Centro de Investigación en Matemáticas, A.C.

CIMAT

Implementación y análisis de algunos modelos de interpretación de redes neuronales

T E S I S

Que para obtener el grado de
Maestro en Ciencias
con especialidad en
Probabilidad y Estadística

P r e s e n t a:
Isaías Manuel Ramírez Bañales

Director de tesis:
José Andrés Christen Gracia

Autorización de la versión final

Guanajuato, Gto. 30 de septiembre de 2019

Centro de Investigación en Matemáticas, A.C.

Acta de Examen de Grado

CIMAT
CENTRO DE INVESTIGACION EN MATEMATICAS A. C.

Acta No.: 155

Libro No.: 002

Foja No.: 155

En la Ciudad de Guanajuato, Gto., siendo las 16:00 horas del día 30 de septiembre del año 2019, se reunieron los miembros del jurado integrado por los señores:

DR. ROGELIO RAMOS QUIROGA **(CIMAT)**
DRA. LILIA LETICIA RAMÍREZ RAMÍREZ **(CIMAT)**
DR. JOSÉ ANDRÉS CHRISTEN GRACIA **(CIMAT)**

bajo la presidencia del primero y con carácter de secretario el segundo, para proceder a efectuar el examen que para obtener el grado de

MAESTRO EN CIENCIAS
CON ESPECIALIDAD EN PROBABILIDAD Y ESTADÍSTICA

Sustenta

ISAÍAS MANUEL RAMÍREZ BAÑALES

en cumplimiento con lo establecido en los reglamentos y lineamientos de estudios de posgrado del Centro de Investigación en Matemáticas, A.C., mediante la presentación de la tesis

"IMPLEMENTACIÓN Y ANÁLISIS DE ALGUNOS MODELOS DE INTERPRETACIÓN DE REDES NEURONALES"

Los miembros del jurado examinaron alternadamente al (la) sustentante y después de deliberar entre sí resolvieron declararlo (a):

APROBADO

DR. ROGELIO RAMOS QUIROGA
Presidente

DRA. LILIA LETICIA RAMÍREZ RAMÍREZ
Secretario

DR. JOSÉ ANDRÉS CHRISTEN GRACIA
Vocal

CIMAT
DIRECCIÓN GENERAL

DR. VÍCTOR MANUEL RIVERO MERCADO
Director General

Agradecimientos

Agradezco a mi madre Karla Bañales por siempre apoyarme y creer en mi, por alentarme a seguir mis sueños, por mostrarme que el trabajo arduo y la honestidad son dos pilares esenciales en la vida, por enseñarme a disfrutar, a vivir con pasión y con entrega, por las desveladas y por toda la guía que me ha brindado.

Agradezco a Victoria Revoreda por mostrarme la ilusión y los sueños a través de la inocencia, por cuestionarme sobre la vida que los adultos deciden tomar y por llenarme de alegrías con una gran variedad de bromas a todas horas.

Quiero agradecer a Rubí Santos por haberme acompañado durante toda la maestría, desde los lapices del examen de admisión hasta la cena del último día de correcciones de tesis, por haberme inspirado y mostrado que puedo ser mejor de lo que creía posible ser, por nunca dejar de creer en mi y hacerme sentir que tenía un apoyo incondicional. Agradezco también a la familia Santos Ríos por su paciencia y motivación.

Al Doctor Andrés Christen le agradezco el apoyo brindado para la realización de esta tesis, por haberme orientado durante la maestría, por mostrar ser un profesor organizado y comprometido con un gran sentido del humor. Aprovecho para externarle mi admiración y respetos.

Agradezco a mis amigos por siempre estar ahí para mi y estimarme como soy, por estar en los días buenos y malos. Agradezco en particular a Brandon que esté donde esté no lo siento lejano y espero pronto tener la oportunidad de brindar juntos de nuevo. También agradezco a Juan porque tras 2 años de convivir de sol a sol, siempre hemos mostrado apoyo y compañerismo.

Gracias al Centro de Investigación en Matemáticas (CIMAT), por haberme permitido ser parte de una comunidad llena de compañerismo, por darme la oportunidad aprender de personas tan entregadas y comprometidas y por haberme brindado condiciones óptimas para cursar un posgrado.

Agradezco también al Consejo Nacional de Ciencia y Tecnología (CONACYT), por el apoyo económico con una beca de maestría.

Resumen

En esta tesis se hace una revisión bibliográfica, implementación y discusión de los algoritmos recientes para la búsqueda de prototipos y generación de mapas de calor para entender y explicar redes neuronales convolucionales, además se presentan los códigos necesarios para poderlos aplicar utilizando Python. También se discuten ventajas y desventajas de los algoritmos presentados y una discusión acerca de la situación actual del entendimiento e interpretación de una red neuronal.

Palabras Clave

Redes neuronales, Máquinas restringidas de Boltzmann, Propagación de relevancia por capas, Activación maximización, Redes adversarias.

Índice general

Agradecimientos I

Resumen III

Introducción 1

1. Introducción a redes neuronales para clasificación 3

1.1. Redes neuronales . . . 3

1.2. Descenso de gradiente y propagación hacia atrás . . . 7

1.2.1. Descenso de gradiente . . . 7

1.2.2. Descenso estocástico de gradiente . . . 8

1.2.3. Propagación hacia atrás . . . 10

1.3. Ajuste de redes en Python . . . 11

2. Técnicas de interpretación y explicación 15

2.1. Motivación . . . 15

2.2. Técnicas de interpretación . . . 16

2.2.1. Activación maximización . . . 16

2.2.2. AM con experto . . . 18

2.2.2.1. Máquinas restringidas de Boltzmann . . . 18

2.2.3. AM con decodificador . . . 24

2.2.3.1. Redes adversarias generativas . . . 25

2.3. Métodos de explicación 27
2.3.1. Análisis de sensibilidad 28
2.3.2. Propagación de relevancia por capas 29

3. Aplicación de las técnicas de interpretación en la base MNIST **31**
3.1. Activación maximización 31
3.2. AM con experto 34
3.3. AM con decodificador 36
3.4. Propagación de relevancia por capas 38

4. Conclusiones y comentarios **41**
4.1. Pruebas con números de 7 segmentos 41
4.2. Ejemplos adversarios 44
4.3. Algoritmos de explicación e interpretación 45

Introducción

Las redes neuronales surgen como un paradigma de programación inspirado en el aprendizaje de los seres humanos. Si bien las personas podemos discernir con facilidad entre una manzana y una piña, resulta complicado explicar algorítmicamente como es que este proceso se lleva acabo en nuestro cerebro, de ahí que cuando queremos crear un programa de computadora el cuál pueda discernir qué fruta se presenta en una imagen, no le saturemos de condicionales sino que permitimos al programa "aprender" poco a poco con una amplia serie de ejemplos hasta ser capaz de apreciar distintas características en las imágenes para poder con ellas realizar la clasificación de manera correcta.

El uso de redes neuronales ha tenido un crecimiento importante durante los últimos años por su gran precisión para la clasificación de imágenes y su fácil implementación. Sin embargo presentan un gran inconveniente, si bien tienen un gran desempeño al momento de clasificar, el gran número de parámetros que tienen hace que sean relegadas al uso de "cajas negras", puesto que no resultan fácilmente interpretables. Actualmente se realiza un esfuerzo por encontrar maneras de entender las redes neuronales (y otros tipos de clasifadores) para poder aprender de ellas, modificarlas y ser consciente de algunas de sus limitaciones.

En las maneras de entender las redes neuronales que clasifican imágenes destaca la búsqueda de "prototipos" para las diferentes clases existentes, es decir buscar (o crear) imágenes consideradas representativas por la red para cada categoría, esto para que el especialista pueda entender cómo la red neuronal asigna elementos a una categoría específica. Además de los prototipos también existen herramientas diseñadas para poder generar "mapas de calor" para entender qué partes de la imagen de entrada son las que estimulan a la red neuronal para ser

asignada a una categoría en particular.

Las redes neuronales para la clasificación de imágenes tienen distintas áreas de aplicación, un área que hace de vital importancia el poder identificar qué está pasando es la que usa imágenes como asistencia para el diagnóstico de enfermedades dermatológicas, como en [5] o en la detección de rostros como en [1].

Es importante que el médico especialista tome medidas para curar a su paciente, sin embargo, si su diagnóstico no coincide con el de la red neuronal es valioso que el médico tenga la capacidad de vislumbrar por qué la red está tomando esa decisión y esto aprovecharlo para generar un mejor diagnóstico. Otra ventaja de comprender mejor una red neuronal en la dermatología es que se puedan representar distintas patologías encontrando prototipos de las categorías en la red neuronal, esto con la finalidad de que médicos en entrenamiento puedan enfocarse en analizar detalles específicos y tengan ejemplos modelo.

Este trabajo es una monografía donde se exponen por medio de ejemplos el uso de algunas técnicas creadas para la interpretación y explicación de redes neuronales y el código necesario para su implementación, teniendo como eje central el artículo [14], el cual es un compendio de algunos métodos para entender y explicar redes neuronales.

El primer capítulo es una breve introducción a las redes neuronales, se presentan conceptos básicos: la notación que se utilizará, algoritmos para su ajuste como lo es el descenso estocástico de gradiente y se ajusta la red con la que se realizarán los ejemplos de la tesis.

En el capítulo dos se exponen los algoritmos de interpretación y explicación de redes neuronales, la idea es que el lector pueda entender el planteamiento de cada uno de los métodos.

El capítulo tres presenta ejemplos de los algoritmos expuestos en el capítulo 2 utilizando la base de datos MNIST, además se dan los códigos necesarios para poder implementar los algoritmos.

Por último el capítulo cuatro corresponde a las conclusiones generadas de haber realizado la investigación y se hacen comparaciones entre los algoritmos.

Capítulo 1

Introducción a redes neuronales para clasificación

En este capítulo se presentan conceptos básicos sobre redes neuronales, esto con la intención de que el texto sea autocontenido y que el lector tenga las bases necesarias para entender los modelos desarrollados en esta tesis, para esto se exponen los algoritmos *ascenso de gradiente* y *propagación hacia atrás*, también se define qué es una red neuronal, las limitaciones que se presentan en su uso y cómo realizar el ajuste de una red neuronal.

1.1. Redes neuronales

Las redes neuronales computacionales (NN, por sus siglas en inglés) son una familia de algoritmos inspirados en las redes neuronales biológicas. El paradigma de las NN, en particular en el caso de los problemas de clasificación con C categorías, consiste en que dado un vector de características de dimensión I, se defina una estructura de red que represente una secuencia de transformaciones al vector de características, hasta obtener un vector de dimensión C que represente la plausibilidad de pertenecer a cada categoría.

A diferencia de un programa clásico que ocuparía operaciones condicionales para discriminar entre categorías, las transformaciones aplicadas en una red neuronal no están predefi-

nidas. En vez de eso se le permite a la red ir *aprendiendo* de forma autónoma, con el único objetivo de minimizar una función de costo que represente el error de predicción. Más adelante aclararemos el concepto de qué significa que una red neuronal aprenda. En este momento solo se pretende desarrollar la intuición que hay detrás de este tipo de modelos.

La importancia de dejar atrás el paradigma del uso de condiciones, es que como seres humanos tenemos una capacidad asombrosa para muchas tareas de clasificación, por ejemplo, distinguir entre dos frutas. Sin embargo no somos igual de buenos para dar la secuencia de pasos que lleva a cabo nuestro cerebro para lograr ese objetivo, por lo cual seguramente omitiríamos algunas condiciones, así que resulta difícil implementarlo computacionalmente. Por otra parte el ser humano es bueno mejorando a través de la experiencia y reajustando los criterios de comparación hasta poder realizar la clasificación de manera precisa. Esta es la idea que inspira las NN, que el programa pueda reajustar sus parámetros hasta tener la capacidad de realizar la distinción de forma adecuada.

Un problema para NN es si tenemos un vector de características de dimensión 3, y queremos saber si un individuo pertenece a una de dos posibles categorías. Para esto se propone realizar una serie de transformaciones. En la figura 1.1 se da una breve estructura de una red

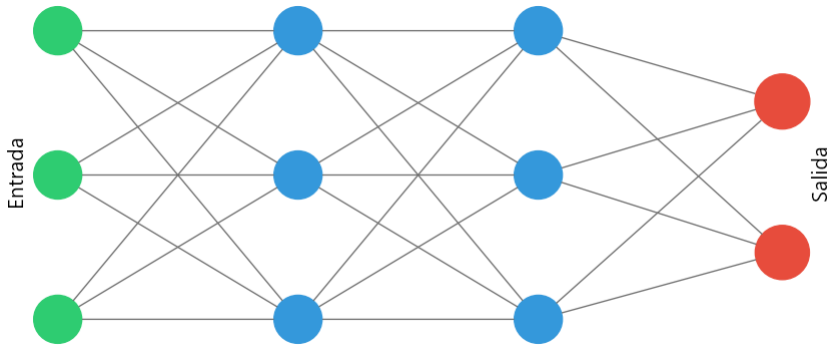


Figura 1.1: Estructura básica de una red neuronal.

