Redes Neuronales
Trabajo Practico 1

Eitan Rosenzvaig Martín Forte

Resumen

En el presente trabajo nos proponemos experimentar sobre redes neuronales. Construyendo las mismas utilizando el lenguaje Python, hemos aplicado las mismas a dos datasets diferentes para lograr crear un clasificador y una regression. En el siguiente informe presentaremos los resultados sobre los distintos datasets.

Index Terms

Python, Redes Neuronales, Regression, Clasificador

I. Introducción

En este trabajo presentamos el desarrollo de una red neuronal y luego su utilizacion en 2 distintos data sets. La red fue construida utilizando el lenguaje Python y la libreria de Numpy para las operaciones elementales, como por ejemplo la multiplicacion de matrices. El primer dataset que se utilizo para la experimentacion con la red neuronal fue desarrollado como experimento para lograr clasificar como benigno o malingo un tumor en las mamas a partir de 10 caracteristicas provenientes de imagenes digitales. Es decir debemos clasificar como positivo o negativo (1 o 0) partiendo de 10 caracteristicas numericas y continuas. El segundo dataset consiste en determinar los requerimientos de carga energetica para calefaccionar y refrigerar edificios en funcion de 8 carateristicas numericas y continuas de los mismos. Los dos valores a predecir son la carga de calefaccion y refrigeracion necesarios. Dichas cargas energeticas son representados por numeros continuos y por ende la red debe aprender a realizar una regression desde las caracteristicas hacia las cargas.

II. Desarrollo

A. Preprocesamiento

Dentro del archivo *DatasetNormalizer.py* hemos decidido implementar una funcion de preprocesamiento *discretization*, ademas de otras funciones de carga de datos, que realiza una transformacion a los datos de entrada a la red. Esta funcion logra transformar los numeros continuos de entrada en vectores de valores binarios, es decir, en vectores de 1s y 0s. Esto permite un mejor y mas eficiente aprendizaje sin perder generalidad a la hora de predecir sobre datos nuevos. La funcion de preporcesamiento realiza la siguiente transformacion a cada columna del dataset (no asi de las variables a predecir):

- Aplicar el logaritmo si la variable esta sesgada
- Calcular media y desvio estandar
- Crear el vector de bits (explicado a continuacion)

El vector de bits de un numero representa un estilo de termometro binario que refleja cuan 'grande' es dicho numero con respecto a la media y n desvios estandar de la misma. Para lograr esto en principio definimos cuantos bits tendra nuestro vector, en nuestro caso son 30 bits. Luego cada bit representa una cantidad de desvios estandar mas grandes o mas chicos que la media, en nuestro caso el primer bit representa 15 desvios estandar menos que la media y el ultimo bit representa 15 desvios estandar mas que la media. Asi el bit 15 representa justo la media, osea, cero desvios estandar mas o menos que la media. Al finalizar para cada entrada de dicha variable prendemos el i-esimo bit (de los 30) si y solo si dicha entrada es mayor que i desvios estandar mas o menos que la media.

Por ultimo este proceso guarda en un archivo pickle todos los datos necesarios (minimos, maximos, desvios, media, etc) para aplicar dicha discretizacion sobre datos nuevos.

B. Red Neuronal

Para programar la red neuronal seguimos el codigo descripto en las clases practicas de la materia, logrando asi plasmar dentro de la clase *NN* todas las funciones detalladas. Ademas se utilizo la libreria de Numpy para las operaciones matriciales. A continuacion detallamos las funciones implementadas y una breve descripcion de las mismas:

