconocer patrones por separado en cada posición de la secuencia [22].

Las RNN_s se puede representar mediante grafos computacionales, donde existe una dependencia temporal entre los estados del sistema en tiempos anteriores y el estado. Esto se puede ver representado en la siguiente expresión:

$$s^{(t)} = f(s^{(t-1)}; \theta). \tag{2.4}$$

La ec. (2.4), el estado en el tiempo t depende del estado en el tiempo t-1 y que el modelo que los relaciona tiene un conjunto de parámetros que permite realizar dicha operación. Esto se ilustra en la figura 2.6.

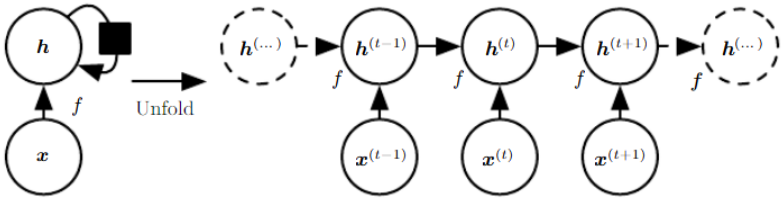


Figura 2.6: Una unidad recurrente desarrollada a través del tiempo. El modelo utiliza los estados previos para así poder generar los estados futuros tanto con la entrada siguiente como con el estado que entregó el modelo en el tiempo anterior. Imagen extraída de [22].

La manera en que se logra iterar sobre los estados pasados para generar futuras salidas y estados se realiza mediante las ec. (2.5), donde b, c son un vector de *bias*, las matrices W, U y V son matrices de pesos. Estas ecuaciones se calculan para cada instante de tiempo, entregando salidas en dichos instantes, por lo que para calcular el error total en el que incurren este tipo de redes basta con sumar los errores particulares en los que incurre cada salida y así calcular la actualización de los gradientes.

El cálculo del gradiente a través de la RNN se hace a través de Back-propagation through time (BPTT), este es el mismo algoritmo que el back-propagation común pero se aplica al grafo expandido del modelo de red recurrente que se detalla en los anexos A.1. Las ecuaciones

de una celda recurrente son:

$$
\begin{aligned}
a^{(t)} &= b + Ux_t + Wh_{t-1} \\
h^{(t)} &= tanh(a^{(t)}) \\
o^{(t)} &= c + Vh^{(t)} \\
y^{(t)} &= softmax(o^{(t)})
\end{aligned}
\tag{2.5}
$$

A pesar de que la idea de utilizar el estado anterior en una celda recurrente para generar mejores predicciones sobre una secuencia de datos es atractiva, dichos modelos sufren de importantes limitaciones. El problema más común que presentan estos modelos es que el gradiente propagado a través de múltiples secuencias muy largas tiende a desvanecerse o explotar (esto último en menor medida). Este problema nace debido que el valor de los gradientes se vuelve cada vez más pequeño debido a las iteraciones sobre largas cadenas de datos. (Esto se debe a que la multiplicación de múltiples Jacobianos para su actualización en comparación a cadenas más cortas [22]. Para solucionar dicho problema en la literatura se crearon las RNN con compuertas que incluyen tanto los modelos "gated recurrent unit (GRU)", además de "long short-term memory (LSTM)". Este tipo de modelos se basa en la idea de crear caminos a través del tiempo en que el gradiente no se desvanezca ni explote. Esto se logra en las RNN con compuertas que cambian su valor en cada instante de tiempo. Para este trabajo de tesis se describirá solamente el modelo de red LSTM.

El uso de bucles interiores que generen caminos por donde el gradiente puede fluir por un mayor tiempo es una de las principales contribuciones de los modelos LSTM. Esto permite que la cantidad de instantes de tiempo que se consideran al momento de realizar BPTT no tiene que ser necesariamente el largo de la cadena sino que puede bastar con una cantidad más acotada de instantes de tiempo, ya que estos se cambian de manera dinámica. La estructura del modelo de red expuesto puede verse en la figura 2.7, donde se muestra que el diseño contempla tres compuertas, una de entrada al modelo, otra es para el estado de la unidad y la última para la salida final. La unidad LSTM puede aprender que partes de la entrada y del estado de tiempo anterior debe considerar o no de manera de aprender de mejor manera.

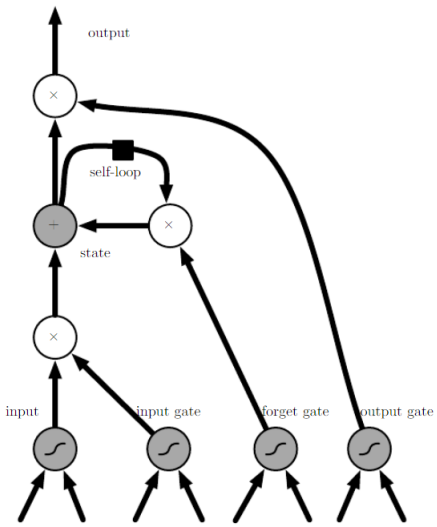


Figura 2.7: Diagrama de la estructura de una celda recurrente LSTM, que tiene la opción de decidir cuanto de la entrada, del estado y de la salida se va a considerar para entregar una salida final. La imagen fue extraida de [22].

Otro problema existente sobre largas cadenas de datos se encuentra en que mientras mayor la secuencia mayor es la información que esta contiene, como sería por ejemplo con el párrafo de un libro. Esto genera que como el estado de la red es de un largo fijo la información deba ser condensada a un vector de menor tamaño [25], haciendo que dicho traspaso de información tenga una pérdida de información, ya que desde un espacio de mayor dimensión se busca llegar a uno con menor dimensionalidad.

2.2.4.2. Transformers

Como se indicó anteriormente uno de los mayores problemas que presentan los modelos de redes recurrentes es la necesidad de condensar la información de la entrada en un vector de contexto de largo fijo [25]. Este problema se genera ya que el vector de contexto debe considerar todos los instantes de tiempo sin obligatoriamente enfocarse en las partes más importantes de dicha oración. Para solucionar este problema en [25] se plantea la base del mecanismo de atención, que propone utilizar una suma ponderada de los contexto de diferentes instantes de tiempo como vector de contexto, donde los coeficientes de dicha suma ponderada son aprendidos mediante una red neuronal. Esto hace que la forma de aprender del mecanismo de atención sea mediante la focalización de dichos pesos sobre instantes de tiempo específicos. El vector de contexto se computa como:

$$c_t = \sum_{j=1}^{T_x} \alpha_{i,j} h_j. \tag{2.6}$$

donde los vectores de estado (h_t) se ponderan con un peso de atención ($\alpha_{i,j}$) para generar el vector de contexto de ese instante de tiempo. Los pesos de atención deben sumar uno y se definen como:

$$\alpha_{i,j} = \frac{exp(e_{i,j})}{\sum_{k=1}^{T_x} exp(e_{i,k})}. \tag{2.7}$$

La red tiene que realizar un equilibrio entre considerar todos los contextos por igual o concentrarse en algunos instantes de tiempo en específico.

El modelo Transformer reemplaza las redes recurrentes por una misma estructura encoder-decoder pero que en vez de utilizar redes recurrentes hace uso del mecanismo de atención para poder usar la información pasada. Esto se hace bajo la idea de los vectores "consulta", "llave", "valor", estos deben pensarse de la siguiente manera. Los vectores de "consulta" son las entradas que se le dan al modelo de decodificación, lo que se puede pensar como una referencia a la que queremos enfocarnos, mientras que la "llave" representa a los estados de cada instante de tiempo (en general se utiliza que la "llave" y el "valor" sean los mismos). De este modo, se puede pensar en el modelo de atención expuesto en los Transformers como un método que busca encontrar la "llave" que más se parezca a la "consulta" y enfocarse en dicha "llave" para utilizar su valor en la ponderación de la expresión (2.7), Lo expuesto anteriormente se puede ver en la figura 2.8.

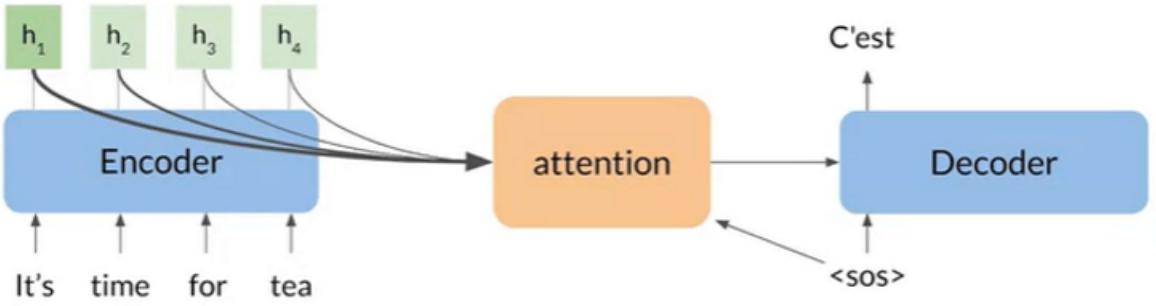


Figura 2.8: Ejemplo de como se utilizan las llaves y los valores en la atención del transformer. Este ejemplo es de un problema de traducción de idiomas donde para una oración se generan 4 vectores de estado (hi), en el *decoder* se utiliza el inicio de oración como vector de consulta y esto permite realizar el cálculo del vector de salida. Imagen extraída del curso 'sequence models and attention mechanism" de Andrew NG

En la práctica, los modelos basados en transformers han mostrado mejor escalamiento hacia secuencias más largas. Esto se debe a que al no utilizar recurrencia, el modelo no presenta problemas de gradiente desvaneciente o explosivo [22] ni tampoco el problema de dimensionalidad limitada expuesto en [25]. Además, la base de las RNN es la recurrencia sobre la celda que permite generar de manera iterativa los estados, mientras que en el modelo de Transformers todos los estados son calculados de manera simultanea (no hace uso de recurrencia). El modelo transformer es más eficiente tanto en costo computacional como facilidad de entrenamiento.

2.2.4.3. Seq2Seq En Regresión Simbólica

El modelo secuencia a secuencia a diferencia de las redes EQL y los Algoritmos Genéticos no busca la expresión en si, sino que encontrar la estructura de la ecuación sin considerar los coeficientes. Como el modelo seq2seq se ha desarrollado dentro del área del procesamiento del lenguaje natural, su uso está basado en la idea de que la solución sea escrita como una secuencia de símbolos dentro de una librería que contenga funciones, variables y caracteres que representen un coeficiente. Como el modelo genera expresiones, para entrenarlo se debe utilizar entropía cruzada. El modelo genera una secuencia que al ser terminada representa

la función predicha del conjunto de datos, de este modo, cada elemento de la cadena de símbolos tiene un error asociado al ser comparado con el símbolo de la secuencia que debiese encontrarse en esa posición y por ende el error total de la secuencia es la suma de dichos errores individuales.

Una vez obtenida la estructura de la función mediante el modelo seq2seq, se determinan los coeficientes de esta mediante un optimizador no lineal que busca minimizar el error cuadrático medio normalizado. Estos métodos de resolución de problemas de regresión simbólica separan los dos subproblemas presentes en la regresión simbólica con un modelo para encontrar la estructura y otro para encontrar los coeficientes de la estructura predicha. Este esquema se puede ver en la siguiente figura 2.9

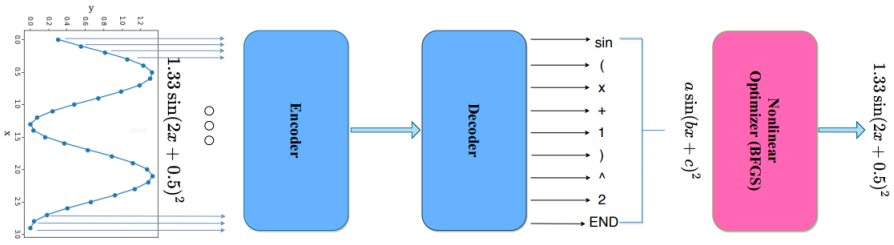


Figura 2.9: El método Seq2Seq de resolución de regresión simbólica utiliza una estructura *encoder-decoder* el que una vez entrenado puede distinguir la estructura de diversas ecuaciones. Al modelo se le entrega una secuencia de datos de la señal que se busca encontrar, este retorna un esqueleto de solución, el que posteriormente es pasado por un optimizador no lineal para entregar la solución final al problema. Esta figura fue extraída de [8].

Una parte fundamental del método presentado es el entrenamiento de la búsqueda de estructuras del modelo, esto ya que debe ser capaz de diferenciar la estructura correcta en el espacio de búsqueda de posibles estructuras. Para cumplir con este fin es necesario generar un conjunto de entrenamiento que esté compuesto de varias señales con diversas estructuras, junto con las expresiones que las generan. Utilizando la señal como una secuencia de entrada de valores y la expresión buscada como la secuencia de salida es posible entrenar el modelo seq2seq con entropía cruzada de la misma manera en que se entrenaría un modelo [24], [26]. Este proceso de entrenamiento del modelo de secuencia tiene dos implicancias importantes para el proceso de búsqueda de ecuaciones. La primera es que el modelo debe entrenarse para varias señales diferentes siendo su principal ventaja poder resolver varios problemas. El segundo punto es que se vuelve un costo excesivo entrenar dicho modelo para solo hallar una ecuación y no tener un modelo que pueda encontrar la estructura de varias ecuaciones a la vez.

Como se mencionó anteriormente en la sección 2.2.4.2 las RNN presentan problemas de gradiente además de compresión de información en secuencias largas. Por esta razón en la literatura se ha pensado en utilizar modelos de tranformers para dar solución a este tipo de problemas, y además poder enfocar la atención en ciertos instantes de tiempo. El modelo [27] es un método de regresión simbólica que se enfoca en el uso de transformers para generar las estructuras de los modelos propuestos en vez de utilizar RNN. Cabe destacar que en [9] se hace uso de transformers para poder encontrar la solución específica buscada al reemplazar en la librería de símbolos, los símbolos de coeficientes por constantes numéricas. Por último, es importante destacar que estos métodos siempre usan un optimizador no lineal para poder

ajustar los parámetros de la función predictora que entrega dicho modelo.

2.2.5. Deep Symbolic Regression (DSR)

2.2.5.1. Aprendizaje Reforzado

El aprendizaje reforzado se basa en que un agente debe aprender las acciones ($\vec{a}$) óptimas que debe tomar para maximizar una recompensa en forma de una señal numérica. El agente busca obtener un objetivo a pesar de la incerteza que tiene sobre su ambiente, este elige acciones utilizando el estado de su entorno, el que incluso puede verse afectado por dichas acciones. El objetivo que define el éxito de la tarea en cuestión debe estar definido necesariamente por la función de recompensa. De este modo, a medida que el agente aprende existe una compensación entre la explotación de ciertas acciones ya aprendidas y la exploración de nuevas acciones a realizar.

En la literatura de aprendizaje reforzado además del agente y su entorno hay cuatro subelementos: una política, una señal de recompensa, una función de valor y opcionalmente un modelo del ambiente [28]. La política ($\pi(\vec{s})$) define la manera en que un agente se comporta en un momento determinado. Esto formalmente se define como un mapeo desde los estados del ambiente a probabilidades de las posibles acciones que puede realizar un agente. Si el agente sigue una política π en un tiempo t, entonces $\pi(a|s)$ es la probabilidad de que el agente incurra en la acción a si el estado es s. En algunos casos la política puede ser de manera tabular, una función o puede involucrar procesos estocásticos para definir las acciones a tomar. En los agentes de aprendizaje, la política es suficiente para definir las acciones que estos tomarán [28].

La señal de recompensa ($r(\vec{a})$) define un objetivo de un problema de aprendizaje reforzado, en cada instante de tiempo el ambiente envía una señal de recompensa al agente. El agente por su parte busca maximizar la recompensa recibida en el largo plazo. De este modo, la señal de recompensa define cuales son malos o buenos estados para un agente en un momento determinado. Por otro lado, mientras que la función de recompensa le indica al agente lo mejor para un instante de tiempo, la función de valor ($V(\vec{s})$) señala que es bueno en el largo plazo. Esto se puede definir como la cantidad total de recompensa que puede obtener un agente acumulado en el tiempo, partiendo desde un estado en específico.

El último subelemento presente en los sistemas de aprendizaje reforzado es el modelo del ambiente (el cual es opcional). Dicho modelo representa el comportamiento del ambiente o de manera más general, permite hacer inferencias sobre cómo se comportará el ambiente ante ciertas acciones del agente. Se utiliza para hacer planeamiento, esto quiere decir que sirven para predecir el futuro ante distintas situaciones. Los métodos para resolver problemas de aprendizaje reforzado que utilizan este modelo para planear se denominan métodos "basados en modelos", mientras que los que no utilizan un modelo para planear se denominan modelos "libres".

2.2.5.2. Tipos de Episodios

Anteriormente indicamos la importancia de la función de valor para poder maximizar la recompensa futura. Para dar más claridad, se a definirán estos conceptos de manera formal. En la práctica cuando se habla de maximizar las recompensas futuras lo que realmente se busca es maximizar el valor esperado de los retornos futuros. El retorno lo definiremos como G_t siguiendo la nomenclatura expuesta en [28]. El caso más simple se encuentra cuando G_t

es la suma de las recompensas futuras:

$$G_t = \sum_{i=1}^{T} R_{t+i}. \tag{2.8}$$

Para este caso en específico, la suma se realiza hasta el tiempo final T. Este modelamiento se basa en casos donde existe una noción de término de la tarea que se busca realizar. Esto quiere decir que las interacciones del agente con el ambiente se dividen naturalmente en subsecuencias que llamaremos "episodios". Un ejemplo de esto son varios juegos de ajedrez que se ven separados cada uno por el término de cada juego, siguiendo este ejemplo, cada juego termina en un estado s_T denominado estado terminal (jaque mate), que lleva a un reinicio del ambiente a un estado inicial (parten jugando las piezas blancas). A este tipo de tareas se les denomina "tareas episodicas". También existe otro tipo de interacciones que no logran ser separadas de manera natural en distintos episodios, sino que la tarea continua sin un límite claro. Un ejemplo de esto podría ser el control de un péndulo invertido, que busca que el péndulo nunca se caiga y por ende no conlleva un instante de término. A este tipo de tareas se les denomina tareas continuas. Notemos que cambiando en la expresión (2.8) T por ∞ la serie no converge haciendo que el óptimo solo exista en infinito. Debido a esto es necesario definir el factor de descuento.

El factor de descuento, hace referencia a la ponderación que se le entrega a las recompensas futuras sobre las nuevas. Este se define por γ y es un número entre $0 < \gamma < 1$. La ecuación (2.8) modificada para tareas continuas se define en la siguiente expresión:

$$G_t = \sum_{i=0}^{\infty} \gamma^i R_{t+i+1}. \tag{2.9}$$

Esto representa la noción de que una recompensa que se obtiene en un tiempo k con $k > t$ tiene un valor γ^{k-1} veces más pequeño que la misma recompensa obtenida en el tiempo t, además de ser una serie con un valor definido. Es fácil ver que si $\gamma = 0$ los agentes serán miopes y por ende buscarán maximizar la recompensa inmediata, mientras que cualquier otro valor de γ menor a 1 generará que el agente pondere más allá del siguiente instante de tiempo.

2.2.5.3. Episodios Periódicos y Ecuaciones de Bellman

En secciones anteriores se definió que $\vec{V}(s_t)$ permite determinar la maximización de retornos que puede obtener una política. En la práctica es interesante no solo preguntarse el máximo retorno en caso de estar en un estado s_t, sino que además como varían los retornos dependiendo de las acciones que toma el agente. Debido a esto se define la función acción-valor para una política π como $q_\pi(s, a)$, que se ve definida en la siguiente expresión:

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a] = \mathbb{E}_\pi \left[\sum_{i=0}^{T} \gamma^i R_{t+i+1} | S_t = s, A_t = a \right]. \tag{2.10}$$

En la literatura, existen métodos para encontrar tanto la función v_π como a_π mediante aproximaciones basadas en métodos de Monte Carlo [28]. Esto sumada a la relación recursiva que presenta la función de valor, nos permite estimar $v_\pi(s)$ mediante la función:

$$v_\pi = \mathbb{E}_\pi[G_t|S_t = s] = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \left[r + \gamma v_\pi(s')\right], \tag{2.11}$$

donde la acción a representa las posibles acciones que puede tomar un agente en un estado, s' representa el estado siguiente del conjunto de los posibles estados siguientes y r toma el valor de la recompensa para dicha situación. la ec. (2.11) se denomina Ecuación de Bellman para v_π y expresa la relación entre el valor de un estado y el valor de los estados sucesores. Esta ecuación hace referencia al hecho de que desde un estado s, el agente podría tomar cualquier posible acción a que permita su política y por ende mediante esta elección es que se define los posibles estados futuros cuya función de valor justamente corresponde a $v_\pi(s')$ (ya que el agente sigue una política definida con anterioridad). La figura 2.10 muestra gráficamente este proceso.

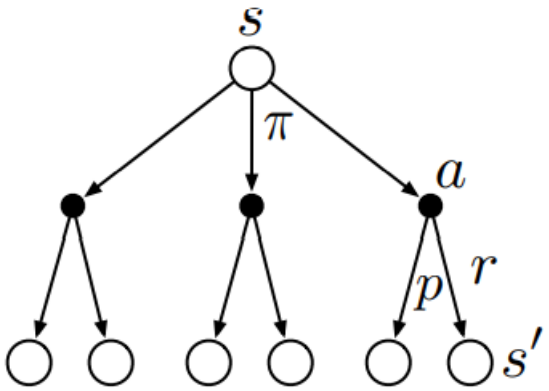


Figura 2.10: Diagrama sobre las posibles acciones que puede realizar un agente y sus posibles efectos en los estados posteriores a los que llegará. Tomada de [28].

De la expresión (2.11) se desprende que v_π es la única solución a dicha ecuación. Esta reutilización de la función v_π conlleva grandes consecuencias, ya que dicha función es fija, es decir no depende del tiempo, por lo que si se conociera la tarea a resolver estaría resuelta. Debido a esto, el objetivo que se busca resolver en este tipo de problemas es encontrar la política óptima que entregue el mayor valor posible o mediante las ecuaciones de valor o acción-valor guiar al agente en cada paso de un episodio. De este modo, las funciones q_* y v_* son las funciones óptimas para un problema en específico y se definen como sigue:

$$v_*(s) = max_\pi v_\pi(s) \tag{2.12}$$

$$q_*(s, a) = max_\pi q_\pi(s, a) \tag{2.13}$$

Reescribiendo la ec. (2.13) en función de v_* utilizando la ec. (2.10) se obtiene la ecuación (2.14) que relaciona la función valor óptima con la función acción-valor óptimas, permitiendo de este modo reescribir la solución solamente en función de la función acción-valor si se toma la acción futura a' que maximice la función de acción-valor óptima según (2.15), resultando

en la ecuación (2.16):

$$q_*(s,a) = \mathbb{E}_\pi[R_{t+1} + \gamma v_*(S_{t+1})|S_t = s, A_t = a] \tag{2.14}$$

$$v_*(s) = max_a q_*(s,a) \tag{2.15}$$

$$q_*(s,a) = \sum_{s',r} p(s',r|s,a)\left[r + \gamma * max_{a'} q_*(s',a')\right]. \tag{2.16}$$

2.2.5.4. Gradiente de la Política

Existen métodos que en vez de utilizar las funciones de valor o de acción-valor intentan aprender la política óptima del problema. Esto de logra haciendo un modelo de la política que sea parametrizado que pueda seleccionar acciones sin necesidad de consultar una función de valor. Dicho valor puede ser efectivamente utilizado para ayudar en el aprendizaje de los parámetros de la política, pero no es requerido para la selección de acciones. Dicho modelo se define de la siguiente manera:

Sea $\theta \in \mathbb{R}^{d'}$ el vector de parámetros de la política. Se define $\pi(a|s,\theta) = P(A_t = a|S_t = s, \theta_t = \theta)$ como la probabilidad que la acción a se tome en un tiempo t, dado que el ambiente se encuentra en un estado s en dicho instante y la política tiene parámetros θ. Si el método utiliza un vector de pesos $w \in \mathbb{R}^d$ entonces el estimado de la función valor se define como $v(s, \mathbf{w})$.

Este método permite aprender la política que maximice los retornos utilizando el gradiente de una función escalar $J(\theta)$ que depende de los parámetros del modelo de la política. Como el objetivo de este algoritmo es maximizar la función J, la actualización de los pesos se hace utilizando la aproximación del gradiente mediante gradiente ascendente,

$$\theta_{t+1} = \theta_t + \alpha \nabla \hat{J}(\theta_t). \tag{2.17}$$

Todos los métodos que busquen aprender la política del agente utilizando este esquema general (utilicen o no la función valor) se denominan “métodos de gradiente de la política".

La política puede ser parametrizada de una manera $\pi(a|s,\theta)$ siempre y cuando este sea diferenciable con respecto a sus parámetros. Para poder utilizar este tipo de métodos dicho gradiente debe existir para todo el conjunto posible de estados, acciones y valores de los parámetros θ, además, para asegurar la exploración generalmente se requiere que dichas políticas nunca se conviertan en políticas deterministas [28].

2.2.6. Teorema del Gradiente de la Política

Teorema del gradiente de la política provee una expresión analítica del gradiente del rendimiento con respecto a los parámetros de la política. Este teorema entrega una aproximación al gradiente de la política de una función de rendimiento J respecto a los parámetros θ, que es proporcional al gradiente del modelo paramétrico de la política respecto a sus parámetros, tal como se expone en (2.18). En esta expresión π es el modelo paramétrico de la política, μ representa la distribución de los estados siguiendo la política π y q_π es la función acción-valor definida por la política π.

Dado que como el gradiente es proporcional al gradiente real de la función J, se puede realizar el aprendizaje de la política por gradiente ascendente, ya que la proporcionalidad es absorbida por la tasa de aprendizaje del entrenamiento expuesta en (2.17). De este modo lo que se requiere es un método de muestreo el cual permita aproximar esta expresión. Nótese,

que en la parte diestra de la expresión se realiza una suma sobre los estados ponderado por la esperanza de ocurrencia de dicho estado en la política π. Esto hace que la expresión pueda reescribirse de la forma (2.19). Por último, dicha expresión puede ser reemplazada mediante desarrollo algebraico [28], [7] por la expresión (2.20), como sigue:

$$\nabla J(\theta) \ \alpha \ \sum_{s} \mu(s) \sum_{a} q_{\pi}(s,a) \nabla \pi(a|s,\theta) \tag{2.18}$$

$$\nabla J(\theta) \ = \ \mathbb{E}_{\pi} \left[\sum_{a} q_{\pi}(s,a) \nabla \pi(a|s,\theta) \right] \tag{2.19}$$

$$\nabla J(\theta) \ \approx \ \frac{1}{N} \sum_{i=1}^{N} R(s^i) \nabla_{\theta} ln(\pi(s(i)|\theta)). \tag{2.20}$$

Uno de los problemas del algoritmo REINFORCE es que el gradiente encontrado en (2.20) tiene una alta varianza [28], [7], por lo que el uso de un *baseline* es necesario. El término *baseline* hace referencia a una función que se le resta a los retornos, la cual debe cumplir la particularidad de no depender de los parámetros de la política y además no variar con las acciones, el uso de este en general no afecta la esperanza del gradiente pero si permite reducir la varianza.

2.2.6.1. Aprendizaje Reforzado en Regresión simbólica

Como se mencionó anteriormente los problemas de SR se dividen en dos subproblemas, el primero hace referencia a la búsqueda estructural de la solución y el segundo se refiere al ajuste de coeficientes de la solución encontrada. En el caso específico visto anteriormente los AG intentan resolver el primer subproblema haciendo cambios en la forma de selección y jerarquización, pero la generación de estructuras correctas sigue dependiendo fuertemente de procesos aleatorios los cuales no necesariamente generarán mejores soluciones. El método de Deep Symbolic Regression (DSR) lleva su nombre debido al uso de redes neuronales para solucionar el problema estructural de la regresión simbólica. Este algoritmo hace uso (al igual que los AG) de estructuras tipo árbol y se basa igualmente en el uso de una población. Una de las principales diferencias con los métodos de AG se encuentra en que no existen operaciones de recombinación o mutación que se realicen entre individuos en este algoritmo, ya que la manera en que se generan nuevas estructuras es mediante el uso de aprendizaje reforzado.

Tal como se dedujo en la sección 2.2.5.4, según el algoritmo Reinforce el rendimiento que se mide con respecto a una función $J(\theta)$ puede ser aproximado por la esperanza del producto entre el retorno obtenido y el gradiente del logaritmo de la salida del modelo de la política. En específico, si se utiliza una red neuronal para aproximar el modelo de política, el gradiente de dicha red con respecto a sus parámetros existe y es finito. De este modo, es posible entrenar una red neuronal utilizando el gradiente de la política de gradiente para que resuelva una tarea que se vea representada por la función $J(\theta)$. En el caso específico de DSR, la finalidad de dicho algoritmo es en utilizar una red neuronal recurrente como modelamiento de la política del agente. En dicho caso el agente tiene la tarea de generar estructuras correctas para así solucionar la búsqueda estructural que se resuelve en AG mediante las operaciones de recombinación y mutación. De este modo, la decodificación de árboles se hace mediante una lista de posibles valores ($\mathcal{L}$) para los símbolos del árbol,tales como: operadores (seno, coseno, logaritmo), variables de entrada (x, y) o constantes. Se define un vector (τ) de tamaño $|\tau| = T$ que representa el genotipo del programa computacional, donde cada entrada de este vector hace referencia a un valor en $\mathcal{L}$.

