Universidad de Buenos Aires

Facultad de Ingeniería

75.06 - Organización de Datos

Luis Argerich
Natalia Golmar
Damian Martinelli
Martín Ramos Mejia
Víctor Podberezski

Trabajo Práctico

Equipo: estoSeVaADescontrolaaar

Francisco Ordoñez 96478
Santiago Lazzari 96735

14 de Abril
1er. Cuatrimestre 2016

Contenido

1 Exploración de datos

1.1 Training set

label	Píxel0	...	Píxel783
0-9	0-255	...	0-255

El training set está compuesto por 42.000 filas y 785 columnas. La columna label indica la clase a la que pertenece la imagen. Las columnas Píxel0...Píxel783 representan cada uno de los 28*28 píxeles que componen la imagen en escala de grises. El set está bien balanceado, como se puede apreciar en el siguiente gráfico.

1.1.1 Imágenes por clase

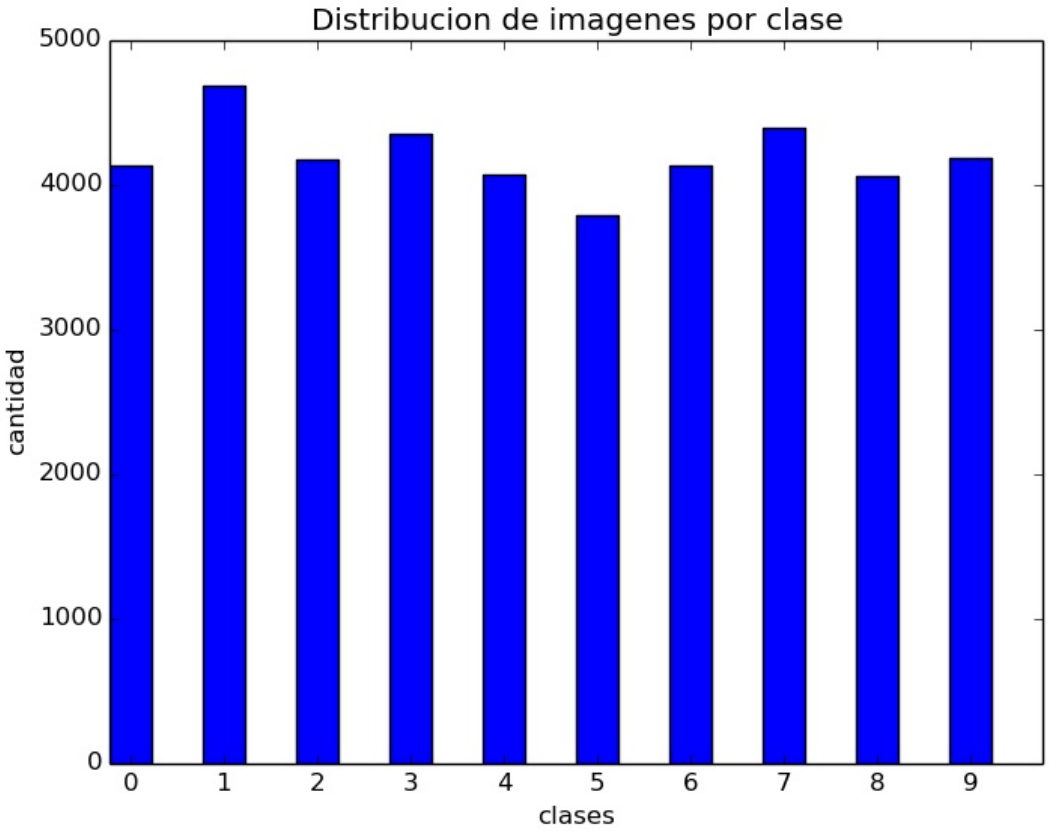

Figure 1: Clases del set

La clase con mayor cantidad de imágenes es la 1 con 4684 entradas y la clase con menor cantidad es la 5 con 3795.

1.2 Testing set

El testing set está compuesto por 28.000 filas y 784 columnas. Las columnas Píxel0...Píxel783 representan cada uno de los 28*28 píxeles que componen la imagen en escala de grises.

