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

TÉCNICAS DE OPTIMIZACIÓN DISCRETA PARA COMPRESIÓN DE REDES NEURONALES

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

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL MATEMÁTICO

FABIÁN ALEX BADILLA MERA

PROFESOR GUÍA:
GONZALO MUÑOZ MARTÍNEZ

MIEMBROS DE LA COMISIÓN:
MARCOS GOYCOOLEA GUZMÁN
FELIPE TOBAR HENRÍQUEZ

Este trabajo ha sido parcialmente financiado por Proyecto Fondecyt 11190515 y CMM ANID BASAL FB210005

SANTIAGO DE CHILE
2023

Resumen

Durante el desarrollo de esta tesis, se estudian diversas aplicaciones de un modelo de optimización entera (MIP) para la representación de redes neuronales *fully connected* con ReLU's como función de activación. Los resultados obtenidos se pueden extender a casos más generales como redes *feedforward* con funciones de activación *piecewise linear* [1].

En el Capítulo 1, se hace una introducción al problema y se presentan los conocimientos teóricos previos de optimización y aprendizaje de máquinas necesarios para el desarrollo de la tesis. También, se introduce el IP maestro que se utilizará (Big-M) y se explican sus restricciones.

En el Capítulo 2, se indican todas las precauciones técnicas que hay que tomar en cuenta para modelar una red neuronal usando este problema IP. Luego se presentan algunos algoritmos para calcular las cotas de activación, las cuales son valores asociados a cada neurona que resultan ser sumamente importante en este modelo IP. Se hacen diversos estudios empíricos para determinar cuales son las maneras apropiadas de calcularlas y que calidad tienen estas cotas calculadas.

En el Capítulo 3, se resuelve un problema de verificación de redes utilizando el modelo IP, con el fin de analizar el impacto que tienen las cotas de activación, su calidad, y el cómo fueron calculadas: versión *naive*, relajada, entera, entre otras. De este capítulo se obtiene el marco teórico ideal para el uso de este modelo IP.

En el Capítulo 4, se utilizan todos los resultados anteriores para implementar un comparador de redes neuronales, utilizando una modificación del modelo IP original. Se realizan diversas pruebas con el comparador, utilizando redes entrenadas con MNIST, y se estudia el comportamiento de las redes, realizando distintas comparaciones.

Se concluye que el desarrollo de esta herramienta es posible, tiene gran versatilidad y puede aportar a ambas comunidades (IP, ML), pero requiere estudios posteriores para analizar como escala con el tamaño de las redes, del dataset y con otras arquitecturas. Finalmente, en las conclusiones, se enumeran los principales aportes de la tesis y se mencionan líneas de trabajo futuro tanto para la comunidad IP como la de ML.

Dedicado a mi madre y mi padre que siempre apoyaron mi educación.
A mis hermanas por aguantarme tantos años.
A mis amigos por siempre estar.
Muchas gracias a todos.

Agradecimientos

Quiero agradecer a mucha gente que me ayudó durante mi formación como ingeniero, matemático y persona a lo largo de mi vida. Antes que nada a mis padres, los cuales fueron pilares fundamentales en mi educación, sobre todo a mi madre que siempre fue exigente con nosotros y trabajó toda su vida para darnos una buena educación a mis hermanas y a mí. También a mi padre que fue el que me mostró el mundo de las matemáticas a temprana edad, lo que a la larga fue incubando mi pasión por ellas, como también me enseñó otras pasiones como mi amor por Colo-Colo. También a mis hermanas, que siempre las miré con admiración, a Alejandra por hacer crecer mi lado humanista e interesarme en muchas más cosas, y a Belén por mostrarme a temprana edad como usar los computadores, piratear y *hackear* en cierta manera. A toda mi familia le agradezco de corazón.

También agradecer a todos los amigos que he hecho durante mi vida, mis amigos del colegio que hasta el día de hoy sigo viendo, como el García, Jaime, Bruno, Tomás, José, Seba, Vicho, Pato. Saber que contaba con ellos para cualquier cosa los fines de semana, sin duda que me aliviaba cuando el estrés del semestre estaba a tope. Las conversaciones que mantengo con ellos siempre me dieron otras perspectivas que no tenía y no valoraba en su momento por mi obstinación. A mis amigos del plan común, como el Cris, Torres, el Mati, Sandoval, el Diego, Harsh, Mitchel, entre muchos otros, con los que jugaba tacataca, juegos de mesa y tomaba más de lo que estudiábamos. Por último, a mis amigos del DIM, con los cuales más he compartido, como el Gonza, Pedro, Leo, Benja, Aldo, Javier, Jota, Manu, Caro, Boca, Clemun. Con algunos he pasado malos momentos, pero siempre nos apañamos estudiando para los ramos más complicados, y ahora últimamente nos vemos más en los buenos momentos, celebrando titulaciones, cumpleaños, viajando a la playa, yendo al estadio, entre otras cosas.

Por último, agradecer a mis profesores guías que me han apoyado enormemente, a Gonzalo por guiarme de muy buena manera y estar siempre atento al desarrollo de la tesis, al igual que Marcos con sus preguntas incisivas y comentarios que siempre son un gran aporte para la investigación. Ambos creyeron en mí y financiaron un viaje en el cual aprendí mucho del área y de la academia, estaré eternamente agradecido con ustedes. También agradecer a Thiago Serra, por recibirme y aportar con ideas para el desarrollo de la tesis como el comparador maxflip y al NLHPC que fueron de gran ayuda para esta tesis. Seguramente se me quedará algún nombre en el tintero, en cuyo caso les agradezco igualmente por aportar en mi formación, sepan perdonarme pero debo respetar el formato de la U jajaja.

Powered@NLHPC: Esta investigación/tesis fue parcialmente apoyada por la infraestructura de supercómputo del NLHPC (CCSS210001)

Tabla de Contenido

Capítulo 1

Introducción al problema y definiciones

1.1. Contexto del problema y motivaciones

En los últimos años, con los avances tecnológicos y las constantes mejoras en capacidad de cómputo, las redes neuronales se han vuelto cada vez más importantes y factibles de utilizar en la vida diaria. Se observan sus aplicaciones en prácticamente todo ámbito, como en la salud, ciencias, economía, entretención entre otros [11, 6]. Es por esto, que el desarrollo de técnicas que permitan evaluarlas y mejorarlas es de suma importancia, por ejemplo, con el paso de los años, cada vez se utilizan redes neuronales más complejas y de mayor tamaño, por lo tanto se vuelve relevante desarrollar técnicas para comprimirlas y aumentar su eficiencia [12].

En esta tesis, se estudiará el uso de modelos MIP para la representación de redes neuronales. Se analizará que tan factible es su uso, sus principales ventajas, desventajas, qué factores se deben considerar y finalmente sus posibles aplicaciones. Para ello, en este capítulo se introducirán distintas definiciones y convenciones que se utilizarán a lo largo de toda la tesis. Éstas abarcan temas básicos de *Machine Learning* y Programación Lineal Mixta.

Notación: Durante el texto se usarán las siguientes convenciones:

i) $[n]_0 = \{0, ..., n\}$; $[n] = \{1, ..., n\}$

ii) Sean $x, y \in \mathbb{R}^n : x \preceq y \iff \forall i \in [n] : x_i \leq y_i$. Análogamente se define $x \succeq y$

1.2. Redes neuronales feedforward

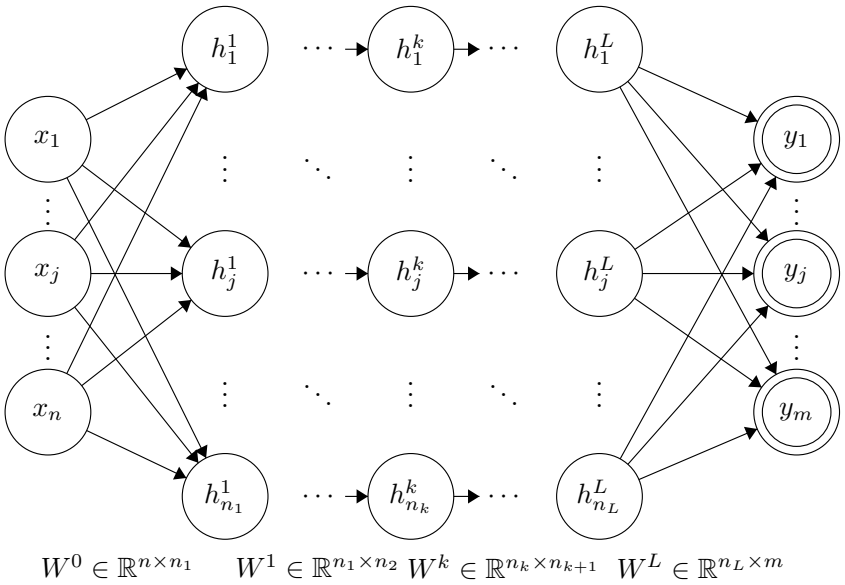


Figura 1.1: Diagrama de una red neuronal feedforward

Una red neuronal feedforward, es un objeto matemático que, como indica su nombre, está compuesto de una unidad básica llamada *neurona* o *nodo*, las cuales están agrupadas en diferentes capas $l \in [L]_0$, con L el número de capas escondidas, que van procesando una señal de una capa a la siguiente.

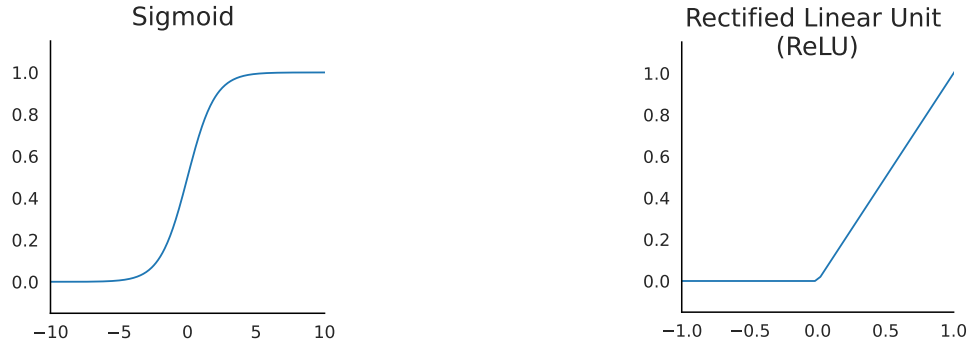
A la primera capa se le denomina capa inicial o *input* y se le denota como $x = (x_1, ..., x_n)^\top$, a la última capa se le denomina capa final o *output* y se le denota como $y = (y_1, ..., y_m)^\top$, a las capas intermedias se les llama capas escondidas o *hidden layers*, las cuales se le denotará como $h^{(l)} = (h_1^{(l)}, ..., h_{n_l}^{(l)})$. Al número de nodos de la capa l se le denomina n_l y este valor puede variar de capa a capa. Cada neurona de la red queda determinada por el par (j, l) que indica que es la j-ésima neurona de la capa l.

Existen también otro tipo de redes que no funcionan de esta forma, como las redes recurrentes, pero no serán objeto de estudio en esta tesis.

Cada neurona (j, l) con $j \in [n_l]$, $l \in [L]_0$, tiene asociada también una *función de activación* $f_j^{(l)} : \mathbb{R} \to \mathbb{R}$, la cual determina si la neurona está *activa* o *inactiva*. Por ejemplo, dada una señal inicial, en cada nodo se obtiene un valor $a_j^{(l)} \in \mathbb{R}$, denominado *activación*, y determinado por los valores de la capa previa. A este valor se le aplica la función de activación obteniendo $h_j^{(l)} = f(a_j^{(l)})$, que se transmite a la siguiente capa. Se dice que la neurona está activa si $h_j^{(l)} > 0$ e inactiva sino.

Existen muchos ejemplos de funciones de activación. Históricamente una de las más usadas fue la sigmoide [2], definida por $f(x) = (1 + e^{-x})^{-1}$ y representada en la Figura 1.2a. Otra función de activación muy utilizada es la unidad lineal rectificada (ReLU por sus siglas en

inglés) [6], definida por $f(x) = \text{máx}(0, x)$ y representada en la Figura 1.2b. Generalmente se utiliza una misma función de activación en las neuronas de una misma capa.



(a) Ejemplo de una sigmoide

(b) Ejemplo de una ReLU

Figura 1.2: Ejemplo de distintas funciones de activación

También, existen muchos tipos de capas, como las capas de *pooling*, las capas de convolución, entre otras. Durante esta tesis, se utilizarán capas *fully connected*, como las que se observan en la Figura 1.1, donde cada arista del grafo une un nodo en la capa l con otro en la capa $l + 1$. En este tipo de capas, la interacción entre los nodos de una capa con la siguiente, está determinada por los *pesos* de cada arista; estos pesos se representan con una matriz $W^{(l)} \in \mathbb{R}^{n_l \times n_{l+1}}$ que premultiplica a los nodos $h^{(l)} \in \mathbb{R}^{n_l}$. Al haber muchas aristas y nodos involucrados en la descripción de estas capas (dos capas consecutivas forman un grafo bipartito completo), sus parámetros suelen gastar espacio considerable en disco, y las redes terminan siendo costosas de entrenar.

Observación 1.1 Algunas capas, como las de convolución y *average pooling*, pueden ser representadas como una fully connected si se definen ciertas relaciones sobre sus pesos. Por lo tanto, muchos de los procedimientos y resultados que se verán en los siguientes capítulos, pueden ser aplicados a redes con dichas capas.

Definición 1.1 *Se definen las siguientes funciones que se utilizarán durante la tesis:*

$$\begin{aligned} relu : \mathbb{R} &\to \mathbb{R}_+ \\ x &\mapsto \text{máx}(0, x) \\ softmax : \mathbb{R}^K &\to [0,1]^K \\ \vec{x} &\mapsto \left(\frac{e^{x_i}}{\sum_{k=1}^{n} e^{x_k}} \right)_{i \in [K]} \\ LogSoftmax : \mathbb{R}^K &\to \mathbb{R}_-^K \\ \vec{x} &\mapsto (\log(softmax(x)_i))_{i \in [K]} \end{aligned}$$

Con todo este preámbulo, se puede definir más formalmente una red neuronal como sigue,

Definición 1.2 *Se llamará red neuronal fully connected (FCNN) de L capas escondidas, a la tupla $\mathcal{N} = (\mathcal{W}, \mathcal{B}, \mathcal{F})$ donde:*

1. $\mathcal{W} = \{W^{(0)}, ..., W^{(L+1)}\}$ *son las matrices de pesos de cada capa,*
2. $\mathcal{B} = \{b^{(0)}, ..., b^{(L+1)}\}$ *son los vectores de sesgos de cada capa,*
3. $\mathcal{F} = \{f^{(1)}, ..., f^{(L+1)}\}$ *son las funciones de activación de cada capa,*

Cada capa $h^{(l)}$ con $l \in [L+1]_0$ está compuesta de n_l nodos, con $x = h^{(0)}$ e $y = h^{(L+1)}$ las capas input y output respectivamente, y cada nodo $h_j^{(l)}$ tendrá un valor dado por $h_j^{(l)} = f^{(l)}({W_j^{(l-1)}}^\top h^{(l-1)} + b_j^{(l-1)})$. Este tipo de redes también se les conoce como perceptrón multicapa o MLP.

En general, se asumirá que $\forall l \in [L] : f^{(l)} = relu$ y $f^{(L+1)} = softmax$ y se denotará simplemente como $\mathcal{N} = (\mathcal{W}, \mathcal{B})$.

Observación 1.2 Las redes neuronales también pueden ser interpretadas como funciones $\mathcal{N} : \mathbb{R}^n \to \mathbb{R}^m$, por lo que muchas veces se abusará de la notación usando expresiones como $y = \mathcal{N}(x)$ o similares.

Observación 1.3 Este tipo de redes se aplica generalmente para resolver problemas de clasificación de más de una clase, como el procesamiento de imágenes [2]. Por ejemplo, si se asume que hay un conjunto de datos $\mathcal{D} = \{\mathcal{X}, \mathcal{Y}\}$, donde $x \in \mathcal{X}$ es la información que procesará la red, e $y_R \in \mathcal{Y}$ es la clase real a la que corresponde x, luego la red clasifica en base a los valores que tome su última capa $y = \mathcal{N}(x)$, la cual puede ser distinta a y_R. Esto último determinará que tan buena es la red para clasificar.

Observación 1.4 Para el caso de $softmax$, se pueden interpretar los valores que entrega como una densidad de probabilidad para cada clase, donde K es el número de clases. Muchas veces se utiliza en cambio $LogSoftmax$ por las buenas propiedades que tiene para los algoritmos de optimización (convexidad), pero perdiendo la interpretabilidad.

1.2.1. Entrenamiento y validación

Otra noción esencial que se necesita introducir, es el proceso de entrenamiento, prueba y validación de las redes neuronales. Como ya se mencionó, las redes quedan definidas por sus matrices de pesos y sesgos, de los cuales aun no se ha mencionado como son determinados. En primera instancia, estos parámetros pueden tomar cualquier valor, pero dependiendo de cual es la finalidad con la que se quiere utilizar la red, se puede considerar que algunas redes son "mejores que otras".

En este sentido, para mejorar una red, se lleva a cabo el proceso de entrenamiento. En esta sección se pondrá el foco en el aprendizaje supervisado, pero existen muchos otros tipos, como el no supervisado (que usan los algoritmos de clusterización), el semi-supervisado, reinforcement learning, entre otros. Como referencia se puede leer la sección de entrenamiento de Bishop [2], o la sección de algoritmos de aprendizajes de Goodfellow et al [6].

En el aprendizaje supervisado, se asume que existe un conjunto de datos $\mathcal{D} = (\mathcal{X}, \mathcal{Y})$ de tamaño N (idealmente muy grande), donde $\mathcal{X}$ es un conjunto de observaciones o datos de entrenamiento e $\mathcal{Y}$ es un conjunto de etiquetas para cada observación. Luego, el proceso de entrenamiento consiste generalmente en minimizar la diferencia entre las etiquetas objetivo dadas por $\mathcal{Y}$ y las etiquetas calculadas por la red neuronal dadas por $\mathcal{N}(\mathcal{X})$. Esto se hace "entrenando" el modelo durante varias épocas, donde cada época es una pasada completa por todos los datos de entrenamiento.

Para esto, es necesario definir una función de pérdida $L(\mathcal{N}, \mathcal{X}, \mathcal{Y})$ para ser usada junto con algún algoritmo de optimización. Un ejemplo común, es utilizar el error cuadrático medio (MSE por sus siglas en inglés) como función de pérdida, junto con el método de descenso de gradientes estocástico (SGD). Otras opciones más comunes actualmente, son usar la entropía cruzada junto con algún algoritmo de optimización como Adam (nombre derivado de *Adaptive moment estimation*)[9].

Definición 1.3 *Sean $\mathcal{N} = (\mathcal{W}, \mathcal{B})$, $\mathcal{D} = (\mathcal{X}, \mathcal{Y})$ con $\mathcal{X} = \{x_1, ..., x_N\}$, $\mathcal{Y} = \{y_1, ..., y_N\}$ e $y_j \in [K]$. Las siguientes son algunas funciones de pérdida estándar:*

$$MSE(\mathcal{N}, \mathcal{D}) = \frac{1}{2} \sum_{n=1}^{N} \sum_{k=1}^{K} \left(\mathcal{N}(x_n)_k - \mathbb{1}_{(y_n=k)}\right)^2 \qquad \text{(Error cuadrático medio)}$$

$$CE(\mathcal{N}, \mathcal{D}) = -\sum_{n=1}^{N} \sum_{k=1}^{K} \ln(\mathcal{N}(x_n)_k) \cdot \mathbb{1}_{(y_n=k)} \qquad \text{(Entropía Cruzada)}$$

Backpropagation, testeo y validación

Uno de los procedimientos más utilizados en ML para entrenar un modelo, es el aprendizaje usando la propagación hacia atrás o *back-propagation*. Este consiste en usar las observaciones en la red para generar predicciones, luego calcular "hacia atrás" los gradientes de la función de pérdida con respecto a los pesos usando la regla de la cadena, para finalmente minimizar los errores de la función objetivo[14]. De esta forma, en cada iteración, se van actualizando los parámetros de la red, para que las predicciones sean más similares a los valores reales.

Este método trae consigo algunos problemas, como el *underfitting*, el cual ocurre cuando el modelo no alcanza a tener un error suficientemente bajo en el conjunto de entrenamiento, ya sea por falta de datos, épocas de entrenamiento u otros motivos. En el caso contrario,

ocurre el *overfitting* el cual genera que las redes queden demasiado determinadas por los datos de entrenamiento, perdiendo así capacidad de interpolación y poder predictivo [6]. En la Figura 1.3 se puede apreciar un ejemplo de este fenómeno.

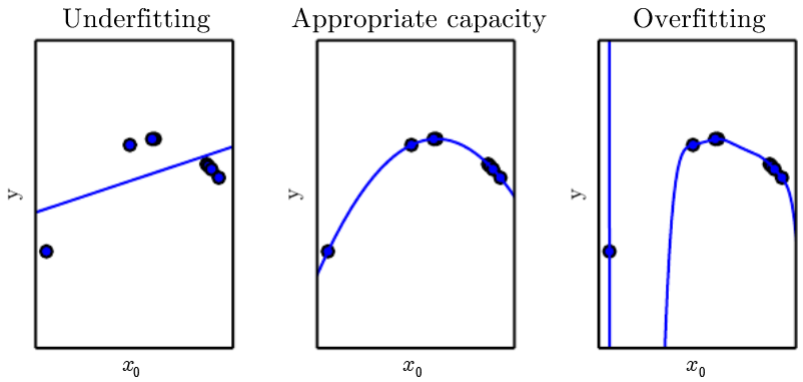


Figura 1.3: Ejemplo ilustrativo del fenómeno, donde se observa una curva siendo ajustada a un conjunto de datos x_0. Fuente: Goodfellow et al [6].

Para evitar el overfitting, es común separar los datos $\mathcal{D}$ en 2 o 3 conjuntos, el de entrenamiento $\mathcal{D}_{train}$, el de validación $\mathcal{D}_{val}$ y el de prueba $\mathcal{D}_{test}$. Los primeros dos, se utilizan durante el proceso de entrenamiento, donde primero se define un número de épocas n_{epochs}, y en cada época $n_e \in [n_{epochs}]$ se le entregan a la red todas las observaciones del conjunto $\mathcal{D}_{train}$, con las cuales se van ajustando los parámetros de la red. Luego, se utiliza el conjunto de validación $\mathcal{D}_{val}$, junto con alguna métrica de desempeño, para compararlo con el conjunto de entrenamiento. De esta forma, se lleva un registro de los desempeños en ambos conjuntos durante las épocas del entrenamiento y se ajustan los parámetros de acuerdo a la función de pérdida. En la Figura 1.4, se puede apreciar un gráfico del proceso de entrenamiento estándar, donde se utilizó un conjunto de entrenamiento y validación.

