Control Inteligente - Proyecto No. 1

Johnnatan N. Ruiz S. [1]
Universidad Nacional de Colombia
[1] jnruizs@unal.edu.co

***Abstract*—In this article it is showed the analysis of three models of neural networks to perform the prediction of the power demand of a subcentral in Bogota, based on the data of the month of November. The models that were realized were: One with an entrance, a hidden layer and an output, with two inputs, a hidden layer and an output, and with three inputs, a hidden layer and an output, respectively. For each of these networks, graphs of training error, generation error were made, a comparison were made between the real data series and the prediction of the neural network. To obtain these graphs it is used the Python programming language and its Tensorflow library. Additionally, a summary of the article "A review on artificial intelligence based load demand forecasting techniques for smart grid and buildings ", since it has a relevant importance with respect to the problem of prediction of the load of the subcentral.**

***Index Terms*—Neural networks, power demand, Python, Tensorflow**

***Resumen*—En este artículo se muestra el análisis de tres modelos de redes neuronales para realizar la predicción de la demanda de potencia de una subcentral en Bogotá, con base en los datos del mes de noviembre.Los modelos que se realizaron fueron: Uno con una entrada, una capa oculta y una salida, con dos entradas, una capa oculta y una salida, y con tres entradas, una capa oculta y una salida, respectivamente. Para cada una de estas redes, se realizaron gráficas del error de entrenamiento, error de generación, se realizó una comparación entre la serie de datos reales y la predicción de la red neuronal. Para obtener estás gráficas se utilizó el lenguaje de programación de Python y su librería de Tensorflow. Adicionalmente, se realizó un resumen del artículo "A review on artificial intelligence based load demand forecasting techniques for smart grid and buildings", dado que tiene una importancia relevante con respecto al problema planteado de predicción de la carga de la subcentral.**

Palabras Clave— Redes neuronales, demanda de potencia, Python, Tensorflow

I. Evolución del error con respecto a toda la tabla

En principio se construyo la tabla de los datos de la subcentral y se normalizaron con respecto a 30, en una tabla de Excel. Estos datos se introdujeron en el programa creado para la red neuronal 1-N-1. A continuación de esto, se realizó una variación de la tasa de aprendizaje y del número de neuronas, para todas las gráficas mostradas en este artículo se dejaron los pesos siempre inicializados en 0.1, para realizar las comparaciones respectivas.

Después de construir las gráficas, se observó que cuanto se tenía una tasa de 0.2, la cantidad de neuronas para llegar a un valor razonable del error, era de 13 neuronas, mientras que para las tasas de 0.1 y 0.01, la cantidad de neuronas superaban las 40 neuronas, lo que quiere decir, que se necesita un mayor tiempo de procesamiento, para llegar a un error de