Píxel0	...	Píxel783
0-255	...	0-255

1.3 Píxeles anómalos

Analizando los datos nos dimos cuenta que hay algunos píxeles anómalos. Hay unos 65 píxeles que están prendidos en solo 10 imágenes. La solución mas sencilla es apagarlos. Algunos ejemplos:

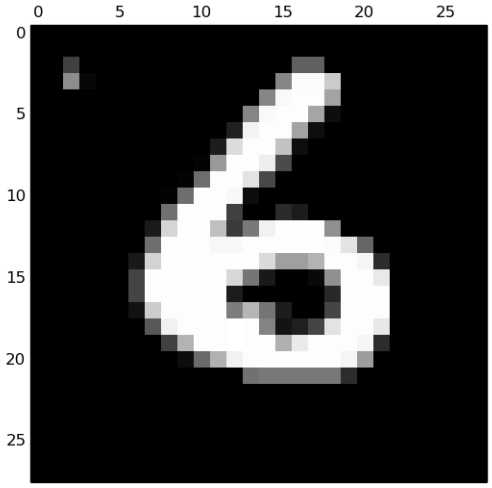

Figure 2: Pixel número 58 encendido en la imagen 15.247 del training set

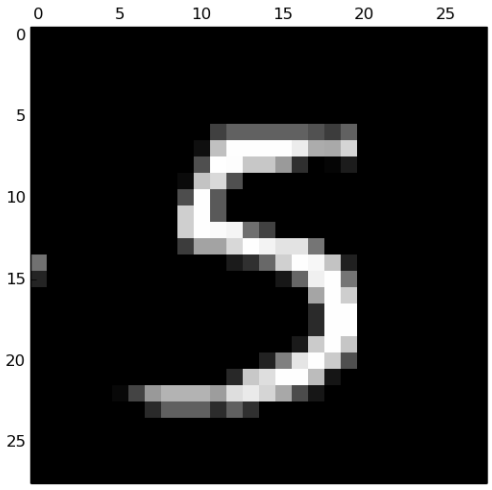

Figure 3: Pixel número 420 encendido en la imagen 17.258 del testing set

y hay algunas imágenes que solamente quieren ver el mundo arder...

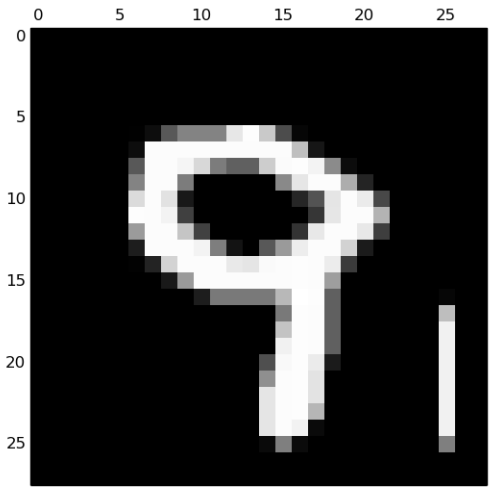

Figure 4: Error en la imagen 7.281 del training set

Curiosamente, no hay ningún píxel que esté encendido en las 70.000 imágenes. El que más veces se enciende es el píxel 435 unas 50.936 veces. Por lo tanto no nos podemos deshacer de ellos.

2 Edición del set de datos

2.1 Quita de píxeles apagados

Como dijimos en la sección de píxeles anómalos, hay muchos que están apagados en todas las imágenes. Más que nada los de las esquinas. Si los quitamos ganamos 65 dimensiones. También hay muchos píxeles que están apagados en el 99% de las imágenes, si los quitamos pasamos de 784 dimensiones a 490. En las pruebas que hicimos en kaggle se puede observar como bajando hasta el 95% no cambia el resultado. Esto nos ahorra muchísimo tiempo a la hora de correr nuestro algoritmo. Otra posible solución para reducir la cantidad de dimensiones es tomar x píxeles consecutivos y hacer su promedio. Por ejemplo, si tomamos $x = 4$, pasamos de tener 784 dimensiones a 196.