Los vértices en verde representan las características del vector de entrada, los vértices en rojo representan los elementos del vector de salida, los de color azul representan los valores que resultaron tras cada transformación que no son parte del vector de salida. A cada vértice se le denomina neurona y cada arista representa que el elemento de la izquierda interviene en

el cálculo del elemento de la derecha con una ponderación w. Cada corte vertical de neuronas se conoce como capa (*layer*), las que corresponden al color verde y rojo son las capas de entrada y salida respectivamente, las capas azules son conocidas como capas ocultas, en el caso del ejemplo se tienen dos capas ocultas.

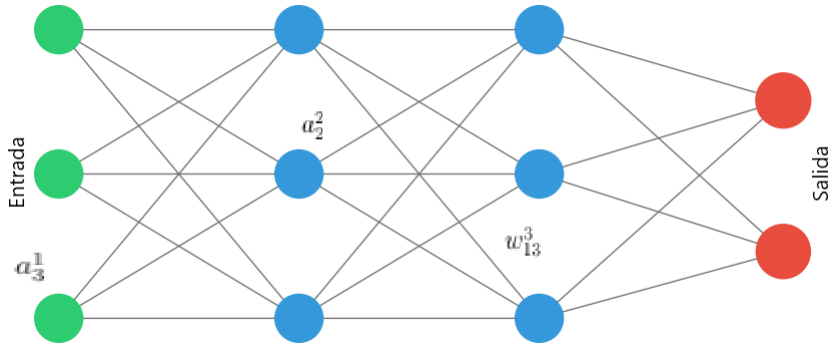


Figura 1.2: Notación básica de una red neuronal.

Siguiendo la notación de [17], tenemos que a_j^i representa el valor de la $j-$ésima neurona en la capa i, y w_{ij}^l es el peso de la neurona a_i^{l-1} para el cálculo de la neurona a_j^l, esto se ilustra en la imagen 1.2.

La relación entre neuronas es la siguiente:

$$a_j^l = \sigma_l\Big(\sum_k w_{jk}^l a_k^{l-1} + b_j^l\Big), \tag{1.1}$$

donde σ_l es la función de activación, la cual permite dar diferentes estructuras a la ponderación, las más habituales son: Identidad, ReLU, Softplus y Logística. Las gráficas de estas funciones puden verse en la imagen 1.3. Las funciones σ_l se fijan desde el inicio, de esta manera se limita el espacio de funciones en las que la red se actualizará. Por último b_j^l es un parámetro de sesgo que también deberá ser ajustado.

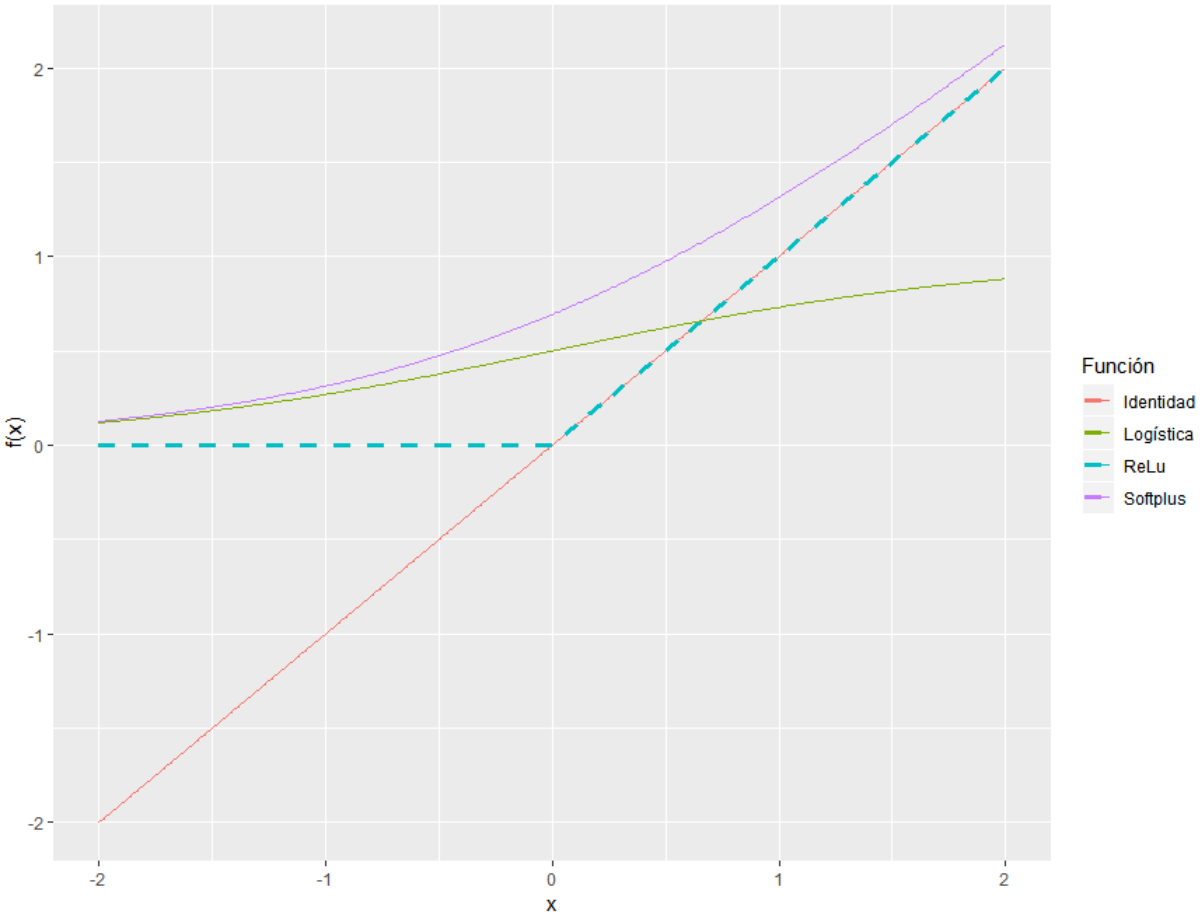


Figura 1.3: Ejemplos de funciones de activación.

Si dos capas forman un gráfo bipartito completo, diremos que se tiene una capa completamente conectada, si esto no ocurre diremos que es una capa convolutiva. La red que se presenta en 1.1 es muy simple, sin embargo cuenta con 32 parámetros que debemos ajustar. Es fácil observar cómo el aumento en el número de neuronas hace que el número de parámetros a ajustar crezca rápidamente. Es claro que aún con una red simple el número de parámetros hacen que no sea claramente interpretable la red y es por esto que se requiere de algoritmos computacionales para que sean entendidas.

Denotaremos por $N(x)$ la función que toma el vector de características y devuelve el vector de salida que se utiliza para la clasificación, es decir $N(x)$ consiste en aplicar las transformaciones de la red neuronal a un dato x, si tenemos un vector de entrada de dimensión

M y C categorías, lo habitual es tener $N(x) : \mathbb{R}^M \to \mathbb{R}^C$.

A manera de resumen debemos tener presente que los a_j^1, con j en $\{1, ..., M\}$ son nuestros datos de entrada, mismos que están fijos, el resto de a_j^i con $i \neq 1$ son escalares que quedan en función de a_j^1, b_j y W_{ij}, siendo b_j y W_{ij} los parámetros de la red neuronal que deben ser ajustados, el objetivo es encontrar los parámetros que permitan que $N(x)$ sea un buen clasificador, en cierto sentido.

La base de datos que se trabajó en esta tesis es la base MNIST, la cual es un conjunto de datos de imágenes de 28×28 pixeles, de números escritos a mano del 0 al 9, en escala de grises, en la imagen 1.4 se presentan algunos de los datos que hay en la base. La base de datos MNIST consta de 70,000 imágenes, de las cuáles tomamos 60,000 para entrenamiento.

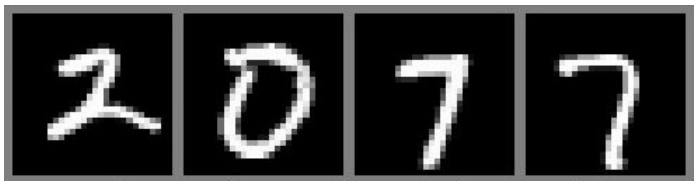


Figura 1.4: Ejemplos de números en la base MNIST, de izquierda a derecha se presentan digitos con las etiquetas 2,0,7 y 7.

1.2. Descenso de gradiente y propagación hacia atrás

Para entender cómo es que los parámetros de una NN se ajustan es fundamental entender los algoritmos de optimización de descenso de gradiente y el método para el cálculo de gradiente de propagación hacia atrás (Backpropagation).

1.2.1. Descenso de gradiente

Este algoritmo iterativo de optimización es útil cuando se quiere resolver

$$\underset{x}{\operatorname{argmin}} f(x),$$

donde $f : \mathbb{R}^n \to \mathbb{R}$ es dos veces diferenciable.

Se tiene que la razón de máximo crecimiento en x está dada por $\nabla f(x)$, de lo anterior $-\nabla f(x)$ será una dirección en la que $f(x)$ decrece, por lo que para algún $\varepsilon > 0$ se tiene

$$f(x - \nabla f(x)) < f(x).$$

De esta manera dado un punto inicical x^0 en el dominio de f, el algoritmo consiste en tomar en cada iteración

$$x^i = x^{i-1} - \nabla f(x^{i-1}).$$

Es importante notar que este algoritmo es suceptible a estancarse en mínimos locales, sin embargo resulta fácil de implementar.

1.2.2. Descenso estocástico de gradiente

El algoritmo de descenso estocástico de gradiente (SGD) es de interés cuando la función que se desea minimizar puede ser escrita como el promedio de costos más pequeños diferenciables. Es decir el problema que queremos solucionar es

$$\underset{x}{\operatorname{argmin}} f(x) = \underset{x}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} f_i(x),$$

por lo que el gradiente de f estará dado por

$$\nabla f = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(x).$$

Al ser este gradiente un promedio podemos en vez de calcularlo sobre los n costos pequeños, usar solo $k < n$ de ellos, que son elegidos a través de muestreo estocástico simple de los n posibles y utilizar un argumento de ley de grandes números para obtener

$$\frac{1}{k} \sum_{i=1}^{k} \nabla f_i(x) \approx \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(x) = \mathbb{E}(\nabla f_i(x)).$$

De manera similar al descenso del gradiente, para un x^0 inicial en cada iteración se toma

$$x^i = x^{i-1} - \frac{1}{k}\sum_{j\in D}\nabla f_j(x^{i-1}); D \subset \{1, ..., n\} \text{ aleatorio con } |D| = k.$$

La ventaja importante que da este algoritmo es que no es necesario calcular el gradiente para la muestra completa, sino que es posible tomar una submuestra, lo que reduce el tiempo computacional en el caso de NN, ya que el tamaño de muestra suele ser grande.

Este algoritmo es útil cuando queremos ajustar los parámetros de una NN, ya que la función C a minimizar es escrita como función de $N(x_i)$ para cada observación.

Suponiendo que tenemos una base de datos X de tamaño n, con x_i el vector de dimensión d de características del individuo i y y_i un vector que codifica la categoría a la que pertenece el dato. Si hay m categorías posibles y_i es un vector de dimensión $m \times 1$, donde $m-1$ entradas toman el valor 0 y una entrada vale 1 que es la entrada correspondiente a la categoría que pertence el dato i. Ahora definimos una función de costo C que sea tratable con el algoritmo SGD, dos ejemplos populares para la función C son:

$$C(X) = \frac{1}{n}\sum_{i=1}^{n} ||N(x_i) - y_i||^2 \quad \text{ó} \tag{1.2}$$

$$C(X) = \frac{1}{n}\sum_{i=1}^{n} (y_i)^T \log(N(x_i)) + (1-y_i)^T \log(1 - N(x_i)), \tag{1.3}$$

donde y_i^T es el vector y_i transpuesto. La primer función es el error cuadrático medio, mientras que la segunda es la función de costos de "entropía cruzada", la popularización de (1.3), es debido a que si los valores iniciales del algoritmo son muy lejanos a los reales la actualización de los parámetros será muy lenta si usamos (1.2), mientras que si ocupamos (1.3) la actualización suele ser más veloz. Una discusión profunda de las ventajas de usar el costo de entropía cruzada se encuentra en [17, cap. 3].

El aprendizaje de una red neuronal hace referencia a la actualización de los parámetros de la red, que si el algoritmo de optimización es SGD, el aprender corresponde a cada iteración del algoritmo.

1.2.3. Propagación hacia atrás

En la sección anterior hablamos de como optimizar una función si contamos con su gradiente, además de haber dado posibles candidatos para la función de costos de una red neuronal. Sin embargo el calcular el gradiente de una red neuronal no es trivial, esto debido al gran número de funciones anidadas que se encuentran involucradas, de ahí la importancia del algoritmo de propagación hacia atrás (*Backpropagation*), el cuál permite calcular los gradientes respecto a w y a b de forma recursiva.

Para facilitar la notación denotaremos por w^l a la matriz de pesos en la capa l, donde el elemento (i, j) de w^l corresponde a w^l_{ij}, es decir $W^l = (w^l_{ij})$ y b^l corresponde al vector de sesgos en la capa l. Además si aplicamos una función f a un vector, hacemos referencia a que la función será aplicada a cada elemento del vector, de lo anterior y de 1.1 tenemos que

$$a^l = \sigma_l(w^l a^{l-1} + b^l) = \sigma(z^l).$$

A la última capa de la red (la de salida), la denotaremos por L, es decir l toma valores de 1 a L.