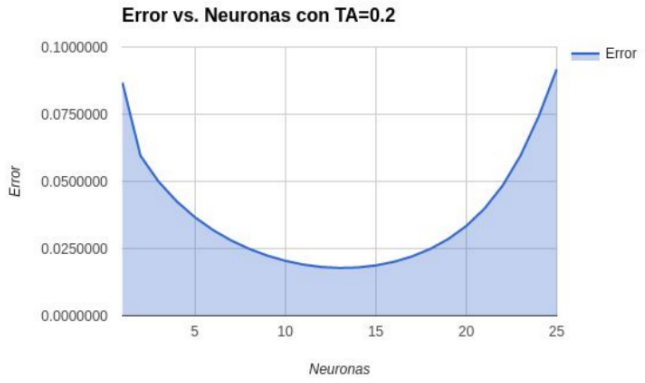

Figura 1. Error vs. Cantidad de Neuronas para una tasa de 0.2

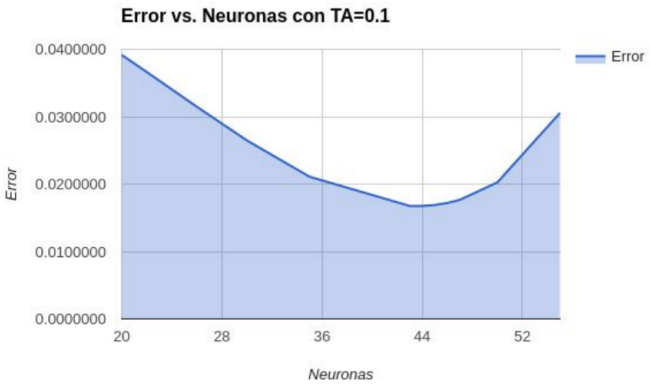

Figura 2. Error vs. Cantidad de Neuronas para una tasa de 0.1

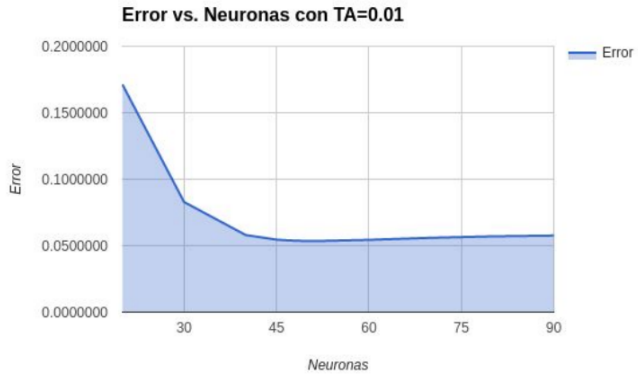

Figura 3. Error vs. Cantidad de Neuronas para una tasa de 0.01

magnitud similar. Sin embargo, también se pudo observar que para la tasa de 0.01, el error se mantenía estable después de una cantidad de neuronas determinadas. El mínimo error que se obtuvo fue de 0.0167560 para la tasa de aprendizaje de 0.1.

II. Evolución del error de entrenamiento con respecto a la cantidad de neuronas

Después de este análisis, se realizó la gráfica de la evolución del error para un entrenamiento de la mitad de la tabla caracterizada. De aquí en adelante se realizaron 6 iteraciones a esta tabla, para poder realizar las respectivas comparaciones.

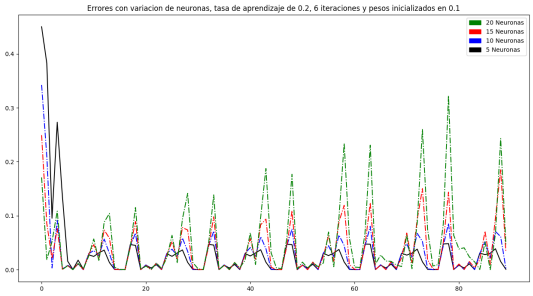

Figura 4. Evolución del error de entrenamiento vs. iteración para una tasa de 0.2

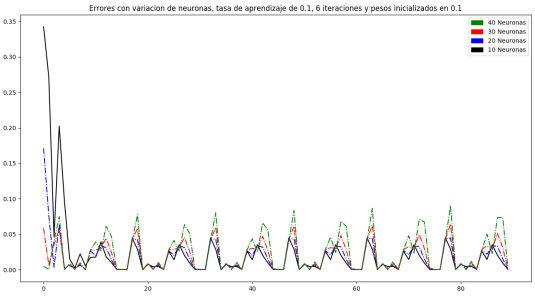

Figura 5. Evolución del error de entrenamiento vs. iteración para una tasa de 0.1

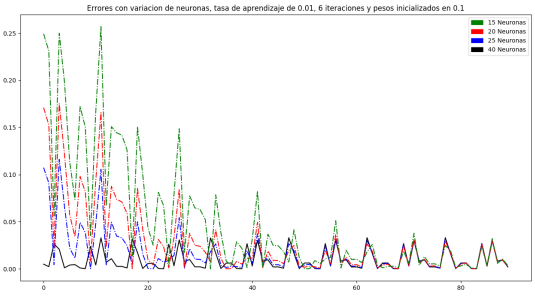

Figura 6. Evolución del error de entrenamiento vs. iteración para una tasa de 0.01