La red recurrente es un modelo secuencia a secuencia, como los vistos en 2.2.4.1, las expresiones de salida de dicho modelo son generadas una a una desde τ_1 a τ_T. Una distribución categórica con parámetros ϕ define la probabilidad de seleccionar cada nodo desde $\mathcal{L}$ [7]. Para poder capturar el contexto a medida que se van generando las salidas τ_i se condiciona la probabilidad sobre todas las salidas anteriores que lleva la secuencia. Por esta razón se utiliza una red recurrente, ya que dicha red permite mediante sus parámetros θ condicionar las salidas futuras de manera autoregresiva. La probabilidad del vector $\phi^{(i)}$ se define como:

$$p(\tau_i|\tau_{1:(i-1)};\theta) = \phi^{(i)}_{\mathcal{L}(\tau_i)} \tag{2.21}$$

Las entradas de la RNN al momento de hacer el muestreo de τ_i son una representación (por ejemplo, un vector con one-hot encoding) del nodo previamente muestreado (τ_{i-1}). El espacio de búsqueda de la regresión simbólica es jerárquico, pero el nodo anterior puede no tener relación alguna al nodo siguiente a ser muestreado por lo que en DSR propone entregar tanto el padre del nodo que se va a muestrear junto con su nodo hermano. Esto se ilustra en la figura 2.11 donde si se sigue la secuencia y se compara con el árbol de la parte C de la figura, se observa que el espacio del hermano se encuentra completo una vez que ya se ha recorrido hasta las hojas la rama hermana del árbol. Cuando el nodo del árbol no tiene un padre o un nodo hermano el espacio de este se reemplaza por un símbolo vacío. Se hace necesario además, restringir el espacio de búsqueda del algoritmo utilizando las siguientes restricciones: 1. Las expresiones se limitan por un largo mínimo de 2 y máximo de 30. Esta restricción permite eliminar expresiones triviales y otras que sean muy complejas de entender. 2. Los nodos hijos de un operador no pueden ser todos constantes, esto ya que se podría reemplazar por otra constante. (3) El nodo padre de un operador unitario no puede ser su inversa. (4) Los hijos directos de una función trigonométrica no pueden ser funciones trigonométricas, esto se debe a que no existen expresiones científicas que tengan este tipo de estructuras. Estas restricciones se realizan a medida que corre el algoritmo simplemente haciendo que las probabilidades de dichos símbolos sean cero.

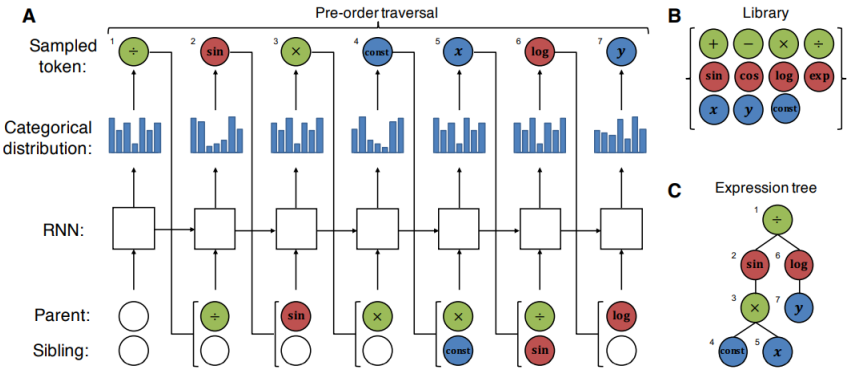


Figura 2.11: Diagrama explicativo del funcionamiento del modelo de DSR , en este los símbolos se seleccionan uno por vez de manera auto regresiva completando el genotipo del programa computacional. Para cada nodo, la RNN genera una distribución categórica del largo de $\mathcal{L}$, de manera de seleccionar uno de estos modelos. Extraído de [7]

Una vez muestreados los símbolos, se procede a generar dicho árbol, las constantes de dicha

expresión se dejan como parámetros del problema y se entrenan dichos parámetros utilizando el error cuadrático medio con respecto a un conjunto de datos de entrada. Esto se hace utilizando un algoritmo de optimización no lineal (ej. BFGS [29], [30]) y se realiza antes de evaluar las expresiones para actualizar la política. Para entrenar la RNN se utiliza (2.20) con *baseline*, siendo este último la media móvil ponderada exponencialmente. Este gradiente se utiliza para maximizar $J(\theta)$ el cual se define como la esperanza de las recompensas obtenidas por las estructuras muestreadas de la política. Como función de recompensa se utiliza la expresión:

$$R(\tau) = 1/(1 + NRMSE). \tag{2.22}$$

Dicha expresión se basa en los AG que utilizan la función de ajuste NRMSE expuesta en (2.26). Además, se definen penalizaciones por complejidad, esto quiere decir que se considera el número de nodos que contiene τ para penalizar su recompensa y además se entrega un valor extra proporcional a la entropía de la expresión muestreada. Esto permite que la RNN explore más expresiones, previniendo así la convergencia prematura a mínimos locales y además que expresiones diferentes pero con mismo *fitness* tengan igual verosimilitud.

Por último, como el valor del gradiente de $J(\theta)$ es una esperanza esta se debe hacer sobre una cantidad definida de muestras que en la práctica no necesariamente significa una gran cantidad de individuos. Para que el modelo busque otras posibles expresiones y no sobreexplote las que ya maneja, se considera una técnica de selección de las expresiones generadas, donde solamente el percentil ϵ de los mejores individuos es considerado para el cómputo del gradiente. Esto según [7] genera mejoras en los resultados ya que considera los mejores individuos y así además sube el promedio del baseline haciendo que sea más difícil que la red tienda a considerar malas expresiones.

2.2.7. AI Feynmann

AI Feynman es una aproximación al problema de la regresión simbólica basado en dividir para conquistar, este método utiliza modelos básicos de regresión simbólica y heurísticas de descomposición para crear un modelo simbólico [12]. El algoritmo va intentando distintas técnicas conocidas que simplifiquen el problema de manera tal de poder utilizar estos modelos básicos de regresión simbólica para encontrar la solución.

En [31] se indica que en general existen funciones (f) genéricas que tienen una complicidad extrema y por ende no pueden ser encontradas mediante regresión simbólica, pero que las funciones que aparecen en la física y otras aplicaciones científicas tienen algunas de las siguientes propiedades simplificatorias:

1. **Unidades:** f está construida de variables que tienen unidades físicas.
2. **Polinomios de bajo orden:** f (o parte de f) tiene polinomios de bajo orden.
3. **Composición:** f está compuesto de un pequeño grupo de funciones elementales, donde cada una no toma más de dos variables.
4. **Suavidad:** f es continua e incluso analítica en su dominio [4].

[4] Las funciones analíticas son dichas funciones que localmente están definidas por una serie de potencias.

5. **Simétrica:** f tiene simetría traslacional, rotacional o de escalabilidad con respecto a una de sus variables.

6. **Separabilidad:** f se puede escribir como una suma o producto de dos partes que no comparten variables en común.

Las propiedades arriba mencionadas permiten realizar diversos análisis que simplifican de cierta manera el problema de regresión simbólica. La propiedad 1 permite el análisis dimensional que generalmente simplifica el problema a uno con menos variables. La propiedad 2 permite un ajuste polinomial de los datos que al resolver un sistema de ecuaciones lineales encuentra los coeficientes del polinomio. La propiedad 3. hace que la función f se pueda expresar como un programa computacional tipo árbol con una baja cantidad de nodos, cuyos sub-expresiones pueden ser encontrados incluso bajo fuerza bruta según [31]. La propiedad 4, permite aproximar f mediante una red neuronal con una función de activación suave. La propiedad 5, puede ser confirmada utilizando dicha red neuronal y así permite reducir el problema a uno con una variable independiente menos (o menos si se encuentra más de una simetría). La propiedad 6 puede ser confirmada usando una red neuronal, permitiendo de este modo que las variables independientes sean particionadas en subgrupos disjuntos y así dividir el problema en dos más simples.

2.2.7.1. Análisis Dimensional

El módulo de análisis dimensional explota el hecho de que muchos problemas en física pueden ser simplificados buscando que los dos lados de las ecuaciones calcen. Esto en general transforma el problema en uno más simple, que tiene un menor número de variables y son adimensionales. En este caso se redefine el espacio por otro conjunto de vectores base, permitiendo de este modo que la expresión quede adimensional y además que se reduzca la cantidad de variables que tiene el problema.

2.2.7.2. Ajuste Polinomial

En se [31] utiliza el hecho de que muchas de las funciones en física utilizan polinomios de bajo orden o noy por ende incluyen un módulo que prueba si un problema puede ser resuelto con un polinomio de bajo orden. Esto se logra mediante un método estándar de resolución de un sistema de ecuaciones lineales que permita encontrar el mejor ajuste a los coeficientes polinomiales. Este intenta ajustarse a los datos del problema mediante un polinomio que puede ser de grado 0 hasta $d_{max} = 4$ y declara exitoso el proceso si se logra un error de RMSE menor a un cierto umbral que se define como hiperparámetro del problema.

2.2.7.3. Fuerza Bruta

El método de fuerza bruta busca generar todas las posibles expresiones simbólicas en un orden creciente de complejidad intentando esto hasta que los resultados obtengan un RMSE menor a ϵ_p o pase un tiempo t_{max} en que ninguna de las expresiones generadas logra dichos valores. En [31] se indica que este método tiene como idea basal ser usado en problemas que ya hayan sido simplificados mediante los otros módulos. El algoritmo también considera el uso de constantes como (π) para evitar expresiones redundantes o operaciones complejas con constantes. Por último, es importante indicar que dentro de los posibles modelos que tengan un error menor a ϵ_b (que es 10 veces el error ϵ_p) se escoge el más corto entre las opciones

ponderando el error y el largo por la siguiente fórmula:

$$DL \equiv log_2 N + \lambda log_2 \left[max(1, \frac{\epsilon}{\epsilon_d}) \right]. \tag{2.23}$$

En esta expresión N es el largo de la cadena de caracteres, λ es un hiperparámetro, ϵ es el error que logra la expresión y ϵ_d hace referencia a un error de referencia de 10^{-5}. De este modo, el algoritmo sopesa tanto el largo de las expresiones como también el ajuste que tienen estas mismas a la curva.

2.2.7.4. Uso de Redes Neuronales

A pesar de utilizar el análisis dimensional, el ajuste polinomial o la fuerza bruta para resolver el problema, es probable que todavía la expresión sea muy compleja. Debido a esto, se utilizan redes neuronales para ver si los datos tienen propiedades simplificatorias que permitan transformarlo en un problema más simple de resolver.En [31] se indica que para poder probar dichas propiedades de simetría, en muchos casos los datos que se tienen de la señal no son suficientes como para poder aseverar que dicha propiedad exista, es por esto que se requiere el uso de redes neuronales, para así hacer un análisis en un espacio de mayor dimensión.

Se entrena una red neuronal para que prediga la salida de la señal que se desea aprender entregándole la entrada que le corresponde, usando la función de error RMSE. Para analizar la traslación lo que se hace es utilizar la red neuronal como modelo tipo caja negra, de este modo lo que se hace es hacer traslaciones de las entradas sumándoles una constante a para luego ver si la diferencia entre las salidas es menor a un umbral ϵ_{sym}, en cuyo caso las entradas a las que se les sumó dicha constante son reemplazadas por una nueva entrada $x_i^{'} = x_i - x_j$. Esta prueba se realiza para todos los pares posibles de entradas, también se hacen pruebas sobre todos los pares de entradas si estas se pueden reemplazar por la suma, producto o razón. Si alguno de estos pares logra cumplir la condición, el algoritmo parte nuevamente pero con una variable menos. La separabilidad se mide seleccionando dos conjuntos disjuntos de variables y se calcula el promedio de dichos conjuntos para utilizar como reemplazo para las componentes los atributos que no fueran seleccionados. Se define que una función es separable por producto o suma cuando $\Lambda_{sep}(x_1, x_2, x_3, ...)$ definida por la siguiente expresión:

$$\Lambda_{sep}(x_1, x_2) \equiv RMSE(f(x_1, x_2) - (f(x1, c1) + f(x2, c2) - f(c1, c2)), \tag{2.24}$$

es menor a un umbral de separabilidad. Si se cumple esta condición se define que la función es separable y por ende las variables se cambian por $x_i^{'} = f(x1, c1)$ y $f(x2, c2) - f(c1, c2)$ y el modelo de AIFeynmann se vuelve a correr desde el inicio.

2.2.7.5. Transformaciones

Una vez terminado el proceso de detección de simetrías y separabilidad, de no encontrarse éstas se pasa a aplicarle transformadas a las variables dependientes e independientes. Para esto se le aplican funciones conocidas tales como seno, coseno, raíz, inversa, entre otras. De este modo, una vez que se transforman los datos se vuelve a correr el algoritmo desde cero pero con las variables transformadas. Una imagen del algoritmo completo se muestra en la sección A.2 en los anexos.

2.3. Métricas

El entrenamiento de los modelos de regresión simbólica se realiza de manera parecida a los modelos de regresión logística. En estos un modelo genera predicciones las cuales son posteriormente comparadas con las respuestas correctas de las observaciones, dicha comparación en los modelos que utilizan *back-propation* generará un gradiente que permita que el modelo mejore y en los casos de algoritmos que no utilizan gradiente entregarán un valor que representará que tan ajustado es el modelo generado a los datos entregados. Al igual que en regresión logística los modelos de regresión simbólica se entrenan y se evalúan con algunas de las siguientes métricas:

1. Error Cuadrático Médio (MSE): El error cuadrático medio es la métrica más utilizada tanto para entrenar modelos como para evaluarlos, dicha métrica representa que tan cerca se encuentran en el plano cartesiano los puntos predichos con los esperados. La fórmula de dicha métrica se representa en (2.25) y como se puede observar tiene la propiedad que todos los errores menores a uno se volverán cada vez más pequeños, de este modo como el cálculo de dicha expresión utiliza el promedio nos entrega una perspectiva general de que tan certera es la aproximación. La fórumla matemática de dicha métrica es la siguiente:

$$MSE(\hat{y}, y) = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2. \tag{2.25}$$

2. Raíz del Error Cuadrático Medio Normalizado (NRMSE): Este error representa la raíz cuadrada del MSE, dicho valor por ende no genera disminuciones de los valores de error cuando estos sean menores a uno. Además, el error se normaliza dividiéndolo por la desviación estándar de los valores esperados. Este tipo de métricas se han utilizado en los últimos trabajos de *benchmarking* sobre regresión simbólica [12]. Dicha métrica se considera estándar en muchos casos [7] y permite, debido a su normalización, ser usada en varios problemas diferentes sin importar mayormente el rango de los datos, ya que dicha diferencia en la distribución de la salida se verá reflejada en la desviación estándar de los datos. La fórmula de dicha métrica es la siguiente:

$$NRMSE = \frac{1}{\sigma_y}\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}. \tag{2.26}$$

3. Coeficiente de Determinación (R^2): Esta métrica es una medida de la calidad global de la regresión. Específicamente es el porcentaje de variación que logra predecir el modelo ($\hat{y}$)en comparación a los datos esperados (y). Esta métrica se considera dentro del trabajo de tesis ya que es una de las métricas utilizadas en [12] al momento de comparar resultados generados por los diversos métodos que fueron revisados. La fórmula matemática de dicha métrica es la siguiente:

$$R^2 = 1 - \frac{\sum_{i=1}^{N}(\hat{y}_i - y_i)^2}{\sum_{i=1}^{N}(\bar{y}_i - y_i)^2}. \tag{2.27}$$

2.4. Regularización

La regularización es una de las formas en que los modelos de *Deep Learning* y *Machine Learning* manejan el sobre ajuste a los datos. Estos métodos aplican una penalización sobre los parámetros (θ) del modelo, esto se hace calculando la norma L_p de θ definida en la ec. (2.28). En este caso para que L_p esté bien definida $p \geq 1$ y $x \in \mathbb{R}^n$ para el caso específico de la norma $p = \infty$ el valor es el máximo componente de θ.

$$L_p(x) = \sqrt[p]{\sum_{i=1}^{N} |x_i|^p} \tag{2.28}$$

Este tipo de modelos se entrena en general utilizando regularización L_1 o L_2, penalizando de este modo la existencia de pesos grandes de distinta manera. Se podría decir que el óptimo es la regularización L_0, pero dicho modelo de regularización carece de las propiedades necesarias para ser entrenado por gradiente descendente, debido a su naturaleza binaria discreta. Algunos trabajos expuestos en la literatura intentan acercarse a representar dicha regularización utilizando técnicas de reparametrización [32], aproximación de gradiente mediante el algoritmo REINFORCE [33] que se explica en la sección 2.2.6 o un estimador que reemplaza el gradiente de la parte muestreada por la función identidad [34]. En la figura 2.12 se muestra un diagrama de las distintas regularizaciones p y como varía el valor dependiendo del valor del peso que se busca regularizar.

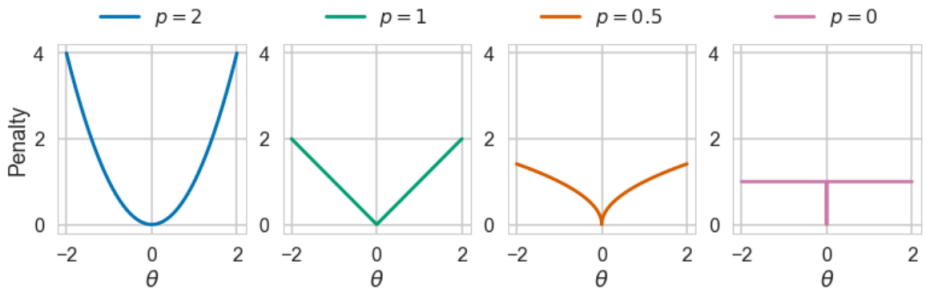


Figura 2.12: Penalización que realizan las diversas normas p en función de los parámetros θ del modelo. Se puede ver como la regularización L_2 aumenta de manera cuadrática la penalización en función del valor θ mientras que las otras regularizaciones aumentan su penalización de acuerdo al peso de una manera menor. Figura extraída de [32].

2.5. Resumen

A través de este capítulo se explicaron los distintos métodos existentes en el estado del arte de la regresión simbólica. En dicho capítulo se explicaron las principales limitantes de los modelos centrados en solucionar el subproblema de la búsqueda estructural dejando en un segundo plano la optimización de coeficientes. También se explicaron conceptos como las distintas métricas que generalmente se utilizan en este tipo de problemas y sus ventajas al momento de evaluar el desempeño del modelo. Por otro lado, se explicó la importancia de la regularización en modelos que hagan uso de "Deep Learning" para evitar el sobre ajuste de estos modelos.

En el siguiente capítulo explicaremos el modelo propuesto en este trabajo de tesis. De este capítulo conceptos como la regularización, las métricas, las redes EQL y el algoritmo "REINFORCE" serán de particular importancia al momento de explicar el modelo propuesto. También Modelos como los "AG" serán particularmente importantes al momento de definir los modelos de comparación.

3. Metodología Modelo SRNet

3.1. Uso de Redes Neuronales

Las redes neuronales utilizan matrices de pesos a las que posteriormente se les aplica una función no lineal como función de activación. Desde una perspectiva estructural, la mayoría de las ecuaciones que se encuentran en la literatura, como es el conjunto de datos AI-Feynman [31], presentan el uso de combinación lineal de pesos y funciones de activación no lineales. Al utilizar funciones de activación comunes en ecuaciones matemáticas y científicas se debiese poder encontrar dichas ecuaciones mediante retro propagación de gradiente. Dicho proceso presenta desafíos que la literatura actual de las redes EQL (sección 2.2.3.2) no ha podido resolver a cabalidad, en específico debido a la dependencia que existe entre la máxima potencia alcanzable y la cantidad de capas que presenta la red. Dichos desafíos son los siguientes:

1. Entrenar redes neuronales mediante funciones no lineales no convencionales no asegura una curva de aprendizaje suave.

2. No es posible entrenar una red con todos los tipos de funciones de activación ya que al momento de retro propagar el gradiente pasará por funciones que estructuralmente no representan la solución buscada, generando ruido en la solución.

3. Se requiere aproximar la función de potencia con un método que no tenga una dependencia en el número de capas para la generación de potencia.

En esta tesis se separan estos problemas en distintas secciones, los problemas de suavidad en las curvas de aprendizaje, al igual que una función de potencia continua, se entablan dentro de los *problemas de ajuste de parámetros* que se presentan con la estructura de red ya definida. Por otro lado los problemas de selección de funciones candidatas se abordan de una manera independiente a los problemas anteriormente descritos, a este tipo de problema se le denomina *problema de búsqueda estructural*.

3.2. Ideas Basales

Como se indicó anteriormente, es necesario generar una red que pueda separar su estructura del ajuste de parámetros y realizar esto último de una manera tal que dicho efecto esté aislado a la estructura de red generada. Para cumplir con este fin, las redes son construidas mediante una estructura previamente definida, que se representa por una máscara binaria.
El uso de máscaras binarias permite separar en bloques los tipos de funciones existentes en cada capa. Es decir, mediante el uso de máscaras, se podrá elegir cada bloque consistente en una función no lineal que no se repite dentro de la misma capa, Cada bloque solamente

utiliza la función de activación que le corresponde a su salida. Un ejemplo de esto se muestra en la figura 3.1 donde cada bloque representa combinaciones lineales de las funciones que indica el nombre de dicho bloque. Luego dichas salidas se concatenan para así aplicar una última combinación lineal que entregue la salida de la capa. Este ejemplo, a pesar de no ser la forma real en que el modelo realiza dicha aproximación, es una buena analogía del proceso interno que ocurre dentro del modelo.

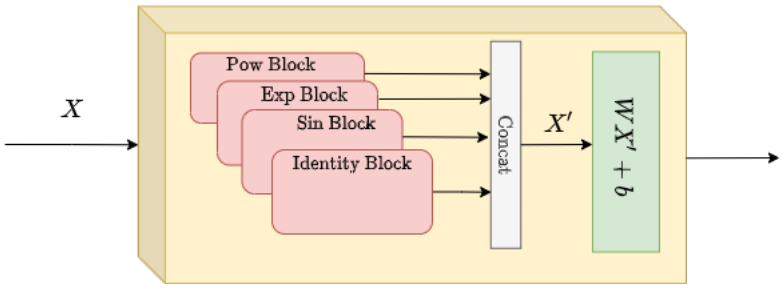


Figura 3.1: Idea basal de una capa de la red propuesta donde cada bloque en rojo representa una salida de un tipo de función, en este caso potencia, exponencial, sinusoidal e identidad. Las salidas de estos bloques se concatenan y se combinan linealmente.

Como se puede observar en la figura 3.1, cada bloque genera salidas con funciones diferentes que son independiente en su cálculo de los otros bloques de la misma capa por lo que es posible extraer o agregar funciones a la red de manera simple, sin afectar mayormente la estructura que presenta. Además mediante un hiperparámetro es posible aumentar la cantidad de salidas de los bloques, tal como se explica en la sección 3.4.9

3.3. Estructura de la Red Propuesta

Una vez explicada la idea basal del modelo es posible introducir la red propuesta denominada SRNetwork (figura 3.2). Esta red como se mencionó anteriormente sigue los lineamientos expresados en la sección 3.2 pero presenta diferencias estructurales.

El modelo de red está compuesto de tres tipos de bloques diferentes, los bloques multivariable $f(\vec{x}, \theta)$, funciones punto a punto $\vec{f_p}(\vec{x}, \theta)$ y funciones mono variables $f(x)$. Además, cada red requiere dos hiperparámetros, el primero es h_f que representa la cantidad de salidas para los bloques multivariables. El segundo es N_{layers} que representa el número de capas que tiene la red. Por último, cada capa tiene la propiedad de observar el input original que se le entrega a la red. Dicho input puede ser usado de diferentes maneras y por esto es que se idearon los bloques de selección que representan la forma en que estos valores son traspasados a capas posteriores.

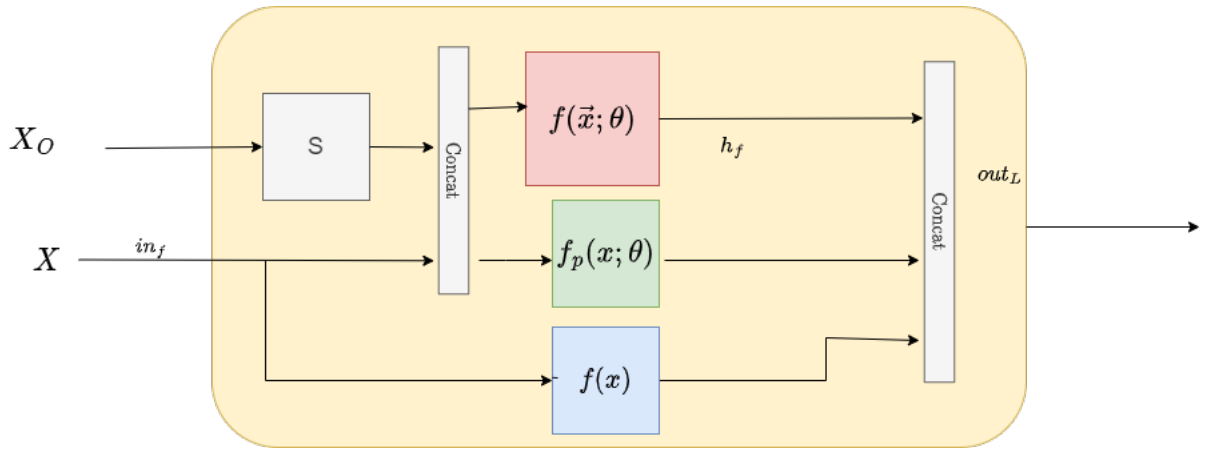


Figura 3.2: Estructura propuesta para una capa del modelo SRNetwork. El bloque rojo representa las funciones multivariable, el bloque verde las funciones punto a punto y el bloque celeste las funciones mono variables. Además, el bloque gris que contiene una S es el bloque de selección.

3.4. Bloques de Funciones

Los bloques de funciones se utilizan para encapsular la lógica y cálculo de un tipo de función en específico. Esto permite que la adición y extracción de los bloques no afecte el diseño del modelo. Para lograr esto la figura 3.3 muestra como se estructuran los diversos tipos de bloques mencionados anteriormente desde una estructura basal común.

Los bloques de funciones univariables, son los más simples, estos aplican la función que los define a la entrada del bloque. Debido a esto, las dimensiones de la entrada y la salida son las mismas, ya que solamente se realiza una transformación mediante una función no lineal.

Los bloques de funciones multivariables, difieren de los univariables en que este para generar una salida utiliza todas las variables de entrada y entrega h_f salidas. Dichos bloques tienen la posibilidad de utilizar una transformación inicial a las entradas mediante la función $\vec{g}_f(\vec{x})$, esta función se le denomina *preprocesamiento*. El caso más común de función de preprocesamiento es el bloque de potencia (ver Fig 3.3), donde se debe aplicar la función $ln(|x|)$ a los datos y combinarlos linealmente para aproximar la función potencia. En otros casos como $exp(\sum_i x_i)$ no es necesaria una función de *preprocesamiento* por lo que la función $\vec{g}_f(\vec{x})$ es reemplazada con la identidad [5]. Una vez aplicada la función de preprocesamiento, se debe realizar una multiplicación matricial entre la salida de dicha función por los parámetros del bloque. Debido a esto es necesario definir con anterioridad el hiperparámetro h_f, ya que este permite definir las dimensiones de la matriz de pesos que tendrán los bloques. Una vez hecha esta multiplicación se aplica la función principal del bloque, para obtener la salida.

Las funciones punto a punto son otra versión de funciones multivariable, que también utilizan la función de *preprocesamiento*. La principal diferencia radica en la naturaleza de la multiplicación que se utiliza, mientras que las funciones multivariables utilizan multiplicación matricial, las funciones punto a punto hacen uso de multiplicaciones punto a punto, debido a esto la cantidad de salidas de este bloque es igual a la cantidad de entradas. En la Fig. 3.3 *Pwise* representa la decisión de si utilizar producto punto a punto o el matricial.