El algoritmo de propagación hacia atrás se basa en computar el gradiente de C con las siguientes ecuaciones recursivas

$$\frac{\partial C}{\partial b^L} = \nabla_a C \circ \sigma'(z^L), \quad \frac{\partial C}{\partial b^l} = \left((w^{l+1})^T \frac{\partial C}{\partial b^{l+1}}\right) \circ \sigma'(z^L), \quad \frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \frac{\partial C}{\partial b^l_j},$$

donde $\nabla_a C$ es el vector cuyas entradas son $\frac{\partial C}{\partial a^L_j}$ y $\circ$ denota el producto de Hadamard.

Las ecuaciones de *backpropagation* son obtenidas a detalle en [17, cap. 2], se obtienen del uso cuidadoso y recursivo de la regla de la cadena y diferenciación usual. Es importante resaltar que si bien el entrenamiento de redes neuronales está fuertemente basado en este algoritmo, para la implementación no es necesario programarlo, ya que la mayoría de herramientas de software para NN ya lo tienen implementado, por lo que solo basta dar la estructura de la red, la función de costo y las funciones de activación σ.

1.3. Ajuste de redes en Python

Todas las redes neuronales que se trabajaron en esta tesis fueron ajustadas utilizando la extensión Pytorch de Python, en particular la red a la cuál se le aplicaron los algoritmos del capítulo 2 para interpretación y explicación siguen una estructura de red tipo *LeNet-5* la cual se presenta en la figura 1.5.

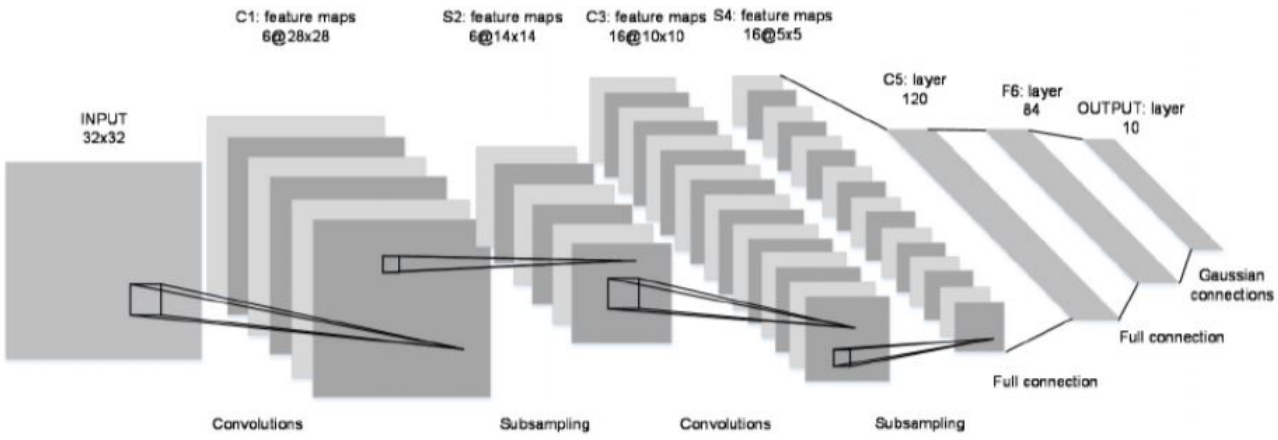


Figura 1.5: Diagrama de la estructura de la red *LeNet-5* tomada de [20].

En nuestro caso la imagen de entrada corresponde a una imagen de 28×28. La estructura para análizar la base MNIST se tomó a partir del ejemplo de pytorch.org para analizar la base CIFAR10. Para la base MNIST se ajustó una red con 6 capas ocultas, que denotaremos por H_i con i en $\{1, ..., 6\}$, las cuales tienen la siguiente estructura:

- H_1 corresponde a una capa convolutiva en 2D[1], con tamaño de kernel 5, que recibe un vector de (784,1) y devuelve 6 imágenes de 24×24. Con función de activación la función sigmoide.

- H_2 corresponde a un submuestreo utilizando *maxpool* en 2 dimensiones[2], toma como

[1]Es decir la convolución se realiza por cuadriculas de la imagen, la base de la cuadrícula es denominada kernel.

[2]La técnica maxpool2d corresponde a recorrer la imagen por cuadrículas y tomar como valor el máximo de cada cuadríicula

datos de entrada 6 imágenes de 24×24 y kernel 2, por lo que devuelve 6 imágenes de 12×12.

- H_3 es una capa convolutiva en 2D con kernel 5, que genera 16 imágenes, de 8×8. Con función de activación la función sigmoide.

- H_4 es una capa completamente conectada con 300 neuronas de salida, con función de activación la función sigmoide.

- H_5 es una capa completamente conectada con 84 neuronas de salida, con función de activación la función sigmoide.

- H_6 es una capa completamente conectada con 10 neuronas de salida, con función de activación la función identidad.

Para el entrenamiento de la red se utilizó como función de costo la entropía cruzada. Para poder trabajar con las salidas como si fueran probabilidades utilizamos la función softmax, que es una generalización de la función sigmoidea.

La red presentó una precisión de 96 % sobre las 10,000 imágenes de prueba con la siguiente precisión para cada número

0	1	2	3	4	5	6	7	8	9
99 %	98 %	97 %	97 %	97 %	95 %	98 %	93 %	97 %	93 %

En la figura 1.6 se presentan a modo de ejemplo algunos números de la base, así como la etiqueta que se les asignó y la probabilidad de pertenecer a cada categoría.

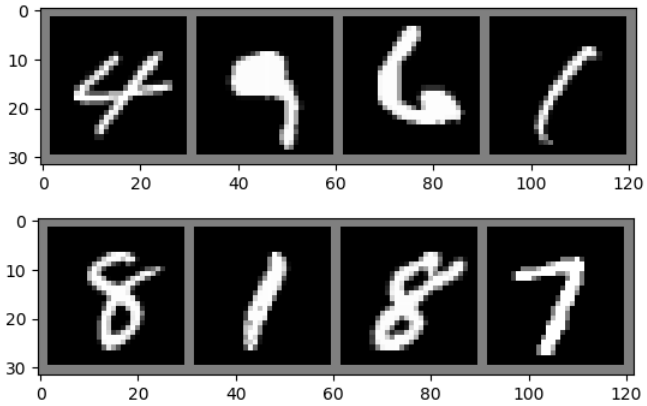


Figura 1.6: Primer renglón, números clasificados como 4, 9, 6 y 1, con probabilidades 0.99, 0.76, 0.99 y 0.99 respectivamente. Segundo renglón números con etiquetas 8, 1, 8 y 7, con probabilidades 0.99, 0.98, 9.99, 9.97. Todas las imágenes pertenecen al conjunto de prueba de la base MNIST.

El código del ajuste de la red puede ser consultado en el archivo MNIST_Ajuste que se encuentra disponible en GitHub. Todos los programas que se presentan en esta tesis se corrieron en una computadora GL502VM con procesador i7-7700HQ a 2.80 GHz y tarjeta gráfica NVIDIA GTX1060.

Capítulo 2

Técnicas de interpretación y explicación

Este capítulo pretende que el lector entienda las problemáticas que se presentan al utilizar redes neuronales para toma de decisiones. Además se presentan algoritmos para obtener imágenes representativas de categorías específicas y también se estudiará un algoritmo para obtener mapas de calor para analizar que parte de la imagen estimula a la red para pertenecer a una categoría específica.

2.1. Motivación

Las redes neuronales para la clasificación de imágenes tienen distintas áreas de aplicación. Como por ejemplo la que usa imágenes como asistencia para el diagnóstico de enfermedades dermatológicas, como en [5] o en la detección de rostros como en [1]. Aplicaciones de las redes neuronales como las anteriores son las que resaltan la importancia de tener un mayor entendimiento sobre las redes neuronales.

El entendimiento de las redes neuronales es un área de estudio que está en una etapa temprana, sin embargo en [14] se recopilan algunos de los algoritmos que se han propuesto con este fin, estos algoritmos se revisaron y programaron para esta tesis. En el artículo se incluyen definiciones de interpretación y explicación, lo que nos permitirá poder tener en claro nuestro marco de trabajo y que estamos buscando en cada algoritmo. Las definiciones

son:

- Interpretación: Es el mapeo de un concepto abstracto a un dominio que el humano puede entender.
- Explicación: Es la colección de características del dominio interpretable, que contribuyen a producir una decisión.

En esta tesis los algoritmos de interpretación se resumen a la busqueda de imágenes prototipo, es decir imágenes que sean un ejemplo representativo de la categoría deseada basado en la estructura de la red. Los algoritmos de explicación serán aquellos que generan mapas de calor para entender que pixeles estimularon a la red para tomar una decisión en específico.

2.2. Técnicas de interpretación

En esta sección se darán las bases teóricas para los algoritmos de interpretación, los algoritmos que se presentarán son *Activación maximización (AM)*, *AM con experto* y *AM con decodificador*. Además se darán algunas observaciones respecto a las características que se observaron durante su implementación.

2.2.1. Activación maximización

Este algoritmo si bien es el menos complicado de los que se estudiaron, sirve como punto de partida para poder entender la idea fundamental de los otros dos algoritmos de búsqueda de prototipos que se presentan en este trabajo.

La idea del algoritmo AM se basa en analizar que pixeles maximizan el pertenecer a una categoría específica, esto se realiza por medio del gradiente de las transformaciones de la red N con respecto a la imágen de entrada x_i. Ya que N puede devolver un vector sin ningún tipo de restricción, el mejorar una de las k categorías no necesariamente garantiza que estamos mejorando las posibilidades de que la red clasifique a la imagen en el valor deseado. Para

evitar este tipo de complicaciones, utilizaremos dos restricciones

$$\sum_{j=1}^{k} N_j(x) = 1,$$
$$\sum_{j=1}^{k} N_j(x) \geq 0.$$

De lo anterior podemos expresar que dado un vector de entrada x_i, $N(x_i)$ representa los parámetros de una variable aleatoria multinomial, por lo que usaremos la notación para representar la probabilidad de pertenecer a la categoría c dado la imagen x_i

$$\mathbb{P}(c|x) = N_c(x),$$

así que el algoritmo *AM* consiste en resolver

$$\underset{x \notin X}{\operatorname{argmax}} \ \mathbb{P}(c|x) - \lambda||x||^2.$$

En el algoritmo AM, se penaliza con la norma euclideana para atraer las soluciones al origen, en el caso de la base MNIST es una imagen completamente en gris, así los colores del prototipo que generan distinción en la clase se encontrarán en trazos específicos mientras que el resto de la imagen será lo más homogénea posible.

Cómo hemos planteado la estructura de la red en la sección 1.3 es fácil observar que existe

$$\frac{\partial \mathbb{P}(c|x) - \lambda||x||^2}{\partial x}.$$

De aquí que para poder solucionar el problema de AM sea suficiente utilizar un algoritmo de ascenso por gradiente, similar a lo que se expuso en la sección 1.2.1.

Este algoritmo presenta un gran inconveniente, la única restricción que se impone a los posibles valores de x es aquella referente a su norma, sin embargo hay imágenes que pueden dar $\mathbb{P}(c|x) \approx 1$, que no pertenezcan al dominio de las imágenes de entrada, para alguna c, además por la gran dimensión de x y la compleja estructura de la red, es muy fácil terminar

en un máximo local, por lo que el algoritmo resulta sumamente sensible al punto inicial. Por eso, se propone el siguiente algoritmo.

2.2.2. AM con experto

Este algoritmo pretende restringir el dominio al que pertenecen a las imágenes de entrada. Para esto se propone un modelo de probabilidad para el espacio de las imágenes de entrada, por lo que en vez de penalizar con la norma como en el algoritmo AM, se usa una penalización que se centre en las imágenes que son verosímiles, esto se consigue ajustando una densidad de probabilidad al espacio de imágenes, a la cual denominaremos experto y lo denotaremos como $\mathbb{P}(x)$.

Para ajustar la densidad se utilizarán máquinas restringidas de Boltzmann, así una vez ajustado el experto, el problema en este algoritmo consiste en resolver

$$\underset{x \notin X}{\operatorname{argmax}} \log \mathbb{P}(c|x) + \log \mathbb{P}(x). \tag{2.1}$$

2.2.2.1. Máquinas restringidas de Boltzmann

Las máquinas restringidas de Boltzmann (RBM) son campos markovianos, con variables visibles, las cuales corresponden a los pixeles de la imagen y variables ocultas, las cuales no son observables, pero permiten al modelo tener mayor flexibilidad y poder así ajustar una densidad que se apegue más a la realidad.

Al ser una RBM un campo markoviano, podemos representar las variables y la relación entre ellas por medio de un grafo, la estructura que deben tener las RBM es que cumplen con ser un grafo completo bipartito, sin aristas entre las variables de un mismo tipo; un ejemplo de esto se presenta en la figura 2.1.