Al final, si se observa que en ambos conjuntos hay desempeños similares, se considera que no hay overfitting. En cambio, si la red presenta un desempeño mucho mejor en el conjunto de entrenamiento que en el de validación, se considera que la red está demasiado determinada por los datos de entrenamiento. Para solucionar esto existen diversos métodos, los más comunes son aumentar los datos del conjunto de entrenamiento (*data augmentation*), cambiar el número de épocas con algún criterio de early stopping, y/o agregar regularización en el entrenamiento, la cual consiste en agregar un factor de penalización en la función de pérdida, con el objetivo de aumentar los parámetros con valor igual a cero. A esto se le llamará también aumentar *sparsity* [6].

Finalmente, para poner a prueba el desempeño de la red se utiliza alguna métrica en el último conjunto $\mathcal{D}_{test}$, el cual no se ha utilizado hasta el momento, entregando así una evaluación insesgada de la red.

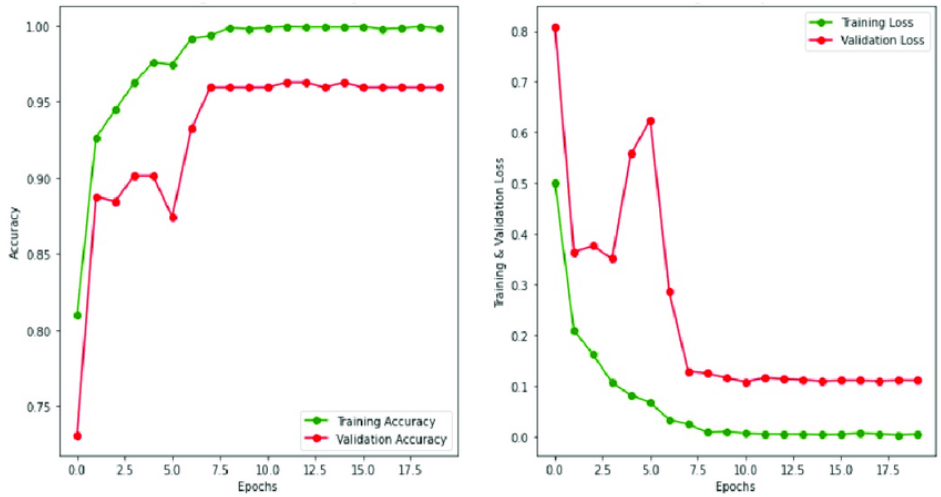


Figura 1.4: Ejemplo del proceso de entrenamiento/validación. Fuente

Observación 1.5 El conjunto de prueba, sólo se usa para la métrica de desempeño (sin cambiar los gradientes de los pesos). Un error técnico común en este proceso, es usar la red en su modo de entrenamiento cuando se está evaluando el conjunto de prueba, lo cual genera que se propaguen los errores, afectando así los valores de los parámetros.

1.2.2. Métricas de desempeño

Para determinar si una red es mejor que otra, es necesario definir *métricas* para evaluar el desempeño de las redes. Estás métricas generalmente se usan en el proceso de validación y prueba de las redes luego de ser entrenadas. Existen varias métricas en la literatura; como esta tesis se enfocará principalmente en problemas de clasificación, se usará recurrentemente la siguiente métrica de *accuracy*.

Definición 1.4 *Dado un dataset* $\mathcal{D} = \{\mathcal{X}, \mathcal{Y}\}$ *con* $\mathcal{X} = \{x_1, ..., x_N\}$, $\mathcal{Y} = \{y_1, ..., y_N\}$, *se define el accuracy de una red* $\mathcal{N}$ *como:*

$$acc(\mathcal{N}, \mathcal{D}) := \frac{\#\{y_n \mid y_n = \mathcal{N}(x_n)\}}{N} \in [0, 1]$$

Donde $\#A$ *denota el cardinal de* A.

Observación 1.6 Notar que esta métrica tiene problemas con los falsos positivos y negativos. Si hay un conjunto $\mathcal{D}$ que es demasiado desequilibrado, a una red le bastaría clasificar todos los input con la clase mayoritaria para tener un buen puntaje. Afortunadamente, en los datasets que se ocuparán más adelante, no ocurre esto.

Observación 1.7 En el gráfico de la izquierda en la Figura 1.4, se observa el uso de ésta métrica de desempeño para determinar que no haya overfitting.

1.3. Modelo MIP para redes neuronales

Debido a la estructura *piecewise linear* de algunas redes, como las que utilizan a la ReLU de funcion de activación [18], recientemente se ha impulsado el análisis *post-training* mediante la programación entera mixta (MIP). Con este apronte se ha logrado hacer verificación de robustez en redes [20], conteo de regiones lineales [16] y también compresión de redes [15]. Este último es un tema en el que se pondrá más énfasis durante el desarrollo de ésta tesis.

Una manera de modelar una red neuronal dentro de un problema de optimización, es utilizar una formulación "Big M", la cuál en general tiene una estructura similar a la del problema detallado más adelante: (Big-M). En este caso particular, las restricciones del problema modelan una red fully connected con ReLU's como funciones de activación, pero con algunos cambios se pueden usar otras funciones suaves como *tanh*, la sigmoide, etc.

Además, existen otras formulaciones como las basadas en particiones que utilizan *disjunctive programming* [21], pero a lo largo de esta tesis solo se estudiarán algunas variaciones del modelo (Big-M), principalmente con cambios en la función objetivo y la cantidad de redes modeladas simultáneamente.

$$
\begin{aligned}
\max_{x\in\mathbb{X}} \quad & F(a,h,z) && && \text{(Big-M)}\\
\text{s.a.} \quad & {W^{(l)}}^{\top} h^{(l)} + b^{(l)} = a^{(l+1)} && \forall l \in [L]_0 && (1.1)\\
& a^{(l)} = h^{(l)} - \overline{h}^{(l)} && \forall l \in [L] && (1.2)\\
& h_j^{(l)} \leq H_j^{(l)} z_j^{(l)} && \forall l \in [L], \forall j \in [n_l] && (1.3)\\
& \overline{h}_j^{(l)} \leq \overline{H}_j^{(l)} (1 - z_j^{(l)}) && \forall l \in [L], \forall j \in [n_l] && (1.4)\\
& z^{(l)} \in \{0,1\}^{n_l} && \forall l \in [L] && (1.5)\\
& h^{(l)} \succeq 0 && \forall l \in [L] && (1.6)\\
& \overline{h}^{(l)} \succeq 0 && \forall l \in [L] && (1.7)\\
& h^{(0)} = x; \quad h_j^{(0)} \in [-\overline{H}_j^{(0)}, H_j^{(0)}] && \forall j \in [n] && (1.8)
\end{aligned}
$$

Donde se tiene que:

- $x = (x_1, ..., x_n)^{\top} \in \mathbb{X}$ es el input y el conjunto donde está definida la red.
- F es una función objetivo que se utilizará dependiendo de qué se busca estudiar. Puede ser una propiedad sobre el output de la red, de los patrones de activación, algunas capas intermedias, etc.
- $W^{(l)}$ es la matriz de pesos de tamaño $n_l \times n_{l+1}$ y $b^{(l)}$ el vector de sesgos asociados de tamaño n_l, con $l \in [L]_0$ las capas de la red,

- $a^{(l)}$ son las activaciones de la capa l.
- $z_j^{(l)}$ es una variable binaria que indica si la neurona (j, l) está activa o no.
- $h_j^{(l)}, \overline{h}_j^{(l)}$ son variables que se utilizan para representar la ReLU, donde $h_j^{(l)}$ toma el valor de la ReLU y $\overline{h}_j^{(l)}$ es la parte negativa de $a_j^{(l)}$. Así, $h^{(l)} = (h_1^{(l)}, h_2^{(l)}, ..., h_{n_l}^{(l)})^\top$ representa el output de la capa escondida $l \in [L]$.
- $H_j^{(l)}, \overline{H}_j^{(l)} \geq 0$ son las cotas Big M del problema, es decir, la cota superior para las variables $h_j^{(l)}$ y $\overline{h}_j^{(l)}$ respectivamente. Esto se puede interpretar como el rango de valores que puede tomar $a_j^{(l)}$, el cual queda dado por $\left[-\overline{H}_j^{(l)}, H_j^{(l)}\right]$. A estas se les llamarán las *cotas de activación*. Notar que basta que estas cotas sean válidas para que el problema esté bien definido, es decir, a priori se pueden usar valores tan grandes como sean necesarios.

La restricción (1.1), es la definición de las activaciones en una FCNN. Las restricciones (1.2)-(1.7) permiten modelar el comportamiento de la ReLU en cada nodo (j, l), ya que si $z_j^{(l)} = 0$, se tiene por (1.3) y (1.6) que $h_j^{(l)} = 0$ y que $\overline{h}_j^{(l)} \in [0, \overline{H}_j^{(l)}]$ por (1.4) y (1.7).

Así, gracias a (1.2) se tiene que $a_j^{(l)} \in [-\overline{H}_j^{(l)}, 0]$, es decir la neurona está inactiva y se cumple que $ReLU(a_j^{(l)}) = 0 = h_j^{(l)}$. De manera similar, si $z_j^{(l)} = 1$ ocurre lo mismo pero intercambiando los roles de $h_j^{(l)}$ con $\overline{h}_j^{(l)}$. Así, se concluye que la neurona está activa notando nuevamente que $ReLU(a_j^{(l)}) = h_j^{(l)}$.

Por último, la restricción (1.8) es una condición inicial para que (1.1) esté bien definida cuando $l = 0$ y que une todo el problema con el input $x \in \mathbb{X}$. Durante el desarrollo de esta tesis, se asumirá que $\mathbb{X} \subseteq \mathbb{R}^n$ es un conjunto acotado, ya que el modelo será ocupado para el procesamiento de imágenes, las cuales están delimitadas por un numero de pixeles n y cada pixel tiene valores que se mueven dentro de un intervalo $[l, u]$.

Esta suposición es bastante estándar por cómo se estructuran los datos en los computadores, así que no es algo muy fuerte para pedirle al modelo y se puede asumir lo mismo para problemas más generales que el procesamiento y clasificación de imágenes.

1.3.1. Alcances del modelo

Es sabido que probar cualquier propiedad en una red neuronal es una tarea compleja, ya que se demostró que pertenece a NP-Hard [8]. Esto justifica la necesidad de tener un modelo como (Big-M).

Este modelo tiene varias características que lo hacen deseable. Por ejemplo, es un tipo de modelo muy usado en la comunidad MIP, por lo que hay muchas técnicas de optimización

que se conocen y se pueden aplicar para resolver el problema. Es también un modelo muy versátil, que teóricamente permite probar muchas propiedades en las redes neuronales. Como se mencionó anteriormente, se puede generalizar para más redes como las que tienen funciones de activación lineales por tramos. Si se fijan relaciones entre los pesos de W se puede utilizar en redes con capas convoluciones o capas de average pooling entre otras.

Sin embargo, este modelo tiene algunas desventajas. Como es un modelo Big-M, este es muy sensible a las cotas $H^{(l)}, \overline{H}^{(l)}$, ya que para resolver el MIP es deseable tener buenas relajaciones lineales del problema, y en este caso mientras más grandes sean las cotas Big M, peor será el LP asociado a la relajación de las variables $z_j^{(l)}$. Esto es, debido a que los valores posibles para $h_j^{(l)}$ y $\overline{h}_j^{(l)}$ crece enormemente en función de dichas cotas.

También, definir los mejores valores de $H^{(l)}, \overline{H}^{(l)}$ no es un trabajo sencillo. En principio, como se asumirá que $\mathbb{X}$ es un conjunto acotado, se podría usar una cota suficientemente grande para asegurar la validez del modelo, pero encontrar cotas que sean suficientemente pequeñas para que la relajación lineal sea buena y además el modelo original sea válido, resultó ser una tarea muy compleja y susceptible a muchos factores como se verá en detalle en el próximo capítulo.

1.4. Objetivos y esquema general de la investigación

El objetivo principal de esta tesis, es generar redes neuronales comprimidas y evaluar sus pérdidas mediante métodos de programación lineal entero. Para esto, como primer paso, se analiza el efecto del *prunning* o redondeo en los modelos MIPs de representación de redes neuronales. Se estudian problemáticas como: ¿Mejora el desempeño del modelo al restar pesos de las matrices? ¿Sigue siendo válido?, ¿Cuándo se puede hacer esto?.

Luego se estudia el efecto de las cotas $H, \overline{H}$ del modelo (Big-M), que importancia tiene lo ajustadas que son las cotas, cómo calcularlas, que precauciones hay que tener en este tipo de problemas. Para esto, se hacen diversas comparaciones y a través de resultados empíricos se determina una guía a seguir en el uso de MIPs en redes neuronales. En estos capítulos se obtienen resultados relevantes para la comunidad MIP, pues se encontraron dificultades técnicas a la hora de implementar y también que hacer para evitarlos.

Por último, se proponen modelos para comparar redes neuronales que serán de utilidad para estudiar la compresión de redes. Este comparador en principio tiene muchas utilidades, como por ejemplo chequear cuanto cambia la red neuronal comprimida respecto a la original y asegurar un margen de error máximo, así si se asume que la red original es suficientemente buena (bajo alguna métrica) se puede asegurar posteriormente que hay una versión comprimida que entrega resultados suficientemente similares.

También, en caso contrario, si ambas redes difieren mucho, este comparador proporcio-

naría un caso adversarial, que es útil al momento de certificar problemas en la compresión.

Este comparador, considerando que es un MILP, probablemente sufrirá algunos problemas de escalabilidad debido a su alto número de variables y restricciones. Basado en trabajos previos donde se utilizan versiones más generales de este MILP [19, 1], y le hacen *speedups* ajustando las restricciones del LP asociado, se pueden mejorar los tiempos de cómputo para redes más grandes. Igualmente, aún en el caso donde no se alcance la optimalidad, este comparador seguiría siendo de utilidad ya que proporciona cotas de optimalidad.

Capítulo 2

Cálculo de las cotas de activación

2.1. Introducción al problema

Con el fin de entender mejor qué ocurre en cada neurona de una red, poder aplicar un modelo de comparación de redes y posteriormente desarrollar técnicas de compresión de redes neuronales, se hace necesario calcular, en primera instancia, el rango de valores que recibirán las funciones de activación en cada neurona del modelo (Big-M). Esta tarea resulta ser altamente no trivial, pues hay muchos factores a considerar y el modelo es muy sensible a problemas prácticos como la cantidad de pesos, el orden de magnitud de los mismos, como fueron entrenadas las redes (con o sin regularización), entre otras cosas.

Por estas razones, se ocupará una versión modificada del problema (Big-M), con el objetivo de calcular las cotas de activación de cada neurona. Esto es, para cada nodo (j, l) se determinará el máximo y mínimo valor que puede tener la combinación lineal:

$$a_j^{(l+1)} = {W_j^{(l)}}^\top h_j^{(l)} + b_j^{(l)}$$

Estas cotas serán muy útiles para las aplicaciones que se verán en esta tesis, pero también son relevantes para cualquier modelo MIP que posea una estructura similar a la de (Big-M), pues se pueden usar en el rol de $\{H_j^{(l)}, \overline{H}_j^{(l)}\}_j^l$. Hasta el momento, en la literatura, no se ha hecho un análisis detallado del impacto de éstas cotas en este modelo MIP para redes neuronales, es por ello que el estudio de cómo calcularlas y analizar los efectos que tienen sobre el modelo es de suma relevancia. Todo esto también motivado por lo que se discutió brevemente en el Capítulo 1 sobre la importancia de tener relajaciones lineales fuertes en este tipo de problemas.

Así, el objetivo de este capítulo es desarrollar una forma de obtener cotas para $a_j^{(l)}$ que sean lo más ajustadas posibles, de manera que al relajar el problema maestro (Big-M) con dichas cotas, esta relajación sea fuerte.

Para calcular la cota superior en el nodo $n \in [n_m]$ de la capa $m \in [L]$, se utilizará el siguiente MILP.

$$
\begin{aligned}
\max_{x \in \mathbb{X}} \quad & F(a,h,z) = a_n^{(m)} && && (\overline{P_{nm}}) \\
\text{s.a.} \quad & {W^{(l)}}^\top h^{(l)} + b^{(l)} = a^{l+1} && \forall l \in [m-1]_0 && (2.1) \\
& a^{(l)} = h^{(l)} - \overline{h}^{(l)} && \forall l \in [m-1] && (2.2) \\
& h_j^{(l)} \leq H_j^{(l)} z_j^{(l)} && \forall l \in [m-1], \forall j \in [n_l] && (2.3) \\
& \overline{h}_j^{(l)} \leq \overline{H}_j^{(l)} (1 - z_j^{(l)}) && \forall l \in [m-1], \forall j \in [n_l] && (2.4) \\
& h^{(l)}, \overline{h}^{(l)} \succeq 0 && \forall l \in [m-1] && (2.5) \\
& z^{(l)} \in \{0,1\}^{n_l} && \forall l \in [m-1] && (2.6) \\
& h^{(0)} = x; \quad h_j^{(0)} \in [-\overline{H}_j^{(0)}, H_j^{(0)}] && \forall j \in [n] && (2.7)
\end{aligned}
$$

Donde se modifican las restricciones de (Big-M), para modelar el comportamiento de la red hasta la capa m y se utiliza como función objetivo el valor de la activación $a_n^{(m)}$. Así, al maximizar se está buscando precisamente el máximo valor que puede llegar a tener el nodo (n, m), o sea, el valor para $H_n^{(m)}$.

Notar que, si en cambio se minimiza $F(a, h, z)$, se puede obtener la cota inferior para la activación del nodo (n, m). A este MILP se le denominará $(\underline{P_{nm}})$.

$$
\begin{aligned}
\min_{x \in \mathbb{X}} \quad & F(a,h,z) = a_n^{(m)} && (\underline{P_{nm}}) \\
\text{s.a.} \quad & (2.1) - (2.7)
\end{aligned}
$$

2.1.1. Algoritmo de resolución *Bounder*

En ambas formulaciones, tanto $(\underline{P_{nm}})$ como en $(\overline{P_{nm}})$, hace falta definir los valores de $H^{(l)}$ y $\overline{H}^{(l)}$ en las capas $l < m$ para poder resolver el problema, pero como $\mathbb{X}$ es acotado, es fácil notar que se pueden obtener recursivamente fijándolos para $l = 0$ con el rango admitido en $\mathbb{X}$. Por ejemplo, con el dataset MNIST normalizado se tendría $\mathbb{X} = [0,1]^{28 \times 28}$, i.e., $\forall j \in [784] : H_j^{(0)} = 1$ y $\overline{H}_j^{(0)} = 0$. Con esto, se pueden obtener los valores de $H^{(l)}$ y $\overline{H}^{(l)}$ para la capa $l = 1$, y así recursivamente encontrar todas las cotas de activación de la red.

Lo anterior también permite controlar el input, por ejemplo, si se quiere analizar cómo se comporta una red para imágenes similares a un dígito d en particular, se pueden fijar estos valores iniciales en una bola de radio ε en torno al dígito $d \in \mathbb{X}$. Esta idea será utilizada en capítulos posteriores.

Así, se puede construir el siguiente algoritmo para conseguir las cotas de activación de todos los nodos:

Algorithm 1: Bounder

Input: $(\mathcal{W}, \mathcal{B})$, $\overline{H}^{(0)}, H^{(0)}$

Output: $\left\{\overline{H}^{(l)}, H^{(l)}\right\}_{l=1}^{L}$

```
Definir las variables del modelo (h^(l), h̄^(l), a^(l), z^(l)) ∀l ∈ [L] con (2.5)-(2.6), a^(l) ∈ ℝ^{n_l}
Agregar la restricción (2.7), usando los valores iniciales H̄^(0), H^(0)
Definir cotas validas H^(l), H̄^(l) ∀l ∈ [L] usando la Proposición 2.1
for 1 ≤ m ≤ L do
    Agregar restricciones (2.1) − (2.4) asociadas a la capa m
    for 1 ≤ n ≤ n_m do
        Cambiar objetivo al a_n^(m) actual
        Resolver (P̄_nm) y (P_nm) asociados
        if Se encuentran cota para a_n^(m) then
            Definir H_n^(m) y H̄_n^(m) con los valores correspondientes
        else
            Propagar peor cota de capa m − 1 con la Proposición 2.2.
Retornar {H̄^(l), H^(l)}_{l=0}^{L}
```

Para obtener las cotas iniciales en caso de alcanzar un límite de tiempo, se utiliza la siguiente proposición:

Proposición 2.1 *La siguiente es una cota válida para las neuronas de la capa $l \geq 2$:*

$$||h^{(l)}||_\infty \leq ||h^{(0)}||_\infty \prod_{m=0}^{l-1} ||W^{(m)}||_\infty + \sum_{n=0}^{l-2} \left(||b^{(n)}||_\infty \prod_{m=n+1}^{l-1} ||W^{(m)}||_\infty \right) + ||b^{(l-1)}||_\infty$$

Para $l = 1$ basta tomar $||h^{(1)}||_\infty \leq ||W^{(0)}||_\infty ||h^{(0)}||_\infty + ||b^{(0)}||_\infty$.

Observación 2.1 Esta cota en general es muy grande, pero al menos asegura cotas factibles $H_j^{(l)}, \overline{H}_j^{(l)}$ para los problemas $(\overline{P_{nm}})$, $(\underline{P_{nm}})$. Notar también, que si nuestros datos están normalizados, se tiene que $x \in \mathbb{X} = [0,1]^{28\times 28}$, así $||h^{(0)}||_\infty = ||x||_\infty = 1$ simplificando el primer término.