2.2 The Hashing Trick

Para hacer al algoritmo más eficiente se intentó reducir dimensiones con el **Hashing Trick**, y sabiendo que podíamos estimar el error con el teorema de Johnson y Lindenstrauss [3, Chapter 4, p. 205] como:

$$k = \frac{4 * log(n)}{\frac{\epsilon^2}{2} - \frac{\epsilon^3}{3}}$$

Donde n es la cantidad de datos en nuestro dataset.

Llegamos a la conclusión de que para reducir significativamente nuestras dimensiones nos iba a quedar un error demasiado grande, 0.5.

Esto nos lleva a la conclusión de que quizás las dimensiones no hacen menos eficientes y precisas las comparaciones con KNN, ya que con la fórmula ya descripta indica que hay muchos mas datos en relación a la cantidad de dimensiones de los vectores. Lo que podría acercarnos a otro approach que es reducir el dataset, sabiendo que hay una cantidad enorme de datos podríamos reducir datos con diferentes técnicas, ésta puede ser de carácter aleatorio y reducir a la n-sima parte todas las muestras de forma

azarosa, también podríamos proceder a recortar utilizando promedios entre vectores y asignando pesos a los mismos, estas técnicas serán desarrolladas más adelante en el trabajo práctico.

2.3 SVD: Singular Value Decomposition

Otra forma de reducir la cantidad de dimensiones es usando la **SVD**. Esta nos da una representación de cualquier matriz y nos permite eliminar datos no representativos. Obviamente, a menor cantidad de dimensiones mayor va a ser el error. [4]

$$\underset{m \times n}{\hat{X}} \approx \underset{m \times r}{U} \underset{r \times r}{S} \underset{r \times n}{V^{\mathsf{T}}}$$

Figure 5: SVD

Donde:

- X es la matriz original.
- U es una matriz de m x r.
- S es diagonal y son los valores singulares de M. Es de r x r.
- V^T es de n x r, se usa de forma transpuesta.

La clave de la **SVD** es ver las r columnas de U, S y V^T que representan conceptos ocultos de la matriz original.

- La matriz U conecta filas con conceptos.
- La matriz S nos da la energía de cada columna de la matriz U
- La matriz V^T conecta columnas con conceptos.

2.3.1 Reducción de dimensiones

Para reducir las dimensiones de la matriz U simplemente tomamos las n primeras columnas(con $n < r$), siempre teniendo en cuenta que las columnas que tomemos tienen que representar **por lo menos** el 90% de la energía total de la matriz S. [4, Chapter 11, p. 424]

Para calcular la energía de la matriz se utiliza la siguiente formula:

$$\sum S_i^2$$

Donde $i = 1..n$.

Se le aplico la **SVD** al set de datos y tomando entre 50 y 200 dimensiones se calculo el porcentaje de conservación de energía.