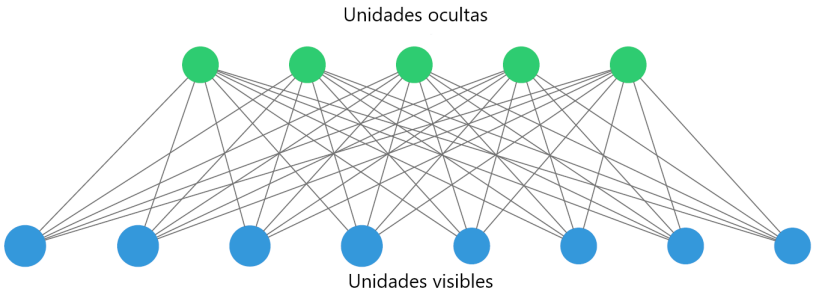


Figura 2.1: Representación gráfica de una máquina de Boltzmann restringida.

En caso de que el lector no esté familiarizado con los campos markovianos o los modelos gráficos, en [7] y [6] se explican de forma detallada en el ámbito de RBM.

Hay dos variantes principales en las RBM, la primera de ellas es cuando tanto unidades ocultas como visibles, son variables binarias, es decir solo pueden tomar valores en $\{0,1\}$, a las que llamaremos máquinas restringidas de Boltzmann binarias (BRBM). Si bien estas son útiles para obtener imágenes que parezcan números tienen la limitación de que no pueden ser usadas para algoritmos del tipo ascenso de gradiente, ya que las unidades visibles no se mueven en un espacio continuo. El segundo tipo de RBM, son aquellas cuyas unidades visibles son continuas y las ocultas son binarias, una de las variantes más comunes son aquellas donde las unidades visibles condicionadas a las ocultas siguen una distribución normal, a estas las llamaremos máquinas restringidas de Boltzmann Gaussianas-Binarias (GBRBM), las cuales serán de particular interés para nosotros, ya que nos permiten optimizarlas usando ascenso de gradiente.

Sea v el vector de M variables visibles y H el vector de N variables ocultas, el modelo de probabilidad para las GBRBM está dado por la siguiente función de densidad

$$\mathbb{P}(v, H) = \frac{1}{z} exp\{-E(v, H)\}, \tag{2.2}$$

donde z es la constante de normalización, y la función $E(v, H)$ de la ecuación (2.2), deno-

minada función de energía, tiene la expresión

$$E(v, H) = \frac{||v - b||}{2\sigma^2} - c^T H - \frac{v^T W H}{\sigma^2}, \tag{2.3}$$

donde v^T es el vector v transpuesto, b y c vectores de $M \times 1$ y $N \times 1$ respectivamente, W matriz de $M \times N$ y σ^2 un escalar, los cuales son los parámetros que se deben estimar, denotaremos a θ un elemento del espacio parametral, es decir θ es de la forma $\theta = (b, c, W, \sigma^2)$.

Con esa estructura se tiene

$$\mathbb{P}(v|h) \sim \mathcal{N}(b + Wh, \sigma^2)$$
$$\mathbb{P}(h_j = 1|v) = \frac{1}{1 + e^{-(c_j + \frac{v^T w_{*j}}{\sigma^2})}}$$

donde w_{*j} representa a la j-ésima columna de W y w_{i*} al i-ésimo renglón de W.

La estimación de los parámetros se realiza usando EMV, es importante resaltar que como no podemos observar las variables ocultas el cálculo de la verosimilitud se basa en la densidad marginal de v, así que dada una muestra V de tamaño L la logverosimilitud promedio está dada por

$$\hat{l} = \frac{1}{L} \sum_{l=1}^{L} \log \mathbb{P}(v_l; \theta) = \frac{1}{L} \sum_{l=1}^{L} \log \Big(\sum_{\mathbf{h}} e^{-E(v_l, h)} \Big) - \log(z)$$

donde h representa la suma sobre todos los posibles valores que puede tomar el vector H. Derivando respecto a θ obtenemos

$$\frac{\partial \hat{l}}{\partial \theta} = -\frac{1}{L} \sum_{l=1}^{L} \log \Big(\sum_{\mathbf{h}} \mathbb{P}(h|v_l) \frac{\partial E(v_l, h)}{\partial \theta} \Big) + \mathbb{E}_v \Big(\sum_{\mathbf{h}} \frac{\partial \mathbb{P}(h|v) E(v_l, h)}{\partial \theta} \Big).$$

Notemos que la derivada de la verosimilitud queda en términos del valor esperado de la densidad teórica, el cuál no podemos calcular. Sin embargo una forma de aproximarlo es por medio del algoritmo de divergencia contrastiva (CD), el cual consiste en reemplazar la esperanza teórica por una media empírica calculada usando una muestra a partir de muestreo de Gibbs.

Si V^k es una muestra obtenida haciendo muestreo de Gibbs con valores iniciales los datos de la muestra e iterando k veces se tiene

$$\frac{\partial \hat{l}}{\partial \theta} \approx -\frac{1}{L}\sum_{l=1}^{L} \log\Big(\sum_{\mathbf{h}} \mathbb{P}(h|v_l)\frac{\partial E(v_l,h)}{\partial \theta}\Big) + \frac{1}{L}\sum_{l=1}^{L} \log\Big(\sum_{\mathbf{h}} \mathbb{P}(h|v_l^k)\frac{\partial E(v_l^k,h)}{\partial \theta}\Big).$$

Una explicación detallada del algoritmo puede encontrarse en [6]. En este artículo hay cotas para el error de CD en el caso de BRBM, también ahí se exponen algoritmos más complejos para la aproximación.

Utilizando divergencia contrastiva tenemos las siguientes aproximaciones para la parcial respecto a cada parámetro

$$\begin{aligned}
\frac{\partial \hat{l}}{\partial b} &= \mathbb{E}_V\Big(\frac{V-b}{\sigma^2}\Big) - \mathbb{E}_{V^k}\Big(\frac{V^k-b}{\sigma^2}\Big)\\
\frac{\partial \hat{l}}{\partial c} &= \mathbb{E}_V(\mathbb{P}(\mathbf{h}=1|V)) - \mathbb{E}_{V^k}(\mathbb{P}(\mathbf{h}=1|V^k))\\
\frac{\partial \hat{l}}{\partial W} &= \mathbb{E}_V\Big(\frac{V\mathbb{P}(\mathbf{h}=1|V)^T}{\sigma^2}\Big) - \mathbb{E}_{V^k}\Big(\frac{V^k\mathbb{P}(\mathbf{h}=1|V^k)^T}{\sigma^2}\Big)\\
\frac{\partial \hat{l}}{\partial \sigma} &= \mathbb{E}_V\Big(\frac{||V-b||^2 - 2V^TW\mathbb{P}(\mathbf{h}=1|V)}{\sigma^3}\Big) - \mathbb{E}_{V^k}\Big(\frac{||V^k-b||^2 - 2(V^k)^TW\mathbb{P}(\mathbf{h}=1|V^k)}{\sigma^3}\Big)
\end{aligned}$$

donde $\mathbb{E}_V$ es la media empírica con respecto a la muestra y $\mathbb{E}_{V^k}$ la media empírica respecto a la muestra obtenida por muestreo de Gibbs y
$\mathbb{P}(\mathbf{h}=1|v) := \Big(\mathbb{P}(h_1=1|v), \mathbb{P}(h_2=1|v), ..., \mathbb{P}(h_N=1|v)\Big)^T$.

Es importante observar que la derivada parcial respecto a σ es computacionalmente inestable, una forma de solucionar esto es reparametrizando (2.3), esta idea se aborda en [19] y consiste en reparametrizar con la siguiente igualdad:

$$\sigma^2 = e^{-t}$$

y estimar t.

Por último nuestro experto no puede tener como variables las unidades ocultas así que requerimos tener la expresión de la densidad marginal, que es

$$\begin{aligned}
\mathbb{P}(v) &= \sum_{\mathbf{h}} \mathbb{P}(v,h) \\
&= \frac{1}{z}\prod_{j=1}^{N}\left(\sqrt{2\pi N\sigma^2}\right)^M [\mathcal{N}(v;b,N\sigma^2 I) \\
&+ exp(\frac{||b+Nw_{*j}||^2-||b||^2}{2N\sigma^2}+c_j)\mathcal{N}(v;b+Nw_{*j},N\sigma^2 I)] \\
&= \frac{1}{z}\prod_{j=1}^{N} C[N_1+\alpha_j N_2]
\end{aligned} \tag{2.4}$$

con

$$\begin{aligned}
C &= \left(\sqrt{2\pi N\sigma^2}\right)^M, \\
N_1 &= \mathcal{N}(v;b,N\sigma^2 I), \\
\alpha_j &= exp(\frac{||b+Nw_{*j}||^2-||b||^2}{2N\sigma^2}+c_j), \\
N_2 &= \mathcal{N}(v;b+Nw_{*j},N\sigma^2 I),
\end{aligned}$$

donde $\mathcal{N}(v;a,b)$, es la densidad de una normal con media a y varianza b evaluada en el valor v e I la matriz identidad.

Las ecuaciones anteriores se derivan a detalle en [13] y en [11]. En la figura 2.2 se presentan datos simulados a partir de este modelo con la base MNIST.

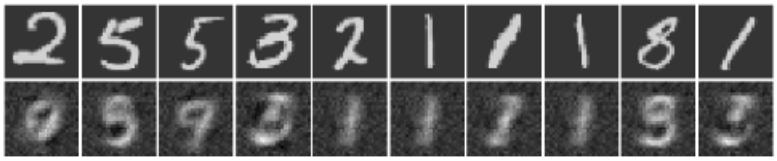


Figura 2.2: Simulación de digitos usando GRBM, en el primer renglón tienen el punto inicial de la simulación y en el segundo renglón el resultado tras cien iteraciones de muestreo de Gibbs .

La función de energía dada en (2.3), no es la única posibilidad para definir una GBRBM, esta es conocida como variante de Cho con varianza constante; en caso de que el lector esté interesado en estudiar otras estructuras para la función de energía y ver casos con varianza diferente para cada unidad visible, se recomienda consultar [19], donde se presentan la variante de Krishevsky, la de Cho y la de Goodfeloow, además se derivan las ecuaciones neccesarias para el algoritmo CD en cada caso.

Ya que en (2.1) deseamos derivar el logaritmo de la ecuación (2.4) con respecto a v_i, primero veamos que pasa con N_1 y N_2,

$$N_1 \propto \prod_{i=1}^{M} e^{-\frac{(v_i-b)^2}{2N\sigma^2}},$$

así que se tiene

$$\begin{aligned}\frac{\partial N_1}{\partial v_1} &\propto -\Big(\frac{v_1-b_1}{2N\sigma^2}\Big)\prod_{i=1}^{M} e^{-\frac{(v_i-b)^2}{2N\sigma^2}} \\ &= -\Big(\frac{v_1-b_1}{2N\sigma^2}\Big)N_1.\end{aligned}$$

De forma análoga se puede ver que

$$\frac{\partial N_2}{\partial v_1} \propto -\Big(\frac{v_1-b_1-NW_{*j}}{2N\sigma^2}\Big)N_2.$$

Con lo anterior tenemos

$$\begin{aligned}\frac{\partial \log(P(v))}{\partial v} &= \sum_{j=1}^{N} \frac{\partial}{\partial v} \log(N_1 + \alpha_j N_2) \\ &= \sum_{j=1}^{N} \frac{\left(\frac{\partial N_1}{\partial v} + \alpha_j \frac{\partial N_2}{\partial v}\right)}{N_1 + \alpha_j N_2} \\ &= \sum_{j=1}^{N} -\frac{v-b}{\sigma^2 N} + \sum_{j=1}^{N} \frac{w_{*j} N_2 \alpha_j}{(N_1 + \alpha_j N_2)\sigma^2} \\ &= -\frac{v-b}{\sigma^2} + \sum_{j=1}^{N} \frac{w_{*j} N_2 \alpha_j}{(N_1 + \alpha_j N_2)\sigma^2}.\end{aligned}$$

En la práctica este algoritmo es suceptible a presentar errores de punto flotante, ya que N_1, N_2 y α_j suelen ser de ordenes de magnitud diferentes, así que es imporante que al implementar este tipo de algoritmos se analice con precaución la realización de cada operación. Una forma de solucionar este problema en Python, es usando la extensión $mpmath$ que si bien resuelve el problema de la precisión numérica hace que realizar las operaciones sea más tardado que en herramientas habituales como $numpy$.

Una de las deficiencias más importantes de usar GBRBM, es que el ajuste del modelo requiere de la realización de un gran número de operaciones, al igual que el ascenso de gradiente, lo que provoca que el encontrar un prototipo sea una tarea sumamente tardada.

2.2.3. AM con decodificador

Los algoritmos AM y AM *con experto*, tenían por objetivo maximizar una función que tenía como valores de entrada elementos del dominio de la red neuronal, que denominaremos por X. El algoritmo que se presenta en esta sección en vez de optimizar en X, pretende maximizar en un dominio Z, con ayuda de una función $\Psi : Z \to X$, a la cual llamaremos decodificador. El artículo en que se plantea este algoritmo es [4].