Demostración. *Caso base $l = 2$:*

Como $h_j^{(1)} = \text{máx}({W_j^{(0)}}^\top h^{(0)} + b_j(0))$

$$\begin{aligned} \implies ||h^{(1)}||_\infty &\leq ||{W^{(0)}}^\top h^{(0)} + b^{(0)}||_\infty \\ &\leq ||W^{(0)}||_\infty ||h^{(0)}||_\infty + ||b^{(0)}||_\infty \end{aligned}$$

Luego para $l = 2$ se tiene que:

$$\begin{aligned}
||h^{(2)}||_\infty &\leq ||{W^{(1)}}^\top h^{(1)} + b^{(1)}||_\infty \leq ||W^{(1)}||_\infty ||h^{(1)}||_\infty + ||b^{(1)}||_\infty \\
&\leq ||W^{(1)}||_\infty \left(||W^{(0)}||_\infty ||h^{(0)}||_\infty + ||b^{(0)}||_\infty \right) + ||b^{(1)}||_\infty \\
&\leq ||W^{(1)}||_\infty ||W^{(0)}||_\infty ||h^{(0)}||_\infty + ||W^{(1)}||_\infty ||b^{(0)}||_\infty + ||b^{(1)}||_\infty \\
&\leq ||h^{(0)}||_\infty \prod_{m=0}^{1} ||W^{(m)}||_\infty + ||b^{(0)}||_\infty ||W^{(1)}||_\infty + ||b^{(1)}||_\infty
\end{aligned}$$

Caso general $l + 1$:

$$\begin{aligned}
&||h^{(l+1)}||_\infty = ||{W^{(l)}}^\top h^{(l)} + b^{(l)}||_\infty \leq ||W^{(l)}||_\infty ||h^{(l)}||_\infty + ||b^{(l)}||_\infty \\
&\leq ||W^{(l)}||_\infty \left(||h^{(0)}||_\infty \prod_{m=0}^{l-1} ||W^{(m)}||_\infty + \sum_{n=0}^{l-2} \left(||b^{(n)}||_\infty \prod_{m=n+1}^{l-1} ||W^{(m)}||_\infty \right) + ||b^{(l-1)}||_\infty \right) + ||b^{(l)}||_\infty \\
&= ||h^{(0)}||_\infty \prod_{m=0}^{l} ||W^{(m)}||_\infty + ||W^{(l)}||_\infty \sum_{n=0}^{l-2} \left(||b^{(n)}||_\infty \prod_{m=n+1}^{l-1} ||W^{(m)}||_\infty \right) + ||W^{(l)}||_\infty ||b^{(l-1)}||_\infty + ||b^{(l)}||_\infty \\
&= ||h^{(0)}||_\infty \prod_{m=0}^{l} ||W^{(m)}||_\infty + \sum_{n=0}^{l-2} \left(||b^{(n)}||_\infty \prod_{m=n+1}^{l} ||W^{(m)}||_\infty \right) + ||b^{(l-1)}||_\infty ||W^{(l)}||_\infty + ||b^{(l)}||_\infty \\
&= ||h^{(0)}||_\infty \prod_{m=0}^{l} ||W^{(m)}||_\infty + \sum_{n=0}^{l-1} \left(||b^{(n)}||_\infty \prod_{m=n+1}^{l} ||W^{(m)}||_\infty \right) + ||b^{(l)}||_\infty
\end{aligned}$$

□

Proposición 2.2 *Dados $H^{(l)}, \overline{H}^{(l)}$ cotas válidas para una capa $l \geq 1$, la siguiente es una cota válida:*

$$||h^{(l+1)}||_\infty \leq ||W^{(l)}||_\infty \max_{j \in [n_l]} \left(H_j^{(l)}, \overline{H}_j^{(l)} \right) + ||b^{(l)}||_\infty$$

Demostración. Trivial de definición de $h^{(l+1)}$ y desigualdad triangular. □

Esta proposición se utilizará en el caso de que se obtengan cotas $H^{(l)}, \overline{H}^{(l)}$ mejores que las proyectadas originalmente con Proposición 2.1, para mejorar las cotas que sean posibles.

2.2. Análisis del algoritmo

2.2.1. Problemas y modificaciones realizadas

El algoritmo 1 presentó problemas de desempeño en su versión original, ya que en general la cantidad de pesos distintos de cero es muy alta. Esto ralentiza la resolución del MILP, incluso con redes entrenadas con algún grado de regularización, ya que muchas veces los

pesos obtenidos son muy cercanos a 0 pero no exactamente 0, lo cual tiene incidencia en el *solver*, llegando al punto de prácticamente no obtener soluciones para capas más profundas.

Para evitar el problema, se crea una pequeña subrutina detallada por el Algoritmo 2, que permite redondear a 0 los pesos que se encuentren en una bola $B(0, \varepsilon)$, con ε dado por el usuario. A esto se le conoce como *prunning* en la literatura.

Algorithm 2: Rounder

Input: $\mathcal{W}$, $\varepsilon \geq 0$
Output: $\mathcal{W}_\varepsilon$

```
1  Definir W_ε = {}
2  for W^(l) ∈ 𝒲 do
3      for fila i ∈ [n_l] do
4          for columna j ∈ [n_{l+1}] do
5              if |W_ij^(l)| ≤ ε then
6                  Definir W_ij^(l)(ε) = 0
7              else
8                  Definir W_ij^(l)(ε) = W_ij^(l)
9      Agregar W^(l)(ε) al conjunto W_ε
10 Retornar el conjunto de matrices redondeadas W_ε
```

Por otra parte, para mejorar los tiempos de ejecución, se desarrolla una versión modificada del Algoritmo 1, donde se cambia la línea 8 para resolver en cambio la relajación lineal de los problemas ($\overline{P_{nm}}$) y ($\underline{P_{nm}}$). Este cambio mejora el desempeño del algoritmo, ya que ahora se resuelve un LP en vez de un MILP, lo cual se puede hacer en tiempo polinomial. El trade-off es que se obtienen cotas que podrían ser menos ajustadas que las del MILP.

También, como para una capa $m \in [L]$ fija, los problemas ($\overline{P_{nm}}$) y ($\underline{P_{nm}}$) son independientes para distintas neuronas $n \in [n_m]$, el algoritmo se puede paralelizar fácilmente por capa, resolviendo simultáneamente los MILPs y LPs en lotes o *batches* $B \subseteq [n_m]$.

Como no importa el orden de estos lotes, se opta por dividir el problema en lotes consecutivos de tamaño a lo más n_c, el número de núcleos del computador. De esta forma, en cada capa m, en vez de realizar n_m iteraciones en la linea 6, se hacen $\left\lceil \frac{n_m}{n_c} \right\rceil$ iteraciones.

Así, finalmente se utilizan 2 versiones modificadas del Algoritmo 1; la versión MILP paralela y la LP paralela, detalladas en el Algoritmo 3.

Algorithm 3: Bounder versión 2

Input: $(\mathcal{W}, \mathcal{B})$, $\overline{H}^{(0)}, H^{(0)}$, $\varepsilon \geq 0$, n_c, relax $\in \{0, 1\}$

Output: $\left\{\overline{H}^{(l)}, H^{(l)}\right\}_{l=1}^{L}$

Obtener W_ε con el algoritmo 2 usando $\mathcal{W}, \varepsilon$

Definir las variables del modelo $(h^{(l)}, \overline{h}^{(l)}, a^{(l)}, z^{(l)})$ $\forall l \in [L]$ con (2.5)-(2.6), $a^{(l)} \in \mathbb{R}^{n_l}$

Agregar la restricción (2.7), usando los valores iniciales $\overline{H}^{(0)}, H^{(0)}$

Definir cotas validas $H^{(l)}, \overline{H}^{(l)}$ $\forall l \in [L]$ usando la Proposición 2.1

for $1 \leq m \leq L$ **do**

    Agregar restricciones $(2.1) - (2.4)$ asociadas a la capa m usando W_ε como las matrices de pesos

    $N_B = \left\lceil \frac{n_m}{n_c} \right\rceil$

    $B_1 = \{1, ..., n_c\}; B_2 = \{n_c + 1, ..., 2n_c\}; ...; B_{N_B} = \{(N_B - 1)n_c + 1, ..., n_m\}$

    for $j \in [N_B]$ **do**

        if *relax=0* **then**

            Resolver simultáneamente $(\overline{P_{nm}})$ y $(\underline{P_{nm}})$ asociados a los nodos $n \in B_j$

        else if *relax=1* **then**

            Resolver simultáneamente las relajaciones lineales de $(\overline{P_{nm}})$ y $(\underline{P_{nm}})$ asociados a los nodos $n \in B_j$

        if *Se encuentran cotas para* $a_n^{(m)}$ *con* $n \in B_j$ **then**

            Definir $H_n^{(m)}$ y $\overline{H}_n^{(m)}$ con los valores correspondientes

        else

            Propagar peor cota de capa $m - 1$ con la Proposición 2.2.

Retornar $\left\{\overline{H}^{(l)}, H^{(l)}\right\}_{l=0}^{L}$

Por último para tener control sobre el tiempo total de ejecución, se agrega también la opción de usar un tiempo límite, el cual se aplica por capa. Es decir, para cada neurona n en un lote en la capa m se utiliza un mismo tiempo limite t_m. De esta manera se sabe que el algoritmo entregará cotas válidas en a lo más $T = \sum_{m \in [L]} N_B^{(m)} t_m$ segundos.

2.2.2. Configuración de pruebas experimentales

Con el fin de entender bajo que condiciones el Algoritmo 3 tiene un buen comportamiento, y poder realizar más adelante otras pruebas con modelos más complejos, se realizan una serie de experimentos para determinar:

1. La configuración de entrenamiento más favorable (con o sin regularización, tipo de regularización, factor de penalización, etc)

2. La cantidad de pesos a redondear, dados por el valor del radio ε

3. El uso de la versión LP o MILP del algoritmo, ¿Cuál es mejor y por qué?

Para estudiar como afectan estos cambios y los hiper-parámetros como el radio de redondeo ε y el grado de regularización λ, se entrenan las redes especificadas en la Tabla 2.1[1] en el supercomputador del NLHPC[2]. En particular, se utiliza un nodo Dell PowerEdge C6420 con 2 CPUs Intel Xeon Gold 6152 @ 2.10 GHz de 22 núcleos. De este nodo, se asigna sólo 1 núcleo y 4 GB de RAM para entrenar cada una de las redes. Éstas se entrenan por 50 épocas, con regularización L1 y L2, donde se utilizan los siguientes factores de penalización:

$$\lambda \in \{0, 10^{-4}, 2 \cdot 10^{-4}, 3 \cdot 10^{-4}\}$$

Luego se guardan los parámetros obtenidos de las redes y se aplican distintos niveles de redondeo a los pesos de las matrices $\mathcal{W}$ con los siguientes valores:

$$\varepsilon \in \{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$$

Finalmente, para todas estas configuraciones (28 en total), se aplica el algoritmo 3, donde se resuelven los LPs y los MIPs utilizando Gurobi 9.1.2. El cálculo de las cotas se realiza en el mismo equipo mencionado anteriormente, pero se asignan 4 núcleos y 6 GB de RAM por cada red.

Tabla 2.1: Arquitecturas usadas en los experimentos. El largo de la tupla indica el número de capas ocultas de la red y cada elemento de la tupla indica la cantidad de nodos en dicha capa. Todas las capas son fully connected y con ReLUs como función de activación.

Nombre	Arquitectura
3000x1	(3000)
800x2	(800,800)
200x5	(200,200,200,200,200)
100x10	(100,...,100)

2.3. Resultados

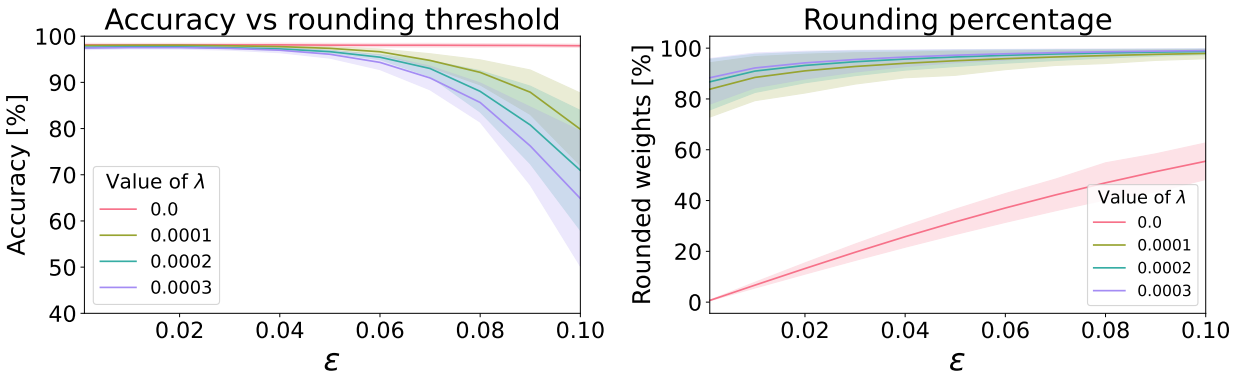
2.3.1. Nivel de redondeo y regularización de la red

Como se observa en las Figuras 2.1a y 2.1b, y como es de esperar, la cantidad de pesos que se redondean afectan negativamente en la precisión de la red, aunque hay un gran margen donde la precisión se mantiene prácticamente igual. Destacamos el rango $[0, 0.04]$. De acá

[1] La última es una tupla de 10 elementos

[2] Laboratorio Nacional de Computación de Alto Rendimiento (NLHPC)

también se desprende que las redes sin regularizar ($\lambda = 0$) nos permiten redondear en un radio mucho más grande sin perder precisión. Esto se evidencia, por ejemplo, para $\varepsilon = 0.1$, donde se alcanza un porcentaje de redondeo cercano al 60 %, sin afectar prácticamente a la precisión de la red, con cambios menores al 1 %.



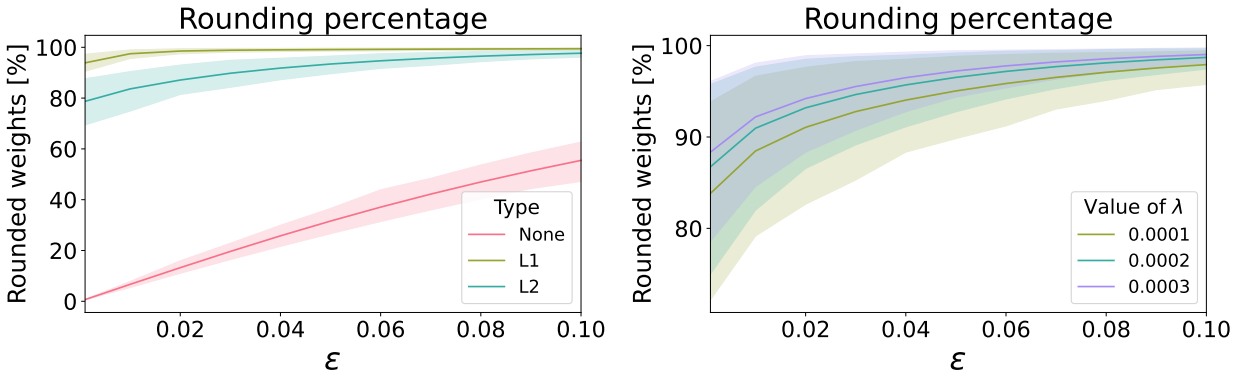
(a) Precisión promedio de todas las redes.

(b) Redondeo por nivel de regularización.

Figura 2.1: Impacto del radio de redondeo ε, tanto en la precisión de la red como en el porcentaje de pesos redondeados.

Otro detalle interesante, es que el tipo de regularización utilizado en el entrenamiento, también tiene un impacto. En la Figura 2.2a, se observa que una regularización de tipo L1 nos permite redondear significativamente más que una L2, para un mismo radio ε.

También, en la Figura 2.2b se puede observar que a un mayor nivel de regularización, es posible redondear muchos más pesos para un mismo nivel ε, sin importar la norma que se esté usando.

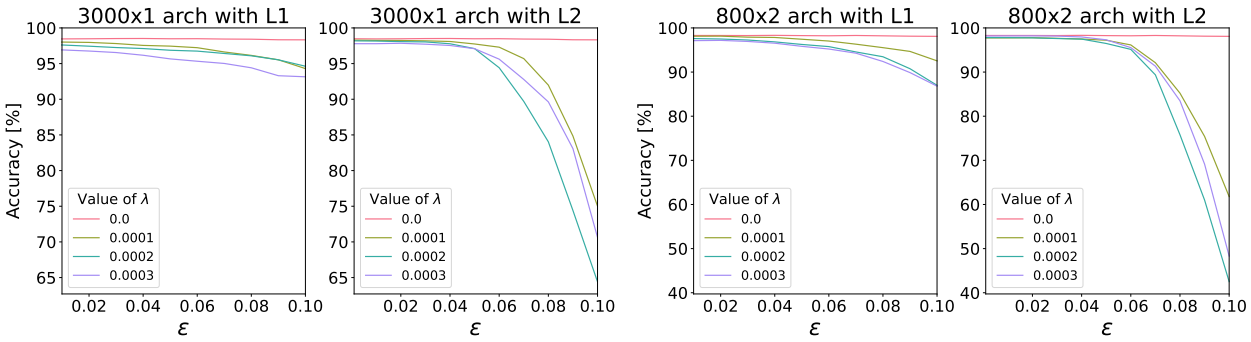


(a) Redondeo por tipo de regularización.

(b) Redondeo promedio en redes regularizadas.

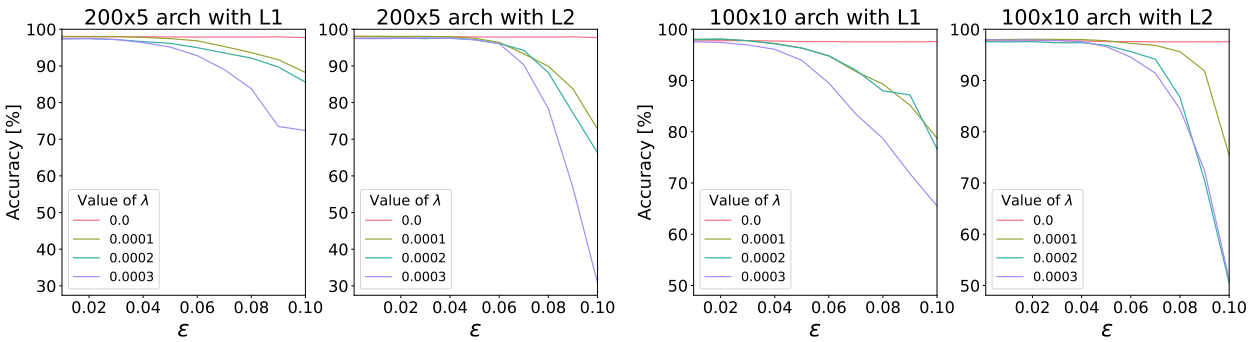
Figura 2.2: Cantidad de pesos redondeados por radio de redondeo ε, según el tipo de regularización (o no) y el factor de penalización.

Es importante analizar cómo afecta la arquitectura de la red al momento de redondear los pesos. Como se observa en la Figura 2.3, en redes más profundas la regularización hace que se pierda precisión más rápidamente. También, redes regularizadas con norma L2, pierden precisión mucho más rápidamente que las de tipo L1.



(a) Red de 1 capa

(b) Red de 2 capas



(c) Red de 5 capas

(d) Red de 10 capas

Figura 2.3: Precisión de las redes separadas por arquitectura y por tipo de regularización utilizada, donde ε es el radio de redondeo utilizado.

2.3.2. LP vs MILP: Calidad de cotas

Un factor importante a considerar, es determinar cuánto se gana y/o se pierde al resolver el LP en comparación con el MILP. Para ello, se analiza la calidad de las cotas obtenidas en el problema relajado, calculando un *gap* de optimalidad relativa definido por (RO Gap).

$$\frac{|Bound_{LP} - Bound_{MILP}|}{|Bound_{MILP}| + 10^{-10}} \tag{RO Gap}$$

Notar que mientras más cercano a 0 sea este valor, se tiene que el LP entrega un cota más cercana al MILP, y por lo tanto se entiende que la cota del LP es buena.

Como muestra la Figura 2.4, una de las primeras observaciones es que las cotas del LP son mejores a medida que se aumenta el radio de redondeo y se aumenta la regularización. También, en esta figura, se observa que las redes sin regularizar presentan una varianza muy alta para el RO Gap, esto sugiere que hay una fuerte dependencia de la calidad de la cota con el tipo de arquitectura que tiene la red.

En la Figura 2.5 se observa de mejor manera el fenómeno anterior, el cual se genera por la calidad del LP en redes más profundas. Las Figuras 2.5c y 2.5d evidencian que la calidad de cotas es comparativamente peor respecto a las redes de 1 y 2 capas, en el caso sin regularizar.

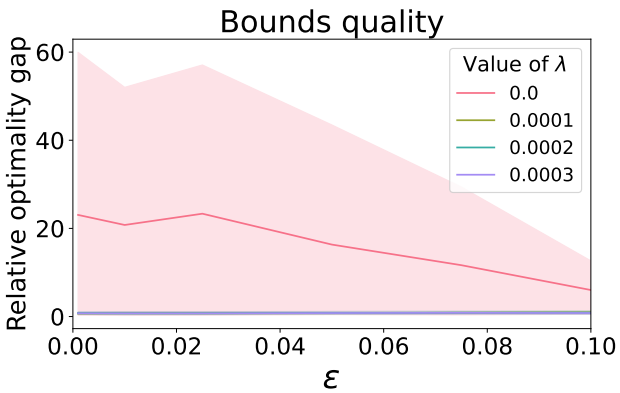
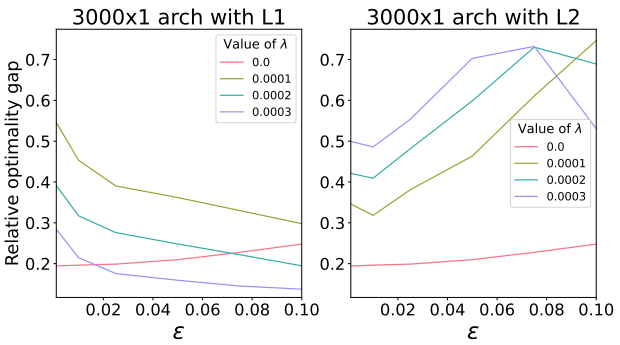
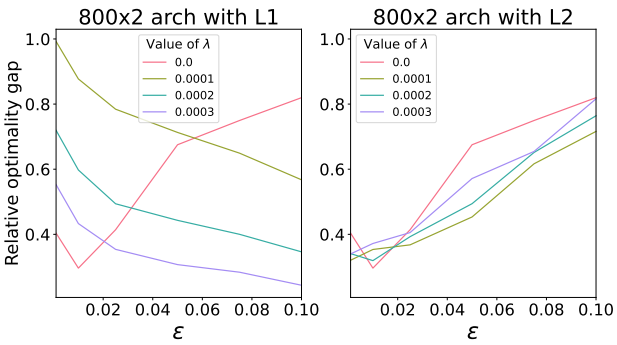


Figura 2.4: Calidad relativa de las cotas obtenidas por el LP.

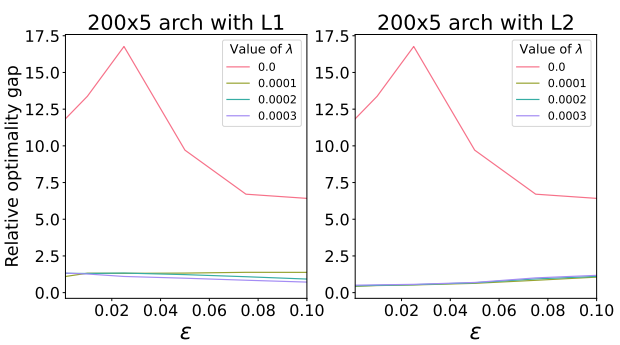
Estos resultados sugieren que hay una propagación de errores, que se acentúa mientras más profunda es la red. La Figura 2.7 muestra de mejor manera este fenómeno, donde se observa un deterioro exponencial de la calidad de la cota obtenida con el LP, en función de la profundidad de la capa.