Con una tasa de aprendizaje de 0.01 requiere menos iteraciones para hacer que el error no dependa del numero de neuronas, es decir, con la misma cantidad de iteraciones, se necesita un menor procesamiento, y el error, no tiene variaciones significativas. Cuando se observa la gráfica con tasa de aprendizaje de 0.2, se muestra evidentemente que al sobrepasar un cierto límite de neuronas, el sistema de la red neuronal tiene a un valor infinito. Y por último, cuando se observa la gráfica de la tasa de aprendizaje con 0.1, tiene un comportamiento similar al de la tasa de aprendizaje de 0.01, pero se puede observar que varía más en magnitud.

III. Evolución del error de generación con respecto a la cantidad de neuronas

Descartando la tasa de aprendizaje de 0.2, por la conclusión de la sección anterior, se crearon las curvas para la evolución del error de generación tomando los datos complementarios de la tabla.

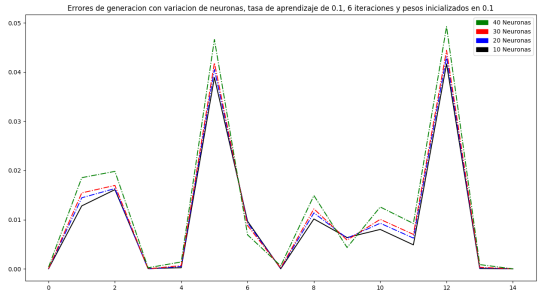

Figura 7. Evolución del error de generación vs. iteración para una tasa de 0.1

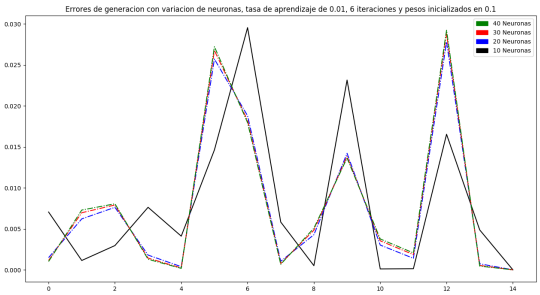

Figura 8. Evolución del error de generación vs. iteración para una tasa de 0.01

Cuando se analizan las gráficas, el máximo local que se obtiene en la curva con una tasa de aprendizaje de 0.01, es mucho menor que el máximo local de la tasa de aprendizaje 0.1. Sin embargo, cuando se tienen pocas neuronas con una tasa de aprendizaje lenta, como lo es 0.01, no se tiene un buen seguimiento del error, en el ejemplo, se muestra que con 10 neuronas el error se desfasa de las demás curvas.

IV. Evolución del error de entrenamiento con respecto a la cantidad de entradas

Teniendo en cuenta todas las conclusiones anteriores, para hacer la red neuronal de dos y tres entradas, con una capa oculta y una salida (2-N-1 y 3-N-1), se usará una tasa de aprendizaje de 0.01, y el número de neuronas será 20, ya que con este número se tiene un poco tiempo de procesamiento y un nivel considerado de neuronas.

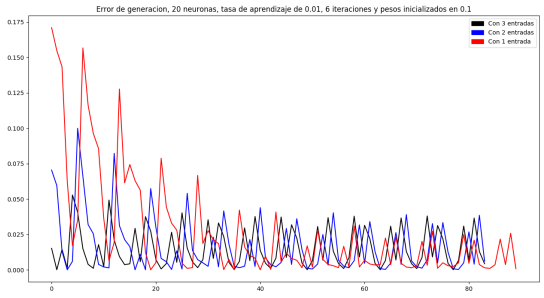

Figura 9. Evolución del error de entrenamiento vs. iteración para una tasa de 0.01 y 20 neuronas en la red.