[5] la función identidad en este contexto se define como la función $f(x) = x$

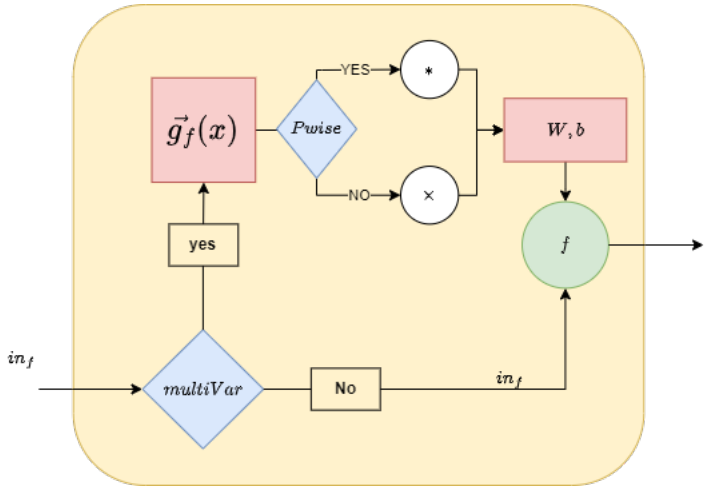


Figura 3.3: Estructura del bloque de funciones propuesto para el modelo SRNetwork. El diagrama engloba los tres tipos de funciones. Los rombos de decisión permiten discernir entre los distintos tipos de funciones, *multiVar* hace referencia a si la función es o no multi variable, *Pwise* hace referencia a si el tipo de operación a realizar es punto a punto o matricial. A modo de ejemplo, en el caso de una función multivariable el valor de *multiVar* sería verdadero y el valor de *Pwise* sería Falso.

3.4.1. Bloques de Selección

A medida que se crean redes con más de una capa, la entrada original de la red se puede perder. Esto hace que expresiones simples que utilicen en su última capa una variable de entrada original deban ser escritas con estructuras más complejas, de manera de poder traspasar la entrada original capa por capa. Dicho proceso genera complicaciones en la generación de expresiones en la red, ya que traspasar el input original capa a capa hace que este sea un input válido para cada capa y por ende al modelo se le agregan parámetros innecesarios.

Por esta razón se idearon los bloques de selección, ver figura 3.4, para que toda capa tenga en todo momento acceso a la entrada original del modelo y así simplificar la estructura de las redes generadas. Una vez seleccionadas, la entrada se traspasa como argumento pero únicamente a los bloques de funciones multivariable o funciones punto a punto, las funciones mono variables no hacen uso de dichas entradas. Como existen casos en que la fórmula solamente utiliza una de las entradas o algún tipo de combinación de estas se tienen tres tipos de bloques de selección.

El primer bloque es el *bloque de selección total*, que utiliza todas las entradas originales y las pasa como argumentos a la capa que tenga el bloque de selección, En este caso no se hace ninguna clase de selección sobre las entradas, sino que todas son pasadas a la capa intermedia que la requiera.

El segundo bloque es el *bloque de selección única*, este bloque selecciona una de las entradas originales al modelo y ésta se pasa a las capas posteriores. Para realizar esto el modelo hace uso de parámetros que representan cada una de las entradas. Al aplicar una función softmax sobre dichos pesos, se tiene la posibilidad de seleccionar cada una de las entradas del modelo. La selección que hace en este bloque se entrena junto con los otros parámetros de la red, para así permitir que la red pueda probar todas las entradas al modelo y luego seleccionar la que más le sirva para generar la salida deseada.

El tercer bloque es el *bloque de selección por combinación lineal*, el que utiliza parámetros

para realizar una combinación lineal de las entradas originales de la red y entregar un solo valor de salida que representa las entradas ponderadas de la red. Este tipo de bloques permite que no sea necesario combinar linealmente dentro de la capa dichas entradas y así ahorrar un bloque en cada capa que este sea necesario.

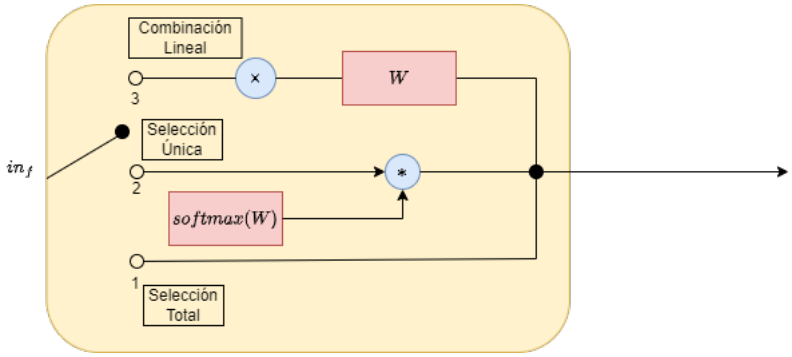


Figura 3.4: Bloques de selección en forma de switch, donde se elige mediante la estructura de la red cual de todas las selecciones se utilizará. El número 1, 2, y 3 representan respectivamente los bloques de selección total, selección única y combinación lineal.

3.4.2. Transformación de Bloques a Ecuaciones Escritas

Una vez entrenado el modelo este se debe analizar para llegar a una representación escrita de la ecuación Fina. Para esto no solamente la red debe poder ser transformada a dicha ecuación, sino que debe estar compuesta por los bloques presentes en la red. Por esto es necesario representar cada bloque de una manera escrita que sea equivalente a los cálculos que se realizan por el modelo. El método es diferente para cada uno de los bloques que se pueden utilizar, ya que las operaciones internas que se hacen difieren.

1. **Bloque univariable:** estos bloques al aplicar solamente la función a la entrada pueden ser descritas por la siguiente ecuación:

$$\text{block}_{1v} \equiv f(x) \tag{3.1}$$

2. **Bloque multivariable:** Estos bloques aplican la función a una combinación lineal de sus entradas, donde la cantidad de salidas es un hiperparámetro de la red. El modelo puede tener bias dependiendo del tipo de función que se está buscando reproducir, la expresión generada se puede observar como sigue:

$$\text{block}_{mv} \equiv f(Wx + b) \tag{3.2}$$

3. **Bloque Multivariable Punto a Punto:** Utilizan parámetros entrenables al igual que los bloques multivariable, pero la principal diferencia radica en el tipo de operación que se utiliza en la multiplicación y la suma. Por esto la expresión de estos bloques es parecida a (3.2) pero se diferencia en que las operaciones son punto a punto como indica la expresión siguiente:

$$\text{block}_{pw} \equiv f(W \odot x \oplus b) \tag{3.3}$$

4. **Bloques de selección:** Estos bloques al contrario de los bloques de funciones no se ven

reflejados como una expresión propiamente tal, ya que solamente transportan la entrada original de la red hacia capas posteriores. Lo que se debe tomar en consideración es que los bloques de selección puntual al momento de ser pasado a expresión escrita seleccionan la entrada con mayor probabilidad y en el caso de la selección por combinación lineal se entrega como una multiplicación entre las entradas y los parámetros de dicho bloque.

3.4.3. Obtención de Ecuaciones

Una parte fundamental del modelo de red propuesto es la habilidad para interpretarlo como una ecuación escrita, la cual pueda ser utilizada sin el modelo como mediador. De este modo, la representación que tienen los pesos del modelo a la hora de ser interpretado sigue una línea completamente diferente a los modelos convencionales que utilizan Deep Learning, dado que en dicho caso no existe una interpretación sobre los pesos y como estos afectan la generación de las expresiones. La red propuesta sigue una línea más parecida a las desarrolladas en [10], [11],[23] donde para generar la ecuación se debe iterar a través de las capas del modelo.

Para interpretar cada bloque, se debe considerar que estos en su interior contienen una matriz de pesos además de seguir la descripción dada en la sección 3.4.2. De este modo, basta con recorrer cada red desde su inicio a su final para reconstruir la expresión que la red completa representa. Al momento de traspasar el modelo a una expresión escrita, es fundamental entender que dicha expresión no es una aproximación, sino que es equivalente a la red misma. Sin embargo, para poder pasar del modelo entrenado a la expresión, se debe hacer una aproximación en el bloque de potencia (sección 3.4.4), dado que dicho bloque utiliza una función sigmoide para aproximar el signo de la expresión, por lo que, una vez terminado el entrenamiento, el signo se define dependiendo de si la probabilidad es o no mayor a 0.5. Una vez hecha esta aproximación la red y la expresión generada son equivalentes.

Además, como la red aprende por gradiente descendente, sus parámetros son números de puntos flotantes, por lo que al momento de reescribir la expresión es necesario definir la precisión que se utilizará. Precisiones de 3 decimales son lo suficientemente buenas como para describir confiablemente la red, mientras que precisiones de un decimal permiten expresiones más compactas pero hacen que la representatividad de dicha expresión tenga cierta diferencia al modelo entrenado.

Para evalúar la expresión generada se utiliza la función de evaluación de strings de python, la cual permite evaluar expresiones en forma de caracteres que representen una expresión. De esta forma, es posible evaluar las expresiones producidas en vez de la red, lo cual permite confirmar de que tanto la red como su representación son equivalentes.

3.4.4. Función de Potencia

El uso de una función de potencia continua dentro de la red es una necesidad que permite abstraer el nivel de potencias al que puede llegar una red de su número de capas. Esto permite que la potencia sea un parámetro de la red y no una constante fija, como es el caso de los trabajos expuestos en la sección 2.2.3.2. Una de las formas para aproximar la función potencia es el trabajo realizado en [35].

3.4.5. Aproximación de Saito-Nakano

En [35] se utilizan redes neuronales para la búsqueda de polinomios. Esto se logra usando una mlp de dos capas, la primera de estas permite generar combinaciones lineales entre las entradas de la red, mientras que la segunda busca aprender las potencias a las que dichas

combinaciones lineales estarán elevadas. La cantidad de términos presentes en el polinomio se define con la cantidad de salidas que presenta la primera capa. A modo general, el tipo de expresiones que encuentra este modelo se representa en la ec. 3.4, por lo que permite encontrar cualquier potencia dentro de ciertos límites. El modelo de Saito Nakano encuentra expresiones de potencia del siguiente tipo:

$$f_{power}(x_0, x_1) = \left(w_{0,0}^{LC} x_0 + w_{1,0}^{LC} x_1\right)^{w_{0,0}^{pc}} \left(w_{0,1}^{LC} x_0 + w_{1,1}^{LC} x_1\right)^{w_{1,0}^{pc}} \tag{3.4}$$

Para aproximar la función potencia Saito-Nakano hacen uso de una propiedad matemática de la función exponencial (ec. 3.5). Usando **a** como un parámetro en vez de un coeficiente en la expresión, se puede utilizar una red neuronal de dos capas para aproximar dicha función con logaritmo y exponencial respectivamente como funciones de activación. La propiedad matemática es la siguiente:

$$|x|^{\alpha} = exp(aln(x)) \tag{3.5}$$

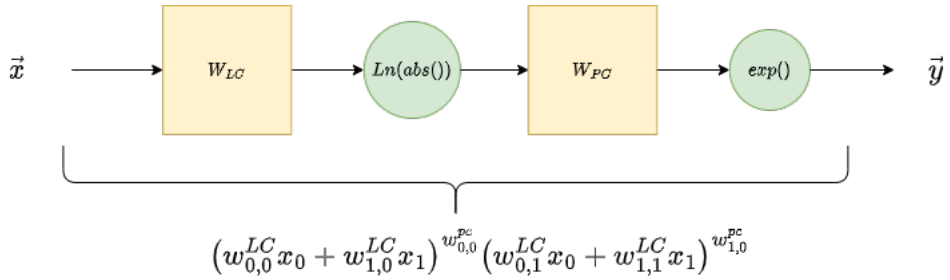


Figura 3.5: Gráfica que muestra cómo se aproxima la función potencia mediante el modelo expuesto de Saito-Nakano.

La figura 3.5 ilustra la aproximación de Saito-nakano para la función de potencia. Este presenta la gran ventaja de que el valor de la potencia es libre, es decir que no se requiere una arquitectura específica para poder encontrar mayores potencias. A pesar de tener grandes ventajas, también presenta limitaciones evidentes en su confección. Primero, el método no contempla los números negativos, esto es un problema cuando el dominio de la entrada contiene estos números. Segundo, todas las potencias aproximables por este método son potencias simétricas [6], esto se debe a que los números negativos no se encuentran en el recorrido de la función exponencial, por lo que aunque se repare el logaritmo natural en el cero o en valores negativos, nunca se podrá recobrar valores que contemplen asimetrías [7] como es el caso de la función x^3, debido a la definición de la función exponencial.

3.4.6. Aproximación de Potencia Propuesta

Tomando como inspiración [35], se buscó la manera de utilizar esta aproximación y ampliar la definición expuesta a todo el dominio de los reales y tanto a las funciones simétricas como asimétricas. Para esto fue necesario reparar la función logaritmo de manera tal que pueda recibir valores negativos e incluso el valor cero. Esto se hizo aplicando el valor absoluto de la entrada al logaritmo y dejando un valor de -6 fijo para los casos donde el valor de la entrada

[6] Función que cumple con la siguiente condición $f(x) = f(-x)$
[7] Función que cumple con la siguiente propiedad $f(-x) = -f(x)$

es menor a 0.019. Esto permite solucionar una limitación de dominio, ya que el polo cercano a cero el valor del logaritmo será cortado en -6, haciendo que dicha función quede definida correctamente en los reales como sigue:

$$ln_p(x) = \begin{cases} ln(|x|) & \text{|x|>0.019} \\ -6 & \text{en caso contrario.} \end{cases} \tag{3.6}$$

Por otro lado, la segunda limitación se resolvió considerando la definición de potencia compleja (ec. 3.9), donde cualquier número que sea negativo (que por consecuencia tenga un ángulo en el plano complejo) de no ser entero hará que el resultado pase de estar en el plano real a estar en el plano complejo. Las ecuaciones correspondientes son las siguientes:

$$z^c = e^{cln(z)} \tag{3.7}$$

$$ln(z) = ln(|z|) + i(Arg(z) + 2n\pi) \tag{3.8}$$

$$z^c = e^{cln(z)} e^{ci(Arg(z)+2n\pi)}. \tag{3.9}$$

De lo anterior se concluye que dentro del plano real solamente existen 2 tipos de potencias: las potencias simétricas y asimétricas. De este modo, la complejidad de generar una función potencia continua radica en poder aprender a medida que el modelo se entrena si la estructura que se está intentando generar es simétrica o asimétrica. Por consiguiente la función que se está intentando encontrar es la siguiente

$$f_p = \begin{cases} exp(aln(|x|) & \text{Si es par} \\ sign(x)exp(aln(|x|)) & \text{Si es impar} \end{cases} \tag{3.10}$$

Para poder aprender la simetría de las funciones de potencia se pasó del modelo de la figura 3.5 al de la figura 3.6. Para realizar esta conversión es necesario utilizar otra matriz de parámetros W_{sign}, que encapsula el aprendizaje de la simetría del bloque a medida que se entrena.

El método consiste en definir la simetría como un valor binario. El valor uno hace referencia a que se elige una estructura simétrica y cero una estructura asimétrica. Para poder aprender dicha simetría el problema se relaja a una representación continua, donde, esta decisión será representada por una función sigmoide. Una vez calculada probabilidad de simetría, esta se suma a la expresión $(1-\sigma)sign(x)$ de manera de generar un *trade-off* entre la existencia o no de la función signo en la expresión de la potencia. Gracias a este método, el modelo decide la simetría mediante gradiente descendente.

En el caso multivariable, se calcula cada entrada elevada a su potencia respectiva y su simetría. Para después multiplicar estos valores, de esta manera se mantiene el signo original de la función de potencia.

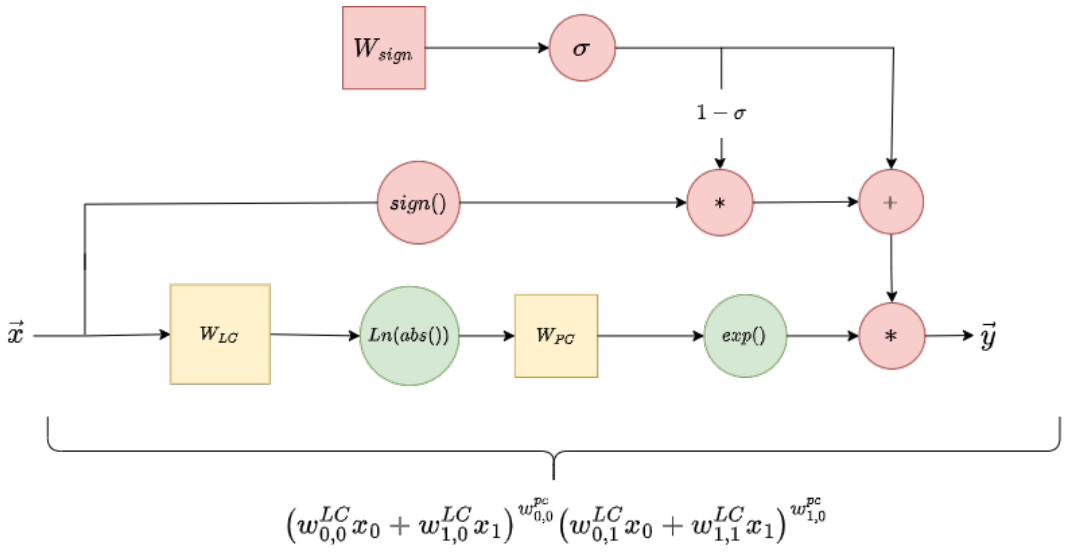


Figura 3.6: Modelo de aproximación de potencias que permite encontrar potencias en todo el espacio de funciones de potencia y en todo el dominio real.

3.4.7. Inicialización de Pesos

Las redes neuronales en muchos casos tienen problemas con la inicialización, ya que dependiendo de la función de activación que utilicen dicha inicialización pueden ser o no útil para resolver el problema. En la literatura un problema conocido es la inicialización de parámetros con valores cercanos, esto ya que generará que la actualización de estos por *backpropagation* terminase llevando dichos pesos distintos a un mismo valor. El modelo de red propuesto al utilizar distintas funciones de activación no es ajeno a dichas limitaciones, por lo que la inicialización de parámetros es un punto fundamental del entrenamiento. Es por esto que es necesario considerar cual es la función que representa el bloque que se está inicializando, además se debe considerar los bloques de funciones que componen la red. Esto se debe a que diferentes funciones de activación tienen diferentes gradientes, los cuales generan por ende diversas superficies de actualización de parámetros. A continuación se presentan las funciones que dentro de la red tienen inicializaciones diferentes a las convencionales, buscando de este modo una mejor convergencia de sus aproximaciones, para todas las otras funciones se utiliza el inicializador de Xavier [36].

3.4.7.1. Inicialización Pesos Función Potencia

La función potencia, es una versión continua a lo que sería una serie de Taylor en la aproximación de funciones. Comenzar la red con valores bajos de potencia genera dos problemas no deseados. El primer problema radica en el valor de los pesos, ya que si los parámetros parten muy bajos la red puede pasar estos rápidamente al plano negativo. Esto genera efectos adversos en el entrenamiento cuando esta convergencia es muy temprana, ya que el paso desde el plano positivo al negativo es más simple por gradiente descendente que el paso inverso. Esto se debe a una divergencia en el cero que existe en el paso de los parámetros desde el plano negativo al positivo, el cual se genera debido a la aproximación que se hace a la función $\frac{1}{x}$ la cual tiene un polo en el cero. Esto genera un estancamiento en el entrenamiento, y que se

ilustra en la figura 3.7. Por otro lado, cuando la función potencia es inicializada en los valores positivos se puede ver que no hay dicho problema ya que la función converge al cero, lo cual no dificulta el entrenamiento. Debido a esto es importante que los pesos de las potencias se inicialicen lejos del cero y no en valores negativos para casos generales.

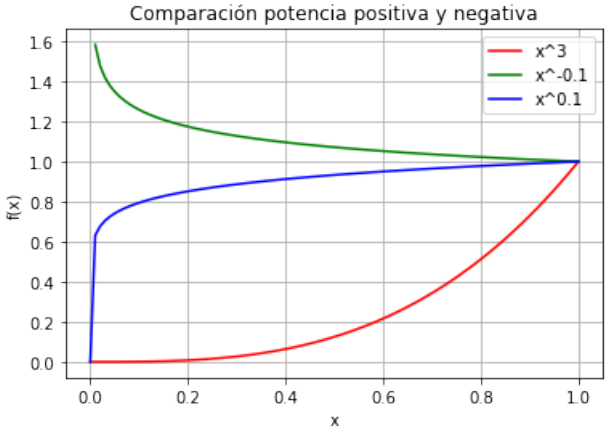


Figura 3.7: A medida que el valor del exponente de la potencia se acerca a cero desde los negativos se genera un polo en el cero. La curva azul nos muestra como el valor de la función cuando el exponente se acerca al cero por los positivos genera una convergencia en cero, esto hace que el paso de la función potencia sea más fácil desde el eje positivo al negativo en el plano de los pesos de la función potencia.

El segundo problema presente en la función de potencia durante el entrenamiento se debe al valor de inicialización de los exponentes. Como se indicó anteriormente, la función potencia corresponde a una representación continua de una serie de Taylor que está limitada a los primeros h_f términos. De este modo, inicializar la red en un valor alejado de uno o con todos los valores de las salidas iguales no genera ningún beneficio sino que aleja al modelo de representar efectivamente una serie de Taylor. La forma correcta de inicializar este modelo es de acuerdo a la siguiente regla: Los exponentes de la función potencia parten en uno, y si h_f es mayor a uno para un bloque entonces las salidas posteriores tendrán los siguientes términos de la serie de Taylor. Por ejemplo, si $h_f = 3$ entonces lo que ocurrirá es que las potencias que salgan del bloque de potencias serán del uno al tres. El único caso en que los pesos de esta función parten en dos es cuando existe en la primera capa una función identidad o un bloque de selección en la capa siguiente. Esta inicialización sigue la idea basal de aproximar una serie de Taylor y por ende los bloques lineales y de selección representan el primer término de la serie de Taylor por lo que sería redundante utilizarla dos veces. Para los casos en que existe más de una variable el modelo genera polinomios donde cada una de los exponentes va subiendo de manera ordenada, por ejemplo, si $h_f = 3$, entonces el polinomio generado será $x_0^1 + x_1^1 + x_0^2$.

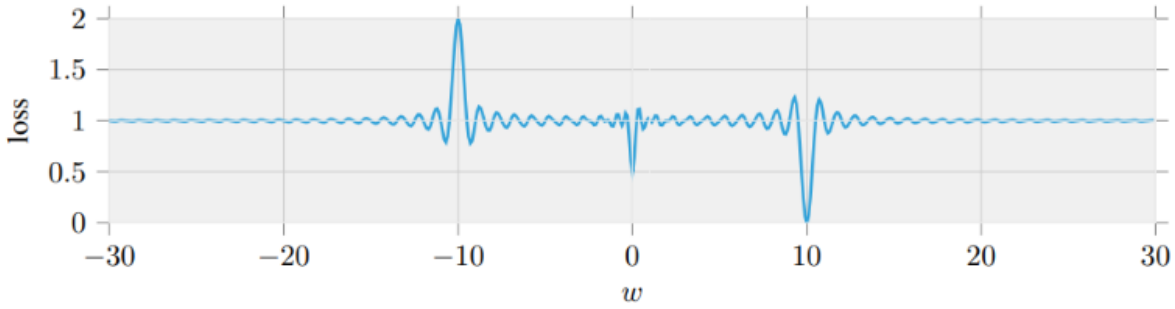
3.4.7.2. Inicialización Pesos Función Exponencial

La función exponencial es una función ampliamente utilizada en las ecuaciones científicas y matemáticas, debido a su naturaleza los exponentes que tiene esta función no suelen ser muy grandes y en caso de serlo éstos son negativas, ya que en caso contrario las ecuaciones divergen y pierden su sentido físico. Debido a esto es necesario considerar su naturaleza

explosiva, por consiguiente, la función exponencial se inicializa con valores negativos para su combinación lineal. Esto hace que sin importar el valor que esta tenga en un inicio la función se iniciará en una zona de estabilidad donde es más probable converger al cero que a un valor alto. Si las entradas a este bloque fuesen negativas, se toma como consideración que en caso de existir una explosión en su valor, el modelo no es el correcto y por ende se debe seguir el procedimiento explicado en la sección 3.6.1.

3.4.7.3. Inicialización de Pesos Función Seno

La función Seno es parte de la familia de las funciones trigonométricas, y tiene la particularidad de que se encuentra en la gran mayoría de las funciones científicas y matemáticas. A pesar de su naturaleza diferenciable en todo el dominio, trabajos como [37], [38] muestran que entrenar una red neuronal que utilice funciones sinusoidales como funciones de activación no es un trabajo simple. En específico, en [37] se demuestra que buscar la frecuencia de la función $sin(wx)$ genera el gráfico de costo de la figura 3.8 cuando la ecuación buscada es $sin(10x)$. En dicho trabajo se concluye que las funciones de este tipo tienen un mínimo local en cero que hace difícil que la red pueda salir de dicho mínimo, además existen una infinidad de curvas de baja amplitud que se forman debido a la aparición de funciones Seno Cardinal (sinc)[8], generando de este modo que el camino de mejora no sea necesariamente suave para el algoritmo de *Backpropagation*.



(a)

Figura 3.8: Función de costos para $sin(wx)$, donde se intenta encontrar w para optimizar la expresión, cuando w=10.

Además si se parte de pesos muy alejados los gradientes son muy pequeños como para poder avanzar de manera satisfactoria hacia dicho óptimo. A esto se le suma el hecho de que agregándole a la función seno la componente de desfase y amplitud el problema tiene otros mínimos locales en diferentes valores de estos, haciendo que los mínimos de w no sean los únicos para dicha aproximación. Esto se ilustra en la figura 3.9.

[8] La función Seno Cardinal es la función definida por la siguiente expresión $sinc(x) = \frac{sin(x)}{x}$

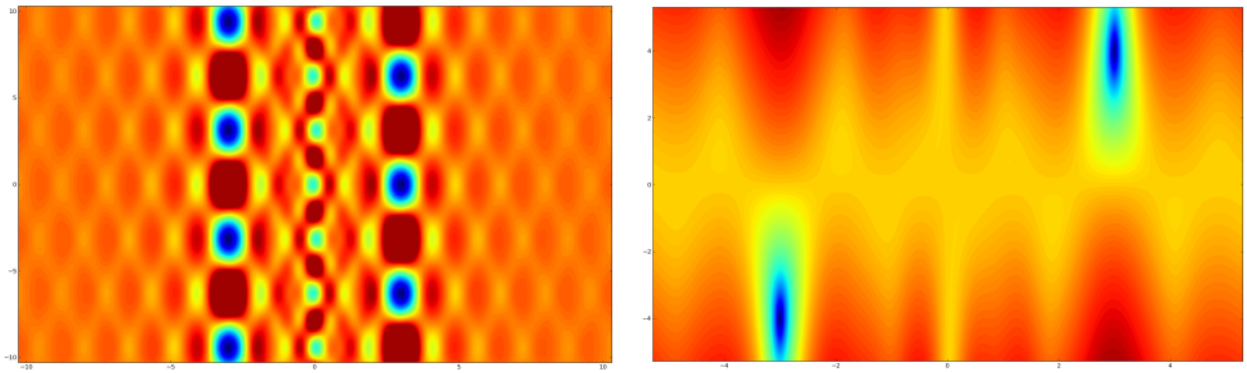


Figura 3.9: Mínimos locales obtenidos cuando se intenta optimizar la función seno considerando otra variable además de la frecuencia, ya sea el desfase (b) o la amplitud (A) representadas en el eje y respectivamente. Mientras tanto, el eje x de los gráficos representa la frecuencia (w). Las imágenes fueron extraídas de [37].

Debido a lo expuesto anteriormente, se puede indicar que la función seno no es una función simple de ajustar mediante *backpropagation*.Otros trabajos como en [38] y [39] indican que utilizar redes neuronales con funciones de activación seno constituye una transformada de Fourier generalizada, por lo que al momento de inicializar dicha red es necesario buscar las componentes principales de la transformada rápida de fourier (FFT) y usar dichos términos como inicialización. Además en [38] se propone que en vez de utilizar la función seno como función de activación se utilice es la función $cos(\pi x)$ asumiendo que el dominio de la función a encontrar es 2π-periódica, dicha función de activación permite que la tasa de cambio de los valores del seno sean mayores, sacando a esta función del mínimo local en 0. De este modo, el bloque de funciones seno tendrá la función de activación $sin(\pi x)$ y por ende, para inicialización de la red se hará con valores cercanos a $W = 0.3$, de manera que el argumento sea cercano a 1 y así los valores iniciales de la red sean cercanos a $sin(x)$.

3.4.8. Cálculo de la Salida de Cada Capa

Para calcular la salida de cada capa primero se debe pasar la entrada por cada función mono variable presente en la capa. Luego se concatena la entrada y la selección hecha por el bloque de selección, si es que este fue seleccionado, para entregarla como entrada a las funciones multivariable y punto a punto. Una vez obtenidos todos estos resultados se concatenan para obtener el vector de salida de la capa. La generación de salida se describe en el algoritmo 1

Algorithm 1 Generación de salida.