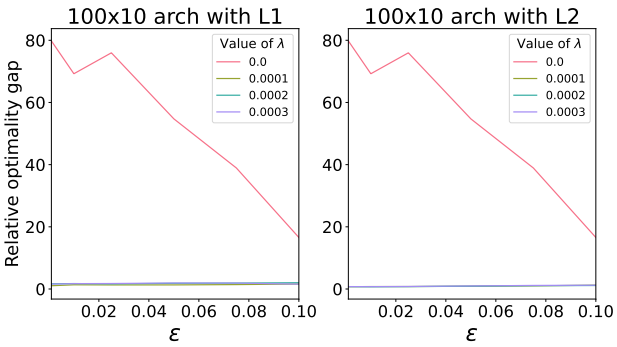
(a) Red de 1 capa



(b) Red de 2 capas

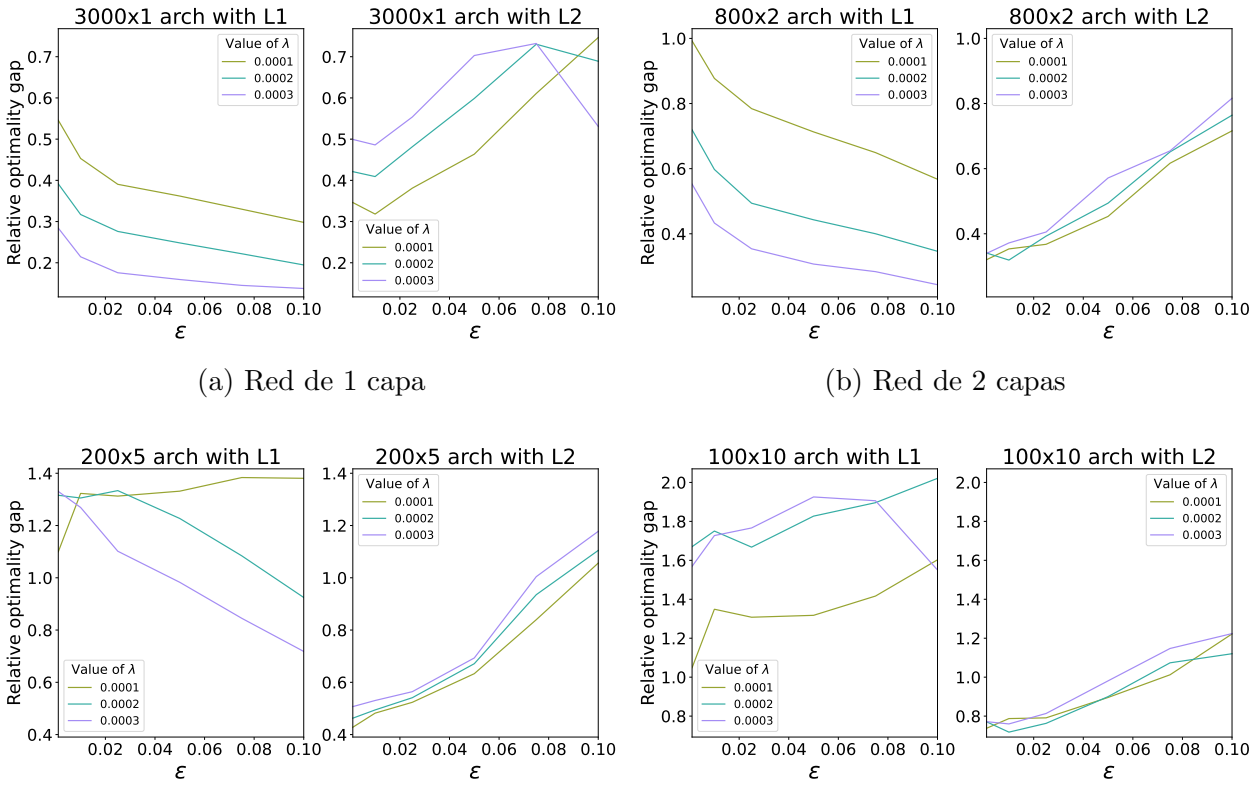


(c) Red de 5 capas



(d) Red de 10 capas

Figura 2.5: Gap de optimalidad relativo en función del nivel de redondeo, separado por arquitectura utilizada. ε representa el *threshold* utilizado para redondear pesos.



(a) Red de 1 capa

(b) Red de 2 capas

(c) Red de 5 capas

(d) Red de 10 capas

Figura 2.6: Gap de optimalidad relativo en función del nivel de redondeo para los casos con regularización. Separado por arquitectura utilizada.

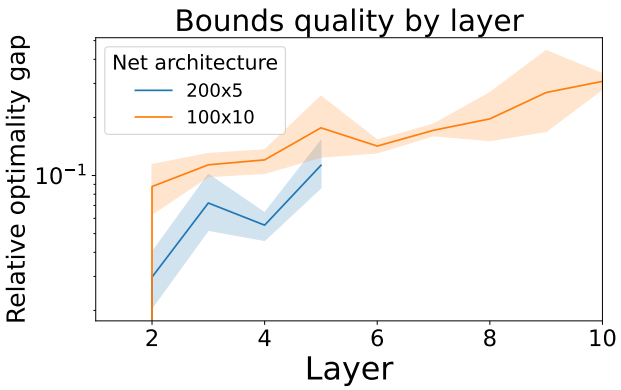


Figura 2.7: Gap promedio de la cota inferior en función de la capa de la red, para las diferentes arquitecturas utilizadas.

2.3.3. LP vs MILP: Tiempos de cómputo

Una variable importante a considerar en todo este proceso es el tiempo que se utiliza para calcular las cotas de activación de la red. Debido a la motivación original, este cálculo es sólo la primera parte de una maquinaria mayor, y a pesar de ser necesario sólo la primera vez, debería ser realizado de manera eficiente. Es por esto que en esta sección se analizará cómo influye la variable temporal en el cálculo de cotas, cómo afecta en la calidad de las cotas y cuánto se gana aumentando el tiempo disponible para calcular.

Lo primero que se analiza es el tiempo de cómputo, tanto del LP como del MILP. En la Figura 2.8 vemos que hay varios ordenes de magnitud de diferencia en el tiempo que conlleva resolver el LP versus el MILP.

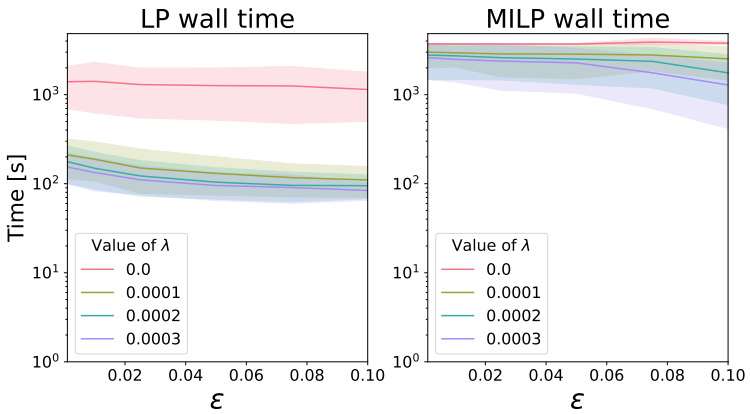


Figura 2.8: Comparación de tiempos de resolución del LP y del MILP (Wall time)

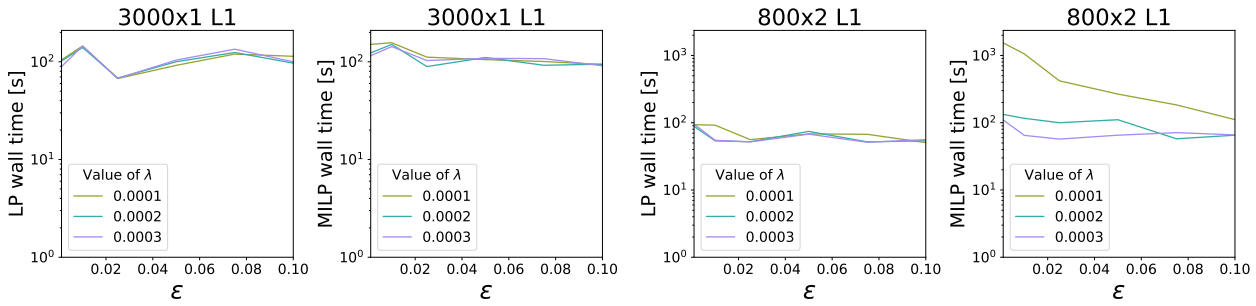
También es interesante ver cómo las instancias sin regularización son, en ambos casos, las que más demoran en resolverse, siendo en el LP mucho más grande el impacto que tiene entrenar sin regularización. Debido al decrecimiento que tienen las curvas, se observa también, que el efecto de redondear pesos es más importante en la resolución del LP que la del MILP.

Por último, en el caso sin regularizar, a pesar de aplicar un radio $\varepsilon = 0.1$ que, como ya se observó anteriormente, provocaba un redondeo del 60 % aproximadamente, no hay una mejora significativa en el tiempo de resolución, tanto en el LP como en el MILP. Esto indica una primera desventaja comparado con las redes regularizadas, que a medida que se redondea más, encuentran cotas más rápidamente.

Si se observan los resultados para las redes regularizadas con norma L1 disgregados por arquitectura, se puede notar que:

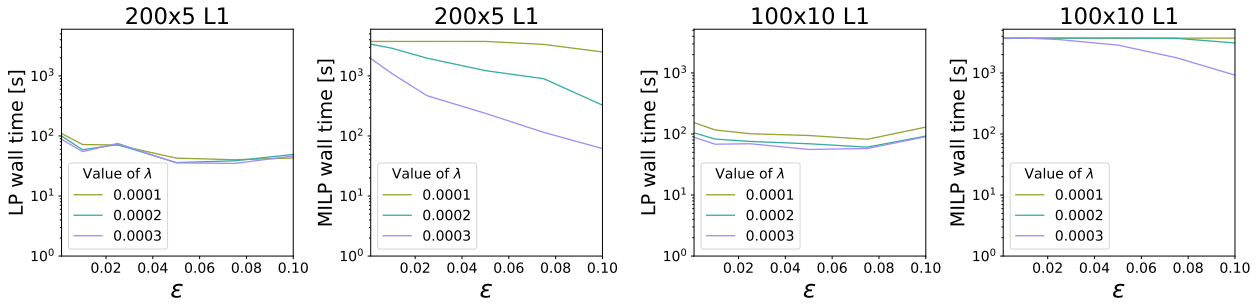
- Para redes poco profundas, los tiempos de cómputo son en la práctica muy similares, como se aprecia en las Figuras 2.9a y 2.9b. En cambio, en las redes más profundas hay ordenes de magnitud de diferencia entre la version relajada y el MIP, como muestran las Figuras 2.9c y 2.9d.

- El efecto de redondear pesos varía dependiendo de la cantidad de regularización y la profundidad de la red, como se observa en las Figuras 2.9b a 2.9d, donde las curva decrecen a tasas distintas dependiendo del factor λ y la arquitectura. Se observa que para redes poco profundas y con poca regularización el redondeo incide más que en los otros casos. A su vez, en redes profundas con más regularización ocurre lo mismo.



(a) Red de 1 capa

(b) Red de 2 capas



(c) Red de 5 capas

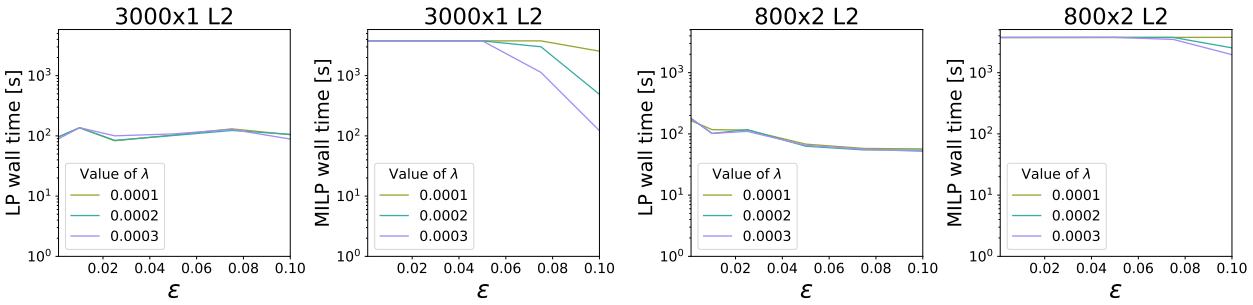
(d) Red de 10 capas

Figura 2.9: Tiempos de resolución LP y MILP para el caso regularizado con norma L1.

Para el caso con norma L2, en cambio, como se aprecia en la Figura 2.10, en todas las arquitecturas el LP es comparativamente mejor en tiempos de cómputo. El MILP en todos los casos tiene un desempeño peor con órdenes de magnitud de diferencia y, salvo en las redes poco profundas, no se beneficia sustantivamente de las bondades del redondeo y la regularización, como si sucede en el caso con norma L1. Sin embargo, las versiones relajadas de estos problemas si se benefician de éstos, y se observan comportamientos similares al caso L1. Esto sugiere que los LP suelen tener comportamientos más estables, sin depender del tipo de regularización que se utilice.

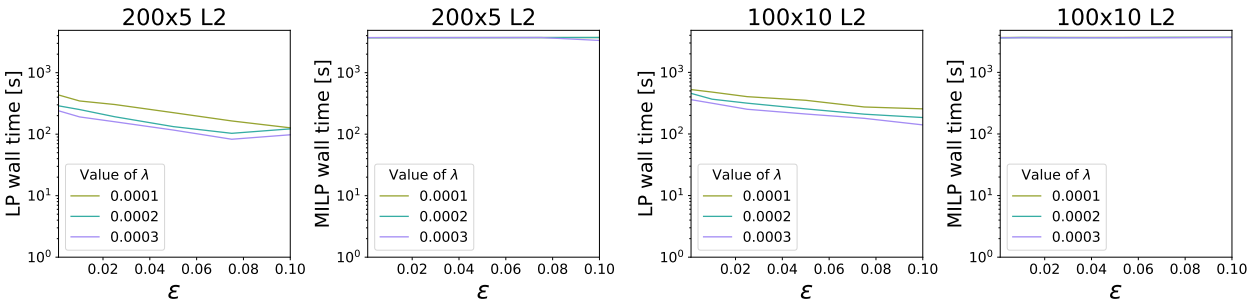
Comparando el desempeño de los LP, para ambas regularizaciones y disgregado por tipo de arquitectura utilizada, se puede notar que en el caso de norma L1, los tiempos de resolución son en general más cortos. Esto se aprecia en las Figuras 2.11c y 2.11d. También, el LP se beneficia más del redondeo en el caso de norma L2. Esto se aprecia en las Figuras 2.9c, 2.9d y 2.11b donde las curvas del caso L2 decrecen más rápidamente que las del caso L1.

Por último, comparando el desempeño de los MILP, para ambas regularizaciones y disgregado por tipo de arquitectura utilizada, se puede notar que en general el caso L1 tiene



(a) Red de 1 capa

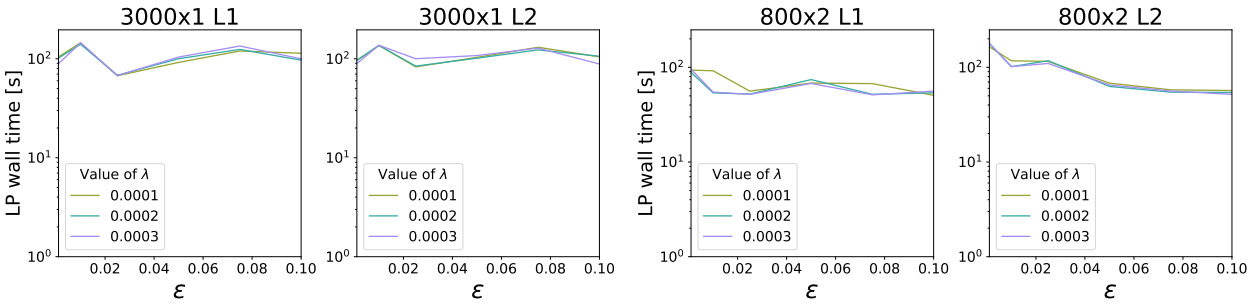
(b) Red de 2 capas



(c) Red de 5 capas

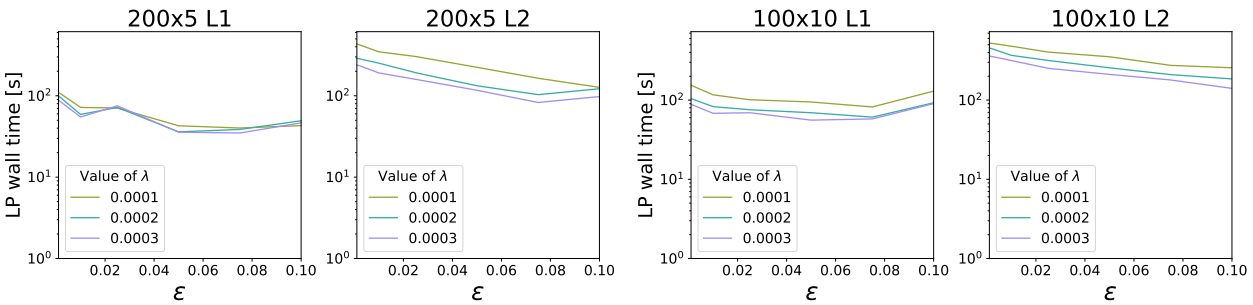
(d) Red de 10 capas

Figura 2.10: Tiempos de resolución LP y MILP para el caso regularizado con norma L2.



(a) Red de 1 capa

(b) Red de 2 capas

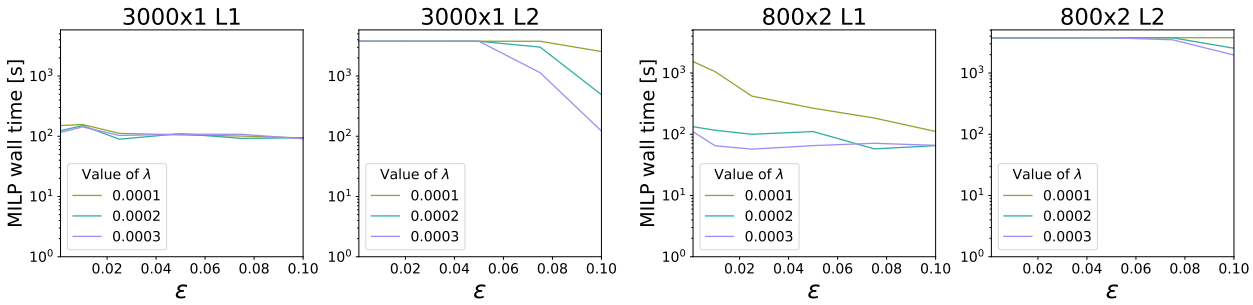


(c) Red de 5 capas

(d) Red de 10 capas

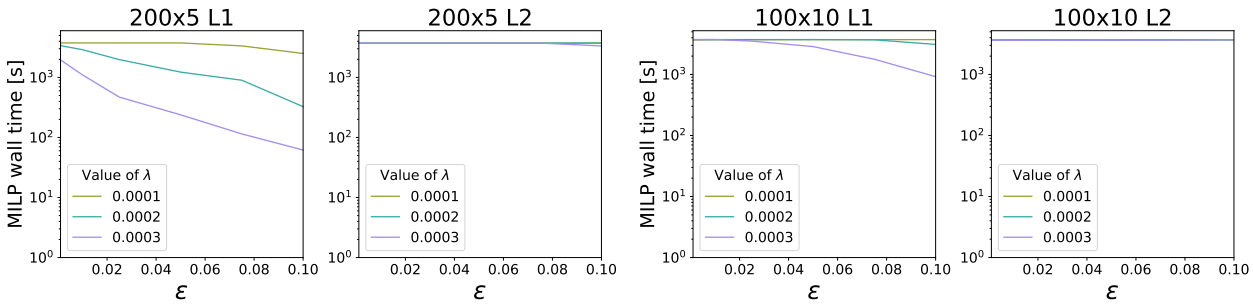
Figura 2.11: Comparación entre las regularizaciones L1 y L2, para los tiempos de resolución LP.

mejor desempeño que el L2 y el factor de regularización y radio de redondeo tienen mayor impacto en los tiempos de resolución en el caso L1.



(a) Red de 1 capa

(b) Red de 2 capas



(c) Red de 5 capas

(d) Red de 10 capas

Figura 2.12: Comparación entre las regularizaciones L1 y L2, para los tiempos de resolución MILP.

2.3.4. Impacto del límite de tiempo empleado.

Como se mencionó en la sección anterior, con el fin de asegurar que el algoritmo no se quede en las capas iniciales todo el tiempo, se desarrolla una subrutina que reparte el límite de tiempo total T de manera inteligente en todas las capas de la red, asignando valores para t_m con $m \in [L]$ de una manera exponencial. Esto significa que a capas escondidas más cercanas a la capa inicial se les asignará menos tiempo límite de resolución, mientras que las capas escondidas más profundas tendrán la mayor cantidad de tiempo para resolver.

Esta decisión se toma considerando los resultados anteriores donde se observa que la complejidad aumenta considerablemente en capas más profundas, por lo tanto es natural pensar que se necesita más tiempo de resolución para alcanzar cotas suficientemente buenas. Además, los resultados experimentales mostraron que en general las primeras capas alcanzan los óptimos con bastante facilidad y en poco tiempo.

Es por esto que se realiza un experimento para evaluar el impacto del límite de tiempo total utilizado en la calidad de las cotas obtenidas. Para ello, en base a los resultados de las secciones anteriores, se selecciona una instancia de 5 capas con 200 neuronas cada una,

entrenada durante 50 épocas y regularizada con un factor $\lambda = 0.0002$ tanto en norma L1 como L2. Luego, estas redes se redondean usando un radio $\varepsilon = 0.04$.

Finalmente, con estas 2 redes, se procede a calcular las cotas de los nodos para distintos límites de tiempo (de 20 a 90 minutos con un paso de 10 minutos), y se obtienen los resultados de la Figura 2.13.

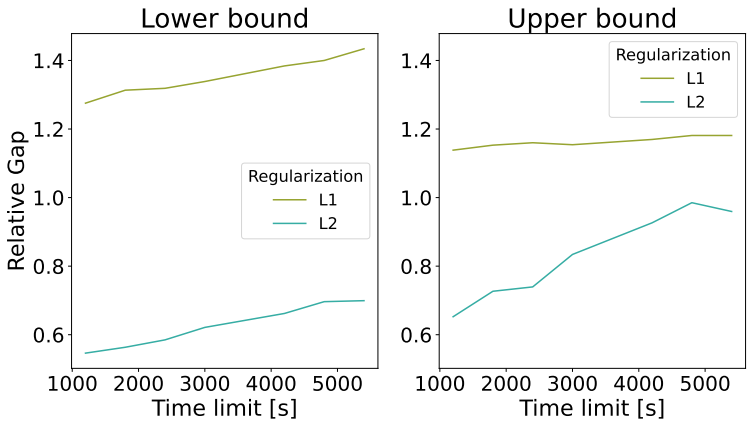


Figura 2.13: Calidad de la cota LP comparada con la mejor cota MILP encontrada, en función del límite de tiempo asignado.