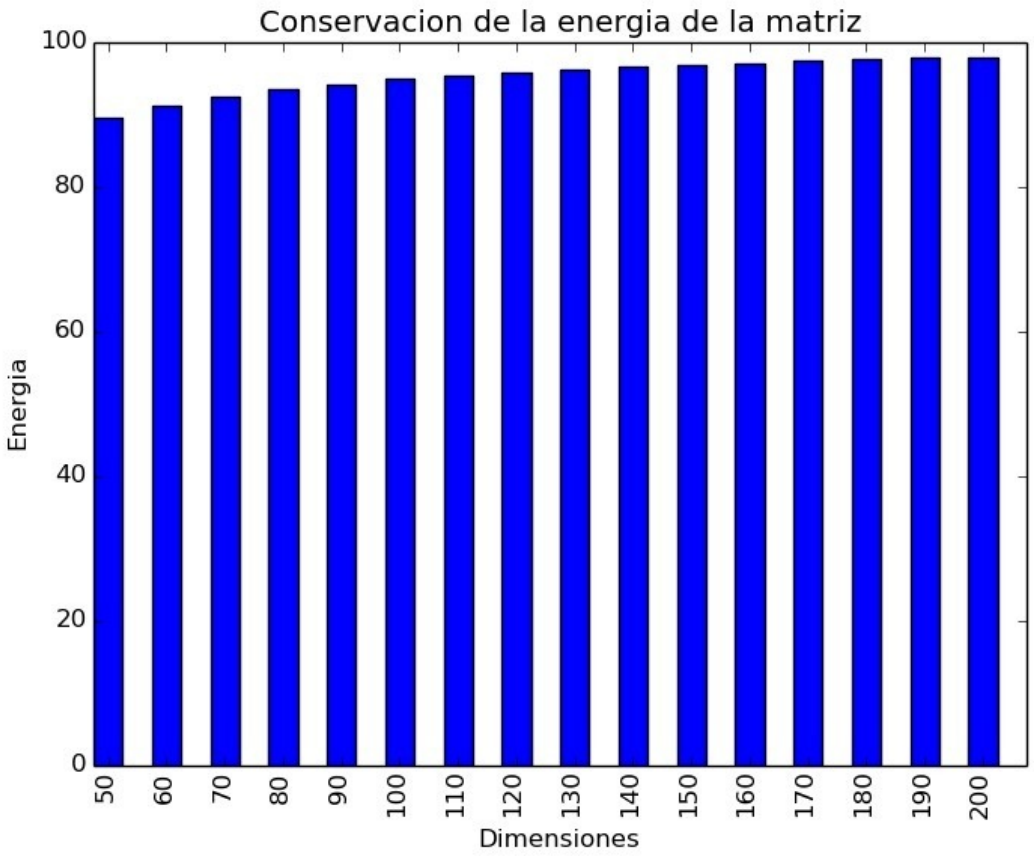

Figure 6: Energía de la matriz tomando distintos valores de n

Como se aprecia en el gráfico todas las dimensiones conservan más del 90% de la energía. Con 100 dimensiones se conserva el 94.96% y con 150 el 96.94%. Viendo esto podemos concluir que la dimensionalidad del set de datos es mucho menor que la original.

3 Investigación de algoritmos

3.1 KNN

KNN (del inglés k nearest neighbors) es un algoritmo que decide cuál es la clase de un elemento a partir de sus k elementos más cercanos, es decir, **1NN** es comparando con todos los elementos en el set de datos, y con alguna métrica (euclideana, Chebyshev, etc) decidir cual es el más cercano, y entonces el dato incógnito será aquel del más cercano. Esto lo convierte en un algoritmo de orden $O(n^2)$.

Como es un algoritmo muy sencillo hicimos pruebas para ver como se comportaba el set de datos aplicándole distintos k y distintas métricas y los resultados fueron los siguientes.

3.1.1 Pruebas de Kaggle

La columna "Ceros" muestra la cantidad necesaria de ceros que debe tener una columna para ser ignorada.

K	Métrica	Ceros	Resultado
9	BrayCurtis	68.600	0.96600
9	Canberra	68.600	0.955529
5	Chebyshev	68.600	0.80257
5	Euclideana	70.000	0.96800
5	Jaccard	68.600	0.959971
5	Jaccard	70.000	0.959971
3	Euclideana	69.300	0.96829
3	Euclideana	70.000	0.96857
3	Kulsinski	70.000	0.10014
1	Euclideana	69.300	0.97114
1	Euclideana	70.000	0.97114
1	Euclideana	67.900	0.97114
1	Euclideana	Null	0.77100
1	Euclideana	Null	0.91129
1	Euclideana	Null	0.91157

Como se puede apreciar, el mejor resultado resultó de usar la distancia euclidiana con $k = 1$. Como dijimos anteriormente, el resultado final no se modifica si reducimos el set de datos quitando los píxeles apagados.

Las últimas 3 entregas fueron creadas reduciendo las dimensiones usando $TheHashingTrick$ a 50, 100 y 150 dimensiones respectivamente. Como parámetros para las funciones de hashing se seleccionaron aleatoriamente los siguientes valores: A = 7689459, B = 89895 y P = 179426549.