- init - Funcion constructura de la clase que toma como parametros obligatorios un array de tamanos de cada capa (incluyendo la de entrada), un array de tuplas de mismo tamano que la cantidad de capas - 1 donde cada tupla contiene la funcion de activacion y su derivada y por ultimo el learning rate de la red. Luego pasa a construir todas las estructuras necesarias para la red (por ejemplo, matriz de pesos)
- activation - Realiza la activacion o "forward pass"de una instancia de entrada a la red
- correction - Calcula el error y el backpropagation del mismo hacia las capas anteriores tomando en cuenta la ultima activacion
- adaptation - Aplica la correccion del error encontrado por la funcion correction a los pesos correspondientes
- predict- Toma un conjunto de instancias y devuelve un conjunto de predicciones correspondientes a cada instancia
- f_batch, mini_batch - Son funciones de entrenamiento que toman un conjunto de datos y entrena a la red de distintas maneras

C. Requrimientos y modo de uso

Para poder ejecutar nuestro codigo se debe contar con los siguientes requrimientos:

- Python 2.6 o mayor
- Libreria Numpy
- Libreria cPickle
- Libreria Scipy

A continuación presentaremos algunos ejemplos para utilizar nuestra imeplementación.

Training Ejercicio 1:

python main.py –problem 1 –mode training –input ./data/tp1_ej1_training.csv –layers 200 –lr 0.01 –epocs 3000 –save nn-ej1

Test Ejercicio 1:

python main.py –problem 1 –mode test –input ./data/tp1_ej1_test.csv –load nn-ej1

Training Ejercio 2:

python main.py –problem 2 –mode training –input ./data/tp1_ej2_training.csv –layers 500 –lr 0.01 –epocs 3000 –save nn-ej2 –testProportion 0.1

Test Ejercicio 2:

python main.py –problem 2 –mode test –input ./data/tp1_ej2_test.csv –load nn-ej2

Para más información: *python main.py –help*

III. Resultados

A. Dataset 1 - Clasificacion

Para determinar el ajuste de la red y su performance en los datos de validacion utilizamos la metrica de area bajo la curva (AUC, por sus siglas en ingles). Dicha metrica permite notar la capacidad de la red de distinguir entre un caso negativo de un caso positivo. Entrenamos la red con 1 capa escondida de diferentes tamaños y siempre utilizando la funcion de activacion sigmoide, inclusive en el output. Debido a tener una funcion sigmoide en la ultima capa no podemos manejar outputs de valores que no esten entre 0 y 1, para eso tomamos 1 para los valores *Malignos* y 0 para los *Benignos*. El primer analisis que realizamos fue encontrar el correcto Learning Rate para una red chica (1 capa hidden de 10 nodos) y ver como dicho learning rate afecta al entrenamiento.

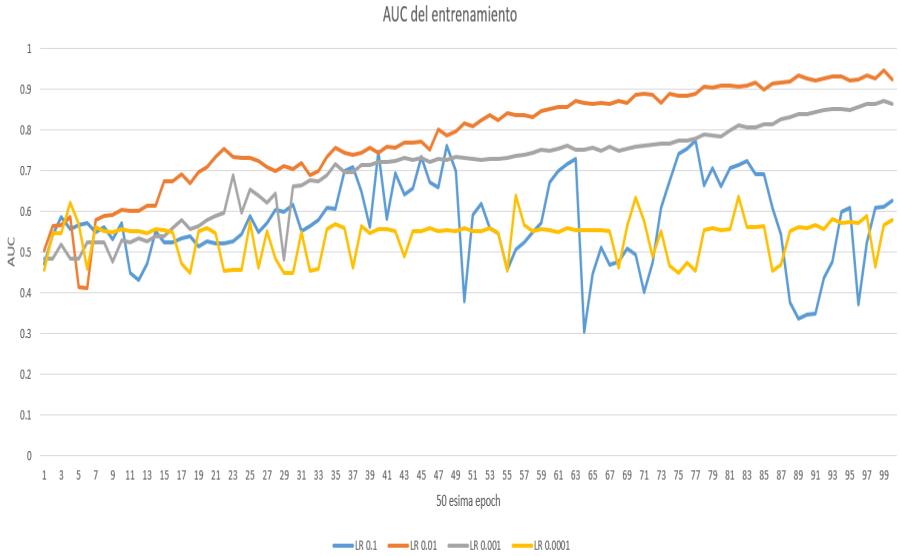

Figura 1. Curvas del AUC para una red de 10 nodos y LR: 0.1, 0.01, 0.001 y 0.0001

Podemos ver como learning rates muy chicos o muy grandes directamente no permiten aprender de los datos, sin embargo, encontrar los learning rates correctos logran un ajuste perfecto de los datos de entrenamiento.