Como se aprecia en la Figura 2.13, la calidad de las cotas LP relativas al MILP, van empeorando a medida que se asigna un mayor límite de tiempo. También el gráfico para la cota superior, parece indicar que con regularización L2, el impacto es mayor.

Este fenómeno, se logra explicar con las Figuras 2.14 y 2.15, las cuales muestran como mejoran las cotas LP y MILP respectivamente con el paso del tiempo, donde valores mayores a 1 representan cotas que aún se pueden mejorar, y el mejor valor equivale a 1 pues es precisamente la mejor cota alcanzada comparada consigo misma.

En estas figuras se ve cómo a partir de un límite de tiempo, el LP deja de mejorar (por alcanzar los óptimos) mientras que el MILP aún tiene margen de mejora. Esto indica que las cotas del LP tienen limitaciones, dadas principalmente por qué tan fuerte es la formulación de la relajación lineal, pero como se ve en ambas figuras, para límites de tiempos razonables, siguen siendo muy buenas comparativamente: a lo más son 2.4 veces más grandes que las del MILP en tiempos mucho mayores de los que se usarían en la práctica.

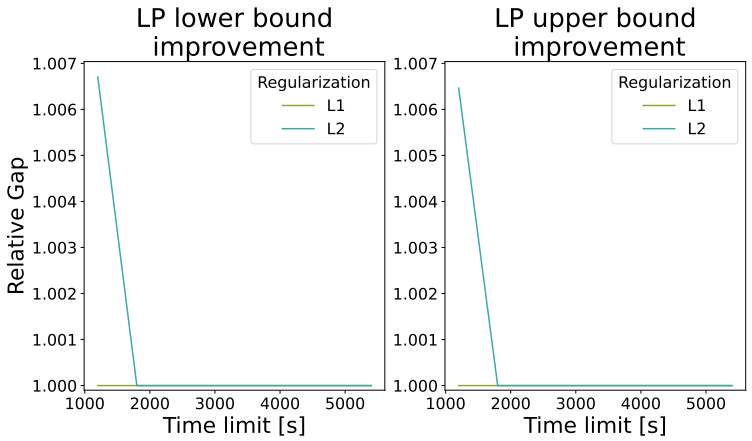


Figura 2.14: Calidad de la cota LP comparada con la mejor cota LP encontrada, en función del límite de tiempo asignado.

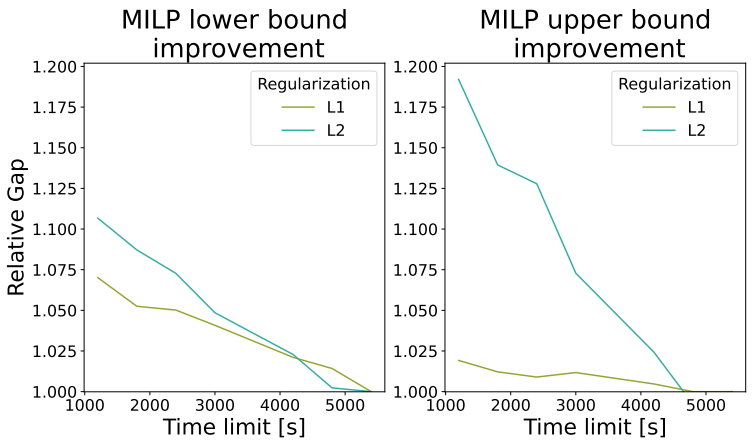


Figura 2.15: Calidad de la cota MILP comparada con la mejor cota MILP encontrada, en función del límite de tiempo asignado.

2.4. Conclusiones

De este primer análisis de las cotas y los resultados experimentales obtenidos, se desprenden varias ideas relevantes para el trabajo que se realizará y que no han sido estudiadas en profundidad previamente en la literatura.

La primera conclusión es que, si se garantiza tener una red con ciertas propiedades deseables, como por ejemplo la regularización, podemos asegurar buenos resultados en el cálculo de las cotas aprovechando las bondades del redondeo de pesos. Como vimos anteriormente, también se pueden asegurar buena calidad de las cotas LP en general, con tiempos de cómputo acotados y con comportamientos estables independiente del tipo de arquitectura.

Incluso en el caso sin regularizar, se observa que se pueden redondear algunos pesos sin perder mucha precisión y ganando también en calidad de las cotas LP, aunque en un grado mucho menor que en el caso regularizado. Es por esto, que de ahora en adelante se aplicará dicho redondeo como un preprocesado, salvo que se mencione lo contrario.

También como la calidad de las cotas LP es buena comparada con las del MILP y se resuelven en un tiempo mucho menor, el preprocesado también realizará el calculo de cotas usando sólo el LP, salvo que se indique lo contrario. En el siguiente capítulo, se evaluará el impacto que tiene lo ajustadas que son las cotas en problemas de optimización que usan restricciones del tipo Big-M en el modelamiento de redes neuronales, para registrar cuanto se pierde al no usar las cotas del MILP que son mucho más ajustadas. ¿Tiene un impacto significativo? ¿Hay algún *trade-off*? Son algunas de las preguntas que se busca resolver en los siguientes capítulos.

Capítulo 3

Impacto de las cotas de activación en modelo Big-M de redes neuronales

3.1. Problema de verificación

En este capítulo, con el fin de estudiar los diferentes efectos que pueden llegar a tener las cotas de activación $H_i^{(l)}, \overline{H}_i^{(l)}$ calculadas en el Capítulo 2, se resolverá un problema de verificación para comparar el impacto que tiene lo ajustada que son dichas cotas. Este tipo de problemas se utiliza mucho en la literatura, por ejemplo para asegurar robustez [20], para la creación de casos adversariales [7], entre otros.

Lo que se buscará con este problema es encontrar un input que, comparado con otra imagen inicial x_0, genere la máxima diferencia posible en la clasificación, en otras palabras, para una imagen referencial inicial, este problema entregará una nueva imagen que es clasificada de manera totalmente distinta a la original.

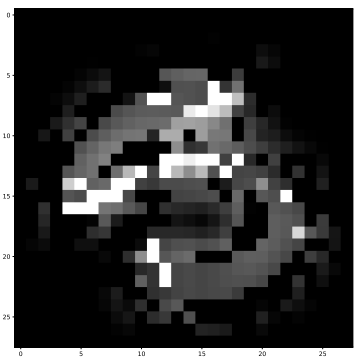


Figura 3.1: Ejemplo de caso adversarial para imagen de referencia con $d = 5$ clasificada como número 8.

Por ejemplo, en la Figura 3.1 vemos una imagen x que se generó tomando como base un x_0 correspondiente a un dígito 5, sin embargo, con ciertos cambios en la imagen de referencia x_0, se logró que la red lo clasificara en cambio como un número 8 con un 93.9 % de certeza.

Para realizar esto, dada una red neuronal $\mathcal{N} = (\mathcal{W}, \mathcal{B})$, se modifica el problema maestro (Big-M) cambiando la función $F(a, h, z)$, generando así el problema (PV):

$$\begin{aligned}
\max_{x \in \mathbb{X}} \quad & F(a,h,z) = \left\| softmax\left(a^{(L)}\right) - y_0 \right\|_1 && && \text{(PV)}\\
\text{s.a.} \quad & {W^{(l)}}^\top h^{(l)} + b^{(l)} = a^{(l+1)} && \forall l \in [L]_0 && (3.1)\\
& a^{(l)} = h^{(l)} - \overline{h}^{(l)} && \forall l \in [L] && (3.2)\\
& h_j^{(l)} \leq H_j^{(l)} z_j^{(l)} && \forall l \in [L], \forall j \in [n_l] && (3.3)\\
& \overline{h}_j^{(l)} \leq \overline{H}_j^{(l)} (1 - z_j^{(l)}) && \forall l \in [L], \forall j \in [n_l] && (3.4)\\
& h^{(l)}, \overline{h}^{(l)} \succeq 0 && \forall l \in [L] && (3.5)\\
& z^{(l)} \in \{0,1\}^{n_l} && \forall l \in [L] && (3.6)\\
& h^{(0)} = x; \quad h_j^{(0)} \in [-\overline{H}_j^{(0)}, H_j^{(0)}] && \forall j \in [n] && (3.7)
\end{aligned}$$

Donde y_0 es un vector de probabilidad fijo definido por $y_0 = \mathcal{N}(x_0)$, el output de la red para la imagen x_0. Por su parte, $softmax(a^{(L)})$ es precisamente el output de la red para imagen $x \in \mathbb{X}$, es decir, $softmax(a^{(L)}) = \mathcal{N}(x)$. Por lo tanto, el problema sencillamente está encontrando un $x \in \mathbb{X}$ tal que $\|\mathcal{N}(x) - \mathcal{N}(x_0)\|_1$ se maximice. También, más adelante se definirá $\mathbb{X}$ de manera tal que la imagen encontrada sea cercana a la imagen de referencia, bajo una cierta semi-métrica.

Las restricciones del modelo, son las mismas que las del problema maestro (Big-M), y están detalladas de mejor manera en la Sección 1.3.

Recordar que si las cotas $[-\overline{H}_i^{(l)}, H_i^{(l)}]$ son muy amplias, al relajar el problema usando $z_i^{(l)} \in [0, 1]$, se obtienen soluciones muy débiles. Para determinar cuánto afecta realmente este margen entre ambas cotas, a continuación se detallará los diversos experimentos realizados para estudiar este efecto.

3.2. Descripción de casos experimentales

Con el fin de estudiar el impacto que puede tener lo ajustada que son las cotas calculadas en el Capítulo 2, se realizan una serie de experimentos usando un verificador de redes modeladas con el MINLP (PV). Este verificador, computa un input x^* tal que:

$$x^* \in B_d(x_0, \varepsilon) \quad \wedge \quad \max_{x \in \mathbb{X}} ||\mathcal{N}(x) - \mathcal{N}(x_0)||_1 = ||\mathcal{N}(x^*) - \mathcal{N}(x_0)||_1,$$

Donde $B_d(\cdot, \cdot)$ es una bola definida por una semi-métrica d, y ε un nivel de perturbación para la imagen inicial x_0.

3.2.1. Restricción del espacio de búsqueda

Para obtener soluciones que no agreguen simplemente ruido aleatorio a la imagen de referencia, se decide buscar formas de restringir el espacio de búsqueda $\mathbb{X}$. Para esto, se define una semi-métrica para imágenes $d^m(\cdot, \cdot) : \mathbb{X} \times \mathbb{X} \to \mathbb{R}$ como:

$$d^m(x, y) = \max_{(i,j) \in 28 \times 28} d^m_{i,j}(x, y)$$

con:

$$d^m_{i,j}(x, y) = \begin{cases} 1 & \text{si } x^m_{i,j} = 0 \wedge y^m_{i,j} \neq 0 \\ \dfrac{1}{|K|} \displaystyle\sum_{k \in K} \left|(x^m_{i,j} - y^m_{i,j})_k\right| & \text{si no} \end{cases}$$

Donde:

- $x^m_{i,j} \in M \subseteq \mathbb{R}^{2m+1 \times 2m+1}$ es la vecindad de distancia a lo más m del píxel (i, j). A esto le llamaremos filtro de tamaño m centrado en el píxel (i, j).
- K son los índices de la matriz en M.

La idea detrás de esta semi-métrica para una imagen de referencia, es tener en cuenta tanto la intensidad de los píxeles, como también la vecindad en torno a ellos. Esto se observa en la definición por partes, donde en el primer caso se tiene en cuenta si el píxel está apagado, mientras que en la segunda se toma en cuenta la vecindad del píxel.

De esta forma, usando esta semi-métrica se pueden imponer condiciones en el espacio de búsqueda $\mathbb{X}$ para que la solución encontrada no difiera tanto de la imagen referencial, tanto en la forma del dígito como en la intensidad de los píxeles.

En la práctica, para restringir el espacio de búsqueda inicial, se realiza el proceso esquematizado en la Figura 3.2, donde para un *padding* p, y un filtro de tamaño $m \times m$ definidos con anterioridad, se recorre la imagen de referencia pixel a pixel. En el caso experimental detallado más adelante, se utilizan valores $m = 3$ y $p = 2$. Con estos parámetros, se calcula el valor $d^p_{i,j}(x^p_{i,j}, 0)$ para cada píxel (i, j) de la imagen inicial que, en palabras simples, es el promedio de los valores encerrados por el filtro. Este valor se utiliza para definir el valor mínimo

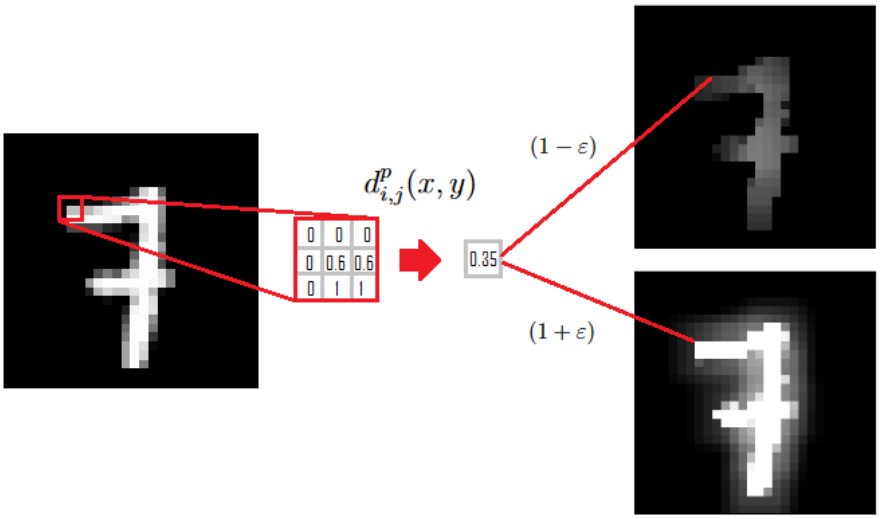


Figura 3.2: Esquema del proceso de cálculo de rangos admitidos para las imágenes referenciales. En este ejemplo se utiliza $m = 3, p = 2$ y un número 7 de imagen referencial.

y máximo que puede tomar dicho píxel en el espacio de búsqueda, tomando respectivamente:

$$\begin{aligned} (1-\varepsilon)\cdot d^p_{i,j}(x^p_{i,j},0) &\qquad \text{para el mínimo} \\ \text{mín}\left(1,(1+\varepsilon)\cdot d^p_{i,j}(x^p_{i,j},0)\right) &\qquad \text{para el máximo} \end{aligned}$$

Para los experimentos que se detallarán más adelante, se fija $\varepsilon = 0.18$ por ser suficientemente grande para generar soluciones factibles y suficientemente pequeño para generar soluciones estructuradas. También, se utiliza un representante de cada clase de MNIST (obtenido aleatoriamente) para utilizar como imagen referencial x_0.

Así, se tienen bien definidos los valores $\{H^0_i, \overline{H}^0_i\}_{i=1}^{784}$, que a su vez, describen el espacio de búsqueda como sigue

$$\mathbb{X} = \prod_{i=1}^{784}[-\overline{H}^0_i, H^0_i] \subseteq [0,1]^{784}$$

Observación 3.1 Para píxeles con valor 0 y con vecindad de valor 0, es claro notar que las cotas mínimas y máximas permitidas, seran iguales a 0. Con esto se asegura preservar la forma de la imagen inicial. A su vez, para píxeles con vecindad distinta de 0, al promediar los valores y ponderar por $(1 \pm \varepsilon)$, se asegura mantener la intensidad de los píxeles en dicha área.

3.2.2. Configuración de los experimentos

Para evaluar el impacto de las cotas de activación empleadas en (PV), se utilizan 4 formas distintas de calcularlas:

- Cotas "*naive*" obtenidas usando la Proposición 2.1.
- Cotas "LP" obtenidas con el Algoritmo 3 con relax=1.
- Cotas "MILP" obtenidas con el Algoritmo 3 con relax=0.
- Cotas "IP+img" obtenidas con el Algoritmo 3 con relax=0 y usando la imagen de referencia inicial para restringir el espacio de búsqueda inicial.

Para estos experimentos, se utiliza una red de cada arquitectura del Capítulo 2. Basado en resultados previos, se considera que las redes con regularización L1 y factor $\lambda = 0.0002$ son una configuración en la cual el Algoritmo 3 se comporta de buena manera. Así, se utilizarán los siguientes 4 casos a estudiar:

- Red de 1 capa escondida de 3000 neuronas, con regularización L1, $\lambda = 0.0002$ y 50 épocas de entrenamiento.
- Red de 2 capas escondidas de 800 neuronas, con regularización L1, $\lambda = 0.0002$ y 50 épocas de entrenamiento.
- Red de 5 capas escondidas de 200 neuronas, con regularización L1, $\lambda = 0.0002$ y 50 épocas de entrenamiento.
- Red de 10 capas escondidas de 100 neuronas, con regularización L1, $\lambda = 0.0002$ y 50 épocas de entrenamiento.

Los experimentos se ejecutaron en el NLHPC, usando la partición general descrita en el Capítulo 2.

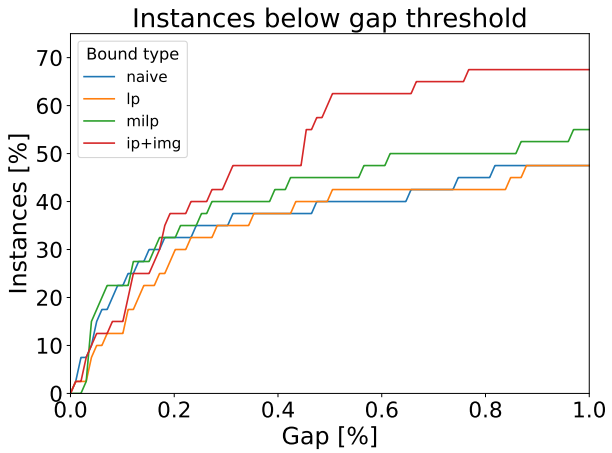
3.3. Resultados

Para distintos porcentajes de gaps de optimalidad en los incumbentes encontrados, se observa que hay distintos comportamientos según el tipo de cotas de activación que se utilice. Como se aprecia en la Figura 3.3, si se busca encontrar incumbentes que están muy cerca de la optimalidad ($\leq 1\,\%$), las cotas del tipo *naive* no aseguran un alto porcentaje de instancias terminadas, mientras que las cotas LP e IP+img presentan la mayoria de las instancias con un gap menor o igual a $1\,\%$.

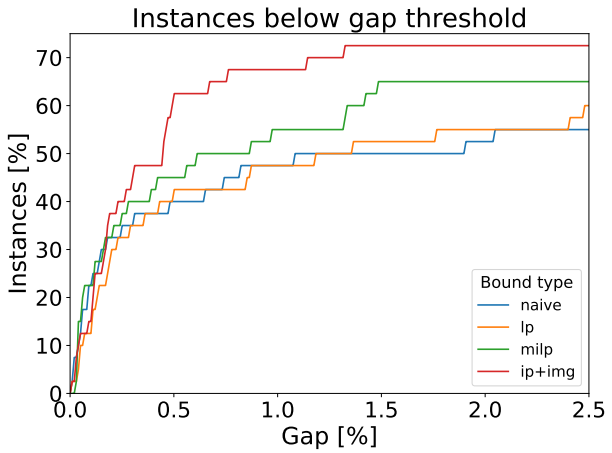
Si se admite tener un gap de optimalidad mayor como en Figura 3.3b, se aprecia que las cotas MILP producen más soluciones en este rango y se mantiene el comportamiento de las cotas *naive*, que siguen siendo peores que las demás.

Se aprecia también en la Figura 3.3c, que las cotas LP tienen mejor desempeño que todos sus competidores en algunos rangos, como entre un 3 % y 5 % de optimalidad.

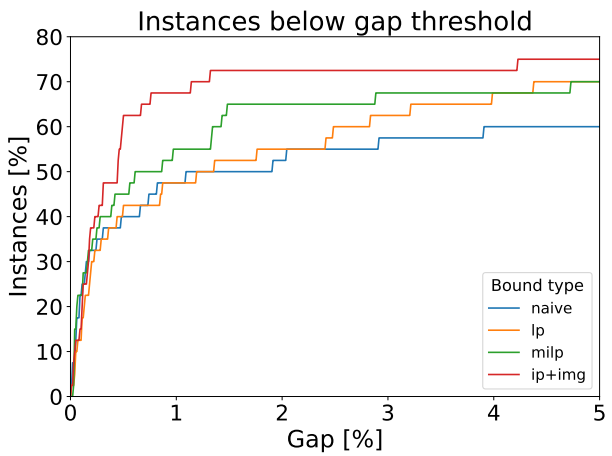
Por último, como se observa en la Figura 3.3d, las cotas *naive* comienzan a ser más competitivas para rangos de optimalidad más altos como el 7.5 %. Esto podría ser utilizado para encontrar rápidamente incumbentes con un gap de optimalidad acotado.