3.2 Perceptrón

El perceptrón fue el primer algoritmo que salió a luz a principio de la investigación, este se trata de la herramienta básica de la inteligencia artificial ya que es el instrumento más parecido y simple para representar una neurona biológica.

Al igual que una neurona biológica, el perceptrón recibe muchos inputs y devuelve un único output, es decir que le aplica una función a un vector de la forma $f : \Re^n \to \Re$ pero donde n representa la dimensión del vector de entrada. En el caso del perceptrón $\Re$ es el conjunto de los números binarios, es decir, recibe un vector de números binarios y devuelve un número en binario.

El perceptrón es un algoritmo que tiene una estructura de pesos w_i al que se lo entrena. El concepto de entrenar es propio de la inteligencia artificial y hace referencia a modificar la estructura interna, en este caso del perceptrón, es decir sus w_i, a partir de un set de datos. Estos pesos se irán modificando mediante la cuenta.

$$w_i = w_i + \alpha * (\delta - y) * x_i$$

Donde x_i representa el i-ésimo elemento de un vector del set de datos. y es la salida del perceptrón. δ el resultado esperado, mientras que α es una constante.

Aplicando éste algoritmo repetidas veces sobre los pesos harán que éstos converjan a un vector de pesos de existir algún patrón en el vector de entrada, entonces se podrá decir que el perceptrón aprendió. Entonces para cualquier vector de entrada responderá acorde al vector de pesos que tiene como estructura interna.

Algunos problemas son que, un solo perceptrón no tendría la posibilidad de responder a un set de datos de gran complejidad ya que éste detectara patrones en una sola dimensión real.

Cabe aclarar que el perceptrón solo funciona en problemas linealmente separables. El problema es que muchas veces los datos no son separables por un hiperplano como se muestra en el siguiente gráfico.

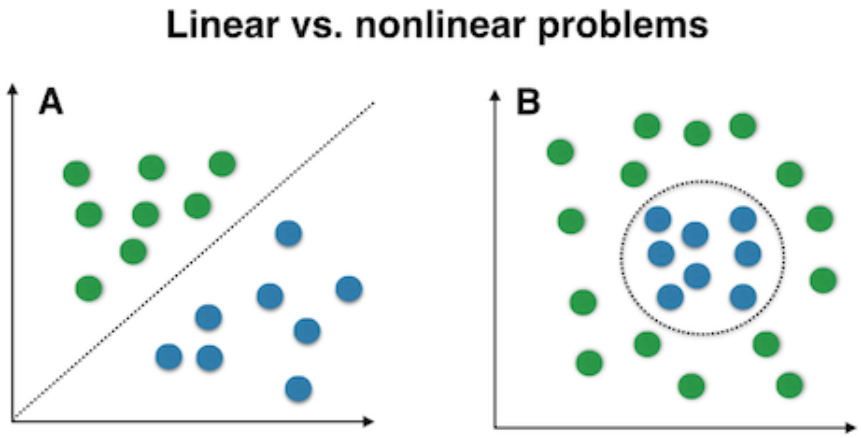

Figure 7: B muestra un problema no linealmente separable

Para este tipo de problemas se pueden usar redes neuronales.

3.3 Red multi capa

Una red multicapa utiliza el concepto de los perceptrones donde donde cada perceptrón es una neurona de una red que conecta varias de estas neuronas. Una red multicapa se puede visualizar de la siguiente manera.