En las curvas de la figura 9, se notó claramente que la mejor evolución del error se da para cuando se tienen 3 entradas. Esto quiere decir, que entre más entradas tenga la red neuronal, más robusta va a ser.

V. Evolución del error de generación con respecto a la cantidad de entradas

Se continuó con la evolución del error de generación de los tres tipos de modelo creados.

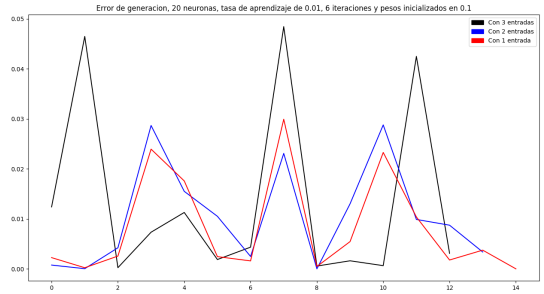

Figura 10. Evolución del error de generación vs. iteración para una tasa de 0.01 y 20 neuronas en la red.

Aquí se observa claramente que aunque el mejor de entrenamiento es para cuando se tienen 3 entradas, se obtiene la peor evolución para los errores de generación. Por tanto, se requeriría una mayor cantidad de iteraciones a la tabla, o una mayor cantidad de datos a entrenar. Sin embargo, el error de entrenamiento, no es mayor que el 0.05, lo que significa que es una buena respuesta del error, para el ejercicio dado.

VI. Comparación entre la serie real y la predicción de la red neuronal

Ya dejando claro, prácticamente todo constante, es decir, pesos inicializados en 0.1, 6 iteraciones a la tabla, 20 neuronas en la red y tasa de aprendizaje en 0.01, se realizó la respectiva gráfica 12 para comparar la serie real y la predicción de la red neuronal.

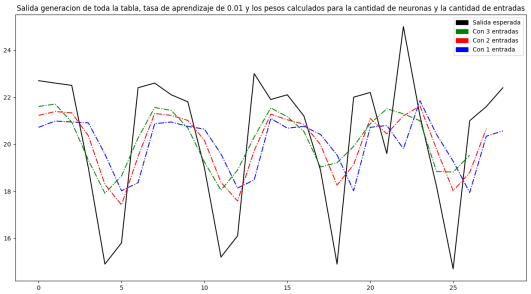

Figura 11. Evolución del error de generación vs. iteración para una tasa de 0.01 y 20 neuronas en la red.

Como previsto, entre más entradas se tengan en la red neuronal, probablemente, más robusto sea el sistema. En las curvas se observa que cuando se tienen 3 entradas, se asemeja más a la curva real, sin embargo, aun se encuentra un poco alejada de la realidad, por tanto se propone encontrar otra función de optimización o poner otra capa oculta, que aproveche la respuesta periódica del error.