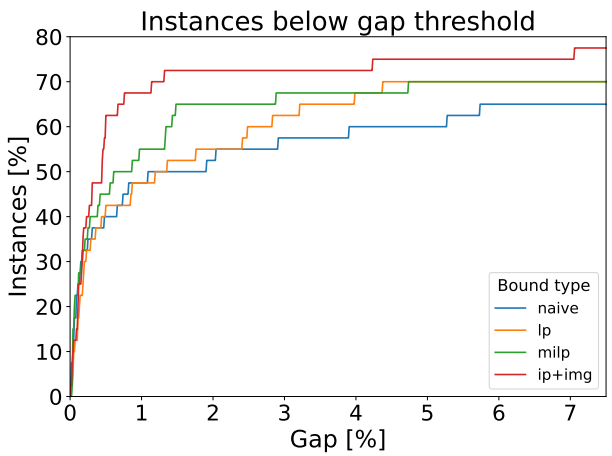
(a) Instancias con gap entre 0 % y 1 %



(b) Instancias con gap entre 0 % y 2.5 %



(c) Instancias con gap entre 0 % y 5 %



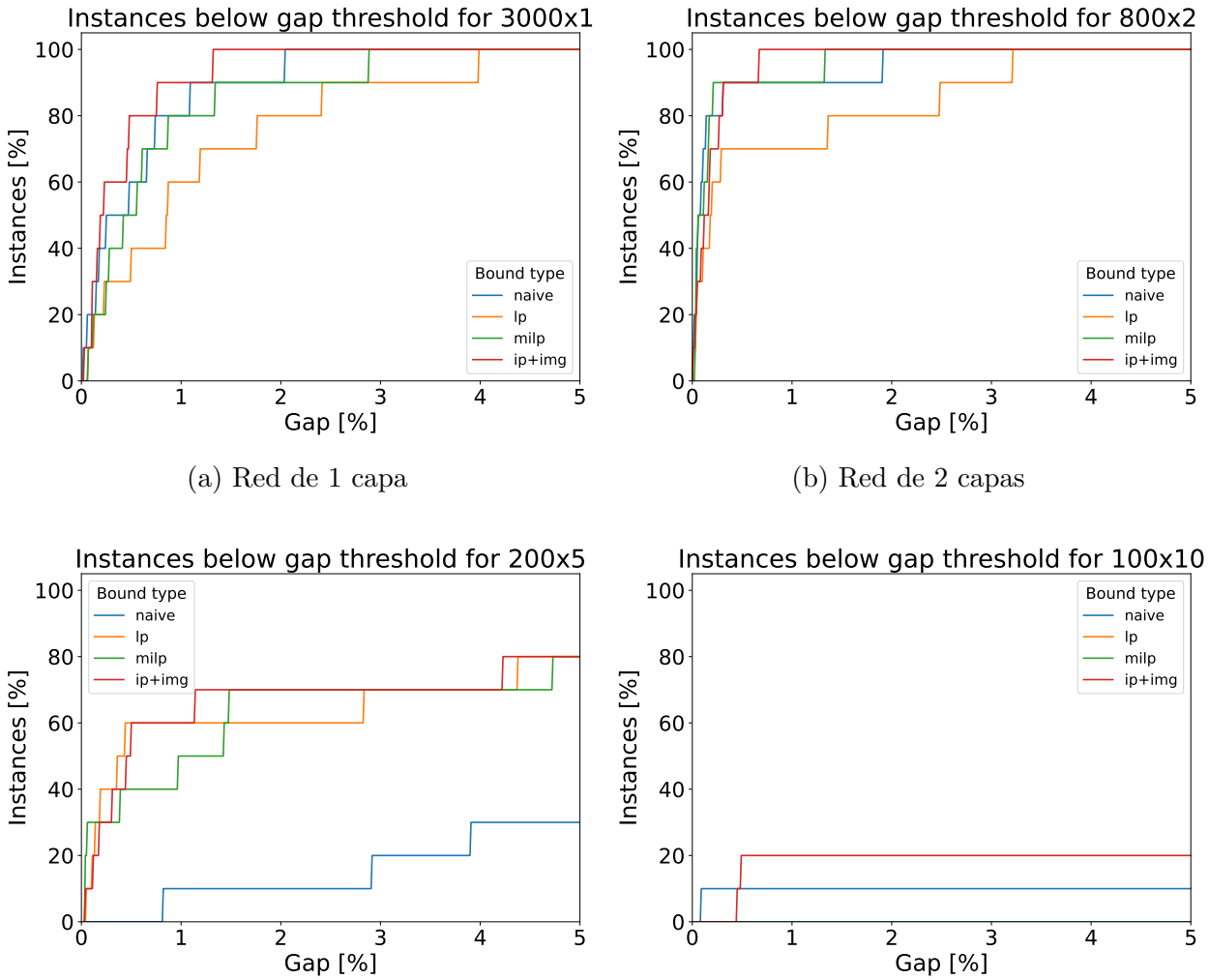
(d) Instancias con gap entre 0 % y 7.5 %

Figura 3.3: Porcentaje de instancias que llegan a un cierto gap de optimalidad dado por el eje X, separadas por el tipo de cota utilizado. En esta figura se consideran todas las redes.

Por otra parte, viendo como se comportan los tipos de cotas de activación dependiendo de la arquitectura de la red, es claro notar que para redes de poca profundidad, el problema se vuelve más sencillo por el porcentaje de instancias que alcanzan incumbentes cercanos al óptimo. En las Figuras 3.4a y 3.4b se aprecia un buen desempeño de las cotas *naive*, esto

sugiere que pueden ser utilizadas para una futura heurística.

Por otra parte, en las Figuras 3.4c y 3.4d se ve un desempeño muy malo de todas los tipos de cotas en general, pero más aún en las cotas del tipo *naive*



(a) Red de 1 capa

(b) Red de 2 capas

(c) Red de 5 capas

(d) Red de 10 capas

Figura 3.4: Porcentaje de instancias que llegan a un gap de optimalidad entre 0 % y un 5 %, separadas por el tipo de cota utilizado y la arquitectura de la red.

Al analizar el tiempo que demoran en computarse los distintos tipos de cotas, se puede observar en la Figura 3.5 que las cotas ip+img no son muy recomendables, ya que la variabilidad de tiempo de cómputo es muy alta, esto puede provocar que algunas instancias demoren mucho en resolverse y sin obtener soluciones suficientemente buenas para que se justifique.

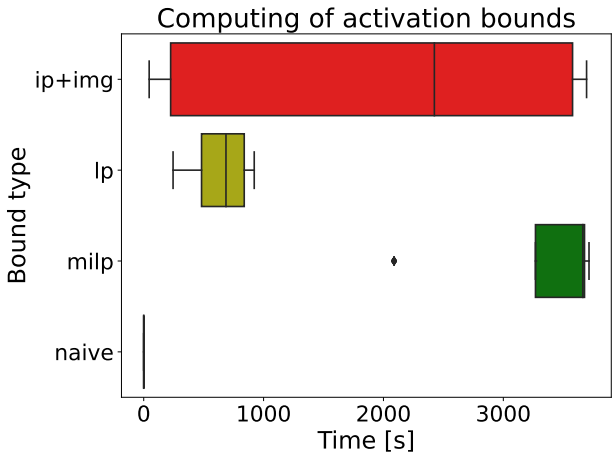


Figura 3.5: Boxplot del tiempo de cómputo de cada tipo de cota.

3.4. Conclusiones

Basado en los resultados experimentales, y contrastándolo con el tiempo que requiere computar cada una de las cotas, es claro ver un trade-off en este punto. Si bien en las instancias de poca profundidad, las cotas *naives* se comportaron de manera competitiva frente a los otros tipos de cotas, es evidente que en casos más cercanos a la realidad tendrían un desempeño peor, como se aprecia en las redes de 5 y 10 capas. Sin embargo, este tipo de cotas igual pueden ser útiles si se piensa en utilizarlas para creación de soluciones factibles en una etapa inicial. Una idea para trabajos futuro, sería utilizar este tipo de cotas, dentro de una heurística primal en los modelos IP.

Por otra parte, las cotas del tipo LP se comportaron de buena manera en general. Como se observa en la Figura 3.5, son las segundas cotas más rápidas en ser calculadas en promedio, y a pesar de que las cotas IP+img en algunos casos son más rápidas en ser calculadas, se considera que las LP son mejores ya que tienen menos varianza, y por lo tanto el usuario tiene una mayor certeza de que tiempos esperar.

Si se permite tener gaps mayores a 1 %, se observa que son mejores que las cotas naives, como indican las Figuras 3.3b a 3.3d. También, se aprecia que presentan rendimientos competitivos, alcanzando los mismos valores que las cotas MIP, como indican las Figuras 3.3c y 3.3d, pero sin el tiempo adicional que conlleva computar dichas cotas. Por último, en los gráficos separados por tipo de arquitectura, se observa que las cotas LP tienen el mismo comportamiento para los casos más cercanos a la realidad, como se observa en las Figuras 3.4c y 3.4d.

Todo lo anterior, posiciona a las cotas LP como el candidato ideal a ser usado en los MIP que modelan redes neuronales, debido a su buen desempeño en mayoría de los casos y su tiempo de cómputo aceptable.

Capítulo 4

Desempeño del modelo MIP para comparar redes neuronales

4.1. Introducción

A partir de los resultados de los capítulos previos, se determinaron ciertas configuraciones ideales donde los MIPs se comportan de manera deseada y se resuelven en tiempos más acotados, tales como: tener redes *sparse* (con regularización en entrenamiento), aplicar niveles de redondeo o *prunning* para evitar errores numéricos, el uso de LP's para el calculo de las cotas de activación, entre otros. Con el fin de probar empíricamente una nueva herramienta que se llamará *comparador MIP de redes*, se usarán estas configuraciones ideales para resolver el problema (PC).

Hasta el momento, en la literatura no se ha empleado este modelo para una tarea semejante, por lo que estudiar este modelo es de gran relevancia para determinar sus alcances y sus principales limitaciones. En principio, este modelo MIP permite estudiar una gran cantidad de problemas, por ejemplo, nos permite cuantificar la diferencia que hay entre las funciones de dos redes neuronales a partir del comparador $C(a, h, z)$. Una aplicación de esto sería, usar el comparador como certificador para una compresión con pérdida de redes neuronales, asegurando así que la versión comprimida de la red difiere solo en un cierto margen dado por el comparador $C(a, h, z)$ utilizado.

El problema (PC) formulado a continuación, es una modificación del problema maestro (Big-M), donde las variables del problema están duplicadas (para representar dos redes) y están relacionadas solo en la capa inicial $x = h_1^{(0)} = h_2^{(0)}$ y por el comparador $C(a, h, z)$. Las restricciones son análogas a las del problema maestro y están explicadas con mayor detalle en la sección 1.3.

$$\max_{x \in \mathbb{X}} \quad C(a,h,z) \tag{PC}$$

$$\text{s.a.} \quad {W_1^{(l)}}^\top h_1^{(l)} + b_1^{(l)} = a_1^{(l+1)} \qquad \forall l \in [L_1]_0 \tag{4.1}$$

$${W_2^{(l)}}^\top h_2^{(l)} + b_2^{(l)} = a_2^{(l+1)} \qquad \forall l \in [L_2]_0 \tag{4.2}$$

$$a_1^{(l)} = h_1^{(l)} - \overline{h}_1^{(l)} \qquad \forall l \in [L_1] \tag{4.3}$$

$$a_2^{(l)} = h_2^{(l)} - \overline{h}_2^{(l)} \qquad \forall l \in [L_2] \tag{4.4}$$

$$h_{1,j}^{(l)} \leq H_{1,j}^{(l)} z_{1,j}^{(l)} \qquad \forall l \in [L_1], \forall j \in [n_l] \tag{4.5}$$

$$h_{2,j}^{(l)} \leq H_{2,j}^{(l)} z_{2,j}^{(l)} \qquad \forall l \in [L_2], \forall j \in [n_l] \tag{4.6}$$

$$\overline{h}_{1,j}^{(l)} \leq \overline{H}_{1,j}^{(l)} (1 - z_{1,j}^{(l)}) \qquad \forall l \in [L_1], \forall j \in [n_l] \tag{4.7}$$

$$\overline{h}_{2,j}^{(l)} \leq \overline{H}_{2,j}^{(l)} (1 - z_{2,j}^{(l)}) \qquad \forall l \in [L_2], \forall j \in [n_l] \tag{4.8}$$

$$z_1^{(l)} \in \{0,1\}^{n_l},\ z_2^{(m)} \in \{0,1\}^{n_m} \qquad \forall l \in [L_1], \forall m \in [L_2] \tag{4.9}$$

$$h_1^{(l)}, h_2^{(m)}, \overline{h}_1^{(l)}, \overline{h}_2^{(m)} \succeq 0 \qquad \forall l \in [L_1], \forall m \in [L_2] \tag{4.10}$$

$$h_1^{(0)} = h_2^{(0)} = x; \quad h_{1,j}^{(0)} \in [-\overline{H}_{1,j}^{(0)}, H_{1,j}^{(0)}] \qquad \forall j \in [n] \tag{4.11}$$

Donde $C(a,h,z)$ es la función comparadora que utilizaremos. Generalmente se aplica en la capa de salida $h^{L+1} = y$.

Con esto, se tiene un conjunto de variables $\{h_i, \overline{h_i}, a_i, z_i\}_{i=1}^2$ que ayudan a modelar dos redes neuronales, las cuales solamente están relacionadas a través del mismo input $x \in \mathbb{X}$ y a través del comparador $C(a,h,z)$ que se definirá más adelante, dependiendo de qué es lo que se busca comparar.

Observación 4.1 Los valores de $\{W_i^{(l)}, b_i^{(l)}, L_i\}_{i \in \{1,2\},\ l \in [L_i]}$ son datos del problema, dados por la descripción de las redes neuronales. Además, por los resultados del capítulo anterior se asumen como conocidos los valores $\{H_i^{(l)}, \overline{H}_i^{(l)}\}_{i \in \{1,2\},\ l \in [L_i]}$, los cuales son calculados usando la versión LP del algoritmo 3.

Notar que a priori, se pueden agregar n redes neuronales a esta formulación, pero como el problema con una sola red ya es suficientemente complejo (es NP-Hard), durante este capítulo, solo se estudiará las limitaciones que tiene en el caso de comparación con $n = 2$.

4.2. Comparadores

4.2.1. Máxima Diferencia entre dos redes

Una de las motivaciones originales de la tesis, era crear un comparador de redes para comprobar de manera cuantitativa, que tanto difería una red neuronal, con su versión comprimida que admite pérdida. Para realizar esto, nace la idea de comparar ambas redes basados en la máxima diferencia posible entre éstas. De esta idea surge un problema adicional, ¿Cómo definir lo que es la "máxima diferencia"?. A priori, esto se puede llevar a cabo de diversas maneras, pero en este caso solo veremos dos formas de hacerlo.

En probabilidad

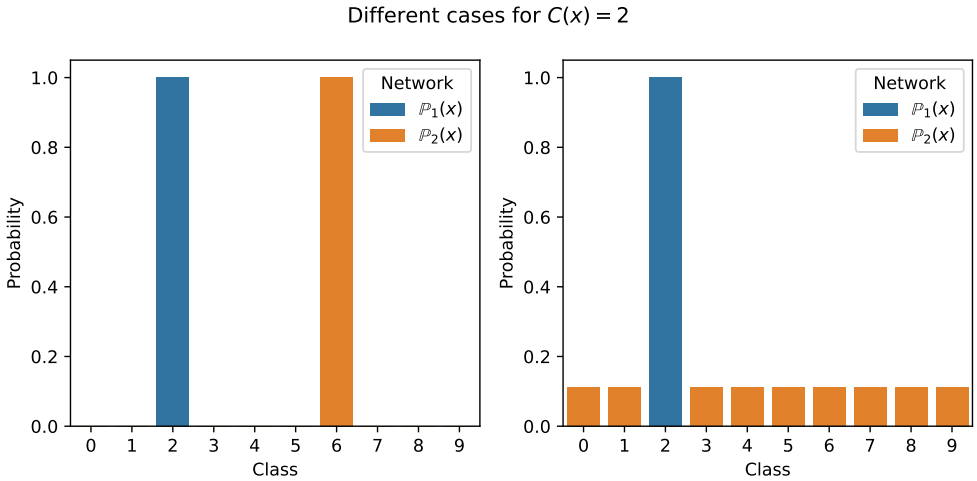


Figura 4.1: Ejemplo ilustrativo de distintos casos donde $C(a, h, z) = 2$ para (maxdiff).

Una manera de medir, la máxima diferencia entre ambas redes, es observar los vectores de probabilidad asociados a una clasificación de ambas redes y luego ver su diferencia en norma. En este caso, se decidió por utilizar la norma $\mathcal{L}_1$, ya que se puede linealizar más fácilmente lo cual es deseado para el IP, y por su simplicidad en términos generales. Así, el comparador quedaría como:

$$C(a, h, z) = \|h_1^{(L_1+1)} - h_2^{(L_2+1)}\|_1 \qquad \text{(maxdiff)}$$

Si bien, ésta es una idea que surge naturalmente, carece de mucha interpretación, ya que para mismos valores de está métrica, se pueden dar situaciones muy diferentes. Por ejemplo, si $C(a, h, z) = 2$, esto puede ocurrir debido a que ambos vectores $h_1^{(L_1+1)}, h_2^{(L_2+1)}$ valen 1 para distintas clases $C_1 \neq C_2 \in [K]$, lo cual indicaría que ambas redes son muy distintas.

También, esto puede ocurrir si un vector digamos $h_1^{(L_1+1)}$, concentra su probabilidad en una sola clase C_1, y el otro $h_2^{(L_2+1)}$ concentra la probabilidad de manera equiprobable entre

todas las otras clases $k \in [K]$ con $k \neq C_1$, lo cual indicaría que la segunda red es bastante mala pues se confunde entre todas las clases salvo una. Este ejemplo se puede observar mejor en la Figura 4.1.

En selección de clase

Otra manera de medir la máxima diferencia, es fijarse solamente en cuál es la clase que selecciona la red. Para clasificar, en general las redes se quedan con la clase de mayor probabilidad, así se tiene sencillamente que:

$$C_k = \underset{i=0,..,9}{\arg\max}\{h_{k,i}^{(L_k+1)}\} \text{ para } k \in \{1,2\}.$$

Luego, para hacer que ambas redes tengan una máxima diferencia en términos de la clase que seleccionan, basta con asegurar que la clase de mayor probabilidad no sea la misma en ambas redes. Esto, nuevamente se puede lograr de distintas formas, pero la que se utilizó en este caso fue agregar las siguientes restricciones:

$$a_{1,k}^{(L_1+1)} \leq a_{1,C_1}^{(L_1+1)} \qquad \forall k \in [K]\backslash C_1 \tag{4.12}$$

$$a_{2,k}^{(L_2+1)} \leq a_{2,C_2}^{(L_2+1)} \qquad \forall k \in [K]\backslash C_2 \tag{4.13}$$

Donde $C_1 \neq C_2 \in [K]$ son clases fijas para las redes $\mathcal{N}_1$ y $\mathcal{N}_2$ respectivamente. A pesar que estas restricciones se aplican sobre las transformaciones lineales $a^{(L+1)}$, estas aseguran que la red $\mathcal{N}_1$ clasificará el input como clase C_1 y la red $\mathcal{N}_2$ clasificará como C_2, ya que $softmax$ es una función creciente. Así, se evita poner restricciones sobre funciones no lineales como $softmax$. Luego, se utiliza la siguiente función objetivo:

$$C(a,h,z) = a_{1,C_1}^{(L_1+1)} + a_{2,C_2}^{(L_2+1)} \tag{maxout}$$

Este comparador tiene la versatilidad de elegir cuales son las clases C_1 y C_2, pero en general produce soluciones muy simétricas como densidades equiprobables, para evitar estas soluciones, en la práctica se suma un término de tolerancia ε en las restricciones (4.12) y (4.13), pero también se podría modificar la función objetivo para evitar que los valores de $a_{i,k}^{(L_i+1)}$ sean muy similares a los de $a_{i,C_i}^{(L_i+1)}$ $i \in \{1,2\}$, y así no obtener estas soluciones tan simétricas.

4.2.2. Máximo número de activaciones cambiadas

Otra forma de comparar redes, orientada en el polítopo asociado a las variables de decisión, es observar la diferencia de los patrones de activación entre dos redes de igual arquitectura, es decir, ver cuales son las neuronas que, para un mismo input $x \in \mathbb{X}$, tienen diferentes estados

de activación en ambas redes.

Para esto, se define la variable binaria auxiliar $\omega_i^{(l)}$ que vale 1 si la neurona (i,l) tiene estados distintos en ambas redes, y vale 0 si tiene el mismo estado. Luego, se puede estudiar el número de activaciones cambiadas, agregando las siguientes restricciones:

$$\omega^{(l)} + z_1^{(l)} + z_2^{(l)} \preceq 2 \qquad \forall l \in [L] \tag{4.14}$$

$$\omega^{(l)} \preceq z_1^{(l)} + z_2^{(l)} \qquad \forall l \in [L] \tag{4.15}$$

Es claro ver que estas restricciones son válidas, pues se tiene que si $z_{1,i}^{(l)} \neq z_{2,i}^{(l)}$ entonces alguna de ellas vale 0 y la otra 1, así se tendría que $w_i^{(l)} \leq 1$ por la primera restricción y $w_i^{(l)} \geq 1$ por la segunda, es decir $w_i^{(l)} = 1$. Por otra parte, si $z_{1,i}^{(l)} = z_{2,i}^{(l)}$, se distinguen dos casos, o bien la suma entre ambos es 0 o 2, así en el primer caso se tendría que $w_i^{(l)} \leq 0$ por la segunda restricción, y en el otro caso se tendría lo mismo pero por la primera restricción, con esto en ambos casos se tiene que $w_i^{(l)} = 0$.

Luego para maximizar el número de activaciones que cambian entre ambas redes, se define el comparador como:

$$C(a,h,z) = \sum_{l=1}^{L} \sum_{i=1}^{n_l} w_i^{(l)}(z) \tag{maxflip}$$

Durante el desarrollo de la investigación, se hicieron pruebas con este comparador, donde se observó que se comportaba de buena manera tanto para el problema primal como el dual. Por temas de tiempo, en esta tesis se omitirán los resultados obtenidos con este comparador, y se profundizará más en futuras investigaciones.

4.2.3. Mínima perturbación

Otro problema interesante, es usar esta herramienta para observar que tan susceptibles a perturbaciones son las redes comprimidas. En este caso, se usa un modo sencillo de compresión con pérdida, que llamamos *redondeo* o *prunning* donde, para un nivel de perturbación ε, se reemplazan los pesos $w \in B(0,\varepsilon)$ por $w = 0$. Con esto en mente, se busca comparar una red con su versión redondeada, y ver cual es el mínimo nivel de perturbación necesario para hacer que la versión redondeada difiera en una clasificación para una misma imagen.

Más formalmente, dadas dos redes $\mathcal{N}_1, \mathcal{N}_2$ y una imagen inicial x_0 perteneciente a la clase $C_0 \in [K]$, se modelará un problema para encontrar un $\varepsilon > 0$ y $x \in B(x_0,\varepsilon) \cap \mathbb{X}$ tal que $\mathcal{N}_1(x) = C_0$ y $\mathcal{N}_2(x) \neq C_0$. Este problema, asume que la red $\mathcal{N}_1$ clasifica correctamente, mientras que la red $\mathcal{N}_2$ se equivoca.

Para modelar esto, se modifican y agregan restricciones a la formulación original (PC). El primer cambio implica agregar una variable ε al problema (PC), y modificar las cotas sobre

el input dadas por la restricción (4.11). En cambio, se utiliza la siguiente restricción:

$$\text{máx}\{x_{0,j} - \varepsilon, 0\} \leq h_{1,j}^{(0)} \leq \text{mín}\{x_{0,j} + \varepsilon, 1\} \qquad \forall j \in [n] \tag{4.16}$$

Con esta restricción, se está utilizando la bola de norma $\mathcal{L}_\infty$, pero se podría usar otra semi-norma como se discutió en capítulos anteriores.

También, para asegurar que la red $\mathcal{N}_1$ clasifique correctamente, se agregan restricciones similares a (4.12). Concretamente, se agrega una variable binaria auxiliar b_k que vale 1 si $a_{2,k}^{(L+1)} \geq a_{2,C_0}^{(L+1)}$ y vale 0 sino. Esta variable, permitirá contar cuantas clases tienen mayor probabilidad que C_0 para la red $\mathcal{N}_2$. Luego, se agregan las siguientes restricciones:

$$a_{1,C_0}^{(L_1+1)} \geq a_{2,k}^{(L_2+1)} \qquad \forall k \in [K]\backslash C_0 \tag{4.17}$$

$$-M \cdot (1 - b_k) \leq a_{2,k}^{(L_2+1)} - a_{2,C_0}^{(L_2+1)} \leq M \cdot b_k \qquad \forall k \in [K]\backslash C_0 \tag{4.18}$$

$$\sum_{k \in [K]\backslash C_0} b_k \geq 1 \tag{4.19}$$

Donde (4.17) asegura que la red $\mathcal{N}_1$ clasifique correctamente. Si M es suficientemente grande, (4.18) asegura que si $b_k = 1$, entonces $a_{2,k}^{(L+1)} \geq a_{2,C_0}^{(L+1)}$ y si $b_k = 0$ se tiene lo contrario. Por último, (4.19), nos asegura que al menos hay una clase con mayor probabilidad que C_0 en la red $\mathcal{N}_2$, es decir, la segunda red clasifica erróneamente.