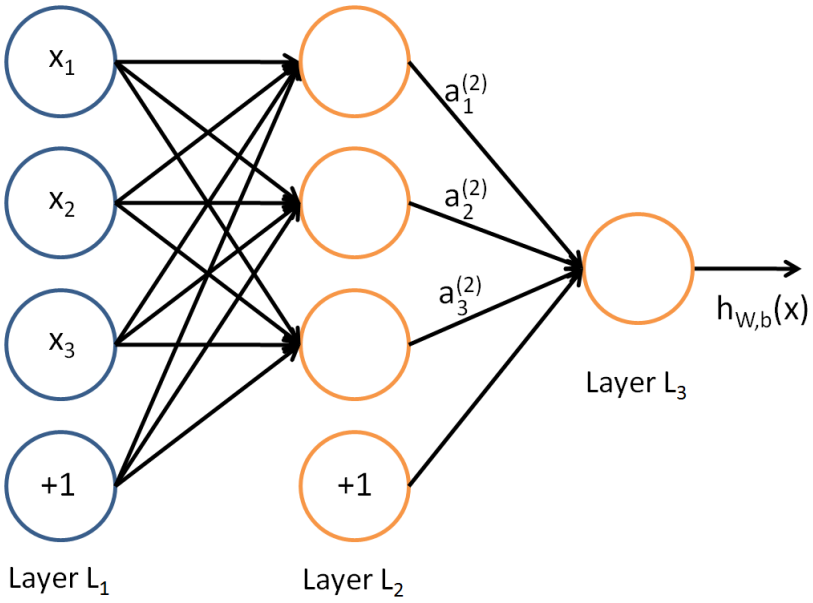

Figure 8: Red multicapa

Donde la primer capa es la **Input layer** que baipasea la información que llega de input y la reparte en la siguiente capa. Las subsiguientes capas son las **Hidden layers** donde se van distribuyendo los datos capa a capa y finalmente se llega al **Output layer** que es el resultado. En la imagen presentada el output layer es una sola neurona, sin embargo no es necesario, podría haber n neuronas en el output layer.

Las redes neuronales se les puede enseñar de muchas formas distintas que se pueden ver en la siguiente sección.

3.3.1 Modelos de aprendizaje

Hay distintos modelos de aprendizaje para una red, por un lado están los **Algoritmos de aprendizaje**:

- **Corrección de error**
- **Aprendizaje Hebbiano**
- **Aprendizaje Competitivo**

Por otro lado están los **Paradigmas de aprendizaje**:

- **Aprendizaje Supervisado**
- **Aprendizaje reforzado**
- **Aprendizaje no supervisado**

El **Aprendizaje Supervisado** es el tipo de entrenamiento en el cual se provee al sistema con información de las entradas al igual que se proveen las salidas esperadas o destinos correspondientes a dichas entradas a modo que el sistema tenga los destinos como punto de referencia para evaluar su desempeño en base a la diferencia de estos valores y modificar los parámetros libres en base a esta diferencia. [5]

En un **Aprendizaje no supervisado** los parámetros libres del sistema son modificados únicamente en base a las entradas del sistema de manera que aprenden a categorizar las entradas y clasificarlas sin necesidad de una referencia. [5]

La **Regla delta** o **Algoritmo de corrección de error** se refiere al modelo de aprendizaje que se basa en minimizar una función de error en relación a los parámetros libres del sistema. Una vez definida la función de costo a minimizar la corrección de error es un problema estrictamente de optimización. [5]

El **Aprendizaje Hebbiano** indica que las conexiones entre las neuronas de entrada activas y las neuronas de salida activas se refuerzan durante el entrenamiento: coincidencias entre actividad de entrada y actividad de salida se intensifican. Mientras que las conexiones entre neuronas de entrada inactivas y neuronas de salida (Activas o Inactivas) no se refuerzan.

Este método de aprendizaje puede ser tanto supervisado como no supervisado. Cuando es supervisado, la respuesta correcta para el dato de entrada es introducida para cada neurona de salida, y los pesos sinápticos entre las neuronas activas se incrementan, mientras que los pesos entre neuronas que no estén activas simultáneamente permanecen igual que estaban. [2]

En el **Aprendizaje competitivo** suele decirse que las neuronas compiten (Y Cooperan) unas con otras con el fin de llevar a cabo una tarea dada. Con este tipo de aprendizaje se pretende que cuando se presente a la red cierta información de entrada, sólo una de las neuronas de salida de la red, o una por cierto grupo de neuronas, se active (Alcance su Valor de Respuesta Máximo). Por tanto las neuronas compiten para activarse quedando finalmente una, o una por grupo, como neurona vencedora y el resto quedan anuladas y siendo forzadas a sus valores de respuesta mínimos. [2]

4 Análisis de los algoritmos

El algoritmo de **KNN**, si bien es un algoritmo muy sencillo de implementar con muy buenos resultados, es de orden $O(n^2)$ costoso computacionalmente hablando y no explora otros algoritmos de la materia que si bien quizás den peores resultados, harán que aprendamos algoritmos muy usados en la actualidad.