Nosotros como desarrolladores debemos elegir el espacio Z. En nuestro caso elegiremos

$\mathbb{R}^l$, por ejemplo, en el caso de las imágenes de MNIST tenemos que $X = (0,1)^{28\times 28}$, es decir tenemos 28×28 pixeles y cada uno de ellos puede tomar un valor en el intervalor $(0,1)$; en el ejemplo que se implementó computacionalmente para esta tesis se tomó $Z = \mathbb{R}^{64}$.

Al igual que en el algoritmo *AM con experto*, vamos a suponer que nuestras observaciones (en el caso de MNIST es cada imagen), son muestras de una distribución de probabilidad (G), pero en este algoritmo no estamos interesados en encontrar una función de distribución, solamente estamos interesados en poder muestrear elementos que provengan de esta distribución.

Para la simulación de datos que provengan de la distribución desconocida G, usaremos simulación por transformación de variables. Hay diferentes ejemplos de transformaciones para la generación de variables aleatorias, el más común de los ejemplos de simulación por transformación es cuando se quiere simular de una v.a. Y con distribución F y simulamos muestras de una variable aleatoria $U(0,1)$ (uniforme en (0,1)) y aplicamos la transformación F^{-1}, con lo que obtenemos un dato con distribución F.

Ya que la distribución de G puede resultar muy compleja no es opción intentar encontrar G^{-1}, sin embargo aquí es donde aparece la función Ψ. Si tomamos $Y \sim \mathcal{N}(\mathbf{0}, \sigma^2 I_{l\times l})$, podemos ajustar por medio de redes generativas adversarias la función Ψ que nos permitirá que $\Psi(Y)$ se distribuya aproximadamente como G. Con esto en mente el algoritmo de *AM con decodificador* consiste en optimizar la función:

$$\underset{z\in Z}{\operatorname{argmax}}\ \mathbb{P}(c|\Psi(z)) - \lambda||z||^2. \tag{2.5}$$

Además Ψ es construida de tal forma que (2.5) pueda ser optimizada usando ascenso de gradiente. En la siguiente sección se explica como se puede encontrar Ψ.

2.2.3.1. Redes adversarias generativas

Las redes adversarias generativas (GAN) son una técnica para la generación de muestras de una distribución que en general es no computable. El algoritmo fue presentado en [8], este algoritmo se ha popularizado por su facilidad de implementación y por su simplicidad con-

ceptual, siendo actualmente uno de los algortimos más eficientes para la generación de datos sintéticos que sean indistinguibles de datos reales, principalmente en el campo de imágenes.

Las redes adversarias consisten en ajustar dos redes neuronales, la primera es denominada como generador G y la otra es un discriminador D, definidas de la siguiente manera:

$$G : Z \to X$$
$$D : X \to [0, 1]$$

Para entender la idea detrás de las redes neuronales, motivaremos su implementación con la red MNIST.

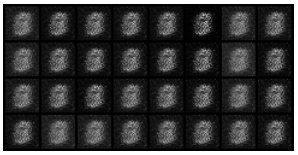
La red D tiene como objetivo discriminar si un x en X proviene de la muestra real o es un dato sintético; $D(x)$ puede interpretarse como una probabilidad, por lo que si somos capaces de crear datos sintéticos indistinguibles de los datos reales y x^* es un dato sintético se tendrá que $D(x^*) = \frac{1}{2}$.

Los datos de entrenamiento para la red D consisten en parejas (x_i, y_i), en el caso de MNIST, x_i es un vector en $\mathbb{R}^{28\times 28}$, y y_i una variable binaria con entradas 0 ó 1, siendo 0 si el dato pertenece a los datos sintéticos y 1 si pertenece a los datos reales, los datos sintéticos se generan en cada iteración del entrenamiento de D usando la red G, de esta forma si C es una función de costo, cada iteración del entrenamiento de D consiste en minimizar

$$\sum_i C(y_i, D(x_i)).$$

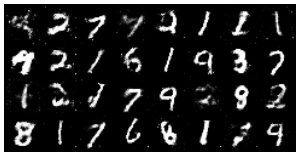
La red G es la que nos va a permitir obtener muestras de una distribución F_G que pretende generar datos similares a los que se generarían por la distribución F de los datos reales; los datos de entrenamiento z_i para G deben ser muestreados de una variable aleatoria continua conocida, que en nuestro caso será una normal l variada, con media 0 y varianza $\sigma^2 I$, así $G(z_i)$ es una variable aleatoria con distribución F_G.

Para entrenar la red G utilizaremos la red D, de ahí el nombre de redes adversarias, G tiene como objetivo confundir al discriminador D, y D intenta detectar las fallas que G pueda



(a) Iteración 1

(b) Iteración 10



(c) Iteración 20

(d) Iteración 250

Figura 2.3: Generación de digitos usando G para distintas iteraciones de entrenamiento.

tener al generar datos sintéticos.

Cada iteración del entrenamiento de la red G consiste en minimizar

$$\sum_i (1 - D(G(z_i))).$$

Es decir buscamos que los datos que genera G, el discriminador los clasifique como reales. Un ejemplo de datos sintéticos para la base MNIST usando GAN se presentan en la imagen 2.3

En este caso el entrenamiento de las redes D y G se realizó alternandamente y cada iteración consiste en el haber entrenado con la muestra a ambas redes; podemos observar como las redes adversarias muestran una excelente capacidad de generar datos sintéticos en este ejemplo.

Por último respecto a (2.5) la función Ψ corresponde a la red G; la red G se conoce como decodificador ya que nos permite cambiar entre los espacios Z y X.

2.3. Métodos de explicación

En esta sección nos enfocaremos en intentar comprender cómo es que la red neuronal asignó una imagen a una categoría. La idea es que a partir de mapas de calor, la persona

encargada de tomar las decisiones con la red pueda observar qué pixeles son los que más influyeron a la red para su decisión.

2.3.1. Análisis de sensibilidad

Un algoritmo básico para la explicación de la decisión de la red neuronal es el denominado *Análisis de sensibilidad*. En este caso la relevancia del pixel i, se calcula a partir del impacto que tiene el pixel en el gradiente de la red, es decir:

$$R_i(x) = \Big(\frac{\partial N^*(x)}{\partial x_i}\Big)^2,$$

donde:

- $N^*(x)$ es la entrada máxima del vector $N(x)$. Recordando $N(x)$ nos da el nivel de plausibilidad para cada una de las C categorías así que

$$N^*(x) = \max_{c \in C} \mathbb{P}(c|x)$$

- x es un elemento del dominio X de la red

- x_i es el i-ésimo pixel

- $R_i(x)$ es la relevancia del pixel i.

Si bien este algoritmo sirve como una primera idea sobre como analizar la respuesta de la red, no tiene algunos aspectos que nos gustaría ser capaces de poder apreciar. Uno de ellos es que aunque la parcial sea grande no distingue en si el efecto es a favor o en contra de pertenecer a la categoría especifica, además de que en la práctica no parecen mostrar resultados que estén relacionados a las características de interés.

2.3.2. Propagación de relevancia por capas

En [3], se plantea la idea del algoritmo *Propagación de relevancia por capas* (*Layer-wise relevance propagation, LRP*). Este algoritmo permite que podamos dar un valor de $R_i(x)$, que sea sensible a si el pixel aporta o no a la clasifiación en la categoría asignada, además de que es un algoritmo pensado completamente para la interpretación y validación por un humano por medio de mapas de calor.

A diferencia del análisis de sensibilidad aquí no basta solamente tener la relevancia del pixel; aquí estaremos interesados en distribuir el valor de $N^*(x)$, a través de cada neurona.

La distribución debe cumplir las siguentes reglas

$$\sum_j R_{j\leftarrow k} = R_k \tag{2.6}$$

$$R_j = \sum_k R_{j\leftarrow k}$$

$$N^*(x) > 0$$

donde $R_{j\leftarrow k}$ es la relevancia de la neurona j que proviene de la neurona k. El algoritmo consiste en recorrer la red neuronal de la capa final hasta la capa de entrada repartiendo la relevancia de cada neurona entre sus predecesoras, este proceso se ejemplifica en la imagen 2.4.

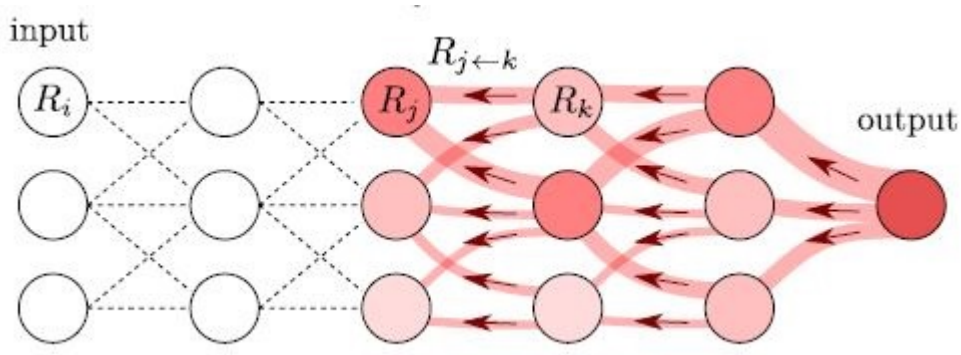


Figura 2.4: Diagrama del algoritmo LRP, imagen tomada de [14].

Las reglas de conservación tienen la utilidad de que la energía de cada neurona al ser distribuida se preserve, así que al llegar a la relevancia de los pixeles tenemos el valor $N^*(x)$ repartido entre cada pixel; sin embargo aún falta decir como la relevancia se distribuye entre las neuornas. Recordando (2.3.1) tenemos que una neurona está dada por

$$a_j^l = \sigma_l\Big(\sum_k w_{jk}^l a_k^{l-1} + b_j^l\Big).$$

Heurísticamente una regla de distribución de relevancias que parece funcionar adecuadamente para la neuorna a_j^{l-1} es:

$$R_j^{l-1} = \sum_k \Big(\alpha \frac{\alpha_j^{l-1}(w_{jk}^l)^+}{\sum_j \alpha_j^{l-1}(w_{jk}^l)^+} - \beta \frac{\alpha_j^{l-1}(w_{jk}^l)^-}{\sum_j \alpha_j^{l-1}(w_{jk}^l)^-}\Big) R_k^l \tag{2.7}$$

donde $()^+$ y $()^-$ denotan la parte positiva y negativa de un número, α y β son hiperparámetros que define el usuario tales que $\alpha - \beta = 1$ y $\beta > 0$. En esta tesis se trabajará principalmente con $\alpha = 2$ y $\beta = 1$. Para una discusión a detalle del impacto de la selección de los hiperparámetros se recomienda ver [14].

A la distribución de relevancias que cumpla las reglas de (2.6), es lo que denominaremos por algoritmo LRP.

Capítulo 3

Aplicación de las técnicas de interpretación en la base MNIST

En el capítulo 2 se presentaron diferentes técnicas que tienen por objetivo interpretar y explicar redes neuornales, este capítulo tiene como objetivo presentar los resultados de aplicar los algoritmos en la base MNIST y hacer breves observaciones acerca de los mismos.

3.1. Activación maximización

En la figura 3.1 se presentan los prototipos x_i de los números del 0 al 9 utilizando el algoritmo AM presentado en la sección 2.2.1. Todos los x_i presentan $\mathbb{P}(i|x_i) > 0.99$ y se eligió $\lambda = 0.001$, ya que heurísticamente presentó los mejores resultados, principalmente al tomar como punto inicial una imagen de la base que perteneciera a la categoría deseada. El algoritmo se inicializó con un vector aleatorio con distribución uniforme en $[-1, 1]$ para cada entrada.

El código puede ser consultado en el archivo MNIST_Ajuste que se encuentra disponible en GitHub.

Una vez encontrado el prototipo una manera de hacer más fácil de visualizar la diferencia entre los números es aplicando $\frac{2}{\pi}tan^{-1}(x_i)$ para que sean más contrastante los valores, esto

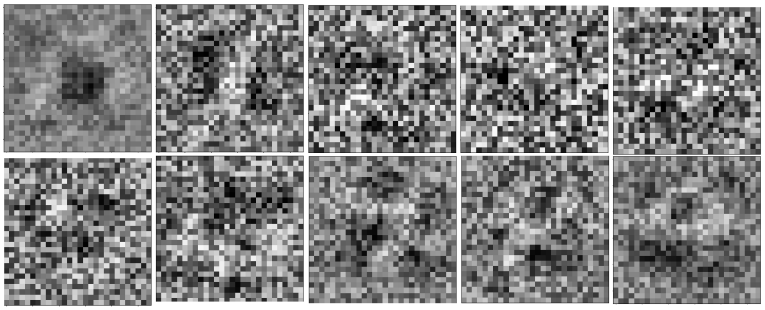


Figura 3.1: Prototipos del 0 al 9 usando AM, en el primer renglón de izquierda a derecha se presentan los números del 0 al 4 y en el segundo renglón del 5 al 9.

da como resultado el prototipo de 5 que se expone en la figura 3.2 con $\mathbb{P}(i|x_i) \approx 1$.