Ahora queremos ver como con el learning rate optimo (0.01) y la red con una capa intermedia de 10 nodos se relacionan el AUC de test contra el AUC de train.

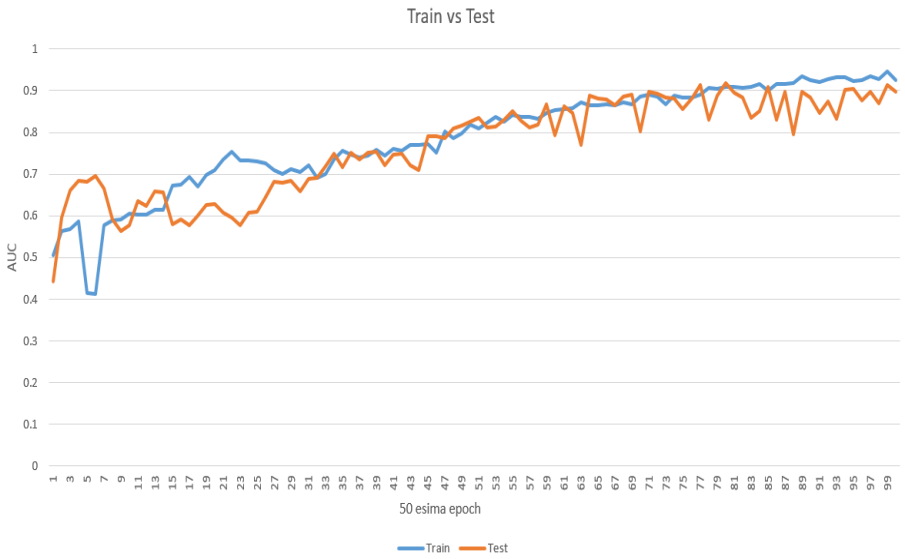

Figura 2. Curvas del AUC para una red de 10 nodos para set de test y train

Sorprendentemente hay una fuerte relacion entre el AUC de train y test lo cual nos confirma una muy buena generalizacion de la red.

B. Dataset 2 - Regression

Para determinar el ajuste de la red y su performance en los datos de validacion utilizamos el error cuadratico medio de la prediccion de la red para (calefaccion y refrigeracion) versus los verdaderos valores de la instancia. Graficamos el promedio de los 2 errores. Entrenamos la red con una pequeña modificacion a las variables de prediccion normalizando sus valores entre 0 y 1 para poder usar correctamente la funcion de activacion sigomide, luego al predecir desnormalizamos las predicciones para poder comparar las contra los verdaderos valores de la instancia. En el siguiente grafico presentamos como diferentes tamaño de la red cambian drasticamente la velocida y eficiencia del aprendizaje de la red.