Require: Fn_L, x, x_o ▷ Fn_L :Cantidate functions, x: layer input, x_0: net input
Ensure: $Fn_L \neq [\,]$
  if $Fn_L[i]$ has selection block **then**
    $x_{extended} = \text{concatenate}(x, x_o)$
  else
    $x_{extended} = x$
  end if
  $x_L = [\,]$
  XWout = [ ]
  for i in len(Fn_L) **do**
    if $Fn_L[i]$ is 1Variable **then**
      x_L.append($Fn_L[i](x)$)
    else
      x_L.append($Fn_L[i](x_{extended})$)
    end if
  end for
  $y = \text{concatenate}(x_L)$
  return y

3.4.9. Generación de Redes

Para poder generar funciones compuestas es necesario hacer una concatenación de las capas anteriormente propuestas para así formar un red. La generación de redes debe hacerse de una manera que simplifique la optimización de parámetros en vez de generar más ruido del necesario, esto último ocurre cuando los bloques tiene más salidas de las requeridas o muchos bloques por capa. De este modo, existen ciertas reglas para crear una red. Primero, el hiperparámetro h_f y la lista de funciones son iguales para todas las capas, esta definición es necesaria porque a priori cuando se está intentando resolver un problema de regresión simbólica no se sabe cuantos términos de cada tipo se requiere.Para encontrar la mejor estructura para dicho modelo de red es necesario hacer pruebas con diferentes valores de capas y h_f.

Además, en la primera capa se pone la condición de que cada bloque de funciones obtenga como salida el mínimo entre h_f y el número de variables de entrada. Como restricción a la estructura de la red se considera que la cantidad de operaciones que se busca obtener por cada bloque no puede ser mayor que la cantidad de variables de entrada que se le entrega. Dicha restricción no se aplica para capas posteriores ya que esta regla no es necesariamente cierta. Para el caso específico de las redes de una sola capa, la función potencia es la única que puede usar un valor mayor al número de variables, ya que se asume que en caso de haber una sola capa esta será utilizada para la búsqueda de polinomios.

Otra consideración es que cuando se genera una red, al final de las capas concatenadas se agrega un bloque de combinación lineal que recibe la salida de la última capa ponderando sus componentes para entregar una sola expresión como salida de la red. Además, al generar una red no se permite que las primeras capas tengan bloques de selección, esto sería redundante ya que la entrada de dicha capa las entradas originales.

En general los modelos de red no debiesen tener más de 3 capas y tampoco más de cuatro h_f por bloque, estos números se definieron de manera experimental, pero al actualizarse

los pesos de la red por gradiente descendente y aumentarse el número de parámetros es muy probable encontrar funciones con un gran número de términos en vez de expresiones acotadas. Es por esto que se busca que este tipo de red no tenga muchos bloques y además que presente pocas salidas por bloque, de manera de limitar lo más posible expresiones sub-óptimas.

3.5. Regularización de Redes

Como se indicó en la sección 2.4 los modelos de redes neuronales utilizados en regresión simbólica por lo general son entrenados con regularización. Esto se debe a que en general las redes neuronales tienden a sufrir de sobre ajuste debido al uso de *backpropagation* para su entrenamiento. En la literatura clásica de las redes neuronales una manera de evitar dicho problema es mediante la regularización, método por el cual las redes tienen que sopesar el ajuste que se genera a la curva con el tamaño de los pesos que dicha red utiliza.

Los modelos de regresión simbólica basados en redes neuronales dedicadas no son la excepción a esta regla y por ende también deben ser entrenados utilizando regularización. En las redes neuronales convencionales la razón del uso de regularización tiene una explicación teórica basada en el sobre ajuste que se genera en la red. Sin embargo, en el caso de las redes neuronales dedicadas utilizadas en regresión simbólica existe otra razón práctica, además de la anteriormente mencionada, para entrenar dichas redes con regularización. La razón radica en el hecho de que la mayoría de las expresiones matemáticas utilizadas en la física no tienen valores extremadamente grandes, haciendo que por ende fórmulas físicas que contengan coeficientes muy grandes no sean las soluciones deseadas para solucionar los problemas de regresión simbólica.

El modelo de red propuesto en este trabajo de tesis busca justamente limitar este efecto, por ende existen pesos de la red cuya penalización por ser grandes es mayor a otros pesos e incluso pesos de la red, que son regularizados de una manera diferente. Esto último ocurre cuando la penalización perjudica la interpretación de esa parte del modelo más que ayudarlo a converger a la mejor solución posible. Los modelos de red son regularizados utilizando una regularización $SR_{\text{reg}(W;\lambda)}$ que usa un ponderador λ para penalizar la norma de cada peso. Se requiere un nuevo método de regularización por la necesidad de controlar no solamente el valor que tienen los parámetros del modelo sino que además los signos de las funciones potencias (sección 3.4.4), esto último se debe a que sin importar el valor que tenga este parámetro, su efecto no se verá reflejado en la expresión como un coeficiente, sino que simplemente definirá si dicha potencia tendrá una naturaleza simétrica o asimétrica. La fórmula de la regularización de la SRNet es la siguiente:

$$L_{SR} = \begin{cases} \lambda|W_i| & si \quad W_i \text{ no es un parámetro de simetría} \\ \\ \min(1-\sigma(W_i), \sigma(W_i)) & si \quad W_i \text{ es un parámetro de simetría.} \end{cases} \tag{3.11}$$

La función mínimo utilizada en los parámetros de simetría está relacionada con el hecho de que no es deseable que el valor de la sigmoide (representada como σ en la ecuación 3.11) quede entre 0 y 1 al momento de decidir el signo. Es necesario por ende que se obligue al modelo a tomar una decisión entre la función simétrica o asimétrica. Esto se logra tomando la mínima distancia al valor 0 o al valor 1, de modo que no exista una predilección sobre ninguna de las simetrías. Aplicar esta regularización resulta fundamental para encontrar un modelo que tenga relación con la realidad, esto ya que para tener una representación de la red que

sea idéntica al modelo de red entrenado es necesario que no existan dualidades entre ambos modelos. Un punto importante a considerar es que la penalización a la simetría del modelo no se encuentra ponderada por un parámetro λ, esto se debe a que no se desea encontrar modelos que compensen el ajuste del modelo con la simetría que este presenta, por lo que todo modelo que no haga una selección definitiva de su simetría tendrá una penalización alta a modo de dificultar ese tipo de soluciones.

Otra regularización que se utiliza en este trabajo de tesis es la regularización L_1:

$$L_1(x) = \sum_{i=1}^{N} |x_i|, \tag{3.12}$$

la cual, permite que las expresiones generadas no tengan coeficientes demasiado grandes y facilitando así el ajuste con los mejores parámetros posibles para dicho modelo. En específico, esta regularización se utiliza en el algoritmo de búsqueda estructural, cuyo objetivo es encontrar el mejor conjunto de funciones para la resolución del problema. Si se utilizara la regularización L_{SR} directamente en dicho algoritmo todos los modelos que utilizan polinomios tendrían una desventaja en comparación a los modelos que usan otras funciones no lineales. Esto se debe a que los polinomios son las únicas funciones con parámetros de simetría y por ende el valor de ajuste que estos presentarían sería menor a la de los modelos que solamente usen funciones no lineales.

3.6. Método de Entrenamiento

El entrenamiento de la red neuronal dedicada es de tipo supervisado, el que es realizado en línea y por etapas, utilizando las entradas $\vec{x}$ como las entradas al modelo y la salida y como la salida esperada, separando el ajuste de parámetros de la búsqueda de estructura (SS). Un pseudo código del método de entrenamiento se muestra en el algoritmo 2 además de un diagrama expuesto en la figura 3.10. En una primera instancia se separa el conjunto de entrenamiento en dos conjuntos, uno de entrenamiento y otro de validación del algoritmo SS. En dicha sección del algoritmo el modelo busca la mejor estructura para resolver el problema, utilizando el método de búsqueda estructural expuesto en la sección 3.7. Dicho algoritmo genera una predicción de los bloques requeridos por cada capa para construir la red. La estructura del modelo requerido se entrega en forma de máscara binaria, dicha matriz tiene un tamaño de $(N_{bloques}, N_{Capas})$ donde cada entrada representa la existencia o no de dicha función candidata i con $i \in N_{\text{bloques}}$ en la capa j con $j \in N_{\text{Capas}}$. Una vez obtenido un subconjunto de funciones candidatas para cada capa, el espacio de posibles estructuras generadas con dicho conjunto se reduce considerablemente, por lo que se generan todas las combinaciones posibles de redes que tengan los bloques seleccionados. De este modo, se construyen C redes diferentes, siendo C el número de estructuras correctas encontradas con las combinaciones de los bloques seleccionados. Posteriormente, la red generada se entrena con la función de pérdida del error cuadrático medio (MSE) por un periodo de 1000 épocas y con un *batch* size de 64. La evaluación del modelo se hace con la métrica NRMSE y R^2, utilizando las expresiones generadas por el modelo. Una vez terminado el entrenamiento, para cada red se selecciona el modelo con el mejor resultado en el conjunto de validación. A cada solución se le aplica una optimización no lineal de sus parámetros, entregando como inicio de búsqueda los parámetros encontrados durante el entrenamiento de la red.

El optimizador no lineales se utilizado es L-BFGS-B. La principal ventaja de L-BFGS-B

Algorithm 2 Algoritmo de entrenamiento de SRNet.

$ss_{train}, ss_{val} = \text{split}_{70-30}(D_{train})$ ▷ Split training data into 70-30
$ss_{epochs} = 50$
$g_{pred} = SS(ss_{train}, ss_{val}, Itr, Res, ss_{epochs}, Fn)$
$N_{list} = [\ ]$
$Cg_{pred} = \text{Combinations}(g_{pred}, Fn)$
for i in len(Cg_{pred}) **do**
    $N_{list}.append(SRNet(i, Fn))$
end for
$train_{eqn} = Train(N_{list}, D_{train}, D_{val}, epchs)$
done, $opt_{eqn} = \text{L-BFGS}(best_{eqn}, D_{train}, bound)$
if not done **then**
    done, $opt_{eqn} = BFGS(train_{eqn}, D_{train})$
end if
$best_{eqn} = check(D_{test}, opt_{eqn}, train_{eqn})$
return Simpy.simplify($best_{eqn}$)

se encuentra en que permite el uso de límites para la de búsqueda de cada coeficiente de la expresión, por lo que asumiendo que el ajuste de parámetros se encuentra cerca del mínimo global, los parámetros correctos no debiesen estar muy lejos de la solución previamente encontrada, es por esto que los límites definidos para la optimización son $[p-1, p+1]$. En caso de fallar se utilizará un ajuste por BFGS que no utiliza ventanas de búsqueda. Para este caso al igual que el anterior la posición inicial de búsqueda será los coeficientes de la expresión generada mediante el entrenamiento. Una vez terminada esta etapa se procede a comparar todas las soluciones generadas con el conjunto de testeo para elegir la mejor como solución final, esta solución antes de ser entregada se simplifica con Simpy[9].

[9] https://docs.sympy.org/latest/modules/simplify/simplify.html

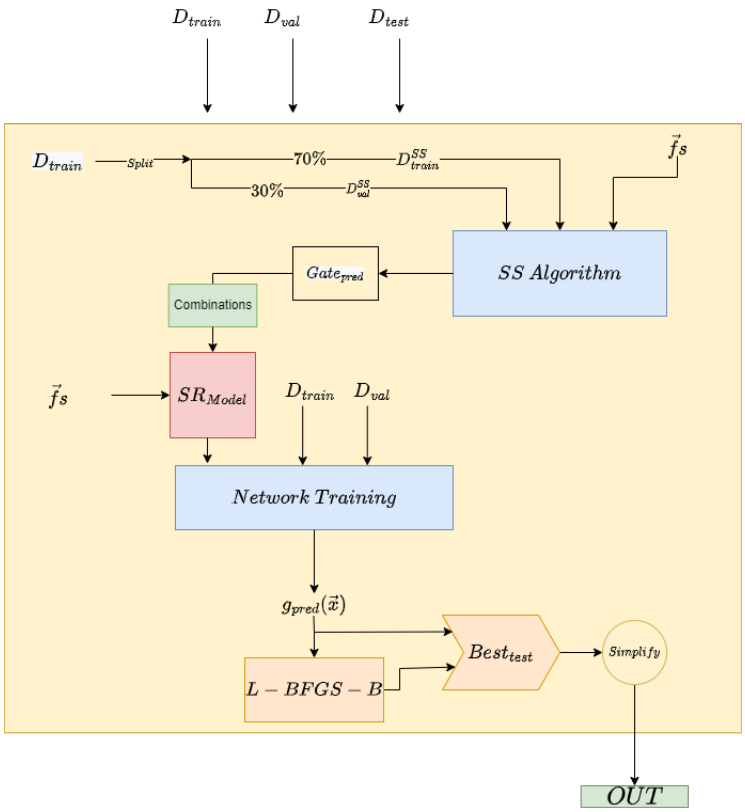


Figura 3.10: Método de entrenamiento de las redes SRNet. En la figura se parte con un conjunto de datos de entrenamiento, validación y testeo. El conjunto de entrenamiento es dividido en dos uno de validación y entrenamiento (ambos para el algoritmo SS). Con estos conjuntos es que se buscan las estructuras de la solución mediante el algoritmo SS. Cuando este ya haya entregado una predicción de funciones, se generan todas las combinaciones posibles de estructuras que no rompan con las reglas de las combinaciones posibles (explicadas en sección 3.7.4). Con la fase anterior terminada las estructuras de redes son entrenadas, generando cada una una solución candidata g_{pred} que posteriormente es optimizada mediante L-BFGS-B. Finalmente se comparan todas las soluciones dentro del conjunto de prueba y la mejor solución es simplificada por Sympy.

3.6.1. Problemas de dominio de expresiones generadas:

A medida que el modelo se entrena y se generan distintas expresiones, existe una alta posibilidad de que se generen problemas en los dominios de dichas expresiones o incluso los parámetros del modelo sufran de valores explosivos de gradiente (problema muy común en algunos modelos de redes neuronales). El entrenamiento de este modelo debe saber lidear con este tipo de problemas y definir un procedimiento para que así este tipo de soluciones no afecten los resultados del algoritmo. Por definición las expresiones que se está buscando encontrar no tienen una naturaleza explosiva, son funciones bien definidas cuyas expresiones deben ser derivables por partes. Por esto en la fase de entrenamiento se asume que los datos no tienen como dominio el punto de discontinuidad de dicha expresión.

Al no considerar ese tipo de funciones dentro del espectro de soluciones posibles, se puede definir con anterioridad que cualquier estructura que genere una explosión de gradiente o que no esté contenida en el plano real debe ser penalizada. La penalización se genera al considerar

un valor máximo en la función de costos que está definido como sigue:

$$P = N_s(3\sigma_y)^2, \tag{3.13}$$

donde N_s es el número de muestras y σ_y es la desviación estándar es de los datos de salidas deseadas. Dicho valor proviene de la condición de que el valor de la raiz del error cuadrático medio debe ser menor a 3 veces la desviación estándar.

Se define así que un valor que se escape del plano real o no pueda ser evaluado tiene un valor mayor que tres veces la desviación estándar, haciendo de este un valor atípico en el entrenamiento que por ende en ningún caso será seleccionado ni por el algoritmo de búsqueda estructural ni por el entrenamiento de posibles combinaciones.

3.7. Algoritmo de Búsqueda Estructural (SS)

Uno de los desafíos de usar redes neuronales dedicadas en regresión simbólica es la condicionante de utilizar todas las funciones candidatas en un modelo, estas generan un mayor ruido en el aprendizaje, ya que no todas las funciones candidatas son parte de las estructura real de la función buscada. El problema estructural es un desafío completamente aparte que deben buscar solucionar los métodos de SR, esto se debe al poder que tienen la mayoría de los modelos para ajustarse a cualquier expresión con una base de funciones candidatas que no necesariamente son las adecuadas. Buscar la estructura correcta de acuerdo al ajuste a los datos probando todas las estructuras posibles no es una forma adecuada de abordar el problema. Por esta razón, es necesario un algoritmo de búsqueda estructural. En esta sección, se propone un método para encontrar la estructura correcta del modelo sin tener la necesidad de entrenarlo hasta converger sino que solo por una cantidad acotada de tiempo. A modo de introducción se muestra en la figura 3.11 un ejemplo del algoritmo generado..

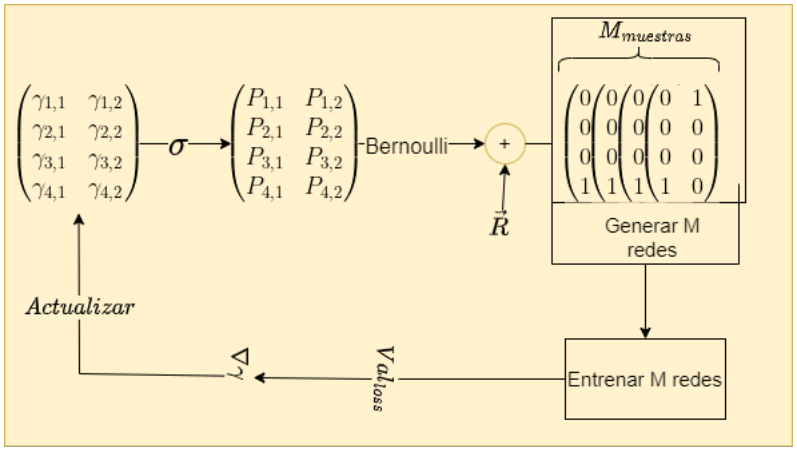


Figura 3.11: Diagrama del algoritmo SS, Primero el modelo genera probabilidades mediante sus parámetros estructurales (γ) y una función sigmoide (σ), se generan las estructuras para la iteración que cumplen con las restricciones (R), Posterior a su entrenamiento se generan los gradientes mediante la función de objetivo y se actualizan los parámetros mediante el algoritmo REINFORCE.

3.7.1. Parámetros Estructurales

Una de las ventajas de la red propuesta es su independencia estructural en la optimización de parámetros, una vez generada la red esta se entrena de la misma manera sin importar su estructura. Esto presenta una gran ventaja ya que de tener la estructura correcta el problema de optimización de parámetros se simplifica considerablemente. A su vez se requiere tener un método que sin información a priori pueda encontrar dicha estructura. Debido a esto nace la idea de parámetros estructurales inspirada en [40] y [41]. Los parámetros estructurales representan los logits de la distribución de cada bloque, donde la probabilidad de existencia se obtiene al aplicar la sigmoide a dicho parámetro estructural.

$$P(\gamma_{(N_{FN},N_{Layers})}) = \sigma(\gamma_{(N_{FN},N_{Layers})}). \tag{3.14}$$

Por esta razón los parámetros estructurales se representan en una matriz con forma (N_{FN} , N_{Layers}), que corresponden al número de funciones candidatas y cantidad de capas. Para calcular la probabilidad conjunta de una estructura completa se utiliza una heurística, que es la suposición de que los bloques son independientes entre si y que por ende la probabilidad de la estructura completa se calcula como el producto de la probabilidad de cada bloque. De no considerarse esta suposición el cálculo de las probabilidades de ocurrencia de los diversos bloques se complejizaría demasiado, ya que cada bloque depende del conjunto de bloques seleccionados en capas posteriores y anteriores.

3.7.2. Función Objetivo a Maximizar

La finalidad del algoritmo propuesto es aprender los valores de los parámetros estructurales y por ende obtener las mejores estructuras para la resolución del problema. El algoritmo REINFORCE [33], es útil para solucionar este problema ya que se puede entrenar un modelo muestreando estructuras y mediante la función de costos definida aprender los parámetros estructurales. En este caso la recompensa definida depende del ajuste del modelo a los datos, siendo esta la función (3.15), por lo que estructuras que obtengan un mejor resultado generarán que el valor de la probabilidad de ésta aumente, haciendo que la red aprenda la mejor combinación de bloques. La función de recompensa se define como:

$$L(x;\theta,y) = \frac{1}{(1+\sum_i((\hat{y}-y)^2))}. \tag{3.15}$$

La razón principal para la elección de la función de recompensa (3.15) que utiliza la reducción por suma y no una basada en el error cuadrático medio es que un error bajo en promedio no quiere decir necesariamente que el ajuste del modelo a la curva sea correcto. En específico, en muchos casos pueden existir incongruencias grandes entre el modelo y la salida deseada y aún así tener un valor promedio de error bajo. Esto ocurre cuando el modelo se ajusta de manera correcta a la mayoría de los puntos, salvo en zonas extremas el ajuste del modelo no es el adecuado, dicha diferencia puede ser una determinante para elegir entre un modelo y otro. Además, como la mejora en los parámetros se guía por gradiente descendente, los modelos a pesar de no tener la estructura correcta pueden ajustarse bien a la curva de entrenamiento, por lo que dichas diferencias sutiles deben ser detectadas por la función de recompensa a utilizar. En el anexo B.1 se ilustra la búsqueda realizada sobre distintas funciones de target utilizadas en el algoritmo SS.

Entre las características deseadas de las expresiones encontradas un punto muy importante es la compactabilidad, es decir, ecuaciones con la menor cantidad de parámetros que se ajusten lo mejor posible a la curva deseada. Dos ecuaciones pueden tener el mismo ajuste pero al momento de seleccionar cual es mejor, siempre se elegirá la que tiene menos parámetros ya que esto permite una mejor análisis de la expresión encontrada.

En [41] propone limitar la cantidad de características seleccionadas dividiendo la función de costos por la cardinalidad del conjunto seleccionado elevado a un parámetro alfa. Esto permite que selecciones con más características se vean penalizadas y por ende su valor de recompensa se vea disminuido. La manera de aplicar esta idea al problema de búsqueda estructural no es contando la cantidad de bloques que presenta el conjunto seleccionado, sino que cantidad de parámetros que generó dicha selección. De esta forma la función que se busca maximizar es la siguiente:

$$T = \frac{L(x;\theta,y)}{|P|^{\alpha}}. \tag{3.16}$$

Donde $L(x;\theta,y)$ es la función de costos presentada en la ecuación 3.15. |P| representa la cantidad de términos que tiene la expresión generada y α es un hiperparámetro que controla que tan rápido aumenta la función de costos cuando aumenta la cantidad de términos en la expresión.

3.7.3. Restricciones

Además, de que dentro del algoritmo existen métodos para el manejo de explosiones en los gradientes y de funciones que no están completamente definidas en el plano real, existen conjuntos de bloques que no se pueden seleccionar juntos o por si solos. Un ejemplo de esto son los bloques de selección que no generan una salida dentro del modelo sino que son una entrada para bloques de la capa donde se encuentran. Otro ejemplo es la existencia de bloques consecutivos que carecen de sentido físico y por ende no es necesario realizar un muestreo de estos para saber que no son una solución válida al problema.

Es importante limitar el espacio de búsqueda a solamente soluciones que sean factibles. Las restricciones definidas en el modelo al momento de muestrear estructuras son las siguientes:

1. **Restricción de contiguidad:** No es posible que dos bloques iguales estén en capas consecutivas. Esto se debe a que dicha estructura carece de sentido físico y por ende una solución de ese estilo no sería útil.

2. **Restricción sobre bloques de selección:** Los bloques de selección no pueden encontrarse en la primera capa, esto se debe a que las entradas de la primera capa son las mismas que entrega un bloque de selección y por ende no es necesario que dichos bloques existan.

3. **Restricción sobre bloques de combinación lineal:** Dicha restricción hace referencia a que los bloques de combinación lineal no pueden estar solos en la última capa del modelo, ya que la red ya tiene considerado un bloque de combinación lineal al final. Además, si dicha capa solamente selecciona un bloque de combinación lineal entonces la misma función se puede regenerar pero con una capa menos.

3.7.4. Algoritmo de Búsqueda Estructural

El algoritmo 3 de búsqueda estructural define su matriz de parámetros estructurales dependiendo de la cantidad de bloques de funciones y capas que de la red. En caso de que solamente exista una entrada la cantidad de bloques de selección se limitarán a solamente al bloque de selección total. Una vez hecho esto los parámetros estructurales se definen en el logit de 0.2 y los bloques de selección de la primera capa se congelan en -10.

Una vez definidas las probabilidades iniciales, se generan M muestras de estructuras candidatas que cumplan con las restricciones previamente descritas. Estas posteriormente son inicializadas como redes poniendo su valor umbral en 10^{-3}, el cual se utiliza cuando esta red sea transformada en una expresión. Los modelos son entrenados en paralelo por 100 épocas, un *batch* size de 128, con regularización $L1$ y $\lambda = 10^{-5}$.

La red es entrenada usando la función de pérdida L_1 y se evalúa el modelo con la función de recompensa 3.16, el mayor valor de validación de cada red es utilizado para optimizar los parámetros estructurales. El uso de *baseline* permite reducir la varianza del estimador de gradiente del modelo y para este caso en particular el *baseline* escogido es el promedio de los valores de recompensa de las redes entrenadas en paralelo. A una corrida de este tipo se le denomina una iteración.

El SSA utiliza en su implementación 500 iteraciones, cuando esto ocurre la probabilidad de salida de los parámetros es cortada por un umbral de 0.8 haciendo que todos los bloques que tengan una probabilidad igual o mayor a esta sean considerados en la etapa posterior. Es importante destacar que durante las iteraciones la validación de la red se realiza utilizando las salidas del modelo y no las expresiones generadas. Esto se debe a que como el SSA entrena cada modelo por un periodo de tiempo más acotado los bloques de potencia no tendrán un valor cercano a los extremos uno o cero y por ende los estos tendrán una desventaja con respecto a los otros bloques debido a la aproximación continua que se hace.

Algorithm 3 Algoritmo de Búsqueda estructural (SS).

$\text{prob}_{min} = 1/Nblocks$ ▷ Set min prob
$\text{prob}_{max} = (1 - 1/500)$ ▷ Set max prob
$\gamma = 1_{(\text{Nblocks, Nlayers})} \cdot \text{logit}(\text{prob}_{\text{min}})$
for i in range(Itr) **do**
  $BD = Bernoulli(\gamma)$
  sampledGates = $list()$
  sampleGate = $BD.Sample()$
  for k in len(Networks) **do**
    while sampleGate is Res or Empty **do**
      sampleGate = $BD.Sample()$
    end while
    sampledGates.append(sampleGate)
  end for
  for i in len(Networks) **do**
    aNetwork[i].mask(sampledGates[i]) ▷ init W
  end for
  networks.train(epchs)
  $V_{loss} = networks.eval(valDset)$
  $target = R(V_{loss})$
  $baseline = EMA.update(target)$ ▷ update Exponential Moving Average
  $loss = target - baseline$
  $\gamma.update(loss * (SampledGates - \sigma(\gamma)))$
  $clip(\gamma, \text{min}{=}prob_{min}, \text{max}{=}prob_{max})$
end for
$\gamma_f = \sigma(\gamma) > 0.8$
return γ_f

3.7.5. Métricas

Los resultados obtenidos serán medidos utilizando las métricas NRMSE, R2 y tiempo de cómputo. Esto permitirá tener una consideración de que tan adecuado es el ajuste del modelo sin importar el problema que se esté resolviendo y los tiempos de cómputo permitirán compararlo con el estado del arte en términos de velocidad.

3.7.6. Modelos Para Comparar

En la última competencia de regresión simbólica basada en el benchmark SRbench [12], los métodos que obtuvieron de los primeros fueron feyn[42], pysr[43] y pstree [10]. Dichos modelos son librerías consolidadas de python y en específico feyn es el algoritmo de regresión simbólica que utiliza una empresa privada para la resolución de problemas. Se estima que los mejores métodos para compararse son los mencionados anteriormente, de manera de tener una comparación fehaciente con el estado del arte y su desempeño en el conjunto de datos Nguyen y del modelo de baterías.

El modelo feyn fue presentado por primera vez en [42], dicho modelo fue desarrollado por una empresa llamada Abzu. El método utiliza Qlattice que es un ambiente de simulación

[10] https://pypi.org/project/pstree/

de grafos desde múltiples entradas a una salida, donde el método feyn utiliza una versión diferente de *backpropagation* para optimizar sus parámetros [44]. De este modo el algoritmo logra aprender la representación correcta para resolver el problema de regresión simbólica.

Los modelos de pysr y pstree son dos modelos que se encuentran en librerías de python establecidas, que encuentran mediante algoritmos genéticos soluciones a los problemas de regresión simbólica. Pysr [45] permite separar un problema de regresión simbólica en varios subproblemas mediante *Graph Neural Networks*, la red permite dividir el problema en distintos nodos (o subproblemas) que deben ser solucionados con algoritmos genéticos. De este modo, los problemas se simplifican y la solución es la composición de la solución de cada sub problema.

Se utilizarán dichos modelos como comparaciones del método propuesto ya que estos tuvieron buenos resultados en la competencia del SR del 2022 y son métodos en constante mejora, por lo que podemos estar seguros que se han seguido optimizando.

3.8. Limitaciones del Método

El modelo propuesto presenta ciertas limitaciones, las cuales están contempladas en la estructura generada, las que son explicadas en este apartado. En primer lugar el modelo no contempla expresiones que tengan la estructura x^y, dicha decisión se debe a la complejidad que conlleva el desarrollo de una función potencia continua cuyo exponente sea un valor real. Ahora, ecuaciones que tengan la estructura anteriormente mencionada son mucho más complejas y no se ajustan a la solución diseñada, por lo que no se abordarán en esta versión del modelo.