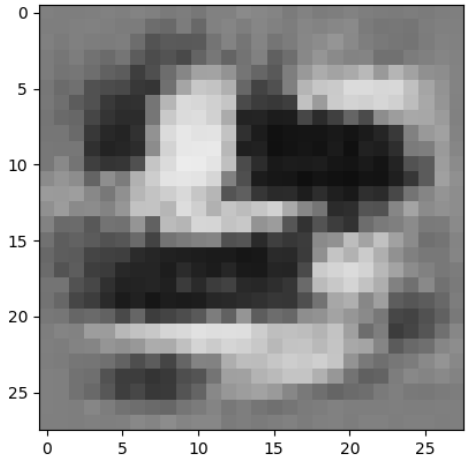
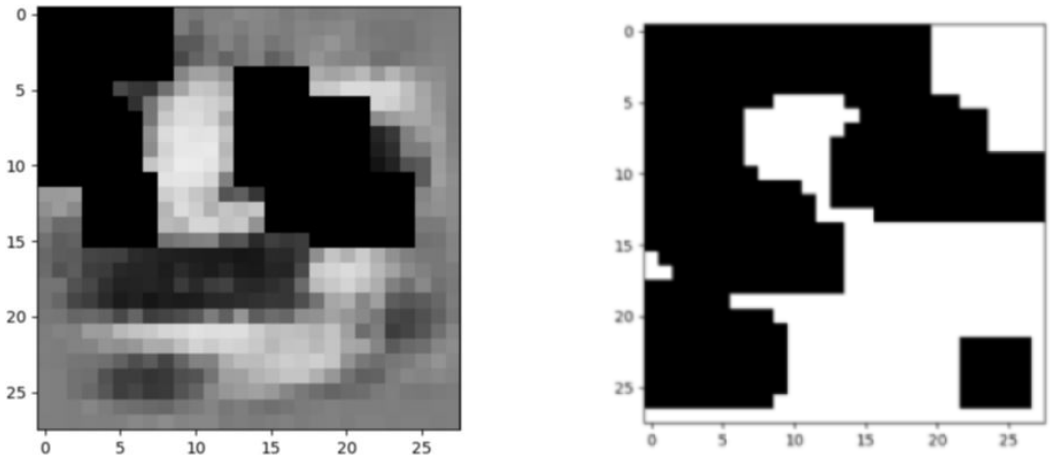


Figura 3.2: Aplicación de $\frac{2}{\pi}tan^{-1}()$ para el prototipo del número 5 en la imagen 3.1.

Aunque la red está casi segura de que los números pertenecen a la categoría deseada, para la mayoría de seres humanos estos prototipos no son satisfactorios. Una primera idea para intentar mejorar los prototipos fue recorrer de izquierda a derecha y de arriba a abajo las imágenes por cuadrículas de tamaño $m \times m$ y ver si al volver la cuadrícula completamente negra $\mathbb{P}(i|x_i)$ aumentaba. La aplicación de este método al prototipo de la figura creó la imagen que se presenta en 3.3a

Una vez obtenida esta imagen podemos repetir el proceso de aplicar el algoritmo AM y después volver algunas cuadrículas fondo.



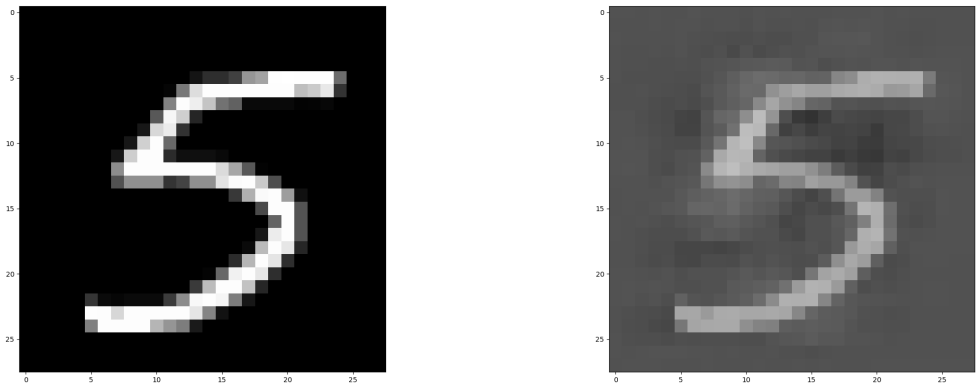
(a) (b)

Figura 3.3: A la izquierda se tiene la adición de fondo al prototipo en 3.2 utilizando una cuadrícula de 5×5; a la derecha se observa una imagen en blanco que se le adicionó fondo con una cuadrícula de 5×5

Una pregunta que nace de forma natural de la idea anterior es ¿Qué pasa si empezamos con una imagen en blanco y le vamos agregando fondo?, siguiendo esta idea se llegó a lo que se presenta en la figura 3.3b.

La imagen 3.3b alcanzó $\mathbb{P}(i|x_i) \approx 1$, pero no es de ninguna manera satisfactoria como representante de 5 para el ojo humano, sin embargo la red la percibe con gran seguridad dentro de la categoría 5.

Aunque el algoritmo AM nos permite encontrar imágenes que la red clasifique con alta probabilidad, es común quedarnos en mínimos locales que no pertenecen al dominio de "números escritos a mano". Una forma de solucionar esto es tomando como puntos iniciales imágenes que pertenecen a la categoría que deseamos, sin embargo esto puede no ser práctico si no conocemos las imágenes de entrenamiento o si no tenemos la certeza de que la etiqueta de la imagen esté correcta, en la figura 3.4 se encuentra un prototipo utilizando el algoritmo AM usando como punto inicial un número de la base MNIST.



(a) (b)

Figura 3.4: A la izquierda se presenta un 5 de la base MNIST, el cuál se clasifica con probabilidad 0.9883; a la derecha un prototipo utilizando algoritmo AM con punto inicial 3.3a, que se clasifica como 5 con probabilidad 0.997.

3.2. AM con experto

Como discutimos en la sección 2.2.2, el usar un experto nos permite restringir nuestro dominio a un espacio de valores verosímiles; si bien ya vimos que las GBRBM permiten muestrear datos de manera sencilla, nos enfrentamos a que debemos poner el hiperparámetro de cuantas unidades ocultas deseamos.

Respecto al hiperparámetro, si bien el aumentar el número de unidades ocultas (h) nos permite ser más flexibles con la distribución ajustada, también hace que el número de operaciones que debemos realizar para calcular el gradiente crezca rápidamente. Un problema que se observó dentro de las pruebas es que si bien el algoritmo permite generar muestras de forma adecuada para h grandes, también se presenta el problema de que comienzan a aparecer máximos locales. En particular se observó que el modelo comienza a asignar masa cada vez más grande a la imagen completamente en blanco. Debido a esto es necesario seleccionar h de manera cautelosa.

Un ejemplo es que al entrenar la GBRBM para 100 unidades ocultas se tiene para los datos de entrenamiento una logverosimilitud (salvo constante de normalización) que oscila

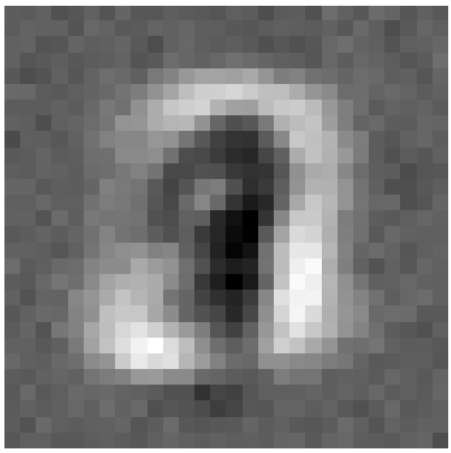


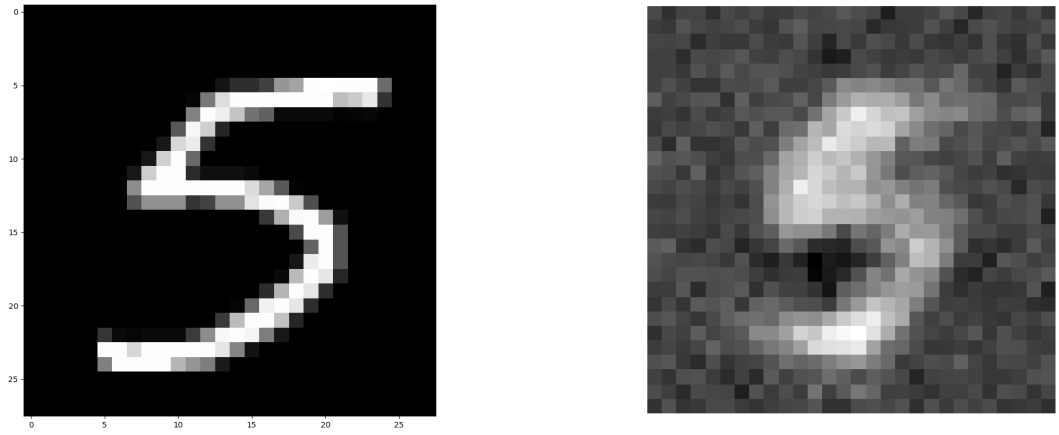
Figura 3.5: Prototipo con $\mathbb{P}(5|x_5) = 0.0535$

en 130, mientras que una imagen totalmente en blanco tiene 410. Esta diferencia solo se incrementará al aumentar h, de ahí que se elija en este trabajo realizar los experimentos con $h = 8$, donde la verosimilitud de la imagen en blanco baja a 95, mientras que la de los datos reales oscila los 120.

Partiendo de un punto inicial aleatorio y buscando un prototipo para el 5 la optimización converge al prototipo presentado en la figura 3.5, coincidentemente con nuestra suposición no parece un número 5, sin embargo se encuentra el mínimo local ahí ya que $\mathbb{P}(x)$ es muy alta, un intento de solucionar esto es volver a tomar como punto inicial el 5 escrito a mano, dando como resultado el prototipo que se observa en 3.6b.

A la izquierda se presenta el 5 real y a la derecha el obtenido con AM con experto, el prototipo presenta $\mathbb{P}(5|X_5) \approx 0.9808$. Es importante notar que en la imagen 3.4 cuando usamos el algoritmo AM y punto inicial el 5 real, el algoritmo se encargó principalmente de volver la imágen más gris y agregar un poco de ruido en regiones clave, mientras que cuando usamos AM con experto el prototipo difiere considerablemente del 5 real y da un valor cercano a una de las modas de la distribución ajustada.

Este algoritmo resultó ser sumamente ineficiente computacionalmente y como lo vemos ser muy sensible a valores iniciales. Si bien presenta un proceso de abstracción mayor al que se presenta en el algoritmo AM básico existen nuevas técnicas que lo vuelven obsoleto, como



(a) (b)

Figura 3.6: Izquierda 5 real, derecha prototipo utilizando algoritmo AM con experto.

lo es el algoritmo AM con decodificador.

El ajuste del modelo se realizó utilizando la extensión $PyDeep$ que puede ser encontrada en [12]. Sin embargo se tuvieron que realizar modificaciones ya que está programada para *Python 2*, las modificaciones solo fueron aplicadas para los modelos de GBRBM con varianza constante y varianzas diferentes por pixel, por lo que el resto de modelos pueden no ser funcionales en *Python 3*. La versión modificada de PyDeep puede ser descargada desde Dropbox, el ajuste del modelo y el ascenso del gradiente pueden ser consultados en GitHub, también se programó para cotejar los resultados con $PyDeep$ la estimación de parámetros para una GBRBM con varianza constante usando divergencia contrastiva y el ascenso de gradiente con Numpy y Pytorch; esta versión puede ser consultada en GitHub.

3.3. AM con decodificador

En este capítulo se muestran resultados de la implementación del algoritmo AM con decodificador presentado en la sección 2.2.3. Las estructuras de las redes adversarias se tomaron del repositorio BeierZhu de GitHub y se hicieron modificaciones menores para que funcio-

nara en Pytorch 1.0.0. La red generadora G es una red completamente conectada con 2 capas ocultas con 256 neuronas, la capa de entrada consta de 64 neuronas y se utilizaron funciones de activación ReLU, salvo para la capa de salida que se utilizó $tanh()$, esto último con el fin de que se obtuvieran números en [-1,1] al igual que en MNIST. El entrenamiento de las redes y el ascenso de gradiente puede ser consultado en GitHub.

En la figura 3.7 se presentan los prototipos para los números del 0 al 9 utilizando este algoritmo, los puntos iniciales se distribuyeron $\mathcal{N}_{64}(\mathbf{0}, \mathbf{I})$ y todos los prototipos presentaron $\mathbb{P}(i|X_i) > 0.999$.

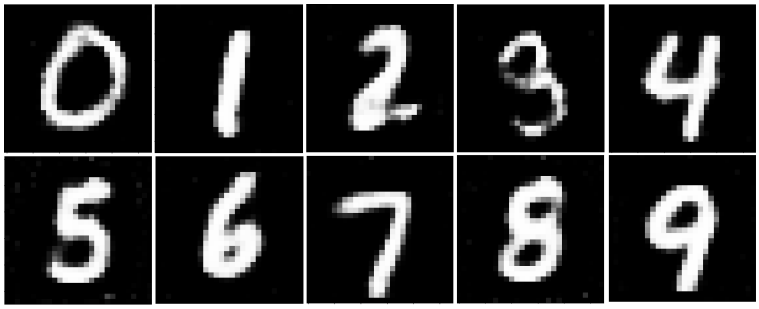


Figura 3.7: Prototipos obtenidos usando AM con decodificador.

