Korean J. Math. **28** (2020), No. 4, pp. 943–953
http://dx.doi.org/10.11568/kjm.2020.28.4.943

SCORE NORMALIZATION FOR A UNIVERSITY GRADES INPUT SYSTEM USING A NEURAL NETWORK

Young Ho Park

Abstract. A university grades input system requires for professors to enter the normalized total scores for the letter grades and to input the scores from six fields such as Midterm, Final, Quiz which sum up to the total. All six fields have specified bounds which add up to 100. Professors should scale in the total scores to match up the letter grades and scale in every field of each student's original scores within the bounds to sum up to the scaled total score. We solve this problem by a novel design of simple shallow neural network.

1. Problem

Suppose a professor has a final score sheet of a university class, part of which is shown at Figure 1. Each row shows scores from six fields labeled 0 to 5 as Midterm(15), Final(15), Quiz(30), Homework(20), Attendance(15), Others(5) of a student. Here, the numbers in parenthesis are the maximum possible scores which sum up to 100. Let N be the number of students. Then raw scores table is an array of size $N \times 6$. The letter grade in field 7 is manually determined by the professor based on the total score entered in field 6. Actually, the cuts for letter grades in this example were given by

$$C = [83, 75, 65, 58, 55, 47, 40, 33].$$

Received November 11, 2020. Revised December 23, 2020. Accepted December 24, 2020.

2010 Mathematics Subject Classification: 68T07.

Key words and phrases: Neural networks, Regression.

	0	1	2	3	4	5	6	7
0	5.6	4.75	15.90	20	15	5	66.25	B+
1	8.2	7.20	23.25	18	15	4	75.65	A0
2	0.0	0.00	0.00	8	9	2	19.00	F
3	3.0	4.25	6.25	14	15	5	47.50	C0
4	3.6	0.50	9.05	18	15	3	49.15	C0

FIGURE 1. Score sheet

For example, if the total score T of a student is such that $75 \leq T < 83$, then the student will get the grade A0.

The university grade input system requires professors to enter *normalized* total scores for the letter grades of students:

A+	A0	B+	B0	C+	C0	D+	D0	F
100∼95	94∼90	89∼85	84∼80	79∼75	74∼70	69∼65	64∼60	59∼0

Then the letter grades will be determined automatically by the system. Furthermore, it also requires to input every score *in integer*. Hence the professor should scale in the total scores to match up the letter grades and scale in every field of each student's scores to sum up the scaled total score. However, it is quite reasonable to fix the Attendance score (column 4), since the absence of more than 1/3 classes causes a failure of the course in most universities. A lot of adjustments must be made to match up the total scores for each students. This is quite a lot of works and time-consuming. Let us look at the row 1. For this student, the professor need to change total score 75.65 to some integer T in $[90, 94)$ and adjust six field scores to sum up to T. A possible example of such adjustments would look like Figure 2.

First, we interpolate total scores in a piece-wise linear manner as in the next picture. More precisely, let $\mathtt{C} = [100, c_1, \cdots, c_8, 0]$ be the cut of the original total scores where 100 and 0 are added to the original cut C. Each point x in each subinterval $[\mathtt{C}_{i+1}, \mathtt{C}_i]$ will be mapped onto a point in the corresponding subinterval $[\mathtt{D}_{i+1}, \mathtt{D}_i]$ of the normal cut $\mathtt{D} =$

	0	1	2	3	4	5	6	7
0	15	7	23	20	15	5	85	B+
1	14	10	27	20	15	4	90	A0
2	1	3	7	13	9	1	34	F
3	12	7	13	18	15	5	70	C0
4	14	3	16	20	15	3	71	C0

FIGURE 2. An example of adjusted scores

$[100, 95, 90, \cdots, 65, 60, 0]$, so that x is mapped onto the *integer part* of

$$\mathtt{D}_{i+1} + \frac{\mathtt{D}_i - \mathtt{D}_{i+1}}{\mathtt{C}_i - \mathtt{C}_{i+1}}(x - \mathtt{C}_{i+1}).$$

Column 6 in Figure 2 shows the results of the normalization of total scores T_i for $i = 0, \cdots, 4$. Now we need to adjust the values of six fields in each row so that the sum of them is equal to the normalized total score.

It is clear that there are many ways or algorithms to do these adjustments. We will adjust each field basically by a linear function. To preserve the order of scores, we require the slope to be positive. The actual adjustment will be done using a neural network. To summarize the conditions we require

1. the Attendance score is fixed.
2. the transformation F of each field is order-preserving:

$$x_1 < x_2 \implies F(x_1) \leq F(x_2).$$

2. Our neural network

Our problem is a kind of regression (fitting) problem. We are going to use a shallow neural network to solve it. Our network is a very simple network as shown in the diagram below, but it contains new fundamental ideas which have never used before. It consists of input layer, a hidden layer and the output layer. We will basically follow the idea of gradient descent algorithm but the direction we use in the iterations is not the exact direction of gradients. Our data consists of six scores as in Figure 1 for each of 32 students. For the generality of neural networks, there

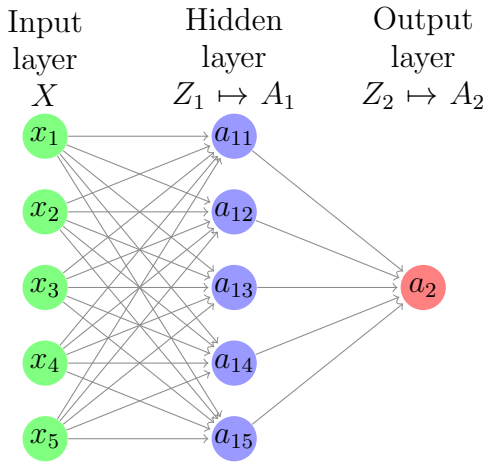

FIGURE 3. our network

are so many references. See [2] for example or Andrew Ng's Machine Learning Youtube videos [4].