El este modelo de red propuesto es entrenado mediante *backpropagation*, esto hace que sea muy propenso a encontrar mínimos locales que se ajustan de manera correcta a la curva pero no sean las soluciones finales que se están buscando. Las combinaciones lineales son procesos donde los mínimos locales son muy fáciles de encontrar y por ende es deseable que en un modelo de red exista la menor cantidad de capas de combinación lineal. Por esta razón una de las limitantes del método propuesto es el ajuste a soluciones que tengan una gran cantidad de combinaciones lineales en su expresión final. Esto se debe a que el modelo propuesto está diseñado como una primera aproximación al problema de regresión simbólica utilizando redes neuronales dedicadas y por ende no es una prioridad de este modelo resolver problemas complejos de combinaciones lineales, sino que su objetivo final es desarrollar un método que permita realizar regresión simbólica.

Otra limitación del método propuesto tiene relación con la cantidad de capas del modelo, en general los modelos de redes neuronales tienen la posibilidad de apilar varias capas dentro de una misma red. En el modelo de red propuesto es factible apilar varias capas, pero no es recomendable la razón tiene relación con el punto anterior, como muchos de los bloques de funciones contienen combinaciones lineales a medida que se van apilando más y más capas, la cantidad de entradas a las capas posteriores son cada vez mayores, dicha situación sumada al efecto de la combinación lineal hace que las expresiones generadas carezcan de representatividad ya que contienen demasiados términos. Esta limitación se une a la expuesta anteriormente, ya que el modelo propuesto de red no está diseñado para encontrar soluciones que tengan grandes sumatorias compuestas con funciones no lineales y como se explicó antes dicha limitación tiene que ver con el método de optimización de la red más que con la estructura de la misma.

En relación a las limitaciones existentes en el algoritmo SS, la principal limitante tiene relación

con la paralelización. La paralelización se realiza con cores de CPU y no por GPU, esto hace que la cantidad de modelos que se pueden correr en paralelo se encuentre limitada por el número de cores que tenga el computador y no así por su memoria. La razón principal para elegir esta paralelización se debe a que cada modelo entrenado en paralelo en el SS es independiente del resto, por lo que la solución más simple era su entrenamiento por CPU. Dicha decisión no condiciona los tiempos de cómputo ya que el modelo en si no tiene más de 50 parámetros, esto permite que las multiplicaciones en CPU sean rápidas. Por último, una de las prácticas más comunes en modelos de redes neuronales es la normalización de las entradas del modelo e incluso capas de *batch normalization*. En el modelo propuesto no es posible realizar dichos procesos ya que esto afectaría el valor de las entradas, en específico a su representatividad a la hora de encontrar las expresiones que generen los datos. Esto produciría que la expresión encontrada no tenga relación alguna con la función real que se busca encontrar utilizando el modelo, por lo que se perdería la representatividad que es fundamental en un modelo de aprendizaje de máquinas interpretativo.

4. Resultados

En este capítulo se presentan los resultados obtenidos al entrenar el modelo propuesto en dos conjuntos de datos distintos. El capítulo empieza introduciendo ambos experimentos para después mostrar los resultados encontrados tanto desde el punto estructural como desde el ajuste de parámetros.

4.1. Experimentos

4.1.1. Nguyen

Nguyen es un conjunto de prueba para métodos de regresión simbólica. Dicho conjunto de datos fue utilizado en [7], siendo ésta la principal razón para su uso. Además, el modelo propuesto en este trabajo de tesis se encuentra limitado a máximo tres variables, por lo que Nguyen es una buena forma de medir el desempeño del modelo generado ya que presenta funciones de hasta dos variables. Los problemas que dicho conjunto de datos contiene se detallan en la figura 4.1.

Name	Variables	Expression	Dataset	Constant?
Nguyen-1	1	$x^3 + x^2 + x$	U$(-1, 1, 20)$	No
Nguyen-2	1	$x^4 + x^3 + x^2 + x$	U$(-1, 1, 20)$	No
Nguyen-3	1	$x^5 + x^4 + x^3 + x^2 + x$	U$(-1, 1, 20)$	No
Nguyen-4	1	$x^6 + x^5 + x^4 + x^3 + x^2 + x$	U$(-1, 1, 20)$	No
Nguyen-5	1	$\sin(x^2)\cos(x) - 1$	U$(-1, 1, 20)$	No
Nguyen-6	1	$\sin(x) + \sin(x + x^2)$	U$(-1, 1, 20)$	No
Nguyen-7	1	$\log(x + 1) + \log(x^2 + 1)$	U$(0, 2, 20)$	No
Nguyen-8	1	$\sqrt{x}$	U$(0, 4, 20)$	No
Nguyen-9	2	$\sin(x) + \sin(y^2)$	U$(0, 1, 20)$	No
Nguyen-10	2	$2\sin(x)\cos(y)$	U$(0, 1, 20)$	No
Nguyen-12	2	$x^4 - x^3 + \frac{1}{2}y^2 - y$	U$(0, 1, 20)$	No
Constant-1	1	$3.39x^3 + 2.12x^2 + 1.78x$	U$(-1, 1, 20)$	Yes
Constant-2	1	$\sin(x^2)\cos(x) - 0.75$	U$(-1, 1, 20)$	Yes
Constant-3	2	$\sin(1.5x)\cos(0.5y)$	U$(0, 1, 20)$	Yes

Figura 4.1: Conjunto de funciones Nguyen, junto con información de cada problema.

En la figura 4.1, cada uno de los problemas tiene 20 muestras. Este número fue modificado

a 1000 de modo de tener una cantidad de datos suficiente para poder entrenar correctamente un modelo de red neuronal. El resto de las definiciones de cada problema se mantiene.

4.1.2. Modelo de Baterías

Con el objetivo de buscar un mejor diseño de empaquetamiento de las baterías de ion-litio, el que permita optimizar el comportamiento térmico de estas, es necesario modelar, de manera lo suficientemente certera, el comportamiento de las baterías. Es por esto que, a partir del *software* de simulación ANSYS®, se obtuvieron datos y luego se diseñó un modelo fenomenológico [46], el cual actualmente entrega diferencias significativas con respecto al *software*. Con el fin de mejorar el desempeño de este modelo, se busca optimizar las expresiones de 3 expresiones de interés (Coeficiente de arrastre, Número de Nusselt y Factor de fricción) que influyen directamente en las salidas. La tabla 4.1 presenta las entradas de dichas expresiones.

Tabla 4.1: nombre y rango de las variables utilizadas en el conjunto de datos de baterías de litio.

Nombre	**rango**
Número de Reynolds (Re)	[574.18, 11015]
Factor de espaciado de celdas (S)	[0.3, 1.5]
Número de Prandtl (Pr)	[0.705, 0.71]

A modo de referencia, para este modelo la última solución aceptada de estos parámetros de interés son las soluciones expuestas en las ec. 4.1, 4.2 y 4.3. Estas soluciones fueron encontradas en [47]. Dichas soluciones se utilizan como referencia para determinar si la estructura generada por el modelo es o no correcta.

$$c_d = S^{0.6} + 5Re^{-0.23} \tag{4.1}$$

$$f_D = 20S^{-1.1}Re^{-0.22} \tag{4.2}$$

$$N_u = 0.5S^{-0.2}Re^{0.63}Pr \tag{4.3}$$

4.2. Resultado

En este capítulo se muestran gráficos tanto de selección de bloques como de ajuste de parámetros. Por esta razón la tabla 4.2 resume las leyendas en los gráficos de búsqueda estructural.

Tabla 4.2: Glosario de leyendas en gráficos.

Nombre en gráfico	**Nombre del bloque**
mvpower	potencia multivariable
mvexp	exponencial multivariable
mvsin	seno multivariable
mvlinear	función lineal multivariable
pwpower	potencia punto a punto
mvlog	función Logaritmo Multivariable
AllSelection	Selección Total
LcSelection	Selección por combinación Lineal
OneSelection	Selección única

4.2.1. Nguyen

La tabla 4.3 resumen de los resultados encontrados al correr el método de regresión simbólica en el conjunto de datos Nguyen.

Tabla 4.3: Comparación de resultados de la ecuación real con la predicha por el modelo propuesto SRNet.

Nombre	**Entradas**	**Real**	**Modelo SRNet**
Nguyen-1	1	$x^3 + x^2 + x$	$x^1 + x^2 + x^3$
Nguyen-2	1	$x^4 + x^3 + x^2 + x$	$x^1 + 0.9x^{1.9} + x^3 + 1.1x^{3.9}$
Nguyen-3	1	$x^5 + x^4 + x^3 + x^2 + x$	$1.2x^{1.1} + 1.8(x)_a^{4.2} + x^2 + x^4$
Nguyen-4	1	$x^6 + x^4 + x^3 + x^2 + x$	$x + 1.2(x)_s^{2.1} + x^3 + 1.8(x)_s^{5.2}$
Nguyen-5	1	$sin(x^2)cos(x) - 1$	$-sin(\pi(0.4x + 0.1x_s^{2.8} + 0.2)) + 0.6sin(\pi(0.6x + 0.2x_s^{2.8} - 0.1)) - 0.2$
Nguyen-6	1	$sin(x) + sin(x + x^2)$	$1.8\sin(\pi\,(0.4x + 0.2x_s^{2.1}))$
Nguyen-7	1	$log(x + 1) + log(x^2 + 1)$	$0.6x(sin(-\pi(0.2x + 0.1)))^{0.2} + 1.3sin(0.2\pi x)^{0.9})x_s^{0.3}$
Nguyen-8	1	$\sqrt{x}$	$\sqrt{x}$
Nguyen-9	2	$sin(x) + sin(y^2)$	$0.3x + 0.9x_a^{1.7} + 0.6sin(0.4\pi x)$
Nguyen-10	2	$2sin(x)cos(y)$	$0.8x - 0.3sin(\pi(0.5x - 0.5y)) - 0.8sin(\pi(0.3x + 0.3y - 0.9)) - 0.2$
Nguyen-12	2	$x^4 + x^3 + 0.5y^2 - y$	$0.1x - 0.9y + 0.6(x)_a^{3.8} - x_s^{2.2} + 0.5(y)_s^{2.1} - 0.1sin(\pi(0.4x + 0.3y + 0.1))$
Constant-1	1	$3.39x^3 + 2.12x^2 + 1.78x$	$3.1(x)_a^{3.2} + 2.1x^2 + 2.1(x)_a^{1.1}$
Constant-2	1	$sin(x^2)cos(x) - 0.75$	$-0.8sin(\pi(0.4x_s^{1.5} + 0.2)) + 0.6sin(\pi(0.7x_s^{1.5} - 0.1) - 0.1$
Constant-3	2	$sin(1.5x)cos(0.5y)$	$0.4sin(pi(0.5x - 0.2y)) + 0.6sin(\pi(0.5x + 0.1y))$

Tabla 4.4: Métricas obtenidas al correr el modelo SRNet diez veces en el conjunto de datos Nguyen.

Nombre	NRMSE	R2	Tiempo [s]	Mejor [NRMSE]
Nguyen-1	0.014±0.015	1.000±0.001	779.043±5.118	0.000
Nguyen-2	0.007±0.005	1.000±0.000	766.868±5.781	0.002
Nguyen-3	0.004±0.001	1.000±0.000	737.712±4.201	0.003
Nguyen-4	0.002±0.000	1.000±0.000	727.843±1.304	0.002
Nguyen-5	0.169±0.054	0.968±0.021	868.095±3.607	0.110
Nguyen-6	0.030±0.008	0.999±0.000	453.927±2.105	0.013
Nguyen-7	0.058±0.052	0.994±0.010	874.009±6.213	0.015
Nguyen-8	0.000±0.000	1.000±0.000	644.466±1.513	0.000
Nguyen-9	0.057±0.021	0.996±0.003	930.542±4.852	0.027
Nguyen-10	0.094±0.068	0.987±0.020	736.982±7.617	0.041
Nguyen-12	0.098±0.020	0.990±0.004	748.219±6.950	0.066
Constant-1	0.006±0.001	1.000±0.000	781.253±2.436	0.004
Constant-2	0.146±0.077	0.973±0.020	866.834±3.562	0.022
Constant-3	0.063±0.037	0.995±0.008	694.055±2.938	0.047

4.2.1.1. Búsqueda Estructural

Los resultados obtenidos con el *benchmark* Nguyen son presentados en dos partes, primero se analizan los resultados del algoritmo de búsqueda estructural en este conjunto de datos. Segundo, se realiza un análisis de las ecuaciones generadas y su comparación con los datos reales del conjunto de testeo.

4.2.1.1.1. Algoritmo de Búsqueda Estructural

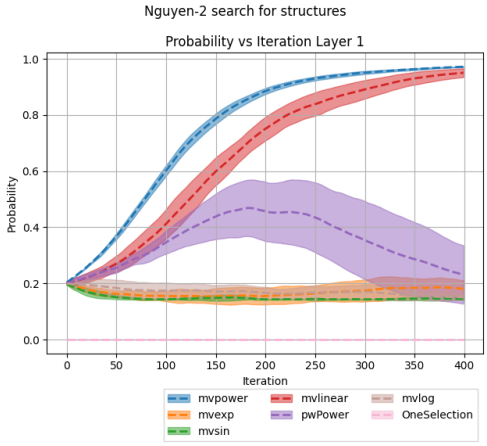
En la tabla 4.4 se presenta un resumen de las mejores métricas encontradas para cada uno de las ecuaciones que constituyen el *benchmark* Nguyen. Para este análisis hay dos casos que nos interesa abordar, el primero es el caso de las soluciones con estructuras finales correctas y el segundo aquellos casos donde los modelos no pudieron ajustar correctamente la estructura al problema requerido.

La decisión sobre si una estructura es correcta o no, no se puede definir a priori, sino que esta cualidad solo es posible definirla con un conjunto de datos particular el cual tenga las expresión reales para poder contrastar. Aquí se define que una estructura es correcta si y solo si dicha estructura tiene el potencial de realizar todas las expresiones que se encuentran en la expresión final. El estudio de estructuras correctas y no correctas permite que podamos verificar una de las hipótesis principales del trabajo de tesis, que es la facilidad que tienen estructuras correctas para ajustarse mejor a las curvas de entrenamiento.

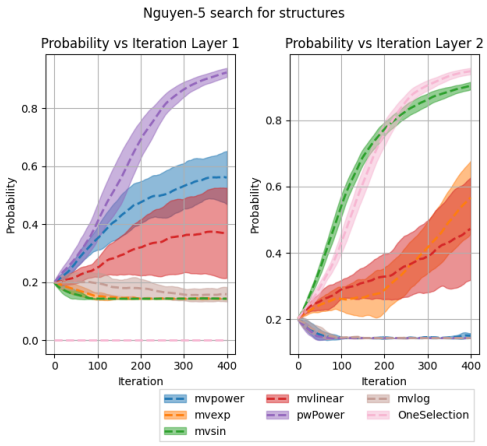
De esta manera, lo que se busca con este análisis es medir que tan alta es la probabilidad de ocurrencia cuando existen funciones que no son parte de la estructura *real* de la expresión buscada y verificar así la hipótesis de esta tesis.

4.2.1.1.2. Estructuras Correctas

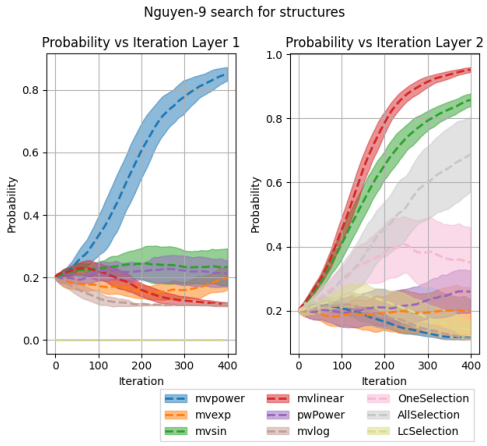
El caso óptimo es que el algoritmo de búsqueda estructural devuelva como resultado la estructura correcta del modelo, pero en la práctica lo que se busca con este algoritmo no es que dicha estructura sea exactamente seleccionada, sino que basta conque la estructura correcta sea un subconjunto de las estructura seleccionadas mediante el algoritmo de búsqueda estructural. Esto permite reducir el espacio de búsqueda de esta etapa a que dentro de las estructuras que forman parte del entrenamiento final esté la estructura correcta. Para hacer dicho análisis de estructuras utilizaremos como ejemplo algunos de los resultados correctos de los problemas 2,5,9, 10 del conjunto de datos Nguyen que se ilustran en la figura 4.2.



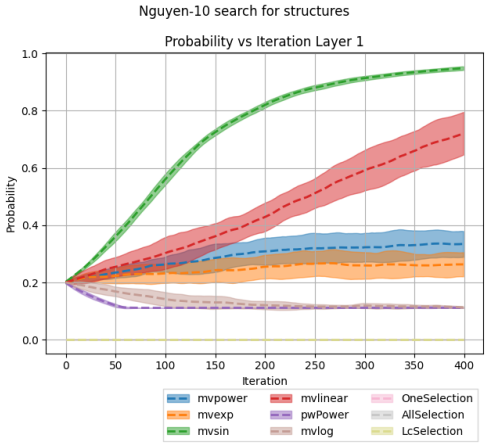
(a) Probabilidad Nguyen-2



(b) Probabilidad Nguyen-5



(c) Probabilidad Nguyen-9



(d) Probabilidad Nguyen-10

Figura 4.2: Búsqueda Estructural en el Dataset Nguyen. Cada experimento fue corrido 10 veces para mostrar la robustez del método. En las figuras a, b,c,d se muestran los resultados de las corridas realizadas sobre los problemas Nguyen-2, Nguyen-5, Nguyen-9 y Nguyen-10 respectivamente.

Al comparar las estructuras obtenidas en la Fig. 4.2 con las expresiones reales que se encuentran en la tabla 4.3, se nota lo siguiente: en primera instancia la probabilidad de los bloques predominantes en la expresión converge de manera rápida y en general con baja desviación estándar, mostrando de este modo que existe una convergencia en estos casos hacia una estructura superior a las otras muestreadas durante el entrenamiento estructural. Se puede observar además que las estructuras con mayor desviación estándar, son de igual manera parte fundamental de la ecuación, pero que el valor añadido que generan en la aproximación del modelo no es tan relevante como el generado por los bloques de menor desviación estándar. Además, existen casos en que es posible generar la misma función con otras estructuras, lo cual puede causar cierta complicación al momento de hacer la selección para cada capa.

Un ejemplo de lo anteriormente mencionado es el caso del gráfico 4.2.b donde la función buscada tiene los términos x y x^2, y se observa que los bloques con menor desviación estándar son los de selección total y seno. Esto se debe al rango de las entradas al modelo, como dichos valores se encuentran entre menos uno y uno claramente el valor de x representa en mayor

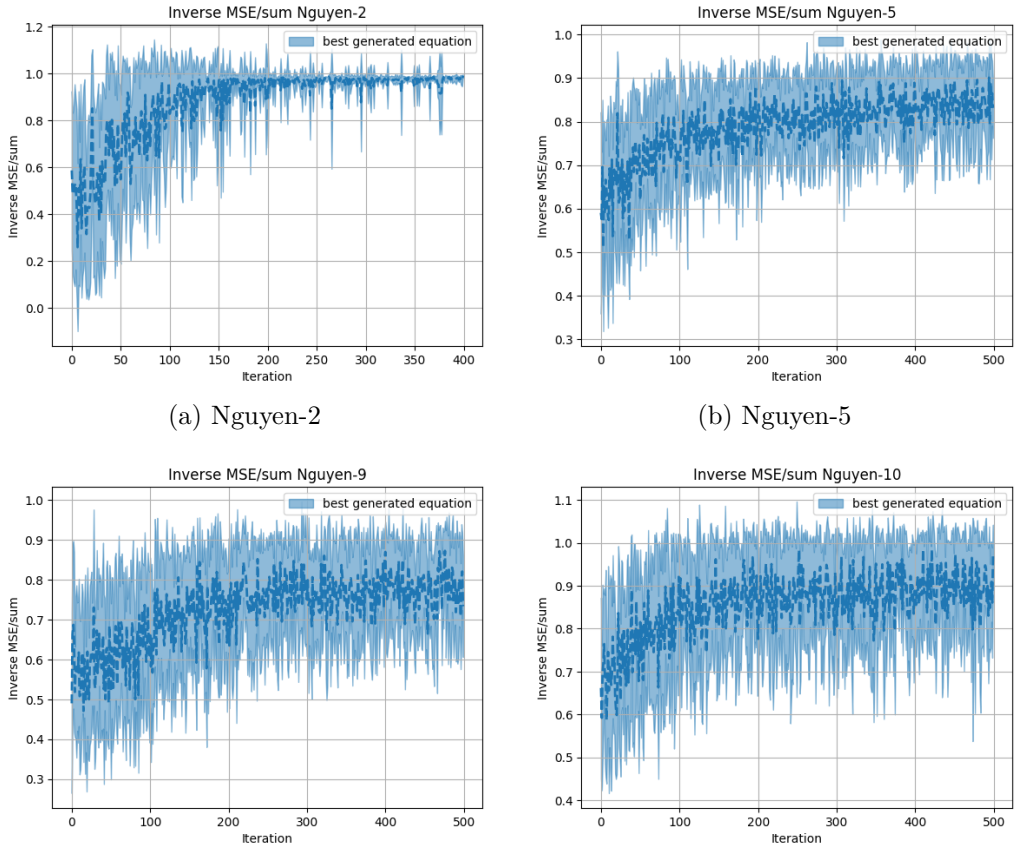
parte la estructura de la solución. De este modo, se observa que al momento de seleccionar polinómios de mayor orden como es x^2 tanto la potencia punto a punto como la potencia multivariable empiezan con la misma fuerza a ser seleccionados por el algoritmo y pasada la mitad de las iteraciones, el bloque de potencia punto a punto es seleccionado por sobre la potencia multivariable. La razón de porque no se seleccionan ambos bloques se debe al efecto que estos generan en la expresión. Como el bloque de potencia punto a punto y el bloque de potencia multivariable se complementan no pueden ser ambos seleccionados ya que cambia mucho la solución encontrada. Además es importante considerar que por la existencia de un bloque de selección total, de elegirse los tres bloques el polinomio generado partiría de x^3 y es por esto que una vez pasado el 50 % de probabilidad del bloque de selección, la probabilidad del bloque de potencia multivariable empieza a decaer.De este modo, se puede observar por esta razón el modelo generado no selecciona más bloques de la cuenta en este caso.

Un caso un poco diferente es el de la figura 4.2.a, el cual al ser un polinomio con entradas entre -1 y 1, los términos de alta potencia se hacen cada vez más pequeños, haciendo por ende que estos tengan cada vez menor relevancia al momento de reconstruir la curva. De este modo, se observa que los términos más relevantes son los dos primeros términos del polinomio x y x^2, x^3 que son justamente los términos que se pueden reconstruir con dicho bloque. Además, se observa como la probabilidad de selección del bloque de potencia punto a punto decae cuando la potencia multivariable es elegida con un 80 % de probabilidad.

En la figura 4.2.c,se puede ver que al momento de seleccionar los bloquees que tienen un umbral mayor al 80 % se elige un bloque de combinación lineal en la segunda capa. A pesar de que el modelo propuesto hace dicha selección, la selección realizada si es correcta, esto ya que en la etapa posterior a la selección de bloques se entrenarán todas las combinaciones posibles de redes generadas con este subconjunto de bloques y dentro de dicho subconjunto se encuentra el conjunto adecuado para la resolución del problema, es importante notar también que como el rango de las entradas es entre -1 y 1 los bloques lineales tienen una semejanza a la función seno, por lo que no es de extrañar dicho efecto.

Con lo expuesto anteriormente, se puede indicar que mediante el algoritmo de búsqueda estructural es posible encontrar una familia de soluciones las cuales pueden aproximar correctamente las expresiones buscadas. A pesar de que la estructura sea correcta, dicha estructura no necesariamente asegura una convergencia a la solución buscada, esto se debe a que el modelo de red al utilizar *backpropagation* es propenso en muchos casos a quedarse en mínimos locales.

Una manera de observar la mejora de las estructuras generadas y así verificar que el modelo está generando un aprendizaje del entrenamiento estructural se ilustra en la Fig. 4.3 la gráfica de la función objetivo (función 3.16), que evidencia como a medida que avanzan las iteraciones, las estructuras generadas mejoran constantemente. Los gráficos de la Fig. 4.3 muestran como mejora de función objetivo por la que pasan los problemas que obtuvieron una estructura correcta.



(a) Nguyen-2

(b) Nguyen-5

(c) Nguyen-9

(d) Nguyen-10

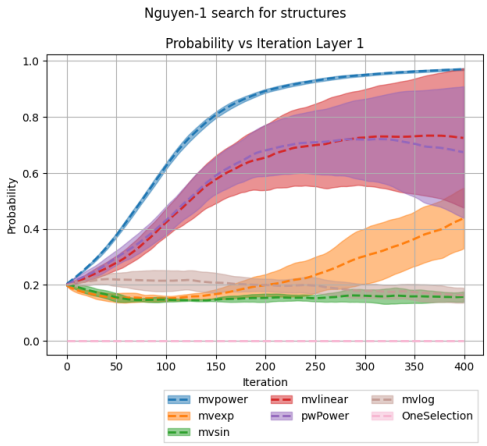
Figura 4.3: Evolución de la función objetivo del entrenamiento estructural en el Dataset Nguyen, Cada experimento fue corrido 10 veces para mostrar la robustez del método. En las figuras a, b,c,d se muestran los resultados de las corridas realizadas en los problemas Nguyen-2, Nguyen-5, Nguyen-9 y Nguyen-10, respectivamente.

Como se observa en la figura 4.3 todas las estructuras que fueron encontradas correctamente mediante el algoritmo de búsqueda estructural lograron una mejora de más del 20 % si se compara el valor máximo (de la media) alcanzado y el valor mínimo (de la media) con el cual se inició. Se observa que a medida que avanza el entrenamiento en el caso de la búsqueda estructural exitosa se obtienen mejoras considerables en el valor de la función objetivo, a través de las diez corridas generadas por cada expresión. Este resultado indica que el método genera mejoras en las estructuras muestreadas de manera robusta y además genera mejoras significativas en su desempeño.

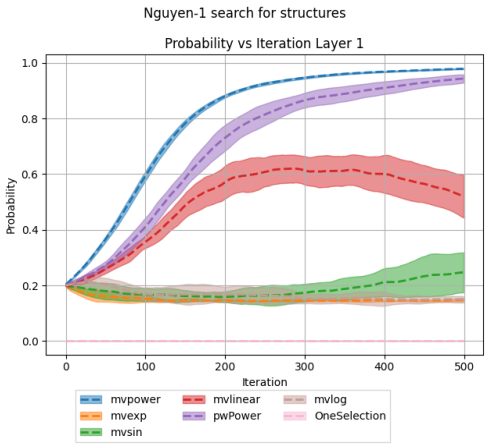
4.2.1.1.3. Efecto de los Hiperparámetros del SSA

El algoritmo se corre 10 veces para así asegurar la robustez del método generado. Uno de los casos en que el modelo entregó resultados diferentes en distintas corridas es el encontrado en el experimento Nguyen-1. Las curvas de probabilidad se pueden observar en la figura 4.4.a

Debido a esto, se consideró que era posible que los hiperparámetros del problema estuvieran afectando la convergencia del algoritmo para las 10 corridas y se decidió correr el algoritmo aumentando el valor de $\alpha = 0.5$, aumentando el *batch size* a 256 y bajando las épocas de entrenamiento a 50. Los resultados encontrados se pueden observar en la figura 4.4.b



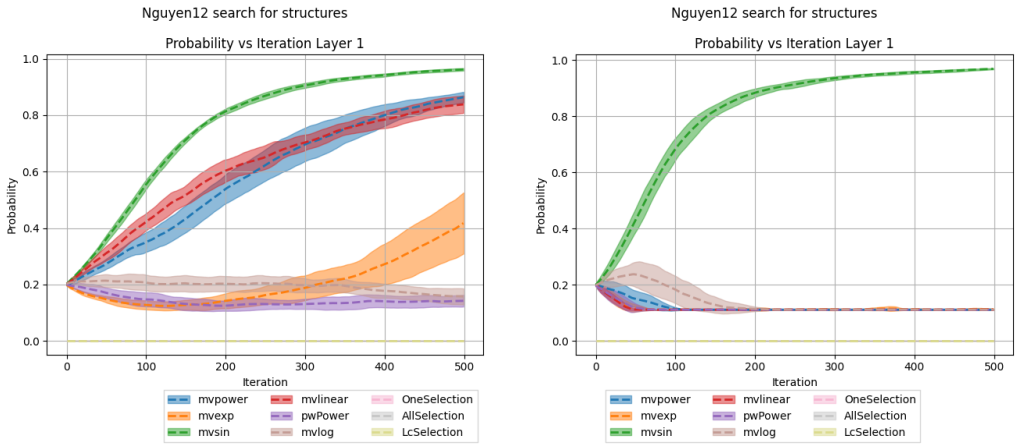
(a) Nguyen-1 obtenido corriendo el algoritmo normal.