A diferencia de los algoritmos anteriores estos prototipos son agradables a la vista, presentan poco ruido y la obtención de cada uno de los prototipos fue computacionalmente eficiente ya que obtener los prototipos una vez entrenada la GAN es una tarea que en promedio tomó 7.86 segundos, además el algoritmo AM con decodificador no presentó problemas respecto a la precisión de punto flotante.

Argumentos como los anteriores hacen que variaciones al algoritmo AM con decodificador, como el que se presenta en [4] representen el estado del arte respecto a la búsqueda de prototipos.

Si bien este algoritmo presenta resultados en la mayoría de las veces satisfactorios también se llegó a dar imágenes extrañas como lo es el siguiente prototipo:

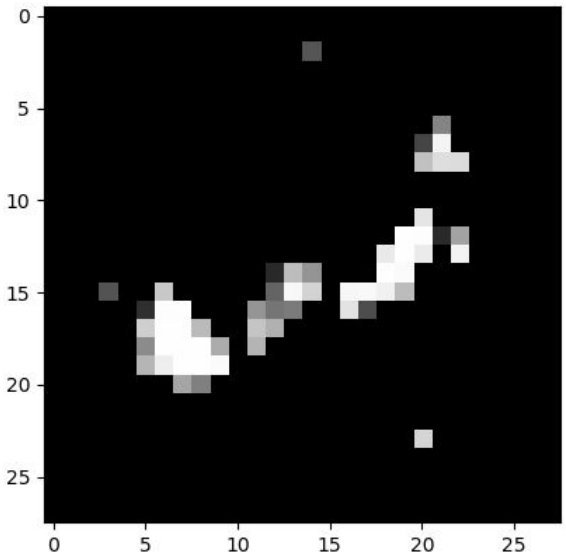


Figura 3.8: Prototipo de 2 con $\mathbb{P}(2|x_2) = 0.9983$ utilizando AM con decodificador.

Este *error* puede deberse a que las redes adversarias no tuvieron un entrenamiento perfecto, por lo que valdría la pena probar con otras estructuras de redes para G y D, buscando mejorar nuestro generador de números escritos a mano. Sin embargo, estos resultados resaltan que lo que entiende la red por un número dista de manera importante de lo que el ser humano puede entender por un número, por lo que es necesaria aún la validación humana para los prototipos.

3.4. Propagación de relevancia por capas

El algoritmo de propagación de relevancia por capas (LRP) visto en la sección 2.3.2, ha sido estudiado a detalle en [14], [3], donde ha sido probado con diferentes bases como lo son ILSVRC y VOC2009 que representan una mayor complejidad que MNIST, ya que son imágenes a color con un mayor número de categorías. En [2] el algoritimo LRP se ha probado en redes para la clasificación de texto mostrando excelentes resultados.

El proyecto heatmapping.org tiene como propósito agrupar publicaciones que desarrollen métodos para la comprensión de algorimos de *machine learning*, en particular tiene un tuto-

rial para la implementación del algoritmo [10], mismo que se utilizó para la presentación de los resultados en esta sección.

En la imágen 3.9 se presenta una aplicación del algoritmo LRP para los prototipos encontrados con las diferentes técnicas presentadas en esta tesis.

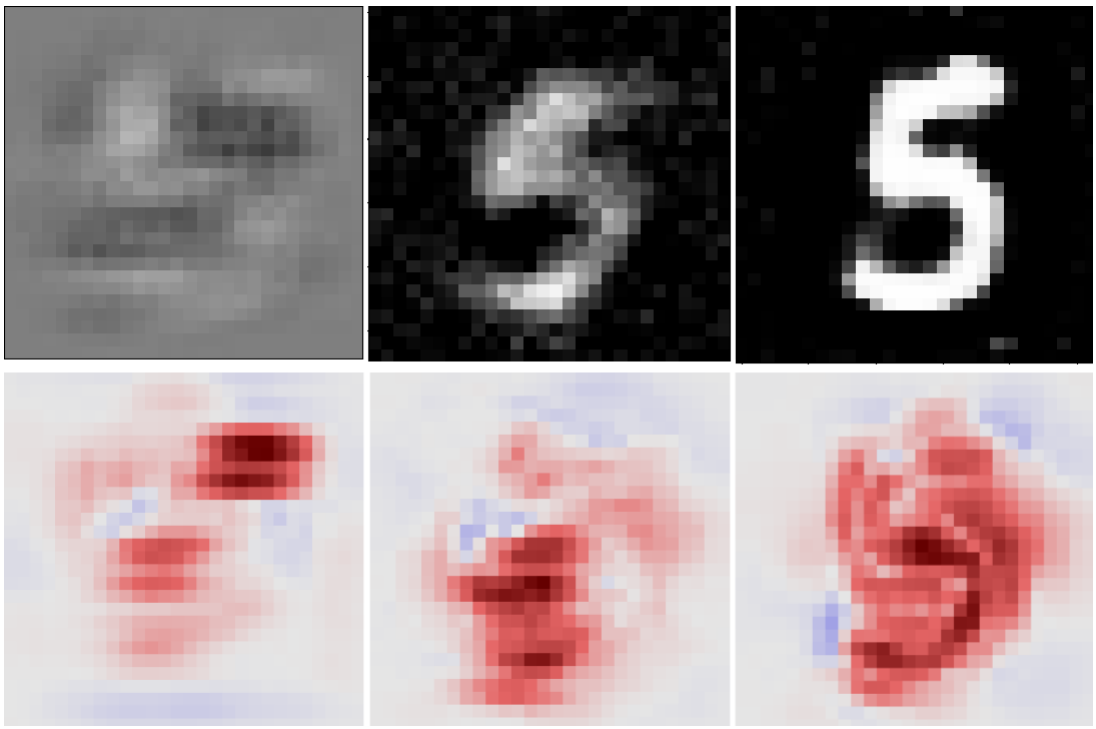


Figura 3.9: En el primer renglón de izquierda a derecha se presentan prototipos encontrados usando los algoritmos AM, AM con experto y AM con decodificador y en el segundo renglón se presentan los respectivos mapas de calor para cada imagen usando LRP con $\alpha = 2, \beta = 1$

El color rojo muestra las regiones que aportaron de manera positiva a que la imagen fuera clasificada como un 5 y en color azul las partes que impactaron de manera negativa al pertenecer a la categoría.

Se puede observar que para el algoritmo AM las regiones rojas están fuertemente concentradas de manera local, que si bien son puntos claves para reconocer un 5 están dispersos. Un fenómeno similar pasa para el caso de AM con experto, mientras que para el algoritmo AM con decodificador podemos ver que la aportación se distribuye sobre casi todo el número, llegando inclusive a resaltar la silueta del digito.

(a) (b)

Figura 3.10: Del lado izquierdo se presentan los prototipos encontrados en la sección 3.3, al lado derecho se observan los mapas de calor obtenidos aplicando el algoritmo LRP a cada prototipo.

Se tomó $\alpha = 2$ y $\beta = 1$ porque heurísticamente han mostrado ser los hiperparámetros que dan mejores resultados, sin embargo si se está interesado en mostrar con mayor peso las características negativas se sugiere usar α un poco mayor.

Es importante resaltar que este algoritmo requiere calibrar dos hiperparámetros (α y β). Además para poder dar una explicación razonable a la imagen la clasificación debe ser correcta, sin embargo, no hay ninguna garantía de que la imagen esté bien etiquetada utilizando solamente redes neuronales, por lo que suele ser necesario una validación humana que corrobore que la etiqueta es adecuada. En la figura 3.10b se presentan mapas de calor para los prototipos encontrados utilizando redes neuronales adversarias.

Capítulo 4

Conclusiones y comentarios

Este capítulo tiene como objeto exponer las ideas que surgieron a partir de haber implementado los algoritmos para interpretación y explicación. Se pretende dar un panorama al lector de la situación actual respecto al entendimiento e interpretación de redes neuronales.

4.1. Pruebas con números de 7 segmentos

Tras haber encontrado prototipos como en las figuras 3.3a, 3.3b y 3.8 podemos concluir que la red neuronal entiende por números cosas muy diferentes a las que nosotros. Si bien cuando le damos un número que está hecho a mano lo clasifica con buena precisión, parece que a ligeras perturbaciones en el dominio de datos de entrada llegamos a falsos positivos que no nos gustaría de un sistema "inteligente".

Con la idea anterior en mente y ya que los números lo permiten, probamos aplicar nuestra a red a una ligera abstracción de los números escritos a mano. Para ello tomamos los números que se presentan en los relojes digitales que se conforman solamente por trazos rectos, por lo que tomamos las imágenes de la figura 4.1.

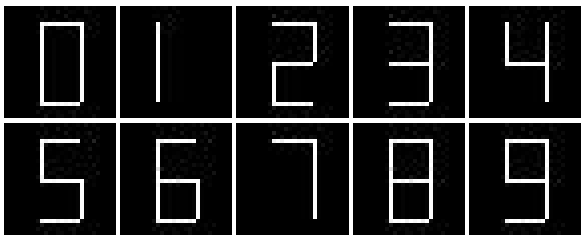


Figura 4.1: Números escritos con matriz de 7 segmentos.

Se generaron las 2^7 posibles imágenes de dígitos en la matriz de 7 segmentos para poder ver a mayor detalle que realizaba la red con este tipo de imágenes, en la imagen 4.2 se presentan algunas de las imágenes creadas.

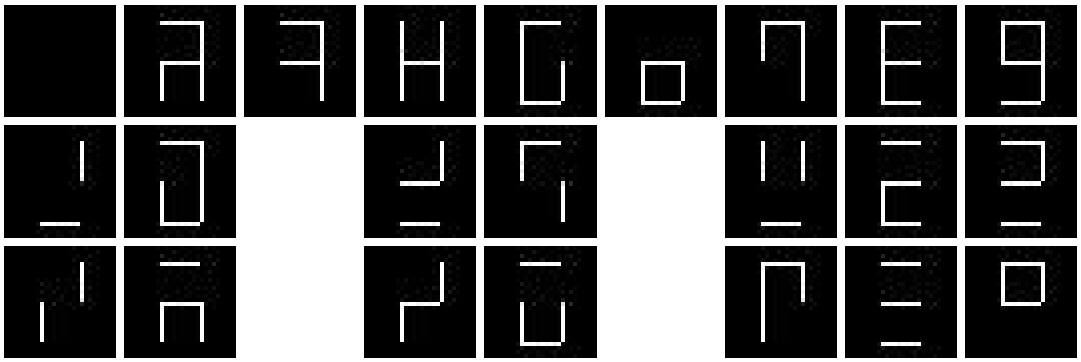


Figura 4.2: Imágenes creadas a partir de las combinaciones de la matriz de siete segmentos; las columnas de izquierda a derecha corresponden a los números del 1 al 9. las imágenes presentadas fueron colocadas en la columna que la red las clasficó.

Las probabilidades de clasificación de las imágenes en la figura 4.2 son:

1	2	3	4	5	6	7	8	9
0.42	0.74	0.53	0.94	0.74	0.37	0.54	0.92	0.59
0.51	0.51		0.66	0.36		0.69	0.89	0.43
0.31	0.38		0.57	0.31		0.24	0.37	0.79

Se procuró presentar tres imágenes por número, sin embargo hubo números para los que no hubo suficientes imágenes clasificadas en esa categoría. Todas las imágenes y resultados pueden ser consultados en Dropbox y el código utilizado para la generación y clasificación de estas imágenes puede ser consultado en GitHub. Es importante observar que las imágenes si

bien no parecen números la red está obligada a clasificar en una de las 10 posibles opciones, a diferencia del ser humano nuestra red es incapaz de decir que algo no es un número, por lo que no es de sorprenderse que todas las imágenes hayan sido clasificadas entre los números del 0 al 9.

El clasificar los números con trazos rectos no representa ningún reto para la mayoría de los humanos, por lo que de manera intuitiva nos gustaría que si la red tiene la capacidad de clasificar de manera adecuada números escritos a mano esta ligera perturbación no tuviera gran impacto, sin embargo las clasificaciones obtenidas fueron:

Número	Probabilidad del número correcto	Etiqueta	Probabilidad para la etiqueta
0	34 %	0	34 %
1	0.2 %	5	73 %
2	11 %	8	81 %
3	90 %	3	90 %
4	96 %	4	96 %
5	95 %	5	95 %
6	7 %	5	53 %
7	23 %	2	31 %
8	86 %	8	86 %
9	71 %	9	71 %

De la tabla anterior y de los prototipos encontrados podemos observar que la red entiende por números cosas que para nosotros no tienen sentido y que nuestro entendimiento de qué es un número puede presentar ligeras abstracciones que confundan drásticamente a una red. Si este tipo de complicaciones se presentan en una red con tan alta precisión con una base de datos que no representa ya un reto su entrenamiento da una perspectiva desesperanzadora sobre poder entender una red.