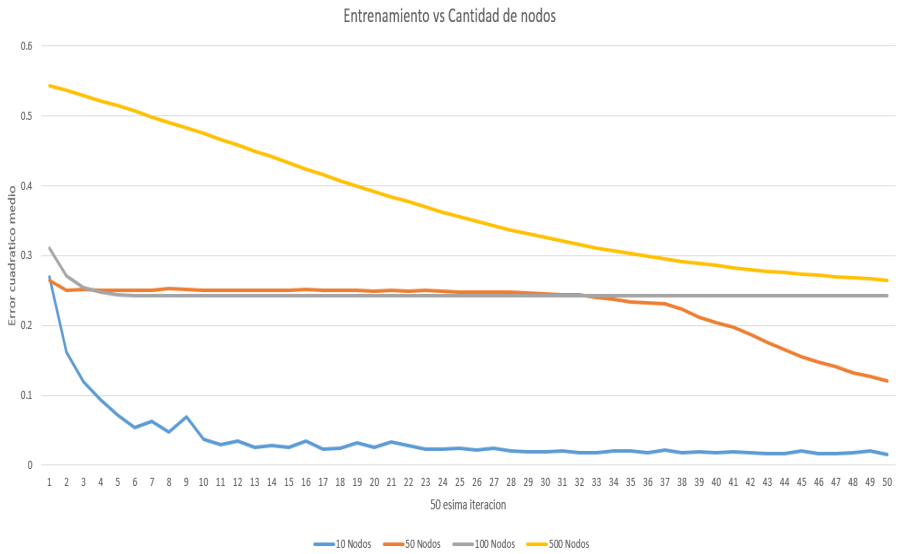

Figura 3. Curvas del error cuadratico medio sobre los datos de entrenamiento y con un LR = 0.01

Podemos notar cuan importante es la correcta eleccion de la arquitectura de la red. Vemos que con 10 nodos intermedios basta para lograr un excelente ajuste de los datos y al incrementar este valor encontramos mayores dificultades para lograr un buen ajuste. Es muy probable que esta arquitectura con una pequeña capa intermedia haya funcionado bien ya que la capa de input es realmente grande (30 nodos por columna original, osea, 240 nodos) y permite un gran grado de flexibilidad al momento de entrenar.

Luego de encontrar un optimo tamaño para la capa intermedia de la red observamos como diferentes valores de Learning Rate modifican el entrenamiento:

Figura 4. Curvas del error sobre los datos de entrenamiento y con una capa intermedia de 10 nodos

Nuevamente, aunque no tan abismal, podemos notar una importante diferencia al modificar el learning rate de la red. Por ultimo luego de encontrar los mejores parametros para la red en terminos de learnig rate y cantidad de nodos de la capa intermedia queremos estudiar las consequencias de agregar mas capas intermedias del mismo tamaño a la red:

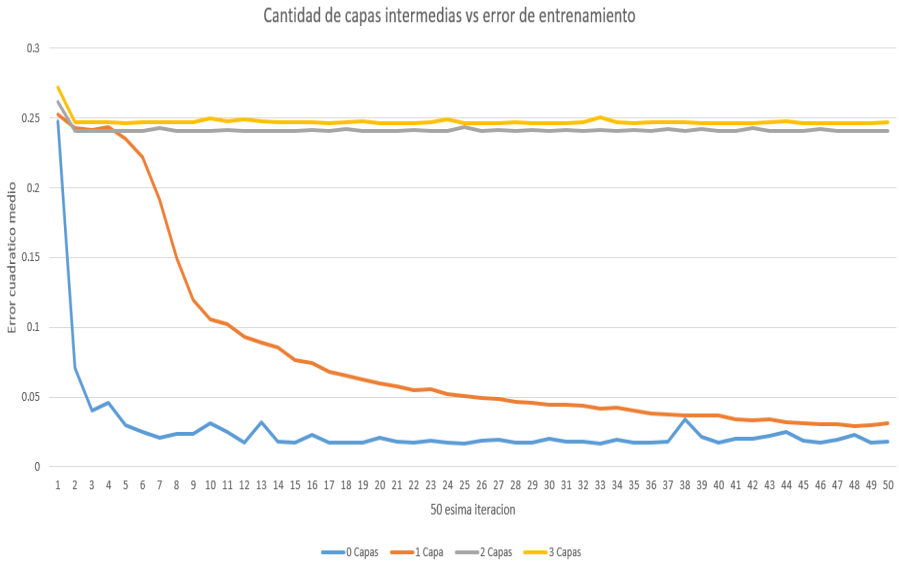

Figura 5. Curvas del error con LR = 0.001 y 10 nodos por capa intermedia

Sorprendentemente tener 0 capas intermedias permite ajustar de manera optima, creemos que esto se debe al valor de learning rate elegido. Por ultimo queremos denotar la performance de la red sobre datos de testeo, para esto separamos 20 % de los datos para testeo y el 80 % restante lo utilizamos para entrenar. Graficamos a continuacion las dos curvas de error cuadratico medio.