(b) Nguyen-1 variando los hiperparámetros del SSA

Figura 4.4: Evolución de las probabilidades encontradas al momento de correr el SSA sobre el experimento Nguyen-1. La figura (a) es el resultado al correr el modelo bajo los hiperparámetros por default, mientras que la figura (b) es el resultado del algoritmo variando los hiperparámetros. Él algoritmo encontró la estructura correcta pero no en todos los casos generados, ya que en ciertas situaciones dejó fuera la parte lineal de la solución.

En este caso, es posible observar que cambiando los hiperparámetros del problema los resultados encontrados para Nguyen-1 mejoran. Realizar dicho cambio no generó mejoras en todos los experimentos, y en algunos casos empeoró la solución, como por ejemplo en el experimento Nguyen-12, el cual al verse afectado por un mayor α no sostuvo la estructura polinomial debido a la cantidad de parámetros que esta contieneEsto se ilustra en la Fig. 4.5.



(a) Nguyen-12 obtenido corriendo el algoritmo normal.

(b) Nguyen-12 variando los hiperparámetros del SSA.

Figura 4.5: Evolución de las probabilidades encontradas al momento de correr el SSA sobre el experimento Nguyen-12. La figura (a) es el resultado al correr el modelo bajo los hiperparámetros por default, mientras que la figura (b) es el resultado del algoritmo variando los hiperparámetros. En este caso se puede ver como el algoritmo encuentra la estructura correcta solamente en la figura (a), mientras que en la figura (b) dichos resultados no corresponden con la estructura real de la solución.

La mayoría de los experimentos Nguyen fueron resueltos con los hiperparámetros estándar pero esta regla no funciona para todos los problemas encontrados. En los casos de Nguyen-1 y Nguyen-12 el uso de otros hiperparámetros generar que el modelo retire partes de la estructura que componen la estructura correcta para solucionar el problema. En caso de que el modelo no logre funcionar, es posible ver si dichos resultados mejoran aplicando un cambio en hiperparámetros.

4.2.1.1.4. Estructuras Incorrectas

Como se indicó anteriormente el método de *backpropagation* al tener asegurada una dirección de mejora hacia un mínimo local o un mínimo global, en muchos casos genera estructuras suboptimas que permiten aproximar la función lo suficientemente bien dentro de un rango definido. En estos casos además se vuelve importante la cantidad de muestreos que presentan las estructuras y que tanto mejor es la solución encontrada con, respecto a las otras. En casos donde existen estructuras suboptimas que representan de manera correcta la curva no se percibirá la mejora debido al efecto del *baseline* en el modelo.

Habiendo descrito las limitantes que sufre el proceso de búsqueda estructural en la sección 3.8 se hará el mismo análisis realizado en la sección anterior, para así esclarecer las diferencias que existen en el proceso cuando la estructura converge correctamente y cuando no. Para esto se muestran los resultados del problema Nguyen-7 en la Fig. 4.6.

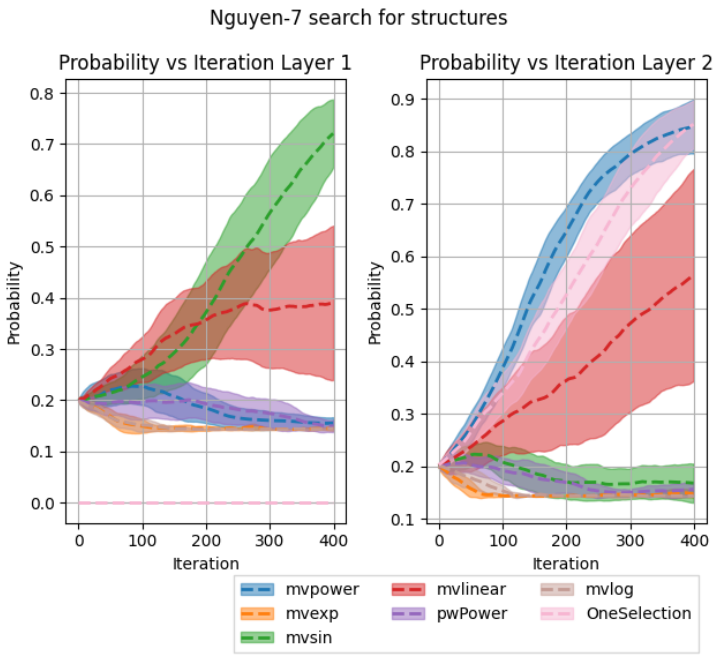


Figura 4.6: Búsqueda Estructural Dataset Nguyen, en esta gráfica cada experimento fue corrido 10 veces para mostrar la robustez del método. En este caso se observan los resultados en el problema Nguyen 7, en dicha imagen se puede observar como el modelo elige en la primera capa una estructura de seno, mientras que en la segunda se escoge una potencia y un bloque de selección. Dichas elecciones son erróneas observando el resulado esperado en la tabla 4.3, ya que en la primera capa debieron seleccionarse bloques de potencia y en el segundo bloques de logaritmo.

Los resultados de la Fig. 4.6, muestran como según el modelo encontrado la estructura correcta es una función trigonométrica elevada a algún exponente. Dicho resultado es incorrecto para el problema Nguyen-7, pero cuando se observa el error generado por dicho modelo en la tabla 4.4, se tiene que la estructura generada tiene métricas que hacen pensar que la solución generada se trata de un mínimo local que fue encontrado por la red.

Otra faceta importante del entrenamiento a considerar se encuentra en la evolución de la función objetivo durante el entrenamiento del SS. Como se observó en la sección 4.2.1.1.2, cuando la estructura es correcta la mejora de la función objetivo es mayor al 20 %. Si se observan las estructuras incorrectas encontradas se tiene que ninguna de las evoluciones de la función objetivo superó el 20 %, siendo incluso en muchos casos casi nula la mejora como se muestra en la Fig. 4.7. Esto es un indicador claro de que la solución encontrada mediante el algoritmo de búsqueda estructural no ha encontrado una solución satisfactoria. Esto se justifica en que al no ser la estructura real del problema, las soluciones que encuentra son soluciones sub-optimas que no permiten que la función objetivo mejore continuamente, debido al hecho de que al no ser una solución global no se puede aproximar todo lo que se quisiera a la curva buscada.

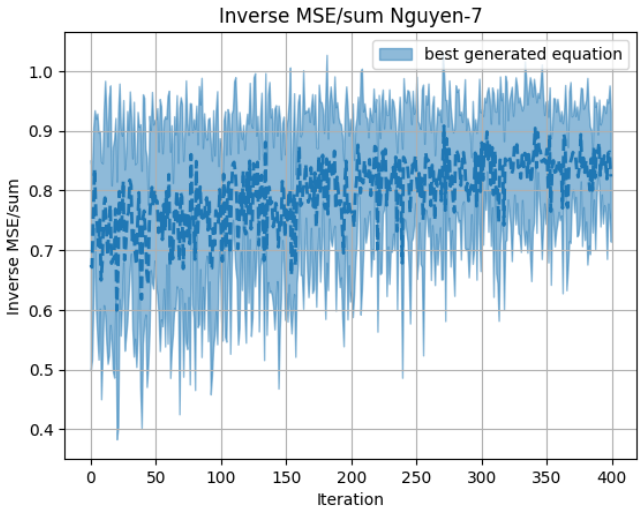


Figura 4.7: Evolución función objetivo en problemas del Dataset Nguyen, en esta gráfica cada experimento fue corrido 10 veces para mostrar la robustez del método

4.2.1.2. Ajuste de Parámetros

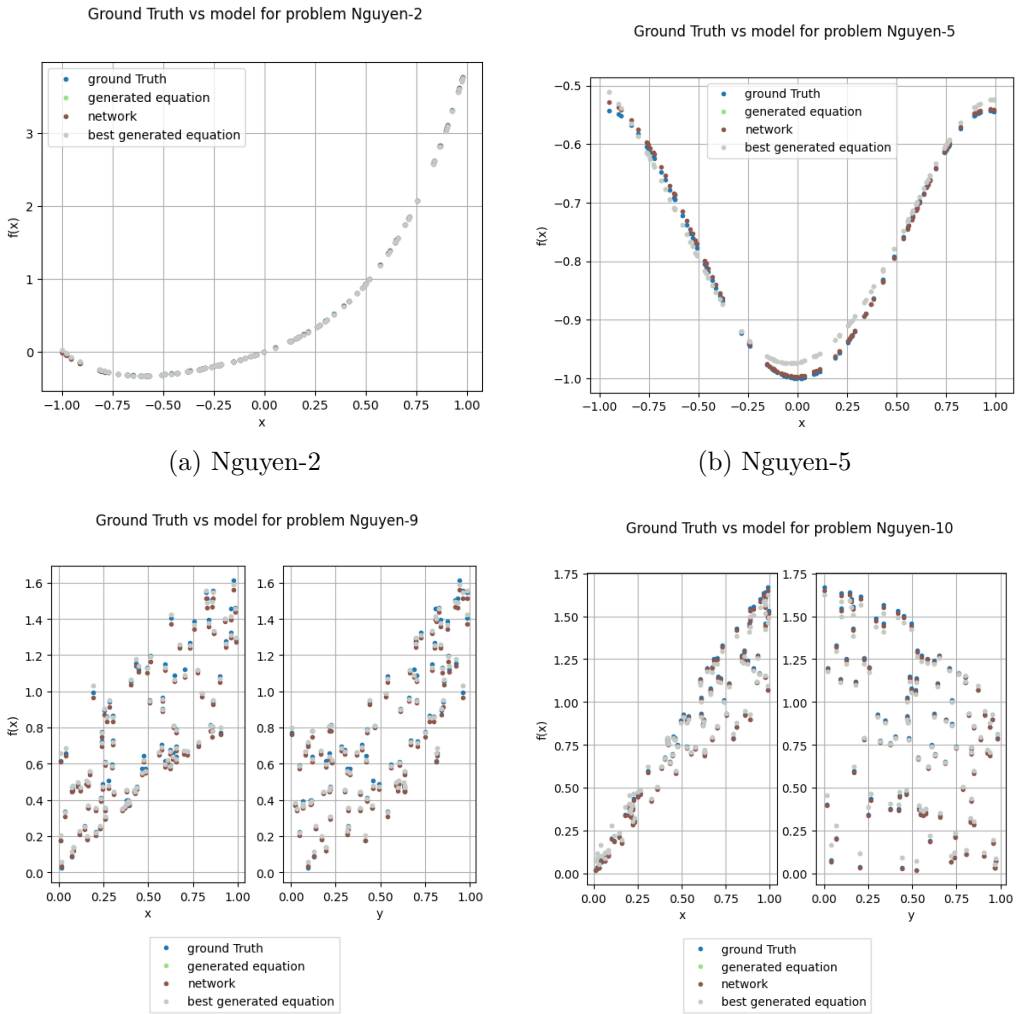
4.2.1.2.1. Ajuste de Expresiones a la Curva

Debido a la estructura del modelo propuesto el problema de ajuste de parámetros es independiente de la búsqueda estructural, este se basa en encontrar los coeficientes correctos una vez que ya se ha encontrado la mejor estructura para solucionar el problema. La principal ventaja que presenta SRNet sobre los métodos de algoritmos evolutivos (pysr, pstree) o modelos como Deep Symbolic Regresion o AI-feynman es que el cambio en las soluciones está guiado por *backpropagation*, esto hace que el ajuste no esté condicionado a la aleatoriedad del método como es el caso de los modelos mencionados, sino que a medida que el modelo es entrenado puede ajustarse cada vez más a la curva.

A pesar de dicha ventaja, existe una limitante al uso de redes neuronales que tiene relación con los mínimos locales y los parámetros continuos. Cuando los parámetros de los modelos son continuos en muchos casos existen soluciones sub-óptimas que se encuentran al utilizar valores continuos en los coeficientes de las expresiones generadas. Al contrario de los métodos anteriormente mencionados, SRNet se entrena mediante *backpropagation*. Cada una de sus funciones tiene por consiguiente una naturaleza continua y por ende se abre el espacio de búsqueda al tipo de soluciones con términos de potencia o coeficientes de punto flotante en los términos de las funciones. Así, se encuentran soluciones como la de Nguyen-3 en la tabla 4.3, donde se observa que existen términos de potencia no enteras mayores a 1, las cuales a pesar de no ser la solución estrictamente correcta del modelo si logran ajustarse de manera correcta dentro del rango.

La figura 4.8 muestra el ajuste que existe en los modelos generados para los mismos problemas expuestos en la figura 4.2. Al ser el modelo generado de tipo IA interpretativo, la única curva generada con redes neuronales es la café que hace referencia a la salida de la red, las curvas en verde y gris son expresiones que se evaluaron en el conjunto de prueba, por lo que dichas expresiones no requieren de una red neuronal para generar las salidas. Esto muestra como las soluciones generadas por el método propuesto si generan un aprendizaje sobre la curva y permiten hacer un ajuste sobre esta sin que al final del método sea necesario un

modelo sino que basta con la expresión generada. Si se observan las soluciones encontradas en la figura 4.2, observa que en el caso de a figura 4.8.a la solución encontrada estructuralmente es la correcta para el problema, pero la solución encontrada no es exactamente la misma que la encontrada en la tabla 4.3. En específico, se puede ver como el modelo hace uso de exponentes no enteros para ajustarse a los datos, siendo esta una solución sub-óptima, pero en la práctica casi igual al resultado real. Por el otro lado, se puede ver como 4.8.b a pesar de tener la estructura correcta tiene un ajuste peor al encontrado en 4.9 que a pesar de no haberse encontrado la estructura real, tiene un ajuste más cercano al resultado. Esto tiene que ver con el hecho de que la función que está en la última capa en Nguyen-5 es la función seno que como indicamos anteriormente su ajuste por *backpropagation* es complejo. Para las soluciones encontradas tanto en 4.8.c y 4.8.d se puede ver que ambas soluciones son bastante cercanas al resultado real, teniendo un ajuste considerablemente mejor que Nguyen-5.



(a) Nguyen-2 (b) Nguyen-5

(c) Nguyen-9 (d) Nguyen-10

Figura 4.8: Ajuste de las ecuaciones a los problemas del Dataset Nguyen, en esta gráfica se sacó el mejor resultado de las 10 corridas que se realizaron del método.

Un punto interesante a considerar es el ajuste de modelos en que el algoritmo SS encontró soluciones incorrectas, esto nos permite analizar que es lo que ocurre con el ajuste de este tipo de estructuras cuando la estructura encontrada no es la adecuada. En la Fig. 4.9 se

observa que la solución encontrada se encuentra cerca de la solución pero tiene ciertas partes con diferencias con respecto a esta, como son el centro y los extremos, de este modo, se cree que debido al acotado dominio de la solución la expresión generada de todos modos logra ajustarse. La solución encontrada es una solución sub-óptima y por ende es un mínimo local.

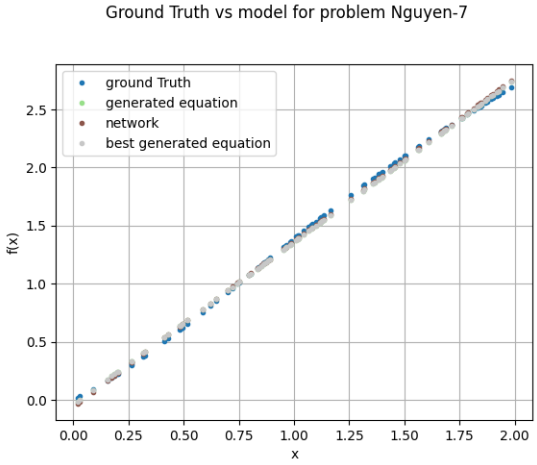


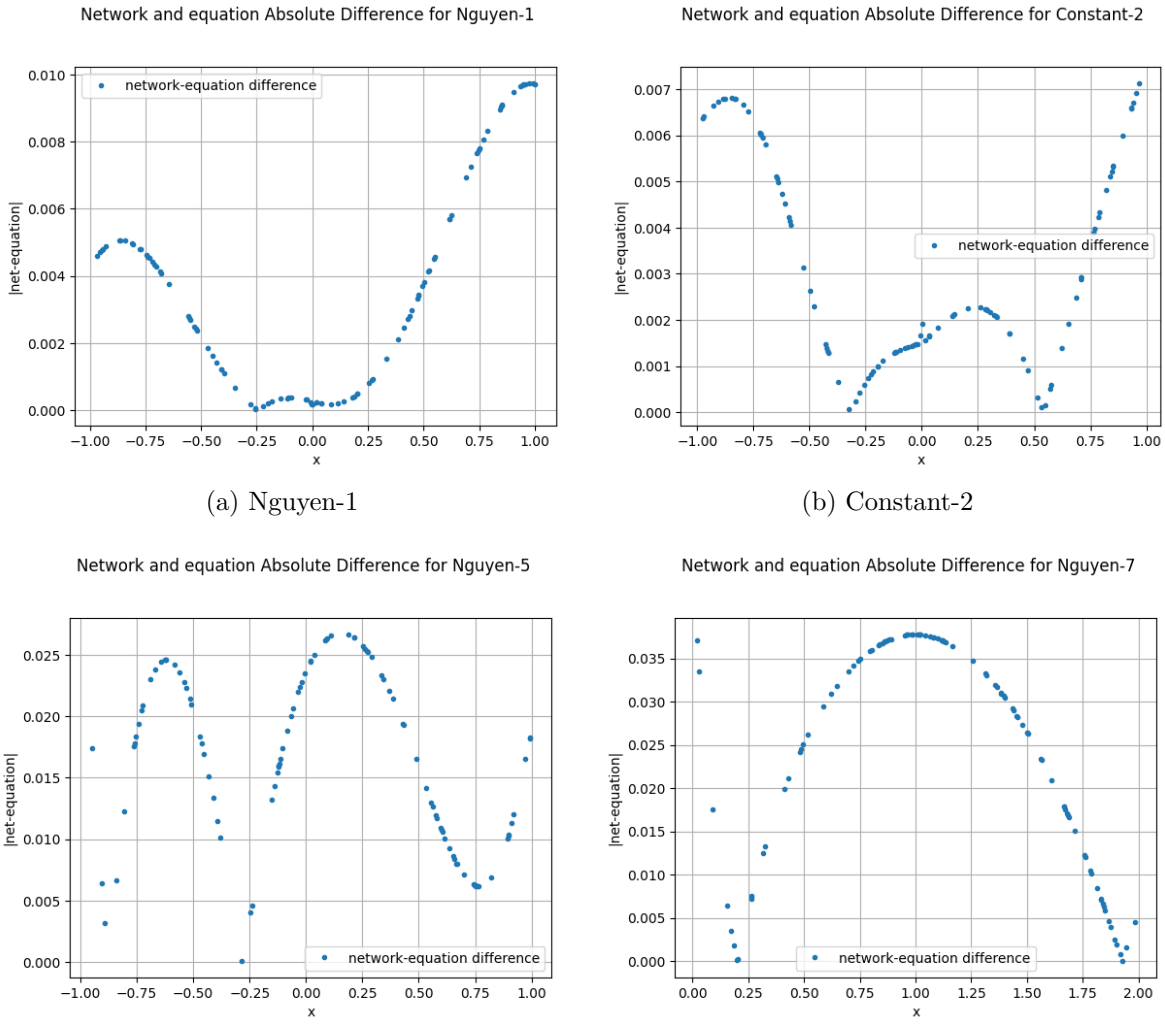
Figura 4.9: Evolución función objetivo en problemas del Dataset Nguyen, en esta gráfica se extrajo el mejor resultado de las corridas realizadas en el experimento Nguyen-7. Para este caso en específico la estructura encontrada no fue la correcta, pero como se puede ver el modelo de red encontró un mínimo local.

4.2.1.2.2. Diferencia entre Expresiones Generadas y Modelo de Red Entrenado

Otro punto a destacar es el efecto del cambio de umbral al momento de generar las estructuras finales del modelo. Dicha diferencia es importante ya que es el paso del modelo de red a la expresión final. Para realizar este análisis se seleccionaron los modelos con estructuras correctas que tuvieran el peor ajuste y estos se compararon con los modelos incorrectos encontrados a través de los experimentos realizados. De esta manera observando la figura 4.10 es simple notar que los modelos con estructuras correctas (Fig 4.10.b, 4.10.c) tienen la nube de puntos a una distancia menor que los modelos con estructuras incorrectas como es el caso de Fig. 4.10.d.

Se puede ver además que dependiendo del ajuste de la solución el efecto del redondeo de los parámetros genera diferentes efectos. Por ejemplo en la figura 4.10 se puede observar que la diferencia entre la curva generada por la red y la expresión recuperada dista de una distancia a lo más de 0.25 para el problema Nguyen-5, dicho efecto se debe a que la solución encontrada no fue la solución que efectivamente reconstruía los datos y por ende el efecto del umbral fue mayor para este caso. Este efecto es menor en el problema Constant-2 que tuvo un ajuste mejor que el problema Nguyen-5. El caso de Nguyen-7 es de sumo interés, se puede notar claramente que la estructura escogida a pesar de no ser una estructura correcta para el problema propiamente tal, sus métricas son similares a la gran cantidad de modelos que obtuvieron las estructuras correctas para los distintos experimentos. De esta manera, se puede indicar que la solución encontrada por el método es una solución sub-óptima local al rango entregado por el problema de Nguyen-7, pero que la diferencia entre el modelo generado y la solución encontrada tienen diferencias más grandes que el resto en comparación. Esto se debe al hecho de que al ser una solución sub-óptima generada por la red los decimales de los

coeficientes de dicha solución son importantes para el ajuste de esta al modelo generado. Al no representar un patrón real en los datos, sino que solamente una aproximación.



(a) Nguyen-1

(b) Constant-2

(c) Nguyen-5

(d) Nguyen-7

Figura 4.10: Diferencia absoluta entre la red entrenada y la expresión generada en función del dominio de validez (x) al final del entrenamiento para los problemas Nguyen-1,Constant-2, Nguyen-5, Nguyen-7 respectivamente.

Existe un trade-off entre la compactabilidad que se busca de las soluciones encontradas y el umbral desde el que se cortan dichas soluciones. Se puede ver que para soluciones lo suficientemente cercanas el efecto de un umbral alto no afecta la representatividad de la curva, pero en el caso de modelos que si obtuvieron la estructura correcta, se puede observar que la diferencia entre soluciones no fue tanto mayor, ya que problemas como Constant-2 y Nguyen-1 tuvieron diferencias de curvas parecidas y en el caso de Nguyen-1 si se obtuvo la solución correcta. Esto indica que el principal problema radica en las soluciones sub-óptimas más que en el punto de corte de las soluciones con el umbral.

4.2.1.3. Curvas de Aprendizaje

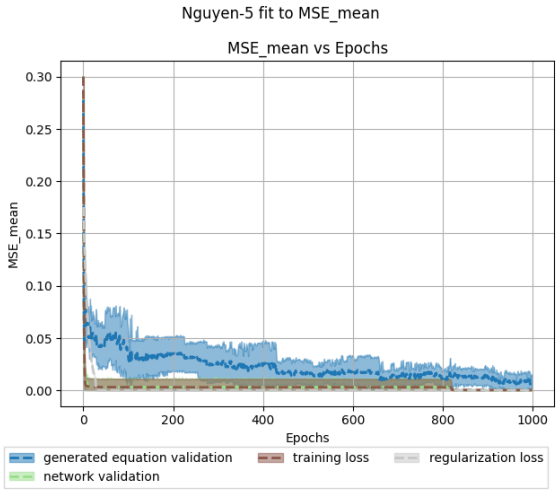
Las curvas de Aprendizaje permiten verificar el progreso de los modelos de red a medida que estos se entrenan, además los conjuntos de validación permiten ver si existe sobre ajuste

por parte de dicho modelo a los datos de entrenamiento o si los errores de validación y entrenamiento se encuentran en órdenes de magnitud parecidos, permitiendo de esta manera saber si existe algún problema con el entrenamiento o si es necesario hacer algo para que las métricas de entrenamiento y validación sean de órdenes parecidos y así el modelo no sufra de sobre ajuste o sub-ajuste.

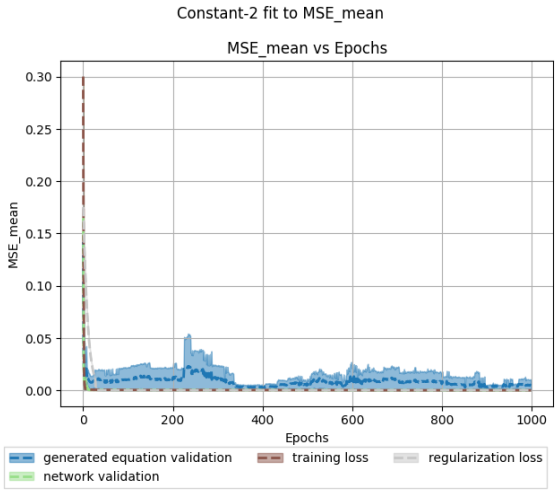
A través de esta sección revisaremos las curvas de aprendizaje de las estructuras correctamente encontradas junto con las incorrectamente encontradas por el algoritmo de búsqueda estructural. De esta manera dicho análisis nos permitirá saber cual es el efecto sobre el entrenamiento de los modelos que tiene la estructura correcta en comparación a los que no los tienen.

4.2.1.3.1. Curvas de Aprendizaje en Modelos con Estructura Correcta

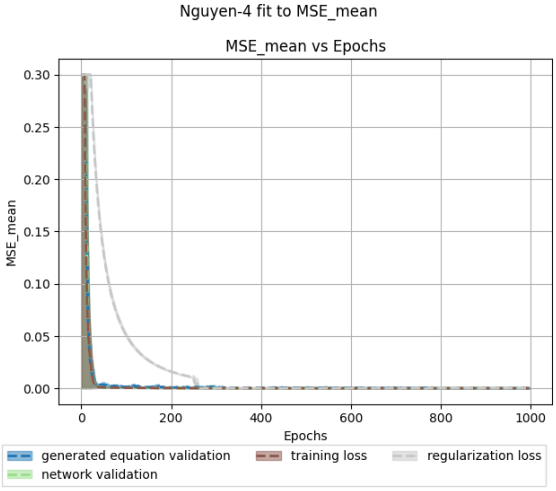
Primero que se hará será analizar el caso de una curva de aprendizaje que tenga un valor de NRMSE menor a 0.06 en comparación a otro modelo que se haya ajustado a la curva con un valor de NRMSE mayor a dicho valor según la tabla 4.4. La figura 4.11 muestra como las distintas curvas de entrenamiento se ajustan a los datos, de dichas figuras podemos claramente notar que el error de predicción (curva café) converge rápidamente a cero en los cuatro casos. La principal diferencia que podemos notar entre los distintos modelos es la convergencia de la regularización L_{SR}, la que está presente para que las potencias tengan la obligación de elegir simetría y no quedarse en puntos intermedios, por lo que podemos ver que a medida que se demora más la selección de simetría el ajuste del modelo tiende a ser mejor. Dicha aseveración nace del hecho de que las redes se inicializan en un punto del espacio de soluciones, si dicha selección se hace de manera tardía y el modelo aún así se ajusta quiere decir que los otros parámetros de dicho modelo generaron dicho ajuste. Por otro lado, si la convergencia es rápida en ambos casos no es posible determinar si ésta se debe a la selección temprana de una simetría para acercarse a un mínimo local o se debe a los parámetros del problema.



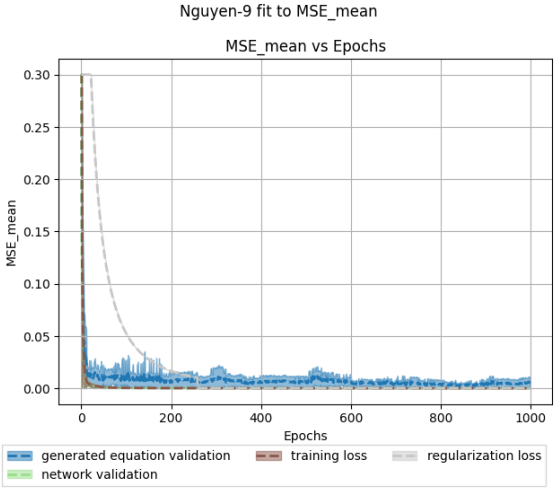
(a) Nguyen-5



(b) Constant-2



(c) Nguyen-4



(d) Nguyen-9

Figura 4.11: Curvas de aprendizaje durante el entrenamiento final de los modelos encontrados. Para esto se consideran 4 gráficos, siendo estos Nguyen 5, Constant-2, Nguyen-4 y Nguyen-9.

Como se indicó anteriormente el MSE es propenso a tener dimensiones diferentes dependiendo de la varianza de la curva a predecir. De este modo, es importante además analizar las curvas de NRMSE a través del entrenamiento. La figura 4.12 muestra el valor de NRMSE promedio a través del entrenamiento.

(a) Nguyen-5

(b) Constant-2

(c) Nguyen-4

(d) Nguyen-9

Figura 4.12: Curvas de entrenamiento durante el entrenamiento final de los modelos encontrados. Para esto se consideran 4 gráficos, siendo estos Nguyen 5, Constant-2, Nguyen-4 y Nguyen-9.