4.2. Ejemplos adversarios

Hasta el momento nos hemos enfocado en intentar entender y explicar redes neuronales, sin embargo una idea que nos muestra la dificultad de este objetivo es lo que se conoce como *ejemplos adversarios*. Siguiendo la notación de la sección 2.3.1, dada una imagen bien clasificada x_i en $\mathbb{R}^m$ se plantea que para alguna pequeña perturbación r ocurre que $N^*(x_i) \neq N^*(x_i + r)$; r es llamada perturbación adversaria si es aquella que resuelve

$$\begin{aligned} &\min_{r \in \mathbb{R}^m} ||r||_2 \\ &\text{sujeto a } N^*(x_i) \neq N^*(x_i + r), \end{aligned}$$

y llamaremos ejemplo adversario a $x_i + r$.

Si nuestra red es en algún sentido robusta esperaríamos que la imagen real y bien clasificada fuera notoriamente diferente de un ejemplo adversario, de lo contrario la red podría ser engañada facilmente, para verificar esto y construir ejemplos adversarios se han diseñado diferentes tipos de algoritmos, de donde resaltan el "método rápido del signo del gradiente" (*fast gradient sign method*) presentado en [9] y el algoritmo "engaño profundo" (*Deepfool*) presentado en [15], la idea en estos algoritmos es crear las perturbaciones a partir del gradiente de $N(x_i)$ para de esta manera agregar los cambios en las regiones de mayor cambio de la red.

La red que hemos trabajado presentó un 96 % de precisión en los datos de prueba, por lo que es razonable buscar ejemplos adversarios para lo cuál usaremos el algoritmo Deepfool, los resultados se pueden ver en la figura 4.3, los cuáles nos hacen ver la falta de robusticidad de nuestra red.

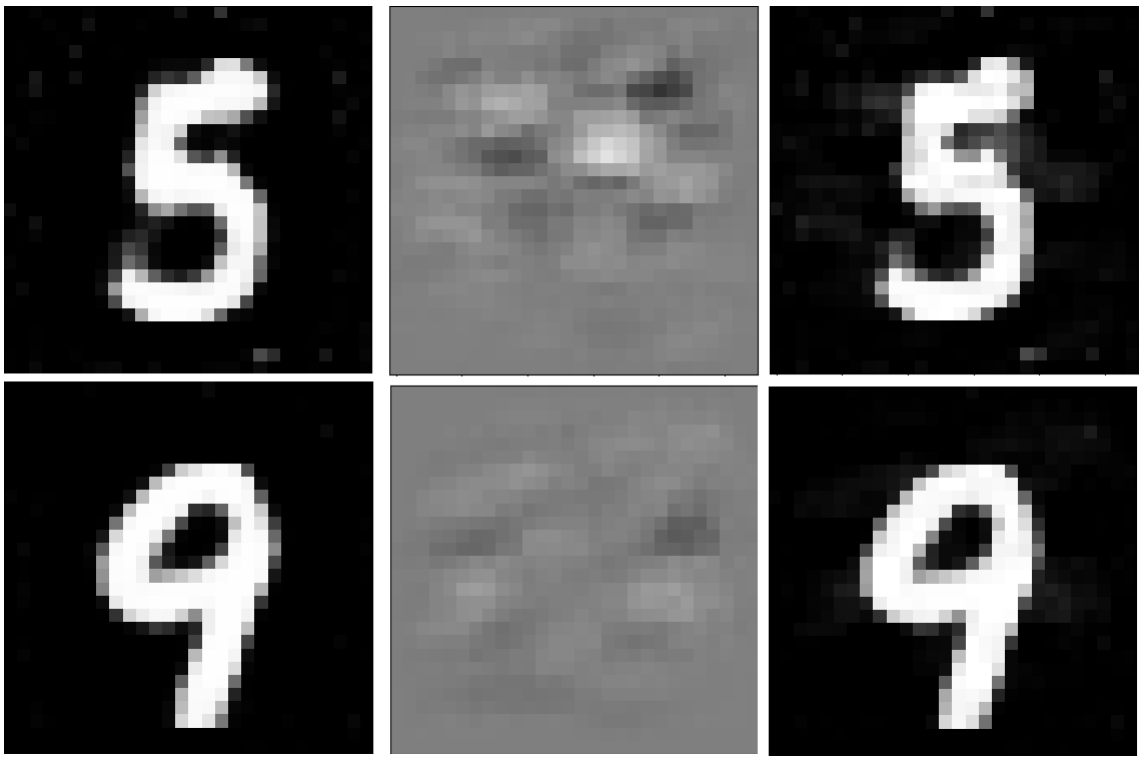


Figura 4.3: En la primer columna se muestran imágenes x_i que la red clasificó en la categoría adecuada con probabilidad mayor a 0.99, en la segunda columna se muestra la estimación de r la tercer columna muestra $x_i + r$ para ambos casos. El ejemplo adversario del primer renglón fue clasificado como 3 con probabilidad 0.57 y para el segundo renglón se clasificó como un 4 con probabilidad de 0.60.

Si bien la creación de ejemplos adversarios puede ser útil para reentrenar la red y aumentar así su robusticidad también nos son útiles para ver lo vulnerables que son los clasificadores y desconfiar de estos.

El código para la generación de estas imagenes puede ser consultado en GitHub

4.3. Algoritmos de explicación e interpretación

Respecto a los algoritmos para búsqueda de prototipos se presentan a contiinuación las series de tiempo de la optimización de la función objetivo:

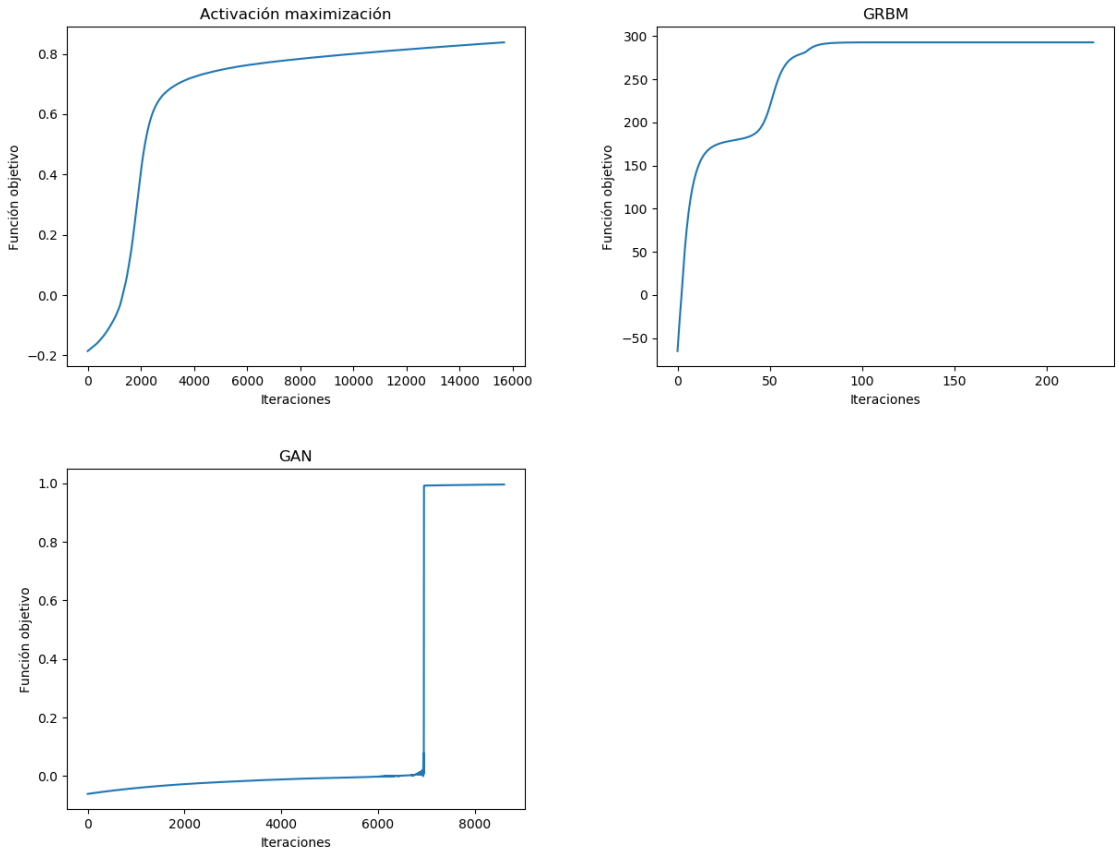


Figura 4.4: Función objetivo por iteración para los algoritmos de búsqueda de prototipos.

Para el algoritmo AM, se requirieron 90 segundos para generar la imagen de la figura 4.4, lo que significa que cada iteración tardó 0.005 segundos en promedio, para el algoritmo AM con experto se requeririeron 45 minutos, cada iteración en promedio tardó 11 segundos y para el algoritmo AM con decodifiicador se utilizaron 270 segundos, por lo que cada iteración tardó en promedio 0.03 segundos. Es importante resaltar que mientras que para AM y AM con experto el crecimiento de la función objetivo es suave, en el caso de AM con decodificador esto no siempre ocurre, durante las repeticiones del experimento se observó que había cambios abruptos en la función de costo. Si bien esta falta de suavidad puede ser corregida realizando el ascenso de gradiente con una tasa de actualización más baja, se encontró que no era recomendable ya que podía quedarse en un máximo local fácilmente, en cambio si se toma una tasa de crecimiento elevada el algoritmo converge en un menor número de iteraciones

y suele hacerlo a un máximo satisfactorio.

Los algoritmos presentados nos permitieron en mayor o menor medida encontrar prototipos de las categorías, sin embargo, no podemos decir que de forma satisfactoria hemos interpretado qué es un número para una red neuronal. La conclusión es que hemos sido capaces de entender que las características que toma la red en cuenta para asignar una categoría son muy distintas a las que los seres humanos tenemos; si bien, los algoritmos presentados en esta tesis requieren aún de la supervisión humana, representan un avance para que el uso de las redes neuronales sea más responsable e ir dismitificando su uso. Sin embargo, nos quedan muchas dudas de que podamos estar en una posición en la que digamos que hemos interpretado como "funciona" una red neuronal como clasificador. Mucho menos que podamos decir que aprendimos algo del clasificador.

Bibliografía

[1] Arbabzadah, F., Montavon, G., Müller, K.-R., and Samek, W. (2016). Identifying individual facial expressions by deconstructing a neural network. In *German Conference on Pattern Recognition*, pages 344–354. Springer.

[2] Arras, L., Horn, F., Montavon, G., Müller, K.-R., and Samek, W. (2017). "what is relevant in a text document?": An interpretable machine learning approach. *PloS one*, 12(8):e0181142.

[3] Bach, S., Binder, A., Montavon, G., Klauschen, F., Müller, K.-R., and Samek, W. (2015). On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, 10(7):e0130140.

[4] Dosovitskiy, A. and Brox, T. (2016). Generating images with perceptual similarity metrics based on deep networks. In *Advances in neural information processing systems*, pages 658–666.

[5] Esteva, A., Kuprel, B., Novoa, R. A., Ko, J., Swetter, S. M., Blau, H. M., and Thrun, S. (2017). Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 542(7639):115.

[6] Fischer, A. and Igel, C. (2012). An introduction to restricted boltzmann machines. In *Iberoamerican congress on pattern recognition*, pages 14–36. Springer.

[7] Fischer, A. and Igel, C. (2014). Training restricted boltzmann machines: An introduction. *Pattern Recognition*, 47(1):25–39.

[8] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014a). Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.

[9] Goodfellow, I. J., Shlens, J., and Szegedy, C. (2014b). Explaining and harnessing adversarial examples (2014). *arXiv preprint arXiv:1412.6572*.

[10] Heatmapping.org (2017). Implementing deep taylor decomposition / lrp. http://heatmapping.org, Accessed: 05/06/2019.

[11] Melchior, J. (2012). Learning natural image statistics with gaussian-binary restricted boltzmann machines.

[12] Melchior, J. (2018). Pydeep. https://github.com/MelJan/PyDeep.git.

[13] Melchior, J., Wang, N., and Wiskott, L. (2017). Gaussian-binary restricted boltzmann machines for modeling natural image statistics. *PloS one*, 12(2):e0171015.

[14] Montavon, G., Samek, W., and Müller, K.-R. (2018). Methods for interpreting and understanding deep neural networks. *Digital Signal Processing*, 73:1–15.

[15] Moosavi-Dezfooli, S.-M., Fawzi, A., and Frossard, P. (2016). Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2574–2582.

[16] Nguyen, A., Dosovitskiy, A., Yosinski, J., Brox, T., and Clune, J. (2016). Synthesizing the preferred inputs for neurons in neural networks via deep generator networks. In *Advances in Neural Information Processing Systems*, pages 3387–3395.

[17] Nielsen, M. A. (2015). *Neural networks and deep learning*, volume 25. Determination press USA.

[18] Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). Automatic differentiation in pytorch.

[19] Schluter, J. (2014). Restricted boltzmann machine derivations. Technical report, Technical Report TR-2014-13, Osterreichisches Forschungsinstitut fur.

[20] Tra, V., Kim, J., Khan, S. A., and Kim, J.-M. (2017). Bearing fault diagnosis under variable speed using convolutional neural networks and the stochastic diagonal levenberg-marquardt algorithm. *Sensors*, 17(12):2834.