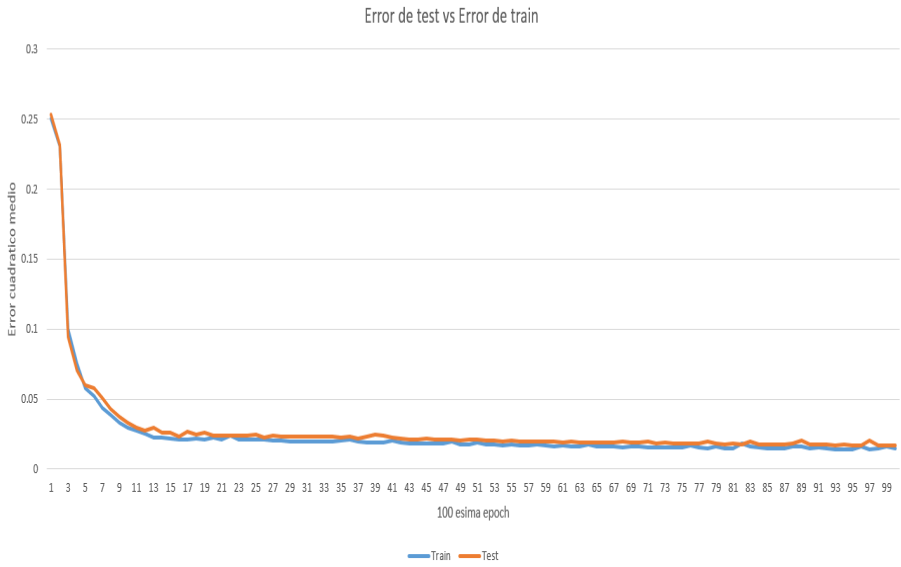

Figura 6. Curvas del error con LR = 0.001 y 10 nodos para la capa intermedia

Podemos ver una gran correlacion entre las curvas mostrando que no se esta sobreajustando a los datos.

IV. Conclusión

Como conclusion queremos destacar varios puntos:

- Codigo de la red - La programacion en si de cada una de las funciones basicas de la red fue relamente tediosa y pequeños errores eran muy dificiles de encontrar, como por ejemplo un error donde la ultima capa de la red no estaba siendo entrenada de manera correcta pero las primeras capas si, por ende, veiamos un entrenamiento correcto durante las primeras iteraciones (ya que las primeras capas lograban aprender algo) pero nunca lograbamos ajustar completamente al data set de entrenamiento a pesar de incrementar las cantidad de nodos de la red exponencialmente. Por otro lado, por el dinamismo que tienen las redes, un pequeño bug puede que no salga a la luz de forma inmediata ya que la red se puede adaptar para corregir la falla.
- Entrenamiento - Pudimos notar que el preprocesamiento de la entrada mejoraba mu-

chisimo la facilidad de entrenar la red (robustez) y lograba obtener mejores instancias de la misma en terminos de generalizacion. Sin embargo, encontramos que no era directo la decision del valor de learning rate y que algunos valores directamente no permitian un aprendizaje. Por suerte los datasets eran relativamente chicos y la adaptacion del learning rate se hacia con varias pruebas que no duraban mas que un par de minutos lo cual no seria asi si contasemos con datasets de varios ordenes de magnitud mayor.

- Proximos pasos- Como proximos pasos nos gustaria poder probar nuestra red sobre otros datasets, utilizar metodos mas avanzados de aprendizaje (como droupout o weight decay) y ademas transformar el codigo para optimizar la performance del entrenamiento, por ejemplo, transformando operaciones iterativas en matriciales.