Luego se utiliza la función objetivo:

$$C(a, h, z) = -\varepsilon \tag{mineps}$$

De esta forma, se pueden encontrar (x, ε) tales que:

$$x \in B_\infty(x_0, \varepsilon) \subseteq \mathbb{X} \text{ y } \mathcal{N}_2(x) \neq \mathcal{N}_1(x) = C_0$$

Esto es, la mínima perturbación ε tal que exista un $x \in B_\infty(x_0, \varepsilon)$ que sea clasificado erróneamente por $\mathcal{N}_2$.

Como este comparador, está pensado para ser utilizado como certificador de una compresión con pérdida, se asumirá que $\mathcal{N}_1$ y $\mathcal{N}_2$ son redes con la misma arquitectura, por lo tanto $\forall l \in [L_1], \forall m \in [L_2] : n_l = n_m$ y $L_1 = L_2 = L$.

La suposición de que $\mathcal{N}_1$ clasifica correctamente está motivada porque $\mathcal{N}_1$ ocupa el rol de la red original y $\mathcal{N}_2$ de la red comprimida. Si es que resolvemos el problema intercambiando los roles de $\mathcal{N}_1$ y $\mathcal{N}_2$., se podrá analizar si existe un trade-off al comprimir la red.

4.3. Descripción de casos experimentales

Con el fin de poner a prueba en la práctica al comparador (PC), se implementan algunos de los comparadores mencionados en la sección anterior, y se realizan diversos experimentos usando el dataset MNIST y algunas redes neuronales entrenadas previamente. Al igual que en capítulos anteriores, todos los experimentos fueron ejecutados en el NLHPC, con las mismas asignaciones de recursos detalladas en la sección 2.2.2.

También, en virtud de los resultados de los capítulos previos, se decide usar ciertas configuraciones en las cuales se observó que el MIP se desempeñaba de mejor manera, como el uso de redes regularizadas, el cálculo de cotas usando Bounder versión LP, entre otras. Esta decisión se toma con el objetivo de reducir lo más posible los problemas de desempeño y poder obtener resultados en tiempos acotados para el desarrollo de la tesis.

4.3.1. Máxima diferencia

Para este comparador, se utilizan las arquitecturas señaladas en la tabla 4.1, entrenadas con regularización L1 con un factor de penalización $\lambda = 0.0002$. Luego se comparan todos las combinaciones posibles de estas redes.

Red	Arquitectura
fcnn1	(3000)
fcnn2	(800, 800)
fcnn5	(200, 200, 200, 200, 200)
fcnn10	(100,...,10)

Tabla 4.1: Arquitecturas para experimentos de (maxdiff)

Tanto para la diferencia en probabilidad como en selección de clase, se realizan 3 tipos de experimentos:

1. Comparar 2 redes usando el espacio inicial por defecto $\mathbb{X} = [0, 1]^{784}$. (`non_imgref`)
2. Comparar 2 redes acotando el espacio inicial a $\mathbb{X} = B_\infty(x_0^C, \delta)$. (`imgref_simple`)
3. Comparar 2 redes acotando el espacio inicial a $\mathbb{X} = B_d(x_0^C, \delta)$ con d la semi-métrica definida en Capítulo 3. (`imgref_neighbours`)

En los últimos 2 experimentos, se resuelve una instancia por cada clase C de MNIST. Para esto, se elige un representante de manera aleatoria que se mantiene fijo en los experimentos. También, se toma un radio $\delta = 0.05$ para obtener soluciones menos triviales, ya que si son similares a la imagen referencial, el modelo tiende a tener dificultades al encontrar una solución que las redes clasifiquen distinto.

Las cotas de activación se calculan usando el algoritmo 3 versión LP con $\overline{H}^{(0)}, H^{(0)}$ dados por el espacio inicial correspondiente, y un ε elegido de manera conveniente tal que se redondee un 90 % de los pesos totales, el valor de ε no es relevante, así que desde ahora solo se mencionará el redondeo en términos del *porcentaje de redondeo* p %.

4.3.2. Mínima perturbación

Para esta sección, con el fin de realizar una gran cantidad de experimentos, se decide utilizar una arquitectura de 2 capas con 15 nodos cada una, de manera de encontrar resultados óptimos en un marco de tiempo acotado.

Para usar el comparador de mínima perturbación, se realizan unos preparativos previos. Para cada clase $d \in [9]_0$ de MNIST, se obtiene de manera aleatoria un representante $x_d \in \mathbb{X}$ tal que $\mathcal{N}(x_d) = d$, el cuál es utilizado como input inicial. También, se generan 40 redes con distintos niveles de redondeos δ_p, dados por los porcentajes de *prunning* p, que van desde un 5 % hasta un 92.75 %, con un paso de un 2.25 %.

Luego, se realizan 400 experimentos (40 por cada dígito), donde se determina el par $(\overline{x}_d, \varepsilon)$ donde ε es la mínima perturbación tal que exista $\overline{x}_d \in B(x_d, \varepsilon)$ con el cual, $\mathcal{N}(\overline{x}_d) = d$ y $\mathcal{N}_{\delta_p}(\overline{x}_d) \neq d$. Esto es, el menor ε tal que la versión original clasifique a la imagen perturbada correctamente mientras que la versión redondeada de la red clasifique de manera distinta. Para todos los experimentos se utiliza la norma infinito, es decir, la métrica `simple`.

Estos experimentos, son realizados en el NLHPC, con las mismas configuraciones de los capítulos pasados, salvo que ahora se le asigna un tiempo límite mayor de 4 horas.

4.4. Resultados

4.4.1. Máxima diferencia: En Probabilidad

A continuación se presentan los resultados obtenidos para todas las combinaciones posibles de las arquitecturas seleccionadas. Primero se muestran las comparaciones entre las dos redes menos profundas, luego entre la menos profunda con la más profunda y finalmente entre las dos redes más profundas.

800x2 vs 200x5

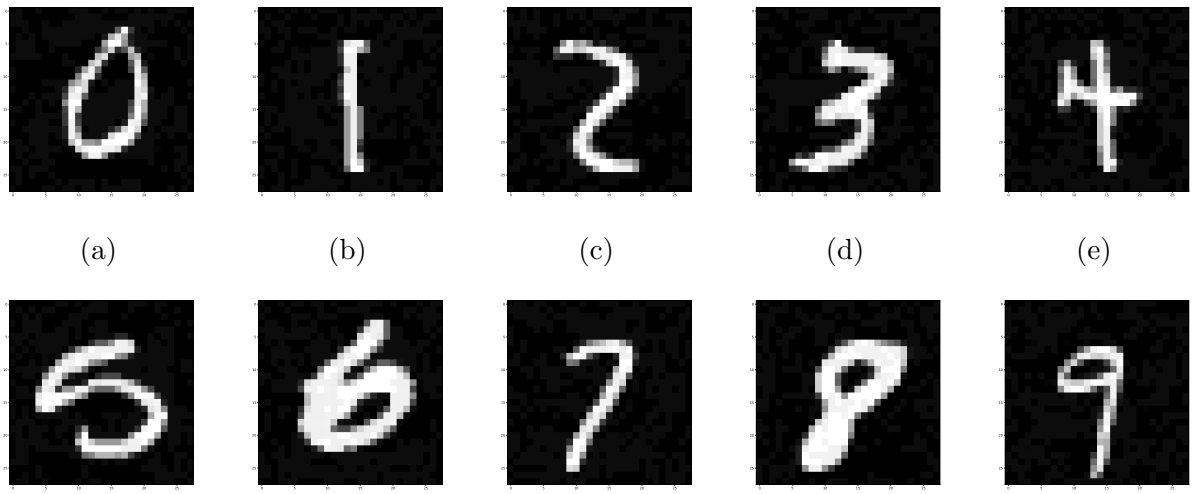
En la Figura 4.3 se muestran las soluciones que maximizan la diferencia en probabilidad para las redes 800x2 y 200x5 usando la métrica `simple` y en la Tabla 4.2, se observa de que manera clasifica cada red usando estas imágenes, y con que probabilidad lo hace. A primera vista, parece ser que sólo son digitos del 0 al 9, pero si se observa más de cerca en detalle, se puede apreciar que todas las imágenes tienen un ruido de fondo.

Tabla 4.2: Clasificaciones dadas por cada solución para 800x2 vs 200x5 usando la métrica `simple`.

Real	Red 1	Prob 1	Red 2	Prob 2	L1 diff
0	0	64 %	0	94 %	0,63
1	1	81 %	1	99 %	0,367
2	8	49 %	2	92 %	0,93
3	3	100 %	3	100 %	0,008
4	9	80 %	4	97 %	1,551
5	5	100 %	5	83 %	0,334
6	6	99 %	8	64 %	1,408
7	7	72 %	7	98 %	0,533
8	8	99 %	9	92 %	1,861
9	9	90 %	9	99 %	0,187

A pesar de este ruido de fondo, la mayoría de las imágenes son bien clasificadas por ambas redes, siendo la excepción las Figuras 4.2c, 4.2e, 4.2g y 4.2i. Se observa que en las imágenes bien clasificadas, sólo varía la probabilidad de certeza de las redes, siendo el máximo una diferencia en norma $\mathcal{L}_1$ de 0.63

Mientras tanto, en las imágenes que confundieron a alguna de las redes, estas tienden a equivocarse con los dígitos 8 y 9. Esto se explica por el ruido de fondo, que para los casos de las Figuras 4.2c, 4.2e y 4.2g actúan cerrando y cambiando la formas de los dígitos, por ejemplo en el dígito 4, se ven algunos píxeles que unen el 4 haciendolo más similar a un 9. En el caso de la Figura 4.2j, se dio la casualidad de que el dígito ya se asemeja bastante a un nueve por lo que no fue muy difícil confundir a la red.



(a) (b) (c) (d) (e)

(f) (g) (h) (i) (j)

Figura 4.2: Soluciones para el caso 800x2 vs 200x5 usando métrica `simple`.

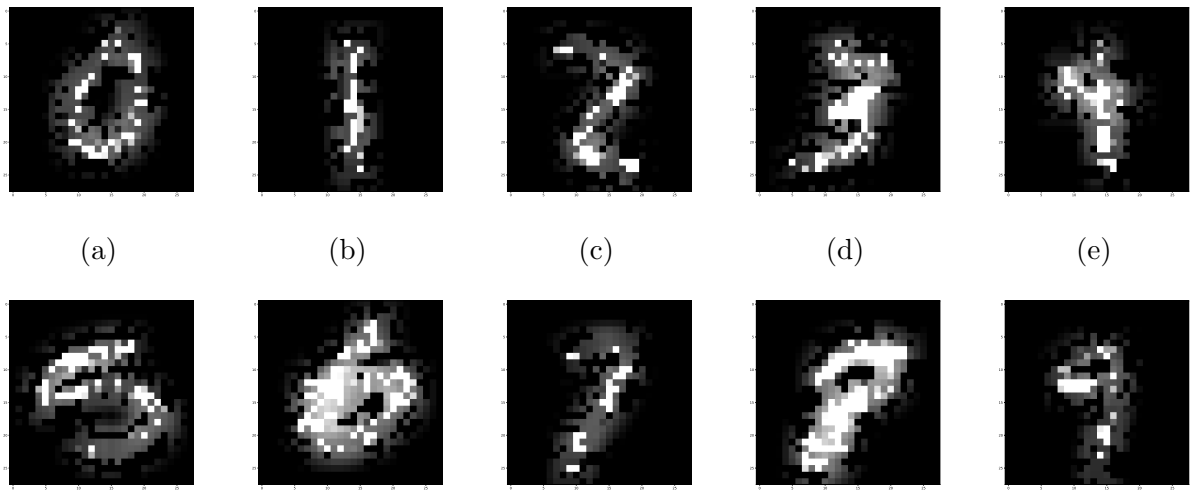
De aquí se puede concluir que esta métrica da soluciones muy cercanas a la imagen de referencia inicial, y en general sólo permite agregar ruido de fondo, utilizando el rango de intensidad de cada píxel que puede variar.

Con la métrica `neighbours` en cambio, es más sencillo confundir a las redes, como se puede observar en la tabla 4.3 donde en todos los casos al menos una de las redes clasifica erróneamente, teniendo incluso casos como los de las Figuras 4.3e y 4.3h donde ambas redes clasifican incorrectamente.

Tabla 4.3: Clasificaciones dadas por cada solución para 800x2 vs 200x5 usando la métrica `neighbours`.

Real	**Red 1**	**Prob 1**	**Red 2**	**Prob 2**	**L1 diff**
0	6	99 %	0	92 %	1,965
1	1	92 %	3	87 %	1,917
2	2	98 %	8	99 %	1,97
3	3	100 %	8	75 %	1,996
4	8	97 %	9	97 %	1,985
5	5	98 %	9	98 %	1,997
6	6	94 %	0	99 %	1,998
7	8	88 %	3	98 %	1,945
8	8	100 %	7	99 %	1,998
9	5	99 %	9	98 %	1,981

A diferencia de la Figura 4.2, en la Figura 4.3 se observan soluciones mucho más complejas. Cómo en este caso el espacio $\mathbb{X}$ se restringe de manera tal que se conserve la forma de la imagen original, no basta con utilizar el ruido de fondo para confundir a las redes.



(a) (b) (c) (d) (e)

(f) (g) (h) (i) (j)

Figura 4.3: Soluciones para el caso 800x2 vs 200x5 usando métrica `neighbours`.

Las soluciones obtenidas en este caso, se aprovechan del rango de intensidad que pueden tener los píxeles dentro de la forma del dígito, obteniendo soluciones que incluso para el ojo humano es difícil identificar de que dígito se generaron originalmente, como las Figuras 4.3e, 4.3h y 4.3i.

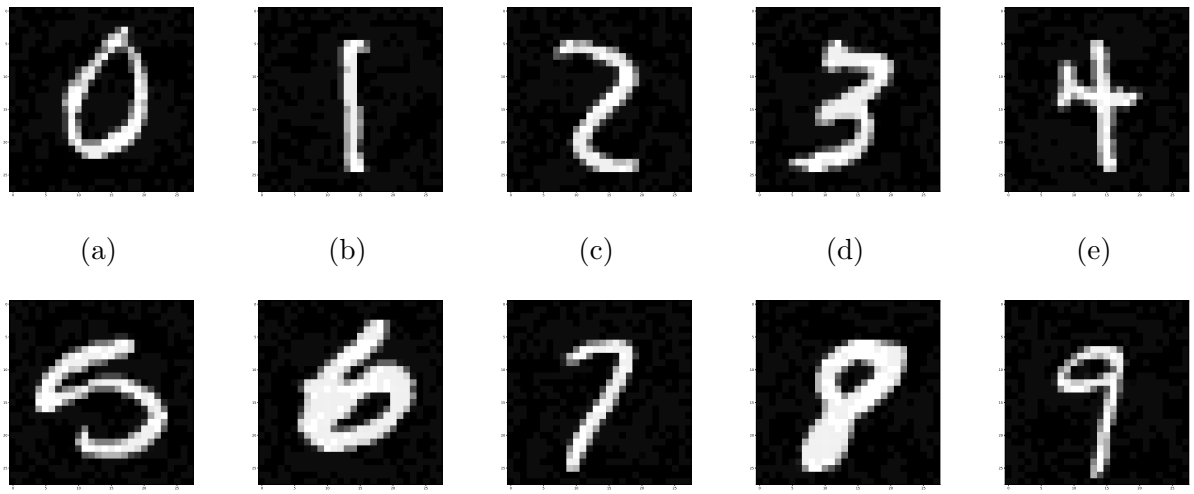
800x2 vs 100x10

En la Figura 4.4 y Tabla 4.4 se observan las soluciones obtenidas para la métrica `simple`, obteniendo comportamientos similares a la comparación anterior por ejemplo, la presencia del ruido de fondo para confundir a las redes.

Tabla 4.4: Clasificaciones dadas por cada solución para 800x2 vs 100x10 usando la métrica `simple`.

Real	Red 1	Prob 1	Red 2	Prob 2	L1 diff
0	0	68 %	0	99 %	0,634
1	1	95 %	8	85 %	1,639
2	2	90 %	8	98 %	1,793
3	3	100 %	3	98 %	0,033
4	9	73 %	4	98 %	1,772
5	5	99 %	5	88 %	0,239
6	6	97 %	6	99 %	0,056
7	7	96 %	2	94 %	1,829
8	8	72 %	8	99 %	0,557
9	9	95 %	4	78 %	1,548

En este caso, se observa que la mitad de los dígitos son bien clasificados por ambas redes



(a) (b) (c) (d) (e)

(f) (g) (h) (i) (j)

Figura 4.4: Soluciones para el caso 800x2 vs 100x10 usando métrica `simple`.

y en la otra mitad una de las redes se equivoca. Este comportamiento es muy similar al caso anterior donde la proporción era 4:6.

En los dígitos bien clasificados por ambas redes, se observa nuevamente que la diferencia $\mathcal{L}_1$ de la probabilidad no es tan grande, siendo el máximo de 0.634. Para los números con clasificación incorrecta, se vuelve a observar la presencia de dígitos como el 8 o 9. Esto sugiere que usando ciertos números es más fácil engañar a las redes.

A diferencia del caso 800x2 vs 200x5, acá se observa una predominancia del error en la red más profunda. También, en los casos donde alguna red se equivoca, en general se observa un valor mayor de la diferencia $\mathcal{L}_1$ comparado con el caso 800x2 vs 200x5. Esto parece indicar que el ruido de fondo en estas imágenes genera un error que se va incrementando en capas más profundas.

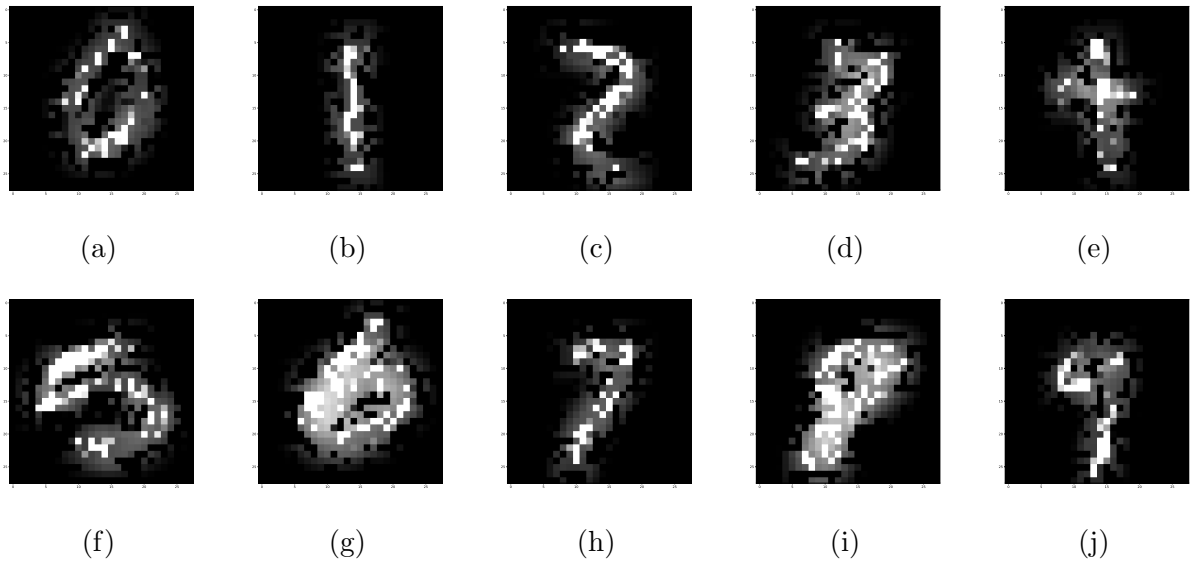
En la Figura 4.5 y Tabla 4.5 se observan las soluciones para la métrica `neighbours`. Al igual que en el caso de las dos redes menos profundas, se observa que con esta métrica en todos los dígitos se logra confundir al menos una red.

Sin embargo, en este caso aumenta la cantidad de dígitos en donde ambas redes se confunden. Esto puede explicarse ya que, como las redes son menos similares (una es corta y la otra más profunda), es más fácil obtener resultados distintos entre ambas redes.

Nuevamente en la Figura 4.5 se observa como las soluciones se aprovechan de la intensidad de los píxeles para confundir a la redes. Algunos ejemplos a destacar son las Figuras 4.5a, 4.5c, 4.5e y 4.5j, donde se pueden encontrar similitudes con los números que erróneamente clasifican las redes.

Tabla 4.5: Clasificaciones dadas por cada solución para 800x2 vs 100x10 usando la métrica `neighbours`.

Real	Red 1	Prob 1	Red 2	Prob 2	L1 diff
0	6	72 %	2	99 %	1,959
1	1	96 %	8	99 %	1,984
2	7	98 %	8	99 %	1,988
3	1	99 %	9	99 %	1,993
4	9	95 %	8	100 %	1,981
5	5	100 %	7	87 %	1,981
6	2	55 %	6	64 %	1,974
7	7	100 %	8	80 %	1,991
8	7	99 %	8	57 %	1,962
9	5	89 %	4	98 %	1,976



(a) (b) (c) (d) (e)

(f) (g) (h) (i) (j)

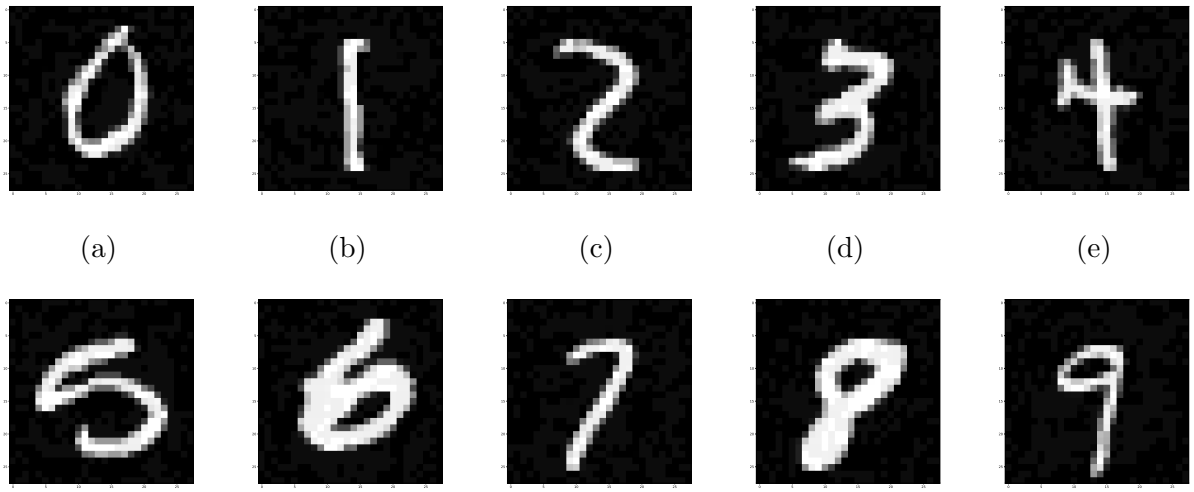
Figura 4.5: Soluciones para el caso 800x2 vs 100x10 usando métrica `neighbours`.