Se puede notar que el ajuste del modelo entrenado es mejor que el de la expresión generada, pero esto se debe a la definición del umbral. Además, se puede observar por las curvas de la figura 4.10 que la diferencia entre la salida predicha por el modelo de red son similares. También se observa que los modelos con peor ajuste (Nguyen-5 y Constant-2) tienen una alta desviación estándar de las curvas de NRMSE tanto en la red como en la expresión generada, mientras que modelos con mejores ajustes tienen una menor desviación.

Se puede decir que la red sufre de sobre ajuste en los casos en que no se encuentra la solución real del problema , pero dicho sobre ajuste no tiene relación con el que sufren las redes neuronales convencionales. Esto se debe a que en este caso, el problema de sobre ajuste nace del nivel de precisión que se puede alcanzar con números de punto flotante, los cuales permiten ajustar la expresión a estructuras subóptimas.

4.2.1.3.2. Curvas de Aprendizaje en Modelos con Estructura Incorrecta

Al analizar los resultados de entrenamiento para el caso con estructuras incorrectas en el modelo es importante considerar que estos sufrirán del mismo sobre ajuste generado por

el nivel de aproximación que permiten los puntos flotantes en las redes entrenadas. De esta manera la figura 4.13 muestra como durante el entrenamiento final, el modelo de Nguyen-7 logró ajustarse a los datos de entrenamiento, pero al igual que en casos anterior, se puede ver que la curva de regularización converge de manera rápida. En este caso la estructura incorrecta no permite que el ajuste del modelo en validación sea mejor, ya que el modelo generado no tiene la posibilidad de generar la función generadora de los datos.

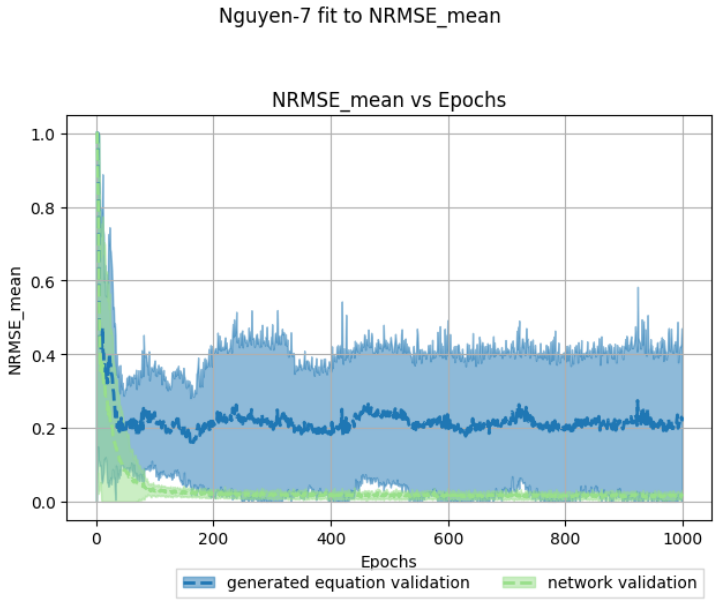


Figura 4.13: Curvas de aprendizaje durante el entrenamiento final del modelo con estructura errónea en el problema de Nguyen-7. En esta figura se puede notar el sobre ajuste que tiene la ecuación generada, ya que no puede ajustar sus resultados a los datos de validación, esto se debe claramente a la falla de la estructura en el modelo.

4.2.1.4. Resultados Finalizado el Entrenamiento

La etapa posterior al entrenamiento, es donde el modelo trata de mejorar la solución encontrada, para esto, la expresión encontrada pasa por un optimizador no lineal para poder encontrar la mejor expresión. Para mostrar que el método utilizado genera efectivamente las mejores soluciones para las estructuras generadas se optimizó la misma estructura que posteriormente fue optimizada mediante tres métodos diferentes, el primero fue utilizar la expresión que se genera directamente posterior al entrenamiento de la red, es decir utilizar la interpretación del modelo (SR). El segundo caso es utilizar L-BFGS-B en la expresión generada por el modelo de red entrenado (L-BFGS-B-SR). Por último, se utilizó la estructura del modelo de red entrenado reemplazando todos los parámetros por valores aleatorios entre $[-4, 4]$ y se utilizó esta solución como inicio de búsqueda para el algoritmo BFGS el cual no tiene límites en su búsqueda (BFGS). Los resultados encontrados de este experimento se muestran en la tabla 4.5.

El modelo generado no presenta mayores diferencias de NRMSE en el conjunto de testeo y validación, dicho efecto se debe a la finalidad existente que tienen los modelos de aprendizaje de máquinas interpretativo y los modelos de regresión simbólica, que es generar soluciones que logren generalizar fuera de los datos de validación y de entrenamiento. De esta manera, se puede ver que el modelo presentado logra efectivamente generalizar con los resultados

obtenidos a través de su entrenamiento. Además, se puede ver que los resultados de mejor función de costos se encuentran con los parámetros generados con el modelo de red neuronal, superando incluso a los casos donde se le entrega solamente la estructura entrenada al modelo de optimización no lineal. Además, se puede ver que en ningún momento modelos como BFGS lograron superar la solución encontrada e incluso se puede decir que dichos modelos empeoraron en muchos casos la solución. De este modo se puede decir que la solución encontrada por el modelo de red presentado es mejor que la obtención de la expresión encontrada utilizando solamente la estructura correcta.

Tabla 4.5: Errores NRMSE al utilizar 3 métodos diferentes de optimización no lineal posterior a selección estructural, las celdas en azul son el tipo de optimización que obtuvo mejores resultados en el conjunto de testeo para un problema Nguyen. Cada uno de los resultados se obtuvo luego de correr 10 veces el algoritmo.

Nombre	**SR**		**LBFGS-SR**		**BFGS**	
Conjunto	Validación	Test	Validación	Test	Validación	Test
Nguyen-1	0.017±0.01826	0.014±0.01520	0.018±0.01876	0.015±0.0158	0.428±1.03736	0.415±0.9878
Nguyen-2	0.007±0.00604	0.007±0.00539	0.010±0.00745	0.008±0.0064	23.431±70.06722	209.965±629.69
Nguyen-3	0.004±0.00158	0.004±0.00163	0.009±0.00343	0.009±0.00342	0.009±0.00000	0.009±0.00000
Nguyen-4	0.002±0.00031	0.002±0.00035	0.002±0.00031	0.002±0.00035	42.747±110.148	69.169±178.56
Nguyen-5	0.175±0.06773	0.178±0.07095	0.257±0.06809	0.262±0.068	0.524±0.542	0.521±0.534
Nguyen-6	0.038±0.02097	0.038±0.02197	0.082±0.03706	0.084±0.03824	0.650±0.933	0.690±0.982
Nguyen-7	0.060±0.05200	0.059±0.051	0.128±0.13048	0.138±0.140	235531.94±706261.79	135.633±306.8
Nguyen-8	0.000±0.00000	0.000±0.00000	0.000±0.00000	0.000±0.00000	0.000±0.00000	0.000±0.00000
Nguyen-9	0.053±0.01543	0.060±0.02020	0.091±0.05984	0.095±0.05628	85527.95±256564.55	1538.870±4608.51827
Nguyen-10	0.097±0.06478	0.095±0.069	0.146±0.15234	0.140±0.14642	0.273±0.26	0.276±0.276
Nguyen-12	0.105±0.02395	0.100±0.01972	0.241±0.21335	0.247±0.23085	100.802±299.733	100.801±299.734
Constant-1	0.007±0.00091	0.006±0.00097	0.008±0.00000	0.007±0.00000	0.015±0.0134	0.014±0.01432
Constant-2	0.143±0.07701	0.147±0.07772	0.369±0.33709	0.363±0.34013	2.478±3.24031	2.424±3.17094
Constant-3	0.068±0.04	0.063±0.03731	0.074±0.038	0.066±0.03618	0.305±0.573	0.284±0.55180

4.2.1.5. Comparación con Métodos SOTA

Para poder entender que tan buenos son los ajustes que produce el modelo SRNet con respecto a los datos, es necesario comparar su desempeño con respecto al estado del arte (SOTA). Como se explicó anteriormente los métodos escogidos para esta tarea son pysr, pstree y feyn. De esta manera para poder comparar los resultados obtenidos con métodos SOTA de regresión simbólica fue necesario obtener resultados de estos métodos en el conjunto de datos Nguyen. Las comparaciones entre estos modelos se pueden observar en el anexo C.1.2, el cual presenta una tabla con los cuatro métodos utilizados.

A modo de análisis, se puede ver que SRNet tiene facilidad para encontrar soluciones del tipo polinómicas, como se puede observar en la tabla 4.4 estos son los problemas con mejores resultados. En esta línea se puede ver que para problemas de regresión simbólica que consideren polinomios el método propuesto obtuvo resultados competitivos con el estado del arte como son los problemas Nguyen-2, Nguyen-3. Existen casos donde incluso el modelo superó el estado del arte como es el caso de Nguyen-8 o Nguyen-4.

Tomando en consideración una visión más global de los resultados obtenidos se puede indicar que el modelo SRNet tiene para la mayoría de los casos (salvo Nguyen-5, Nguyen-12 e Constant-2) soluciones muy similares a las encontradas con el método de pstree. Esto presenta una consideración importante considerando el desempeño que obtuvo pstree en la última competición de regresión simbólica. Por otro lado, se puede indicar que el método feyn

de Qlatice logra resultados considerablemente mejores que los generados mediante el método propuesto. Aún así, es importante mencionar que los algoritmos expuestos en la literatura utilizan potencias enteras para aproximar los polinomios, de este modo, estos modelos presentan la ventaja comparativa de no tener dentro del espacio de posibles soluciones expresiones con potencias no enteras y en caso de existir, dichas soluciones son de baja probabilidad ya que dependen de multiplicaciones consecutivas además de divisiones.

Al considerar los tiempos de cómputo que utilizó cada uno de los modelos, se puede mencionar que el más rápido es feyn y el más lento es el método SRNet propuesto. Específicamente los tiempos de cómputo son aproximadamente entre ocho y diez veces los que presenta pstree. De este modo, se puede indicar que dicha arista del modelo es una desventaja comparativa importante del método propuesto en comparación al estado del arte. La principal razón de las diferencias de tiempos de cómputo encontradas se deben en parte al uso de BP para actualizar los parámetros de la red, esto sumado al hecho de que el código está completamente compilado en python y no hace uso de otros programas como si los utiliza pysr, feyn. De este modo, el tiempo de cómputo es una arista a mejorar del método propuesto en especial buscando formas de optimizar su trabajo en paralelo y optimizar las actualizaciones por BP.

4.2.2. Modelo de Baterías de Litio

El conjunto de datos del modelo de baterías es desafiante, en especial para modelos que utilicen redes neuronales. La principal razón de esto se encuentra en las entradas al modelo. Como se mencionó en la sección 4.1.2 cada una de las entradas al modelo tiene una distribución muy distinta en comparación a las otras variables. En específico se tiene el caso del número de Reynolds que se encuentra definido entre $[574, 11015]$ y el número de Prandtl que su rango esta entre [0.705, 0.71] siendo casi constante. Este tipo de problemas hace una demostración explicativa sobre una de las limitaciones de este modelo, el modelo propuesto no puede realizar normalización de la entrada. El proceso de normalizar los datos de entrada a un modelo de aprendizaje de máquinas es parte fundamental del pre procesamiento de los datos.

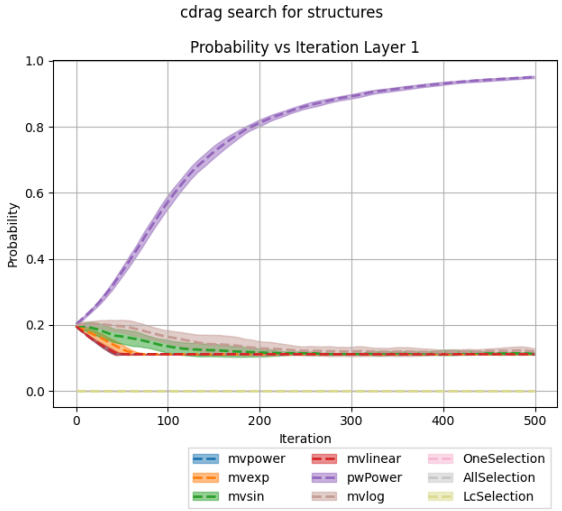
Los resultados generales obtenidos por el SRNet se pueden encontrar en la tabla 4.6, donde el ajuste a este nuevo conjunto de datos fue considerablemente más bajo a los que se encontraban en los problemas Nguyen. Una parte interesante del proceso de ajuste del modelo se encuentra en el hecho de que a pesar de no obtener las expresiones correctas, las estructuras encontradas en el coeficiente de arrastre (c_d) y el número de Nusselt si fueron las adecuadas, mientras que para el factor de fricción (f_D) el algoritmo pudo encontrar estructuras polinómicas pero mediante la función potencia punto a punto que no realiza multiplicaciones, dichas multiplicaciones era según la teoría lo que el modelo debía encontrar (Anexo C.2.1). El principal problema que tuvo el método al momento de ajustarse a la curva fue el ajuste de parámetros y como se dijo anteriormente, el aprendizaje en redes neuronales sin haber normalizado la entrada se complejiza en comparación al entrenamiento con entradas normalizadas.

Tabla 4.6: Tabla de resultados obtenidos luego de aplicar SRNet propuesto al problema de Baterías de Litio.

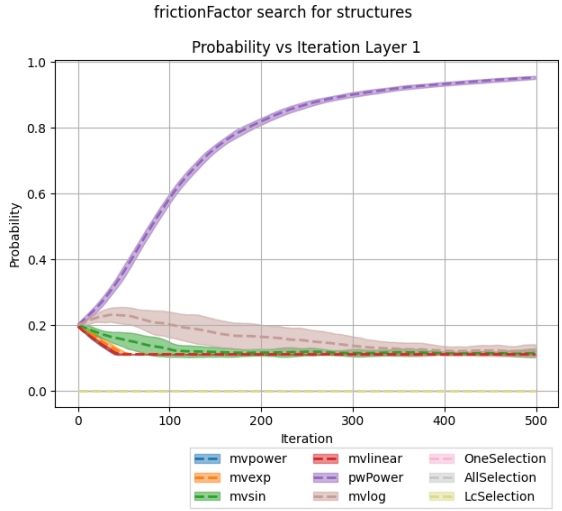
Nombre	NRMSE	R2	Tiempo [s]	Mejor [NRMSE]	Mejor Expresión
c_d	0.283±0.012	0.920±0.007	1214.177±2.273	0.272	$2.4S_a^{-0.2} - 0.1Re_a^{0.2}$
f_D	0.359±0.001	0.871±0.001	1233.898±38.781	0.358	$3S_a^{-1.1} + 16.3Re_a^{0.3} - 1$
n_u	0.168±0.037	0.970±0.013	1370.947±49.860	0.119	$3.4S - 2.9P_r + 0.3S^{-0.2}Re^{0.6}P_r^{-1.4} - 4.5$

Tomando en consideración los resultados obtenidos en la tabla C.3, se puede notar que a pesar de no generarse el ajuste adecuado a la curva, el método propuesto tiene resultados comparables con los métodos del estado del arte, siendo el segundo en dos de los tres coeficientes, quedando bajo feyn que fue uno de los ganadores de la competencia de SR.A pesar de existir una limitación al momento de poder normalizar las entradas a la red, el ajuste del modelo igualmente es comparable con el estado del arte para este tipo de casos.

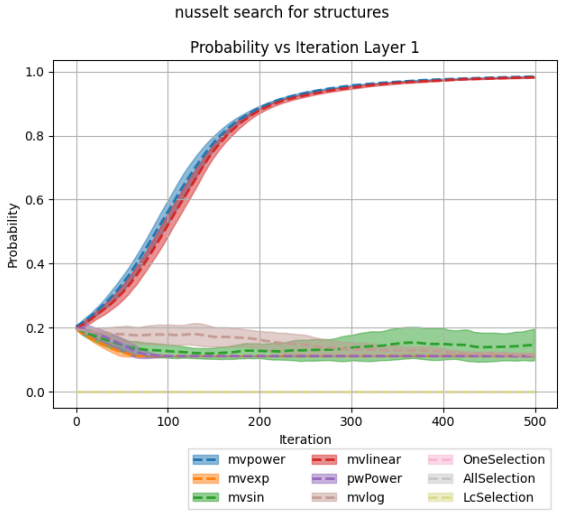
Por último, en la figura 4.14 se muestran los ajustes que tuvo el modelo, en específico el resultado menos ajustado es el generado en en el Número de Nusselt, mientras que el que tiene mejor ajuste es el generado para c_d.



(a) Cdrag



(b) Factor de Fricción



(c) Número de nusselt

Figura 4.14: Estructuras generadas mediante el algoritmo de Búsqueda estructural. En este caso cada gráfico representa 10 corridas realizadas sobre cada una de los 3 coeficientes del modelo de baterías.

4.2.2.1. Normalizando los Datos

A modo de demostración explicativa del efecto que produce la normalización de los datos

sobre el modelo, se intentó normalizar el valor del número de Reynolds de una manera que no generara dependencia en los datos anteriors, por lo que dicha normalización será dividir el número de Reynolds en 10000. De este modo, posterior a su normalización, se realizó el entrenamiento del modelo propuesto, logrando los siguientes resultados.

Tabla 4.7: Resultados obtenidos al aplicar el método propuesto de normalización de entradas en problema de Baterías.

Nombre	NRMSE	R2	Tiempo [s]	Mejor [NRMSE]	Mejor Expresión
c_d	0.268±0.002	0.928±0.001	1089.670±2.961	0.266	$2.2exp(-2S - 0.3Re) + 1.1exp(-0.3Re) + 0.5$
f_D	0.398±0.033	0.840±0.027	1190.840±41.804	0.376	$3.2s^{-1.1} - 1.1Re^{0.7} + 0.5$
n_u	0.168±0.037	0.970±0.012	1101.296±2.793	0.126	$8.9exp(-0.2S + 0.8Re + 2.9z) - 5.9exp(-0.1S - 3Re + 3P_r) + 0.4$

De este modo, según lo explicado anteriormente las soluciones que históricamente mejor han funcionado para este problema son las soluciones polinómicas y multiplicativas entre potencias. Como se puede ver el modelo normalizado pierde dicha solución y la solución generada hace uso de exponenciales las cuales no se condice a la teoría detrás del modelo de baterías. De esta forma, se puede notar que es posible normalizar las entradas, pero el efecto que dicha práctica genera en el modelo es la pérdida de representatividad de la teoría que ya se conoce sobre el problema, por lo que no sería una solución factible.

5. Conclusiones

En este trabajo de tesis se desarrolló un modelo nuevo de regresión simbólica el cual logra solucionar los dos tipos de problemas presentes en la regresión simbólica que son: *la búsqueda de estructuras* y *la búsqueda de parámetros*. Esto se logra mediante el uso de un algoritmo de búsqueda estructural y un entrenamiento posterior que utiliza bloques de funciones y selección. El método propuesto llamado SRNet permite el aprendizaje tanto de la estructura correcta para resolver el problema como de los coeficientes que minimicen el error de salidas. El modelo tiene la ventaja de ser un método de aprendizaje de máquinas interpretativo, el cual permite analizar y entender los resultados que dicho modelo entrega, haciendo que un experto pueda analizar los resultados obtenidos. El método propuesto tiene la ventaja de utilizar métodos de aprendizaje de máquinas y además contar con un bajo número de parámetros globales, siendo estos no más de 50 parámetros.

El modelo propuesto tiene la particularidad de ser el único en la literatura que ajusta la función potencia de manera continua siendo el valor de la potencia un parámetro del modelo. De esta manera el modelo se escapa del método convencional de aproximar las potencias con el uso de multiplicación (algoritmos genéticos) o funciones del estilo x^{α} siendo α el exponente de la función definida que se encuentra en un set de funciones construido por $\alpha \in \{-n, ..., n\}$. Dicho cambio permite entregarle mayor libertad a los modelos haciendo que el exponente generado no esté ligado a la estructura de la solución sino que desde una estructura escueta puedan generarse soluciones con variados exponentes sin dicha dependencia. Como se explicó anteriormente dicha necesidad es primordial al momento de realizar modelos de regresión simbólica con redes neuronales dedicadas ya que de no realizarse de esa manera la solución estaría ligada al número de capas haciendo que cada modelo que necesite un exponente alto tenga un gran número de capas y por ende existan variadas soluciones subóptimas en el camino entorpeciendo la búsqueda estructural.

Además, a través de este trabajo de tesis, se diseñó e implementó un modelo de red dedicada el cual puede cambiar su estructura simplemente cambiando las funciones que se le entregan en un inicio. Dicho tipo de estructuras hace que este algoritmo se parezca más a un modelo de computación evolutiva que a uno de *Equation Learning Networks (EQL)*. Al modelo propuesto se le pasa un conjunto de funciones y este elige las mejores para solucionar el problema, mientras que modelos como EQL deben hacer todos los parámetros 0 ya que se encuentra obligado a utilizar todo el conjunto de funciones candidatas. En esta línea es importante también destacar el diseño del algoritmo SS, el cual en los variados problemas mostró su robustez al momento de seleccionar estructuras, siendo este un método nuevo implementado para este tipo de estructuras. Además, se mostró que el aprendizaje de estructuras, cuando es correcto, mejora en más del 20 %, haciendo así que exista una heurística que permita discernir cuando este se ha equivocado.

Otro punto importante del trabajo de tesis realizado tiene relación con la optimización

de parámetros, a través de este proceso no solamente se diseñó una versión mejorada de la función potencia continua, sino que además se hicieron análisis de puntos de inicialización de redes para distintas funciones de activación (problema recurrente en modelos de redes neuronales). Se logró diseñar un modelo de red con robustez en funciones polinómicas, el cual es competitivo con algunos modelos del estado del arte como pstree en términos de los resultados de la solución obtenida. También se puede concluir que el método propuesto debe mejorar los tiempos de cómputo para así poder compararse a los métodos del estado del arte, ya que actualmente sus tiempos de cómputo son prácticamente 10 veces mayor que el de los modelos del estado del arte. Mediante este trabajo se dieron los primeros paso en la metodología sobre como entrenar dichos modelos de redes tanto por su inicialización como también definiendo que tipo de regularizaciones ocupar.

También a través de este trabajo de tesis se puede destacar el experimento realizado sobre el modelamiento de baterías de litio. En dicho problema se pudo ver como el modelo encontró las estructuras adecuadas para el coeficiente de arrastre y el número de Nusselt. Pero a pesar de que la estructura generada era la correcta, no se pudieron encontrar los coeficientes adecuados para dichas expresiones. De este modo, se puede recalcar la robustez del SSA, además de ver el efecto que genera en la optimización de coeficientes la diferencia existente entre los ordenes de magnitud de las entradas. Por último, es interesante notar que a través de este trabajo de tesis se pudo mostrar como la normalización de la entrada afecta la representatividad del modelo, haciendo que este seleccione estructuras erróneas las cuales no permiten dar una representatividad clara a la expresión generada. Como se pudo ver en los resultados encontrados esto no tiene un efecto directo en el valor de la función de pérdida del modelo pero si en como podemos analizar los resultados y las relaciones existentes entre las variables independientes y dependientes del problema.

A modo de mejora para futuros procesos de tesis se propone estudiar la estructura del modelo de red neuronal dedicada y hacer cambios que permitan limitar las combinaciones lineales a términos sin decimales que se combinen con las entradas, esto permitirá restringir el espacio de soluciones y así limitar la existencia de soluciones intermedias sub-óptimas que se puedan generar con este modelo de red. Además, se propone estudiar las posibles formas de aproximar la función potencia de manera de no permitir que la elección de simetrías que no tengan correspondencia con la función potencia real, es decir no permitir soluciones del tipo x^3 simétricas. Otra mejora posible al modelo generado tiene relación con la normalización de las entradas, como se dijo anteriormente esto es una limitante del modelo de red debido a la interpretación de la expresión resultante, por lo que solucionar dicha limitante podría facilitar la optimización de los modelos encontrados y mejorar su interpretabilidad. También se propone separar el algoritmo SSA de la optimización de parámetros de manera de poder obtener la estructura de red predicha y entrenar sobre esta cuantas veces sea necesario hasta encontrar una respuesta satisfactoria, donde incluso se podría volver a correr el SSA para buscar otras estructuras posibles y limitar la ya encontrada. También, se propone estudiar la posibilidad de paralelización de las estructuras que puedan ser corridas en GPU y no en cpu como se realizó en esta tesis ya que siempre dichos modelos están limitados al número de cores que tenga el computador utilizado.

Por último, se enfatiza el hecho de que el desarrollo de modelos como el propuesto permite que los algoritmos de regresión simbólica tengan una manera de aproximar los coeficientes de las expresiones generadas que es uno de los principales problemas actuales de los modelos de regresión simbólica con algoritmos genéticos u optimizadores no lineales, ya que dicha estructura puede ser la correcta pero si los coeficientes no son los adecuados el ajuste del

modelo no será el correcto. El desarrollo de modelos que utilicen *backpropagation* en modelos de aprendizaje de máquinas interpretativo permite que el área de la regresión simbólica avance en los problemas de búsqueda de estructuras y además, permite reducir los desafíos existentes en el ajuste de coeficientes de las expresiones generadas.

Bibliografía

[1] Rudin, C., "Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead," Nature Machine Intelligence, vol. 1, pp. 206–215, 2019, doi:10.1038/s42256-019-0048-x.

[2] Koza, J. R., Genetic programming : on the programming of computers by means of natural selection. MIT Press, 1992.

[3] La Cava, W., Spector, L., y Danai, K., "Epsilon-lexicase selection for regression," en Proceedings of the Genetic and Evolutionary Computation Conference 2016, GECCO '16, (New York, NY, USA), p. 741–748, Association for Computing Machinery, 2016, doi:10.1145/2908812.2908898.

[4] Zhang, Q. y Li, H., "Moea: A multiobjective evolutionary algorithm based on decomposition," IEEE Transactions on Evolutionary Computation, vol. 11, pp. 712–731, 2007, doi:10.1109/TEVC.2007.892759.

[5] Deb, K., Pratap, A., Agarwal, S., y Meyarivan, T., "A fast and elitist multiobjective genetic algorithm: NSGA-II," IEEE Transactions on Evolutionary Computation, vol. 6, pp. 182–197, 2002, doi:10.1109/4235.996017.

[6] Schmidt, M. D. y Lipson, H., "Age-fitness pareto optimization," en Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation, GECCO '10, (New York, NY, USA), p. 543–544, Association for Computing Machinery, 2010, doi: 10.1145/1830483.1830584.

[7] Petersen, B. K., Larma, M. L., Mundhenk, T. N., Santiago, C. P., Kim, S. K., y Kim, J. T., "Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients," en International Conference on Learning Representations, 2021, https://openreview.net/forum?id=m5Qsh0kBQG.

[8] Biggio, L., Bendinelli, T., Lucchi, A., y Parascandolo, G., "A seq2seq approach to symbolic regression," en Learning Meets Combinatorial Algorithms at NeurIPS2020, 2020.

[9] Kamienny, P.-A., d'Ascoli, S., Lample, G., y Charton, F., "End-to-end symbolic regression with transformers," en Advances in Neural Information Processing Systems (Oh, A. H., Agarwal, A., Belgrave, D., y Cho, K., eds.), 2022, https://openreview.net/forum ?id=GoOuIrDHG_Y.

[10] Sahoo, S., Lampert, C., y Martius, G., "Learning equations for extrapolation and control," en Proceedings of the 35th International Conference on Machine Learning (Dy, J. y Krause, A., eds.), vol. 80 de Proceedings of Machine Learning Research, pp. 4442–4450, PMLR, 2018, https://proceedings.mlr.press/v80/sahoo18a.html.

[11] Kim, S., Lu, P. Y., Mukherjee, S., Gilbert, M., Jing, L., Ceperic, V., y Solja, M., "Integration of neural network-based symbolic regression in deep learning for scientific discovery," IEEE Transactions on Neural Networks and Learning Systems, vol. 32, pp. 4166–4177, 2019.

[12] La Cava, W., Orzechowski, P., Burlacu, B., de Franca, F., Virgolin, M., Jin, Y., Kommenda, M., y Moore, J., "Contemporary symbolic regression methods and their relative performance," en Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks (Vanschoren, J. y Yeung, S., eds.), vol. 1, 2021, https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/file/c0c7c76d30bd3dcaefc96f40275bdc0a-Paper-round1.pdf.

[13] Kubalik, J., Derner, E., Zegklitz, J., y Babuska, R., "Symbolic regression methods for reinforcement learning," IEEE Access, vol. 9, pp. 139697–139711, 2021, doi:10.1109/ACCESS.2021.3119000.

[14] Błądek, I. y Krawiec, K., "Solving symbolic regression problems with formal constraints," pp. 977–984, Association for Computing Machinery, Inc, 2019, doi:10.1145/3321707.3321743.

[15] Bartoli, A., Castelli, M., y Medvet, E., "Weighted hierarchical grammatical evolution," IEEE Transactions on Cybernetics, vol. 50, pp. 476–488, 2020, doi:10.1109/TCYB.2018.2876563.