En cuanto a la reducción de dimensiones, el método de **SVD** demostró ser eficiente así que puede ser utilizado para pelear con el problema de la dimensionalidad. Por otro lado, si se redujeran demasiadas dimensiones entonces el resultado del mismo es que el error de la reducción seria demasiado grande, es decir que se requiere encontrar un equilibrio de lo que sea conveniente reducir en dimensiones.

En cuanto a la inteligencia artificial por el lado de los perceptrones y las redes neuronales, encontramos que en la página de kaggle muchas soluciones iban por ese lado. Como positivo es que una vez entrenado el perceptrón o la red neuronal, el algoritmo torna a resolver el problema en orden $O(1)$, es decir constante. Ésta es una ventaja muy grande comparada con **KNN**. Cabe destacar que sin embargo, el perceptrón sufre de la maldición de la dimensionalidad, esto quiere decir que de tener un input con muchas dimensiones le costará mucho más aprender y se puede caer en un problema de **Underfitting**. Esto significa que la cantidad de datos tiene que ser suficientemente grande como para que el perceptrón aprenda correctamente. Por esto a nuestra red neuronal la vamos a entrenar con una cantidad pequeña de capas ocultas, al usarse demasiadas se corre el peligro de que el algoritmo tome mucho tiempo en aprender.

Con respecto a los algoritmos de aprendizaje se deben destacar las siguientes problemáticas. En cuanto al paradigma de **Aprendizaje no supervisado** el principal problema es que no deja detectar anomalías y como fue analizado previamente había imágenes con anomalías. El algoritmo de **Aprendizaje Hebbiano** no tiene en cuenta la eficacia de la red. Así, aunque la red ya este entrenada y los valores de entrada generen valores de salida correctos, la regla de aprendizaje continua incrementando los pesos sinápticos entre neuronas activas. Finalmente el **Aprendizaje Competitivo** puede presentar el problema de que algunas unidades de proceso no ganen nunca, es decir, no se activen, lo que quiere decir que esos patrones nunca van a ser reconocidos por la red.

5 Algoritmos a utilizar

Siendo que **KNN** es de orden $O(n^2)$ y de naturaleza tan simple lo descartamos de los algoritmos posibles.

Por otro lado, nos quedamos con los algoritmos de machine learning y optamos por generar una red neuronal con **corrección de error** y **aprendizaje supervisado**, la regla de aprendizaje será la de **back propagation**.

Para compensar el problema de la dimensionalidad le aplicaremos al set la **SVD** y reduciremos a 100 dimensiones ya que empíricamente obtuvimos resultados favorables y la energía que se conserva es muy grande con esa cantidad de dimensiones como fue explicado en la sección de modificaciones del set de datos.

La red neuronal final será de la siguiente forma:

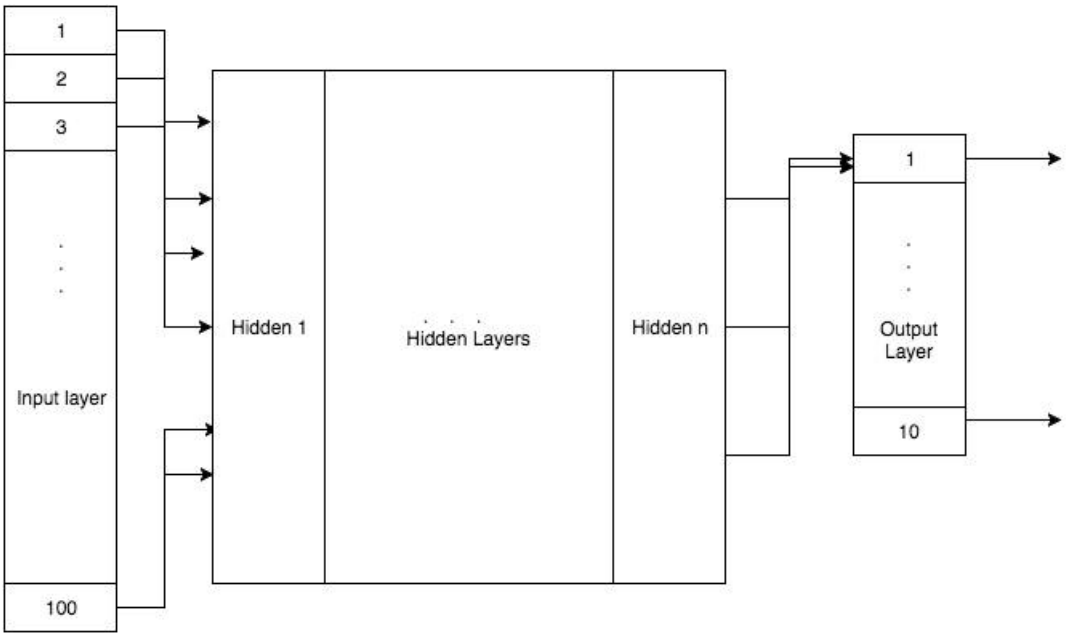

Figure 9: Modelo de red neuronal

Se puede ver que el input es 100 que equivale al número de dimensiones del vector ya reducido con **SVD** y el output son 10 salidas. Cada salida representa a un dígito del set de datos, es decir, si la imagen ingresada es 0, entonces la salida de la red neuronal debería ser 1 en el primer output, y 0 en los restantes.

La cantidad de **hidden layers** va a variar entre 2 y 4, agregar más no aportaria valor alguno y haria a la red extremadamente lenta. Una forma podria ser **Input layer** = 100, **Hidden layer 1** = 50, **Hidden layer 2** = 25 y finalmente **Output layer** = 10 donde la igualdad representa la cardinalidad, es decir la cantidad de neuronas que tiene cada capa. Si se llega a detectar **underfitting** u **overfitting** entonces estas capas ocultas se deberán modificar.

Finalmente, el último detalle reside en cómo se le enseña a la red. En este caso decidimos usar **Backpropagation**, un algoritmo de aprendizaje supervisado. El algoritmo consiste de dos fases. En la primera, luego de ingresar los datos a la red y generar una salida, esta es comparada con el output deseado y se calcula un error para la misma. La segunda fase consiste en propagar ese error hacia las capas ocultas de la red distribuyendo dicho error según la responsabilidad que tuvo cada neurona con el

mismo. Este proceso se repite hasta que todas las neuronas que hayan contribuido reciban una señal del error. La importancia de este proceso consiste en que, a medida que se entrena la red, las neuronas de las capas ocultas van a reconocer patrones. Entonces, cuando se le ingrese un nuevo input, las neuronas van a saber reconocer el patrón y van a responder con una salida activa si el input se parece a algún patrón conocido [1].
En el caso de que haya algún conflicto en la output layer, por ejemplo que 2 neuronas devuelvan 1. Se van a resolver teniendo guardado en memoria una imagen promedio de cada dígito y a partir de éste si, por ejemplo, los resultados activos fueron el 3 y el 5 entonces comparar el input con los representativos y ver cuál es el más cercano, si el más cercano es el 3 entonces el resultado final será el 3 y se descartará la solución del 5. Como con las pruebas de Kaggle el mejor algoritmo encontrado fue **1NN** con métrica **euclideana** entonces procederemos a calcular la distancia con dicha métrica.

References

[1] Backpropagation. `https://en.wikipedia.org/wiki/Backpropagation`.

[2] Redes neuronales. `http://www.hugo-inc.com/RNA/Unidad%204/4.1.2.html`.

[3] Luis Argerich. *Apunte Organización de datos*. Febrero 2016.

[4] Jeffrey D. Ullman Jure Leskovec, Anand Rajaraman. *Mining massive Datasets*. Marzo 2014.

[5] UDLAP. Perceptrón multicapa. `http://catarina.udlap.mx/u_dl_a/tales/documentos/lep/mejia_s_ja/capitulo3.pdf`.