VII. Código utilizado

Para obtener todas estas gráficas se usaron muchos versiones del mismo programa, sin embargo, los algoritmos eran iguales, por tanto, únicamente se mostrarán los de la última versión, para cuando se tenían 3 entradas.

```
import numpy as np
import tensorflow as tf

'''Numero de iteraciones a la tabla'''
iteraciones = 6

'''Numero de neuronas de la red'''
neuronas = 20

'''Numero de entradas totales'''
entradas = 3

'''Numero de entradas mostradas'''
entM = 3

'''Numero de salidas totales'''
salidas = 1

'''Tasa de aprendizaje'''
ta = 0.01

'''Numero de datos de la tabla de entrenamiento'''
ndatosE = 14

'''Numero de datos de la tabla de generacion'''
ndatosG = 13


W = tf.Variable(0.1*tf.ones([neuronas,1+entradas]),'float')
C = tf.Variable(0.1*tf.ones([neuronas,salidas]),'float')
eT = tf.Variable(tf.zeros([ndatosG]),'float')
eTT = tf.Variable(tf.zeros([ndatosE*iteraciones]),'float')
yF = tf.Variable(tf.zeros([ndatosG]),'float')

x = tf.placeholder(tf.float32,[ndatosE,1+entradas])
yd = tf.placeholder(tf.float32,[ndatosE,salidas])
xg = tf.placeholder(tf.float32,[ndatosG,1+entradas])
ydg = tf.placeholder(tf.float32,[ndatosG,salidas])

for i in range(iteraciones):
        for m in range(ndatosE):
                y = 0
                for n in range(neuronas):
                        s = 0
                        for p in range(entM+1):
                                s = s+W[n][p]*x[m][p]
                        h = tf.tanh(s)
                        y = y + h*C[n]
                        e = yd[m]-y
                        ad = 2*ta*C[n]*e*(1-y*y)

                eTT = eTT + tf.one_hot(ndatosE*i+m,ndatosE*iteraciones,e[0]*e[0])

                W = W + ad*x[m]
                C = C + 2*ta*e*y

for m in range(ndatosG):
                y = 0
                for n in range(neuronas):
                        s = 0
                        for p in range(entM+1):
                                s = s+W[n][p]*xg[m][p]
                        h = tf.tanh(s)
                        y = y + h*C[n]
                        e = ydg[m]-y
                        ad = 2*ta*C[n]*e*(1-y*y)
                eT = eT + tf.one_hot(m,ndatosG,e[0]*e[0])
                yF = yF + tf.one_hot(m,ndatosG,y[0])

xt =    [[1,    0.7533  ,       0.7567  ,       0.7333  ],[1,   0.7500  ,       0.7533  ,       0.7567  ],
        [1,     0.6333  ,       0.7500  ,       0.7533  ],[1,   0.4967  ,       0.6333  ,       0.7500  ],
        [1,     0.5267  ,       0.4967  ,       0.6333  ],[1,   0.7467  ,       0.5267  ,       0.4967  ],
        [1,     0.7533  ,       0.7467  ,       0.5267  ],[1,   0.7367  ,       0.7533  ,       0.7467  ],
        [1,     0.7267  ,       0.7367  ,       0.7533  ],[1,   0.6333  ,       0.7267  ,       0.7367  ],
        [1,     0.5067  ,       0.6333  ,       0.7267  ],[1,   0.5367  ,       0.5067  ,       0.6333  ],
        [1,     0.7667  ,       0.5367  ,       0.5067  ],[1,   0.7300  ,       0.7667  ,       0.5367  ]]

ydt =   [[      0.7500  ],[     0.6333  ],[     0.4967  ],[     0.5267  ],[     0.7467  ],[     0.7533  ],
        [       0.7367  ],[     0.7267  ],[     0.6333  ],[     0.5067  ],[     0.5367  ],[     0.7667  ],
        [       0.7300  ],[     0.7367  ]]

xgg =   [[1,    0.7367  ,       0.7300  ,       0.7667  ],[1,   0.7067  ,       0.7367  ,       0.7300  ],
        [1,     0.6300  ,       0.7067  ,       0.7367  ],[1,   0.4967  ,       0.6300  ,       0.7067  ],
        [1,     0.7333  ,       0.4967  ,       0.6300  ],[1,   0.7400  ,       0.7333  ,       0.4967  ],
        [1,     0.6533  ,       0.7400  ,       0.7333  ],[1,   0.8333  ,       0.6533  ,       0.7400  ],
        [1,     0.7067  ,       0.8333  ,       0.6533  ],[1,   0.6067  ,       0.7067  ,       0.8333  ],
        [1,     0.4900  ,       0.6067  ,       0.7067  ],[1,   0.7000  ,       0.4900  ,       0.6067  ],
        [1,     0.7200  ,       0.7000  ,       0.4900  ]]

ygg =   [[      0.6067  ],[     0.4900  ],[     0.7000  ],[     0.7200  ],[     0.7467  ],[     0.7067  ],
        [       0.6300  ],[     0.4967  ],[     0.7333  ],[     0.7400  ],[     0.6533  ],[     0.8333  ],
        [       0.7067  ]]

eT=tf.expand_dims(eT,0)
eTT=tf.expand_dims(eTT,0)

init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)

cW, cC, ce, ceTT, cYF  = sess.run([W,C,eT,eTT,yF], {x:xt, yd:ydt, ydg:ygg, xg:xgg})
print("\nW: %s\nC: %s\ne: %s\ny: %s\ny\neTT: %s"%(cW, cC, ce, cYF, ceTT))