200x5 vs 100x10

Por último, en la Figura 4.6 y la Tabla 4.6 se presentan las soluciones para las dos redes más profundas usando la métrica `simple`. Se vuelven a observar los mismos comportamientos que en los dos casos anteriores como: presencia de ruido de fondo, preponderancia del error en la red más profunda, confusión con dígitos 8 o 9, y proporción entre números mal clasificados y bien clasificados cercanos a 50 %/50 %.

Tabla 4.6: Clasificaciones dadas por cada solución para 200x5 vs 100x10 usando la métrica `simple`.

Real	Red 1	Prob 1	Red 2	Prob 2	L1 diff
0	0	86 %	0	100 %	0,278
1	1	100 %	8	82 %	1,655
2	2	95 %	8	99 %	1,977
3	3	100 %	3	99 %	0,026
4	4	64 %	4	97 %	0,71
5	5	82 %	5	99 %	0,351
6	8	55 %	6	100 %	1,34
7	7	99 %	8	92 %	1,944
8	9	97 %	8	100 %	1,957
9	9	99 %	4	62 %	1,5



(a) (b) (c) (d) (e)

(f) (g) (h) (i) (j)

Figura 4.6: Soluciones para el caso 200x5 vs 100x10 usando métrica `simple`.

Para la métrica `neighbours`, se obtienen las soluciones de la Figura 4.7 y la Tabla 4.7, con comportamientos muy similares al caso 800x2 vs 100x10 salvo pequeñas diferencias.

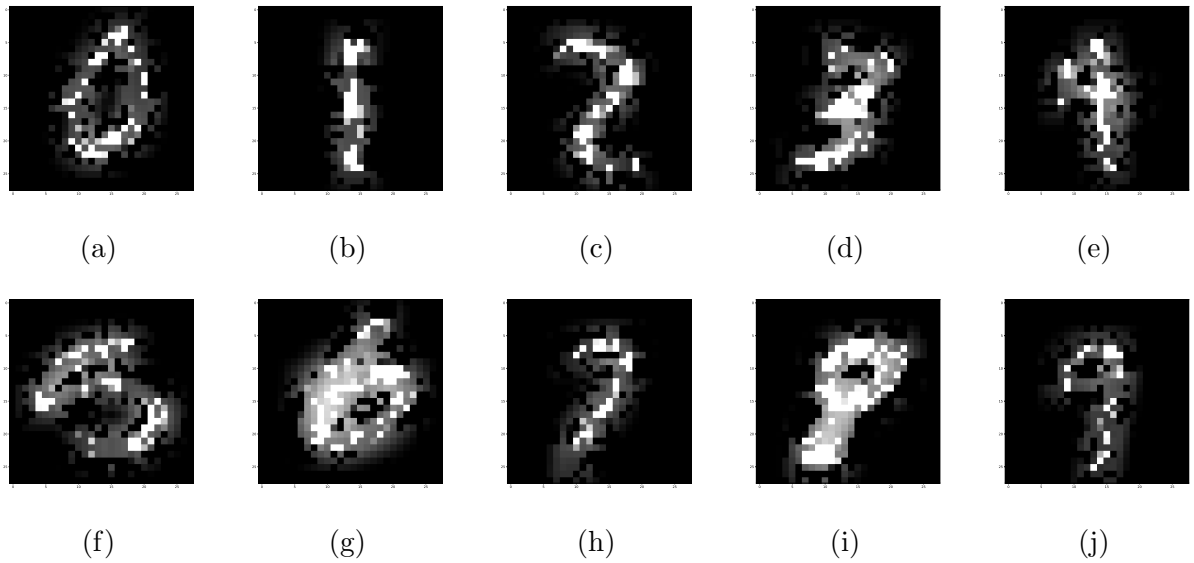
En este caso por ejemplo, aparece el fenómeno de confundirse con un 8 o 9 pero usando una métrica distinta a `simple`, ya que en 10 de las 14 clasificaciones incorrectas una red se confundió con uno de estos dígitos.

También, los casos en donde ambas redes se equivocan bajaron en comparación al caso 800x2 vs 100x10, pero siguen siendo más que en 800x2 vs 200x5. Esto parece confirmar, que la similitud de las arquitecturas influye en que tan difícil se hace confundir a las redes.

Se destacan soluciones como Figuras 4.7c, 4.7e y 4.7i que son suficientemente buenas para confundir al ojo humano.

Tabla 4.7: Clasificaciones dadas por cada solución para 200x5 vs 100x10 usando la métrica `neighbours`.

Real	Red 1	Prob 1	Red 2	Prob 2	L1 diff
0	2	97 %	0	76 %	1,948
1	3	100 %	8	100 %	1,996
2	7	71 %	8	100 %	1,999
3	4	64 %	3	99 %	1,989
4	9	98 %	8	94 %	1,992
5	9	59 %	8	99 %	1,984
6	8	97 %	6	99 %	1,974
7	7	97 %	9	74 %	1,98
8	9	100 %	8	99 %	1,989
9	9	98 %	8	98 %	1,979



(a) (b) (c) (d) (e)

(f) (g) (h) (i) (j)

Figura 4.7: Soluciones para el caso 200x5 vs 100x10 usando métrica `neighbours`.

4.4.2. Mínima perturbación

A continuación se presenta el resumen de los resultados obtenidos para los experimentos de mínima perturbación. En la Figura 4.8 se aprecia cómo se comporta el valor de ε para cada dígito en función del porcentaje de redondeo que se utiliza. Vemos que hay ciertos dígitos que poseen un ε mayor en general, como es el caso del número 8. Esto se condice con resultados que se observaron en la sección 4.4.1, donde ciertas clases parecen ser más fuertes que otras.

También en la Figura 4.8, se observa que para redes con un porcentaje de redondeo menor a 40 %, el modelo MIP asociado tiene más problemas para encontrar una solución óptima. Los registros de Gurobi indican que esto se debe a la dificultad de encontrar soluciones factibles para el problema primal. También, en general para valores menores a 20 %, no se encontraron

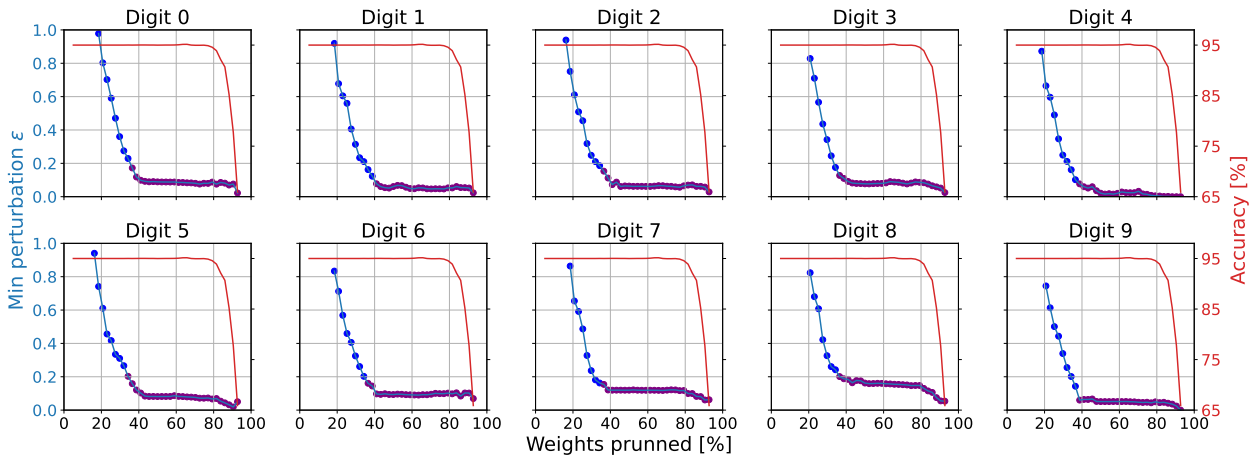


Figura 4.8: Eje izquierdo: Mínimo nivel de perturbación ε necesario para confundir a la red redondeada. Eje derecho: Accuracy de la red para el porcentaje de pesos redondeados. Los puntos morados representan las instancias donde se alcanza el óptimo, mientras que las azules son la mejor solución factible encontrada en ese caso.

soluciones factibles, esto puede ser por dos razones: no existe solución para estos casos ya que las redes son demasiado similares, o bien, se vuelve al problema anterior de la dificultad de generar soluciones para el problema primal. En cualquiera de los casos, se encontró que las heurísticas por defecto de Gurobi tienen grandes dificultades para estos tipos de problemas, lo que presenta un espacio de mejora en futuras investigaciones.

Se puede notar que hay un intervalo en el cual el accuracy aumenta levemente entre el 60 % y 70 %. En la Figura 4.9, se hace énfasis en este intervalo de redondeo para observar como afecta al valor de ε.

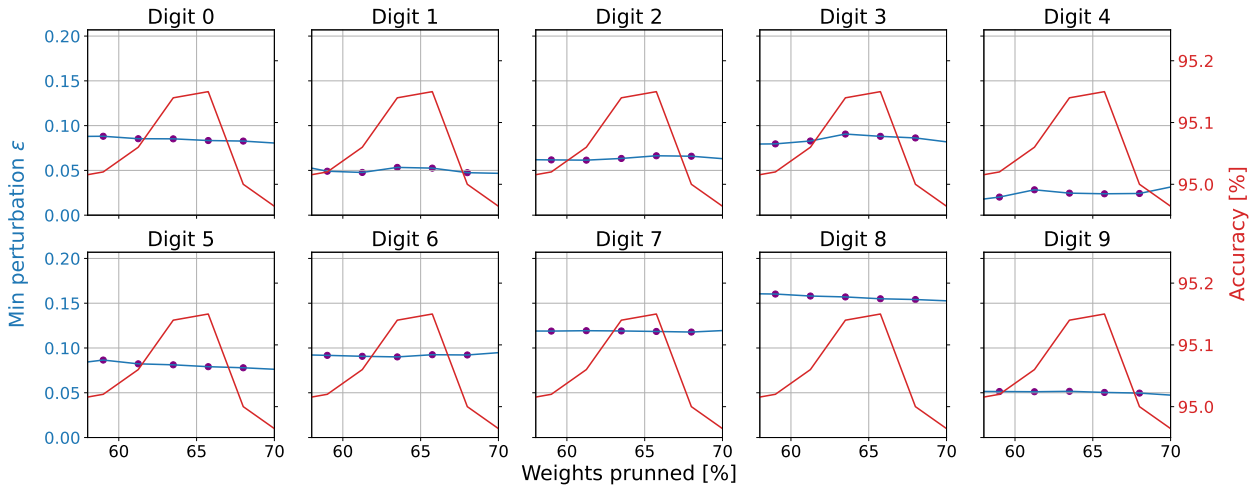


Figura 4.9: Eje izquierdo: Mínimo nivel de perturbación ε necesario para confundir a la red redondeada. Eje derecho: Accuracy de la red para el porcentaje de pesos redondeados. Los puntos morados representan las instancias donde se alcanza el óptimo, mientras que las azules son la mejor solución factible encontrada en ese caso. Enfoque en intervalo de interés.

Como se observa en la Figura 4.9, la mínima perturbación necesaria parece ser constante en este intervalo a pesar de lo que se esperaría. La intuición nos indica, que si la red redondeada es más precisa en este rango, se debería necesitar un mayor nivel de perturbación para confundirla. Esto en la práctica no ocurre, y puede deberse a que el cambio en la precisión no es tan significativo o bien el accuracy no es una muy buena métrica para determinar esto.

También, se pone énfasis en el tramo de redondeo donde la red empieza a perder rápidamente su precisión. Como se observa en la Figura 4.10, como regla general la mínima perturbación necesaria para confundir a la red comprimida tiende a bajar, lo cual es un resultado esperado, ya que mientras menos precisión tenga, uno esperaría que fuera más fácil encontrar una imagen que se clasifica bien por la red original y mal por la comprimida.

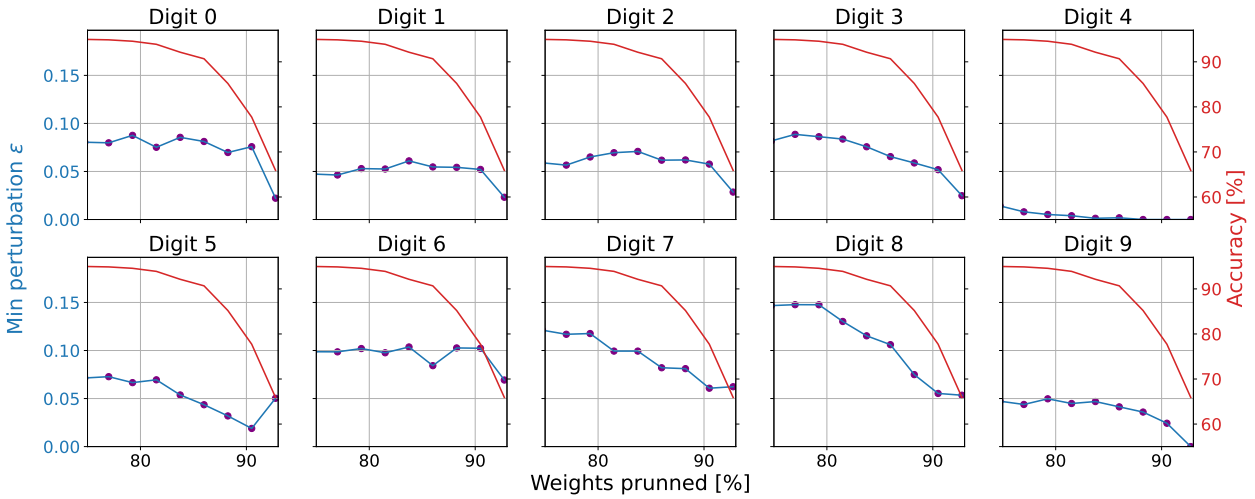


Figura 4.10: Eje izquierdo: Mínimo nivel de perturbación ε necesario para confundir a la red redondeada. Eje derecho: Accuracy de la red para el porcentaje de pesos redondeados. Los puntos morados representan las instancias donde se alcanza el óptimo, mientras que las azules son la mejor solución factible encontrada en ese caso. Enfoque en intervalo con 75 % y 93 % de redondeo.

4.4.3. Discusión y comentarios

En este capítulo, se pusieron a prueba algunos de los comparadores presentados y se realizaron pruebas empíricas con ellos. Estos resultados pudieron ser obtenidos por usar un modelo MIP como el del problema (PC).

Si bien los resultados en particular no son muy relevantes, por ser un estudio sobre el dataset MNIST usando redes no muy comunes, sí tienen un valor intrínseco, pues demuestran que el uso del comparador MIP es factible y además es una herramienta que permite responder preguntas complejas sobre las redes, realizar análisis muy puntuales y demostrar algunas propiedades sobre ellas.

Estos tipos de experimentos, se pueden extrapolar a redes más comunes hoy en día, como

redes con capas de convolución, con capas de pooling, entre otras, y por lo tanto se pueden hacer investigaciones más ambiciosas en el futuro con esta herramienta. Por ejemplo, se podría idear un método de *hackeo* de redes, donde se entrena una red y se compara con la red a vulnerar, usando el MIP (PC), para ver si son idénticas. Si no es el caso, se aplican cambios a la red hasta que el comparador indique que ambas redes son iguales. Esto se podría realizar si se tiene una gran capacidad de cómputo y si se conocen algunas propiedades de la red a vulnerar como su arquitectura, sobre que inputs opera, etc.

Cómo nota general de este capítulo, se desprende que el comparador es una herramienta factible de usar, es muy versátil pues puede servir para una gran cantidad de problemas que se planteen, pero presenta problemas de escalamiento y por lo tanto queda la interrogante de cómo se comportará con redes más similares a las que se usan actualmente, que son más profundas pero en general con más *sparsity* en sus matrices de pesos. También, queda pendiente poner a prueba el comparador con redes sin regularización y ver cuánto empeora su desempeño en estos casos.

Conclusiones finales

En esta tesis se presentaron e investigaron diversos problemas abarcando tópicos de aprendizaje de máquinas, compresión de redes neuronales, optimización entera, cálculo numérico, entre otros. Se hicieron varias pruebas de concepto sobre aplicaciones de modelos IP a la representación de redes neuronales y se obtuvieron resultados de interés para ambas áreas.

Para la comunidad de Machine Learning, se presentaron herramientas versátiles como el verificador del Capítulo 3, el cual se puede utilizar por ejemplo para la creación de casos adversariales. También, se presentó el comparador que permite estudiar dos o más redes neuronales a través del modelo del Capítulo 4, el cual se puede utilizar por ejemplo para la certificación de compresiones con pérdida. Se estudiaron y mostraron otras posibles aplicaciones del comparador y, a la vez, sus limitaciones y desventajas.

Para la comunidad de optimización entera, se mostró el impacto de las cotas Big-M en este tipo de aplicaciones. Cómo estas cotas influyen en los tiempos de resolución, cómo calcularlas y el impacto que tienen en la calidad de las soluciones en los modelos (Big-M). También, se observó que Gurobi presenta dificultades para encontrar soluciones primales en estos modelos, siendo necesario el uso de *callbacks*. Esto presenta una posible línea de investigación sobre heurísticas para estos MIPs de redes neuronales.

Se observó también en el Capítulo 4 que la definición de un comparador $C(a, h, z)$ es altamente no trivial, y debe estar muy bien pensado para lo que se pretenda hacer con las redes. Esto presenta una linea de trabajo futuro para investigadores que conozcan muy bien ambas áreas (ML y MIP), donde se definan y estudien nuevos comparadores que logren un objetivo motivado desde el aprendizaje de máquinas, pero de manera eficiente con los conocimientos de formulaciones MIP. ¿Cómo definir un comparador para lograr un objetivo sin complicar tanto la formulación del problema IP? ¿Qué comparadores son realmente útiles para problemas de ML?.

Trabajo Futuro

De los resultados de esta tesis se desprenden muchas líneas de investigación futura, cómo las ya mencionadas, pero el más directo es el estudio del escalamiento del problema. De

estos resultados, surgen algunas preguntas naturales: ¿Cómo escalan estos modelos MIP con el tamaño, profundidad y complejidad de la red?, ¿Se pueden utilizar estos modelos en arquitecturas de redes más modernas?, Si es así, ¿Empeora el desempeño?

Cómo se mencionó en algunos capítulos, es posible extender el trabajo realizado durante la tesis a otras arquitecturas como LeNet o ResNet, como también se pueden usar otros datasets más complejos como CIFAR. Sin embargo, no se tiene certeza de si empeorará el desempeño del modelo o no, ya que se presenta un trade-off entre el aumento de variables y la profundidad de estas arquitecturas con el sparsity de sus matrices y el número de parámetros necesarios para representarlas. Si bien la profundidad y el número de variables aumentará los tiempos de resolución, el sparsity de las matrices disminuirá los tiempos de resolución como se vio en el Capítulo 2.

Bibliografía

[1] Ross Anderson, Joey Huchette, Will Ma, Christian Tjandraatmadja, and Juan Pablo Vielma. Strong mixed-integer programming formulations for trained neural networks. *Math. Program.*, 183(1):3–39, September 2020.

[2] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, New York, NY, USA, 2006.

[3] F. Ceccon, J. Jalving, J. Haddad, A. Thebelt, C. Tsay, C. D. Laird, and R. Misener. Omlt: Optimization & machine learning toolkit, 2022.

[4] Sanjeeb Dash, Oktay Gunluk, and Dennis Wei. Boolean Decision Rules via Column Generation. *Advances in Neural Information Processing Systems*, 31, 2018.

[5] Jonathan Frankle and Michael Carbin. The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks. *arXiv*, March 2018.

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

[7] Joey Huchette, Gonzalo Muñoz, Thiago Serra, and Calvin Tsay. When Deep Learning Meets Polyhedral Theory: A Survey. *arXiv*, April 2023.

[8] Guy Katz, Clark Barrett, David Dill, Kyle Julian, and Mykel Kochenderfer. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. *arXiv*, February 2017.

[9] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv*, December 2014.

[10] Qianli Liao, Brando Miranda, Andrzej Banburski, Jack Hidary, and Tomaso Poggio. A Surprising Linear Relationship Predicts Test Performance in Deep Networks. *arXiv*, July 2018.

[11] Alejandra Martins. Qué es el "aprendizaje profundo" de la inteligencia artificial y cómo ya está cambiando la vida de millones de personas en todo el mundo - BBC News Mundo, August 2017. [Online; accessed 31. Aug. 2022].

[12] James O' Neill. An Overview of Neural Network Compression. *arXiv*, aug 2020.

[13] Sebastian Pokutta, Christoph Spiegel, and Max Zimmer. Deep Neural Network Training with Frank-Wolfe. *arXiv*, October 2020.

[14] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, October 1986.

[15] Thiago Serra, Abhinav Kumar, and Srikumar Ramalingam. Lossless Compression of Deep Neural Networks. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 417–430. Springer, Cham, Switzerland, September 2020.

[16] Thiago Serra, Christian Tjandraatmadja, and Srikumar Ramalingam. Bounding and Counting Linear Regions of Deep Neural Networks. In *International Conference on Machine Learning*, pages 4558–4566. PMLR, July 2018.

[17] Thiago Serra, Xin Yu, Abhinav Kumar, and Srikumar Ramalingam. Scaling Up Exact Neural Network Compression by ReLU Stability. *Advances in Neural Information Processing Systems*, 34:27081–27093, December 2021.

[18] Qinghua Tao, Li Li, Xiaolin Huang, Xiangming Xi, Shuning Wang, and Johan A. K. Suykens. Piecewise linear neural networks and deep learning. *Nat. Rev. Methods Primers*, 2(42):1–17, June 2022.

[19] Christian Tjandraatmadja, Ross Anderson, Joey Huchette, Will Ma, Krunal Kishor Patel, and Juan Pablo Vielma. The Convex Relaxation Barrier, Revisited: Tightened Single-Neuron Relaxations for Neural Network Verification. *Advances in Neural Information Processing Systems*, 33:21675–21686, 2020.

[20] Vincent Tjeng, Kai Xiao, and Russ Tedrake. Evaluating Robustness of Neural Networks with Mixed Integer Programming. *arXiv*, nov 2017.

[21] Calvin Tsay, Jan Kronqvist, Alexander Thebelt, and Ruth Misener. Partition-Based Formulations for Mixed-Integer Optimization of Trained ReLU Neural Networks. *Advances in Neural Information Processing Systems*, 34:3068–3080, December 2021.