[16] Lourenço, A. e. a., "Structured grammatical evolution: A dynamic approach," en Handbook of Grammatical Evolution, pp. 137–161, Springer International Publishing, 2018, doi:10.1007/978-3-319-78717-6_6.

[17] O'Neill, M. y Ryan, C., Grammatical Evolution. Springer US, 2003, doi:10.1007/978-1-4615-0447-4.

[18] Virgolin, M., Alderliesten, T., y Bosman, P. A., "Linear scaling with and within semantic backpropagation-based genetic programming for symbolic regression," pp. 1084–1092, Association for Computing Machinery, Inc, 2019, doi:10.1145/3321707.3321758.

[19] Kubalík, J., Derner, E., y Babuška, R., "Symbolic regression driven by training data and prior knowledge," en Proceedings of the 2020 Genetic and Evolutionary Computation Conference, GECCO '20, (New York, NY, USA), p. 958–966, Association for Computing Machinery, 2020, doi:10.1145/3377930.3390152.

[20] Cava, W. L., Singh, T. R., Taggart, J., Suri, S., y Moore, J., "Learning concise representations for regression by evolving networks of trees," en International Conference on Learning Representations, 2019, https://openreview.net/forum?id=Hke-JhA9Y7.

[21] Topchy, A. y Punch, W. F., "Faster genetic programming based on local gradient search of numeric leaf values," en Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation, GECCO'01, (San Francisco, CA, USA), p. 155–162, Morgan Kaufmann Publishers Inc., 2001.

[22] Goodfellow, I., Bengio, Y., y Courville, A., Deep Learning. MIT Press, 2016. http://www.deeplearningbook.org.

[23] Martius, G. y Lampert, C. H., "Extrapolation and learning equations," en ICLR (Workshop), 2017, https://openreview.net/forum?id=BkgRp0FYe.

[24] Sutskever, I., Vinyals, O., y Le, Q. V., "Sequence to sequence learning with neural

networks," en Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2, NIPS'14, (Cambridge, MA, USA), p. 3104–3112, MIT Press, 2014.

[25] Bahdanau, D., Cho, K., y Bengio, Y., "Neural machine translation by jointly learning to align and translate," 2015. 3rd International Conference on Learning Representations, ICLR 2015 ; Conference date: 07-05-2015 Through 09-05-2015.

[26] Brown, T., Mann, B., Ryder, N., y Subbiah, "Language models are few-shot learners," en Advances in Neural Information Processing Systems (Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., y Lin, H., eds.), vol. 33, pp. 1877–1901, Curran Associates, Inc., 2020, https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

[27] Valipour, M., You, B., Panju, M., y Ghodsi, A., "Symbolicgpt: A generative transformer model for symbolic regression," CoRR, vol. abs/2106.14131, 2021, https://arxiv.org/abs/2106.14131.

[28] Sutton, R. S. y Barto, A. G., Reinforcement Learning: An Introduction. The MIT Press, second ed., 2018, http://incompleteideas.net/book/the-book-2nd.html.

[29] Fletcher, R., Practical methods of optimization. John Wiley & Sons, 2013.

[30] Nocedal, J. y Wright, S. J., Numerical Optimization. New York, NY, USA: Springer, 2e ed., 2006.

[31] Udrescu, S.-M. y Tegmark, M., "AI feynman: A physics-inspired method for symbolic regression," Science Advances, vol. 6, 2020, doi:10.1126/sciadv.aay2631.

[32] Louizos, C., Welling, M., y Kingma, D. P., "Learning sparse neural networks through l0 regularization," en International Conference on Learning Representations, 2018, https://openreview.net/forum?id=H1Y8hhg0b.

[33] Bengio, Y., Léonard, N., y Courville, A. C., "Estimating or propagating gradients through stochastic neurons for conditional computation," CoRR, vol. abs/1308.3432, 2013, http://arxiv.org/abs/1308.3432.

[34] Srinivas, S., Subramanya, A., y Babu, R. V., "Training sparse neural networks," en 2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), pp. 455–462, 2017, doi:10.1109/CVPRW.2017.61.

[35] Saito, K. y Nakano, R., "Numeric law discovery using neural networks.," en ICONIP (2), pp. 843–846, 1997.

[36] Kumar, S. K., "On weight initialization in deep neural networks," CoRR, vol. abs/1704.08863, 2017, http://arxiv.org/abs/1704.08863.

[37] Parascandolo, G., Huttunen, H., y Virtanen, T., "Taming the waves: sine as activation function in deep neural networks," 2017, https://openreview.net/forum?id=Sks3zF9eg.

[38] Silvescu, A., "Fourier neural networks," en IJCNN'99. International Joint Conference on Neural Networks. Proceedings (Cat. No.99CH36339), vol. 1, pp. 488–491 vol.1, 1999, doi:10.1109/IJCNN.1999.831544.

[39] Lapedes, A. y Farber, R., "Nonlinear signal processing using neural networks: Prediction and system modelling," 1987, https://www.osti.gov/biblio/5470451.

[40] Ke, N. R., Bilaniuk, O., Goyal, A., Bauer, S., Larochelle, H., Pal, C., y Bengio, Y.,

"Learning neural causal models from unknown interventions," 2020, https://openreview.net/forum?id=H1gN6kSFwS.

[41] Urrutia, J. A., Estévez, P. A., y Vergara, J. R., "Subset feature selection with structural variables," en 2021 IEEE Latin American Conference on Computational Intelligence (LA-CCI), pp. 1–6, 2021, doi:10.1109/LA-CCI48322.2021.9769843.

[42] Broløs, K. R., Machado, M. V., Cave, C., Kasak, J., Stentoft-Hansen, V., Batanero, V. G., Jelen, T., y Wilstrup, C., "An approach to symbolic regression using feyn," CoRR, vol. abs/2104.05417, 2021, https://arxiv.org/abs/2104.05417.

[43] Cranmer, M., "PySR: Fast & Parallelized Symbolic Regression in Python/Julia," 2020, doi:10.5281/zenodo.4041459.

[44] Rumelhart, D. E., Hinton, G. E., y Williams, R. J., "Learning representations by back-propagating errors," Nature, vol. 323, pp. 533–536, 1986, doi:10.1038/323533a0.

[45] Cranmer, M., Sanchez-Gonzalez, A., Battaglia, P., Xu, R., Cranmer, K., Spergel, D., y Ho, S., "Discovering symbolic models from deep learning with inductive biases," NeurIPS 2020, 2020.

[46] Reyes, J., "Reporte del modelo paramétrico," Universidad de Chile, 2014.

[47] De la Sota, R., "Optimización de modelo térmico paramétrico de baterías de litio mediante algoritmos evolutivos.," Universidad de Chile, 2022.

Anexos

Anexo A. Marco Teórico

A.1. Secuencia a Secuencia

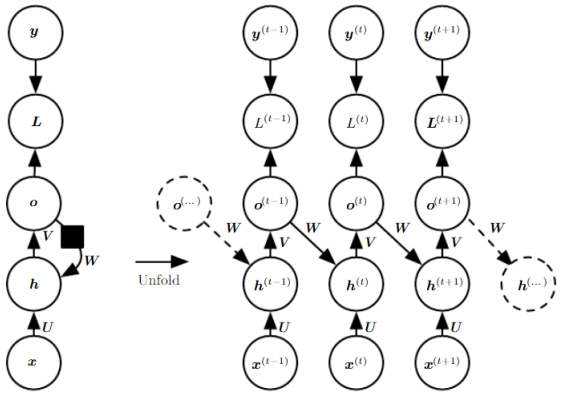


Figura A.1: Red recurrente desenvuelta en a través del tiempo. En este se puede ver como las matrices de peso son constantes a través del tiempo que permiten el uso de Backpropagation Through time (BPTT)

A.2. Algoritmo AIFeynmann

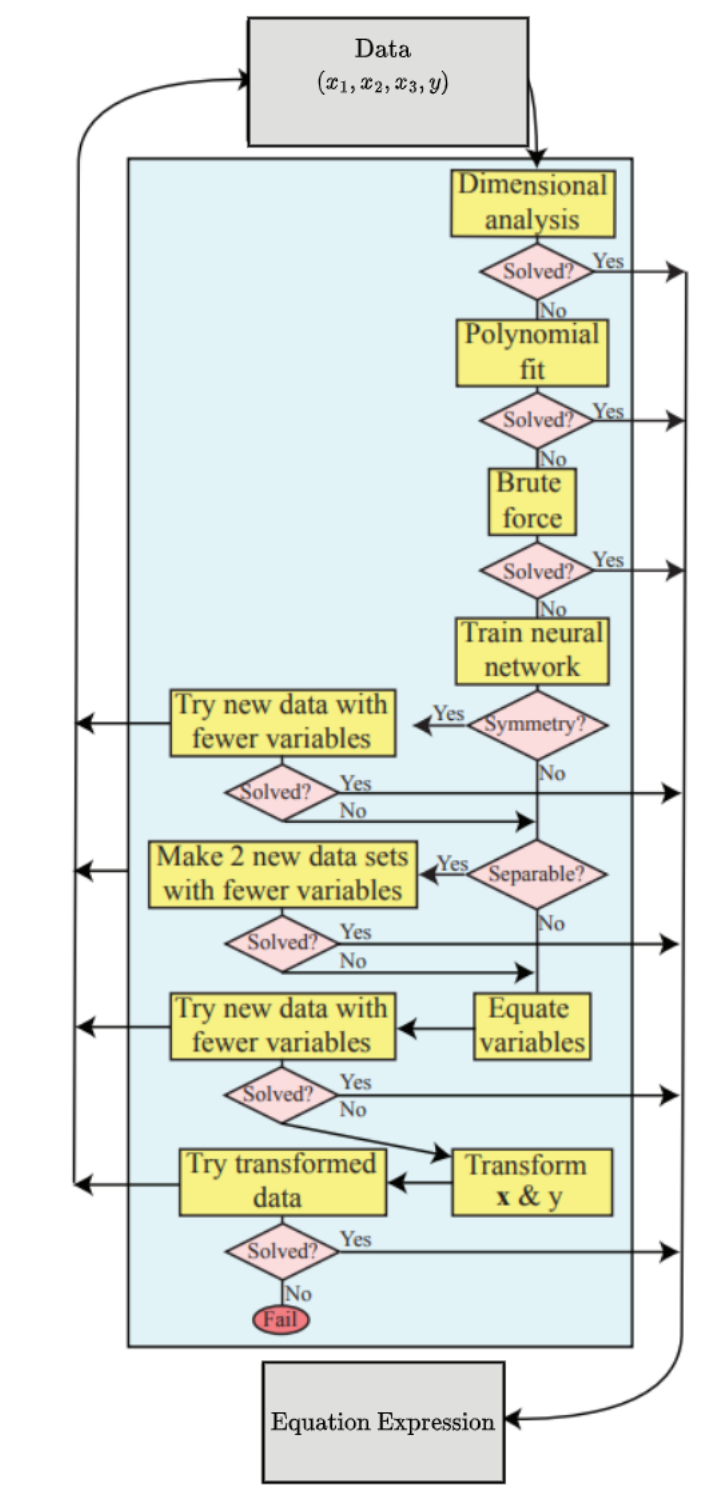


Figura A.2: Se presenta el diagrama del algoritmo AIFeymann, primero se analizan las dimensiones del problema, luego se trata de hacer un ajuste polinomial para luego intentar buscar las simetrías que pueda tener la señal mediante redes neuronales. Por último, si después de todos los pasos anteriores todavía no se encuentra dicha función, entonces se procede a aplicarles transformaciones y volver a iterar el proceso.

Anexo B. Metodología

B.1. Función Objetivo SSA

La función objetivo en el algoritmo de búsqueda estructural, se definió como la función (3.15). Esta tiene la particularidad de hacer más pequeños los errores cercanos entre si y mayores los errores mayores a uno. Utilizar la suma permite que todos los errores sean contabilizados de la misma manera y así no perder información de zonas en que el ajuste del modelo no sea el correcto. Es más, un problema común con las métricas que utilizan el promedio como función de costos se encuentra en zonas de alto poder de predicción y de otras zonas con un bajo poder de predicción. De este modo, para probar la heurística planteada en la elección de dicha función de costos, se corrió el algoritmo con cuatro funciones de costos diferentes, donde se reemplazó la suma del error cuadrático en la expresión (3.15) por las funciones MAPE (Error absoluto medio porcentual), NRMSE, RMSE (Raiz del Error Cuadrático Medio) y SSE, de manera tal de poder evidenciar los efectos que genera el cambio de métrica en la mejora de la selección de estructuras.

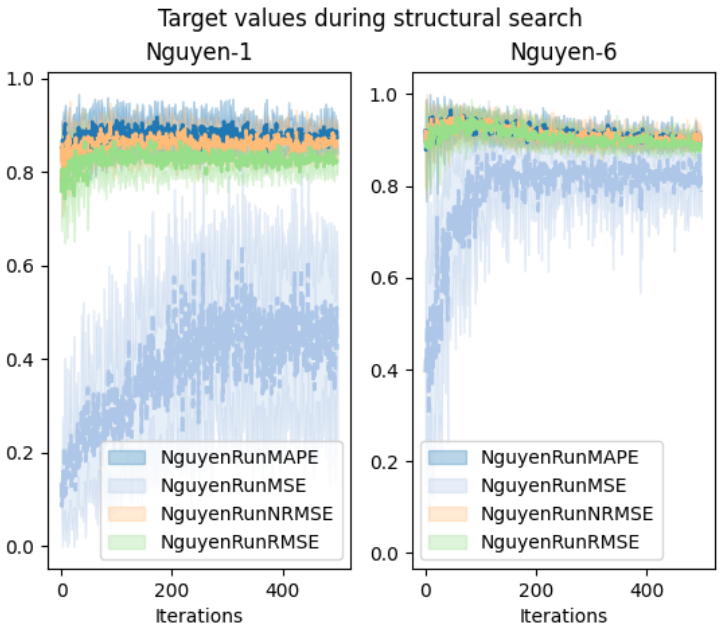


Figura B.1: Valor de las distintas métricas utilizadas para el algoritmo SSA en los problemas de Nguyen-1 y Nguyen-6.

Al observar la figura B.1 se puede evidenciar como la única función objetivo que logró generar una mejora considerable dentro de las diversas iteraciones del SSA fue el de la expresión (3.15), esto se debe a lo indicado anteriormente, las redes neuronales mediante BP tienen un método asegurado para ajustarse a los datos, por lo que métricas como las que utilizan promedios tienen ponderaciones muy altas que no permiten diferenciar entre estructuras y por ende no logra generar un aprendizaje sobre los diversos bloques disponibles en cada capa. Por último, se puede notar como a medida que avanza el algoritmo mejora considerablemente la selección que se genera sobre los diversos bloques al ser su ajuste cada vez mejor. En el caso de las otras métricas la mayoría de los casos corresponden a mejoras parciales, lo que no permitiría diferenciar correctamente entre las diversas estructuras en el modelo si se llegasen

a utilizar dichas métricas.

B.2. Cálculo de Gradiente con Estructuras:

$$
\begin{aligned}
\frac{\partial J}{\partial \gamma_{i,j}} &= \nabla \sum_B P(B) MSE(y, f(x, B, \theta)) \\
&= \sum_B MSE(y, f(x, B, \theta)) \nabla P(B) \\
&\quad \text{Using the property from [7]: } \nabla ln(x) = \frac{\nabla x}{x} \\
&= \sum_B MSE(y, f(x, B, \theta) P(B) \nabla ln(P(B) \\
&= \sum_B P(B) MSE(y, f(x), B, \theta)) \nabla \sum_l ln(P(B^l)) \\
&= \sum_B P(B) MSE(y, f(x), B, \theta)) \sum_l \frac{\partial ln(P(B^l))}{\partial \gamma_{i,j}} \\
&= \sum_B P(B) MSE(y, f(x), B, \theta))(B_{i,j} - \sigma(\gamma_{i,j})) \\
&= E_B(MSE(x, f(x), B, \theta)(B_{i,j} - \sigma(\gamma_{i,j}))) \\
&= \frac{\sum_B MSE(x, f(x), B, \theta)(B_{i,j} - \sigma(\gamma_{i,j})))}{\mid B \mid}
\end{aligned}
$$

Anexo C. Resultados

C.1. Nguyen

C.1.1. Hiperparámetros

Tabla C.1: Hiperparámetros utilizados para correr el conjunto de datos Nguyen.

Nombre	h_f	N_{layers}
Nguyen-1	2	1
Nguyen-2	3	1
Nguyen-3	4	1
Nguyen-4	4	1
Nguyen-5	2	2
Nguyen-6	2	2
Nguyen-7	2	2
Nguyen-8	1	1
Nguyen-9	2	2
Nguyen-10	2	1
Nguyen-12	3	1
Constant-1	2	1
Constant-2	2	2
Constant-3	2	2

C.1.2. Resultados Comparativos

Tabla C.2: Contiene los resultados de las métricas de NRMSE, R2 y tiempo de cómputo obtenidos al correr los modelos pysr, pstree, feyn además del modelo propuesto SRNET en el conjunto de datos Nguyen. Cada uno de estos resultados fue corrido 10 veces de manera de obtener resultados que representen correctamente el funcionamiento del modelo.

Name	Bench	R2	NRMSE	Time [S]
Constant-1	feyn	1.000±0.000	0.000±0.000	18.654±0.000
	pstree	1.000±0.000	0.005±0.000	60.144±0.000
	pysr	1.000±0.000	0.001±0.002	193.324±0.002
	SRNet	1.000±0.000	0.006±0.001	781.253±2.436
Nguyen-2	feyn	1.000±0.000	0.000±0.001	18.572±0.001
	pstree	1.000±0.000	0.004±0.000	59.817±0.000
	pysr	1.000±0.000	0.000±0.000	175.155±0.000
	SRNet	1.000±0.000	0.007±0.005	766.868±5.781
Nguyen-6	feyn	1.000±0.000	0.001±0.000	18.778±0.000
	pstree	1.000±0.000	0.003±0.000	60.711±0.000
	pysr	1.000±0.000	0.000±0.000	191.895±0.000
	SRNet	0.999±0.000	0.030±0.008	453.927±2.105
Nguyen-9	feyn	1.000±0.000	0.004±0.001	19.631±0.001
	pstree	0.992±0.000	0.091±0.000	56.906±0.000
	pysr	1.000±0.000	0.000±0.000	197.439±0.000
	SRNet	0.996±0.003	0.057±0.021	930.542±4.852
Constant-2	feyn	1.000±0.000	0.002±0.001	18.239±0.001
	pstree	1.000±0.000	0.006±0.000	60.098±0.000
	pysr	1.000±0.000	0.002±0.002	243.570±0.002
	SRNet	0.973±0.020	0.146±0.077	866.834±3.562
Nguyen-5	feyn	1.000±0.000	0.003±0.000	18.312±0.000
	pstree	1.000±0.000	0.006±0.000	60.541±0.000
	pysr	1.000±0.000	0.003±0.002	238.207±0.002
	SRNet	0.968±0.021	0.169±0.054	868.095±3.607
Nguyen-8	feyn	1.000±0.000	0.001±0.001	18.606±0.001
	pstree	1.000±0.000	0.004±0.000	60.095±0.000
	pysr	1.000±0.000	0.002±0.003	223.516±0.003
	SRNet	1.000±0.000	0.000±0.000	644.466±1.513
Constant-3	feyn	1.000±0.000	0.002±0.001	19.818±0.001
	pstree	0.999±0.000	0.036±0.000	57.217±0.000
	pysr	1.000±0.000	0.003±0.003	215.289±0.003
	SRNet	0.995±0.008	0.063±0.037	694.055±2.938
Nguyen-1	feyn	1.000±0.000	0.000±0.000	18.826±0.000

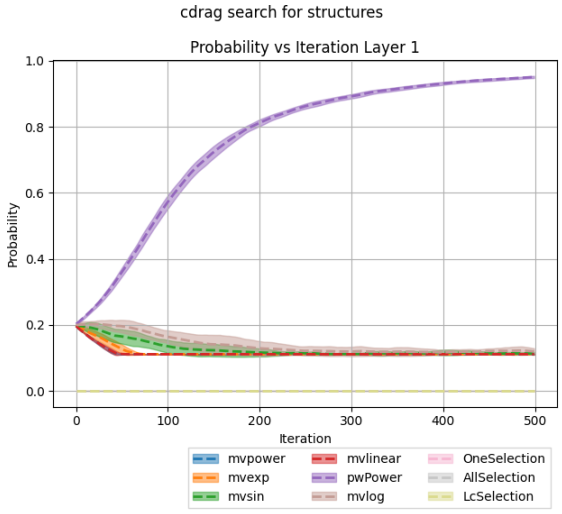
	pstree	1.000±0.000	0.004±0.000	61.229±0.000
	pysr	1.000±0.000	0.000±0.000	185.121±0.000
	SRNet	1.000±0.001	0.014±0.015	779.043±5.118
Nguyen-10	feyn	1.000±0.000	0.002±0.001	19.746±0.001
	pstree	0.996±0.000	0.062±0.000	57.336±0.000
	pysr	1.000±0.000	0.000±0.000	218.084±0.000
	SRNet	0.987±0.020	0.094±0.068	736.982±7.617
Nguyen-3	feyn	1.000±0.000	0.003±0.002	18.638±0.002
	pstree	1.000±0.000	0.006±0.000	60.824±0.000
	pysr	1.000±0.000	0.000±0.000	192.261±0.000
	SRNet	1.000±0.000	0.004±0.002	737.712±4.201
Nguyen-4	feyn	1.000±0.000	0.005±0.002	18.388±0.002
	pstree	1.000±0.000	0.006±0.000	60.730±0.000
	pysr	1.000±0.000	0.002±0.002	187.127±0.002
	SRNet	1.000±0.000	0.002±0.000	727.843±1.304
Nguyen-5	feyn	1.000±0.000	0.003±0.000	18.312±0.000
	pstree	1.000±0.000	0.006±0.000	60.541±0.000
	pysr	1.000±0.000	0.003±0.002	238.207±0.002
	SRNet	0.968±0.021	0.169±0.054	868.095±3.607
Nguyen-6	feyn	1.000±0.000	0.001±0.000	18.778±0.000
	pstree	1.000±0.000	0.003±0.000	60.711±0.000
	pysr	1.000±0.000	0.000±0.000	191.895±0.000
	SRNet	0.999±0.000	0.030±0.008	453.927±2.105
Nguyen-7	feyn	1.000±0.000	0.000±0.000	18.675±0.000
	pstree	1.000±0.000	0.004±0.000	60.504±0.000
	pysr	1.000±0.000	0.001±0.000	189.092±0.000
		0.994±0.010	0.058±0.052	874.009±6.213
Nguyen-8	feyn	1.000±0.000	0.001±0.001	18.606±0.001
	pstree	1.000±0.000	0.004±0.000	60.095±0.000
	pysr	1.000±0.000	0.002±0.003	223.516±0.003
	SRNet	1.000±0.000	0.000±0.000	644.466±1.513
Nguyen-9	feyn	1.000±0.000	0.004±0.001	19.631±0.001
	pstree	0.992±0.000	0.091±0.000	56.906±0.000
	pysr	1.000±0.000	0.000±0.000	197.439±0.000
	SRNet	0.996±0.003	0.057±0.021	930.542±4.852
Nguyen-12	feyn	1.000±0.000	0.013±0.005	19.711±0.005
	pstree	0.993±0.000	0.081±0.000	57.050±0.000
	pysr	0.998±0.002	0.044±0.017	237.407±0.017
	SRNet	0.990±0.004	0.098±0.020	748.219±6.950

C.2. Modelo de Batería:

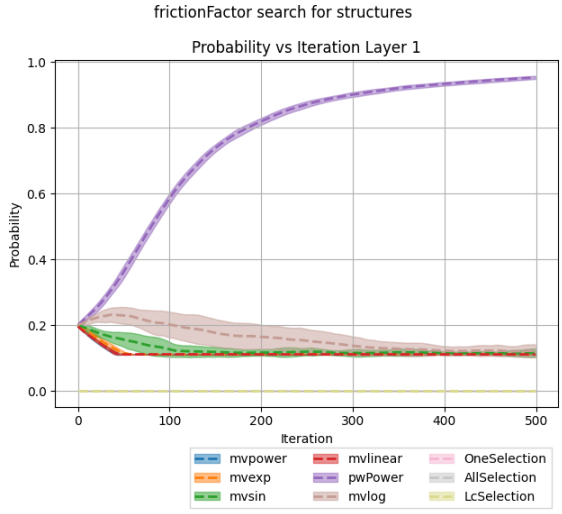
Tabla C.3: Resultados comparativos al correr métodos de regresión simbólica en el modelo de baterías.

Name	Bench	R2	NRMSE	Time [S]
c_d	feyn	0.931±0.001	0.263±0.002	28.252±0.002
	pstree	0.872±0.000	0.358±0.000	201.233±0.000
	pysr	0.912±0.009	0.296±0.014	638.052±0.014
	SRNet	0.920±0.007	0.283±0.012	1214.177±2.273
f_D	feyn	0.904±0.009	0.310±0.014	32.377±0.014
	pstree	0.868±0.000	0.364±0.000	171.544±0.000
	pysr	0.863±0.024	0.369±0.032	410.075±0.032
	SRNet	0.871±0.001	0.359±0.001	1233.898±38.781
n_u	feyn	0.991±0.000	0.096±0.002	32.705±0.002
	pstree	0.987±0.000	0.113±0.000	151.579±0.000
	pysr	0.980±0.003	0.141±0.009	327.402±0.009
	SRNet	0.970±0.013	0.168±0.037	1370.947±49.860

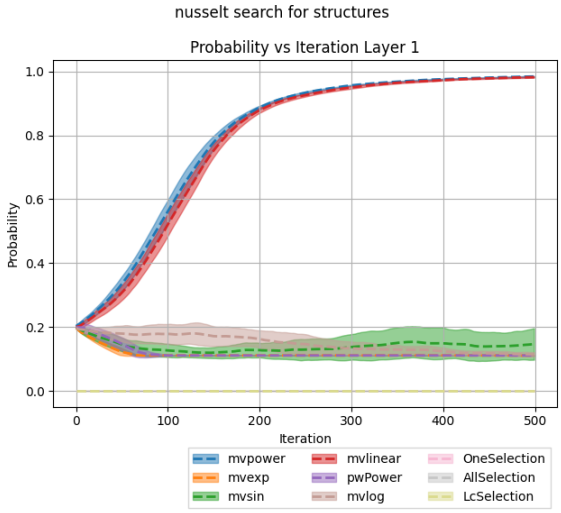
C.2.1. Búsqueda Estructural para Modelo de Baterías:



(a) Cdrag



(b) Factor de Fricción



(c) Número de nusselt

Figura C.1: Estructuras generadas mediante el algoritmo de Búesqueda estructural. La figura muestra la búsqueda de estructuras para los coeficientes Coeficiente de Arrastre, factor de fricción y número de nusselt, con las letras a), b) y c) respectivamente. Dichos resultados fueron ejecutados 10 veces para asegurar su robustez.

Nombre	SR		LBFGS-SR		BFGS	
Conjunto	Validación	Test	Validación	Test	Validación	Test
Nguyen-1	0.017±0.01826	0.014±0.01520	0.018±0.01876	0.015±0.0158	0.428±1.03736	0.415±0.9878
Nguyen-2	0.007±0.00604	0.007±0.00539	0.010±0.00745	0.008±0.0064	23.431±70.06722	209.965±629.69
Nguyen-3	0.004±0.00158	0.004±0.00163	0.009±0.00343	0.009±0.00342	0.009±0.00000	0.009±0.00000
Nguyen-4	0.002±0.00031	0.002±0.00035	0.002±0.00031	0.002±0.00035	42.747±110.148	69.169±178.56
Nguyen-5	0.175±0.06773	0.178±0.07095	0.257±0.06809	0.262±0.068	0.524±0.542	0.521±0.534
Nguyen-6	0.038±0.02097	0.038±0.02197	0.082±0.03706	0.084±0.03824	0.650±0.933	0.690±0.982
Nguyen-7	0.060±0.0520	0.059±0.051	0.128±0.13048	0.138±0.140	235531.94±706261.79	135.633±306.8
Nguyen-8	0.000±0.00000	0.000±0.00000	0.000±0.00000	0.000±0.00000	0.000±0.00000	0.000±0.00000
Nguyen-9	0.053±0.01543	0.060±0.02020	0.091±0.05984	0.095±0.05628	85527.95±256564.55	1538.870±4608.51827
Nguyen-10	0.097±0.06478	0.095±0.069	0.146±0.15234	0.140±0.14642	0.273±0.26	0.276±0.276
Nguyen-12	0.105±0.02395	0.100±0.01972	0.241±0.21335	0.247±0.23085	100.802±299.733	100.801±299.73
Constant-1	0.007±0.00091	0.006±0.00097	0.008±0.00000	0.007±0.00000	0.015±0.0134	0.014±0.01432
Constant-2	0.143±0.07701	0.147±0.07772	0.369±0.33709	0.363±0.34013	2.478±3.24031	2.424±3.17094
Constant-3	0.068±0.04	0.063±0.03731	0.074±0.038	0.066±0.03618	0.305±0.573	0.284±0.55180