```

Figura 12. Código utilizado para obtener los diferentes vectores y matrices de los pesos, errores y salidas.

VIII. Resumen artículo

En pocas palabras el artículo describe cada uno de los métodos de técnicas de pronóstico de la demanda de energía para las redes inteligentes y para edificios. Técnicas como ” Algoritmo genético con regresión vectorial de soporte”, ” Red Neuronal Artificial con genética y modelo Propagación inversa”, ” Neuronas difusasz ” Algoritmo genético caótico simulado recocido con soporte a regresión de vectores”. Describe algunas ventajas y desventajas de los respectivos métodos y adicionalmente, describe con detalle todos los parámetros que son necesarios para implementarlas.

Adicionalmente, en el artículo, se referencia la red de ISO en New England, y se muestra una serie de datos entre 2004 y 2009, en horas. Luego menciona todos las variables o parámetros de entrada que se necesitan para implementar el pronóstico de la demanda de energía en esta central. Los datos más importantes son los punteros del día y hora, saber si el día es laboral o no laboral, el punto de rocío, el bulbo seco y los datos de carga co-relacionados. Estás entradas se deben escoger correctamente, ya que de esto depende de que el sistema a implementar, funcione.

En la actualidad hay diferentes tipos de pronóstico, como el pronóstico a largo plazo, este es muy útil para la planificación de acuerdo con la demanda energética y la energía política del Estado, se pronostica entre 1 año y 10 años en adelante. También esta el pronóstico a medio plazo, se utiliza para saber el funcionamiento eficiente y el mantenimiento del sistema de energía, va entre 1 mes a un año. Y por último se tiene el pronóstico a corto plazo, este se utiliza para conocer control de la reserva de hilatura y la evaluación de los contratos de compraventa entre varias empresas.

Una de las ventajas del pronóstico a corto plazo es asegurar el sistema de seguridad. La predicción exacta de la carga es una herramienta determinar el estado operativo óptimo del sistema de potencia. Además, esta información también se puede utilizar para preparar la energía sistema de acuerdo con el estado de carga futuro y aplicar unas las acciones correctivas.

En conclusión el artículo muestra que es muy importante saber la predicción del valor de la carga en las centrales de energía, ya que por ejemplo, se podría saber con anticipación la programación, mantenimiento, ajustes de los tipos arancelarios y la evaluación de los contratos que se puede llevar a cabo por pronóstico de carga precisa. Con una planificación eficaz de los sistemas eléctricos se puede ahorrar millones de dólares, lo que juega un papel importante en el crecimiento económico de un país.

Referencias

[1] Todas las figuras en este artículo fueron creadas por Johnnatan Ruiz.

[2] La información y fórmulas utilizadas se extrajeron de las diapositivas de la clase Control Inteligente. Universidad Nacional de Colombia. Alberto Delgado.

[3] Raza, M. Q. and Khosravi, A.: A Review on Artificial Intelligence Based Load DemandForecasting Techniques for Smart Grid and Buildings. Renewable and Sustainable Energy Reviews, Vol. 50, pp. 1352 – 1372, Junio 2015.