The input data X consists of 5-dimensional vectors of scores from all fields but Attendance. It is an array of size $5 \times N$, where N is the number of samples. Using the numpy operation symbols and the vectorization, the forward propagation is given as follows :

$$Z_1 = W_1 @ X + b_1, \qquad A_1 = M * \sigma(Z_1/M)$$
$$Z_2 = W_2 @ A_1, \qquad A_2 = Z_2$$

Here W_1 is a 5×5 matrix, b_1 is a 5×1 array, $M = (15\ 15\ 30\ 20\ 5)^T$ is the 5×1 array of maximums of five fields, $\sigma(z)$ is the sigmoid function defined by $\sigma(z) = 1/(1 + e^{-z})$ and $W_2 = (1\ 1\ 1\ 1\ 1)$, all-one vector.

For a single input vector $X = (x_1\ x_2\ x_3\ x_4\ x_5)^T$, the output from the hidden layer is $A_1 = (a_{11}\ a_{12}\ a_{13}\ a_{14}\ a_{15})^T$, where a_{1i} satisfy

$$0 \leq a_{1i} \leq M_i$$

for all $1 \leq i \leq 5$. Here M_i is the upper bound of values from field i. It is a new idea to use the sigmoid function in this way to guarantee this bound condition. The array A_1 represents the adjusted scores in five fields of a student. Note also that we scaled the domain of the sigmoid to distribute its values more evenly on the domain $[-M_i, M_i]$. Compare it with $M_i * \sigma(z)$ without the scaling as shown in the Figure 4.

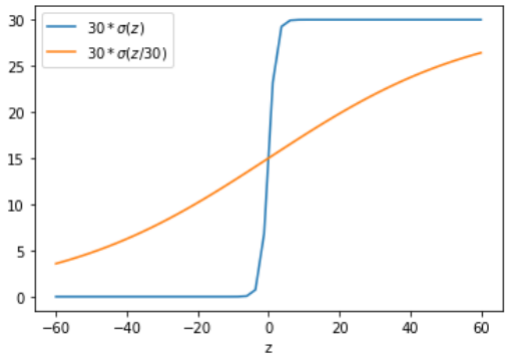

FIGURE 4. $M \cdot \sigma(z)$ versus $M \cdot \sigma(z/M)$ with $M = 30$

The network's output is $A_2 = a_{11} + a_{12} + a_{13} + a_{14} + a_{15}$, which represents the adjusted total score at the iteration stage.

Remember that the target array Y of the network is the difference between the adjusted total score (column 6 in Figure 2) and the Attendance score (column 4 in Figure 2).

The cost function of the network is given by

$$J = \frac{1}{2N} \sum (A_2 - Y)^2$$

Let dQ denote the Jacobian $\frac{\partial J}{\partial Q}$ for any Q. Then the derivatives in the back propagation are given in the vectorized form as follows (see [6]):

$$\begin{aligned} dZ_2 &= (A_2 - Y)/N \\ dA_1 &= W_2^T @ dZ_2 \\ dZ_1 &= \sigma'(Z_1/M) * dA_1 \\ dW_1 &= dZ_1 @ X^T \\ db_1 &= \texttt{numpy.sum}(dZ_1, \texttt{axis=1}, \texttt{keepdims=True}) \end{aligned}$$

Usually, the updating parameters W_1, b_1 is done by

$$\begin{aligned} W_1 &\leftarrow W_1 - \alpha \cdot dW_1 \\ b_1 &\leftarrow b_1 - \alpha \cdot db_1 \end{aligned}$$

where α is the learning rate. However, we modify this update rule in a new way as

$$W_1 \leftarrow W_1 - \alpha \cdot dW_1 * I_5 \tag{1}$$

$$b_1 \leftarrow b_1 - \alpha \cdot db_1 \tag{2}$$

Here $*$ denotes the pointwise multiplication as usual in numpy. Note that our network design is different from the conventional network. Since the weight matrices must stay in the diagonal form, we can not apply the usual gradient descent algorithm, which results in non-diagonal weight matrices. This is why we only take into account of diagonal components of derivative dW_1 in Equation (1). So the direction of the convergence is little bit off the direction of gradients. It turns out that this kind of modified iteration still works. We start with $W_1 = I_5$. Then this iteration will always yield diagonal

$$W_1 = \text{diag}(w_1, \cdots, w_5).$$

It seems that $w_i > 0$ for all i in practice with initial $W_1 = I_5$. We make sure that this condition holds for the final W_1. For the final output, it took less than 7 seconds to get the cost less than 1 after 80000 iterations. See Figure 5 for the cost function. Because we need more adjustments for the network's output as explained below, costs less than 4 would work as well.

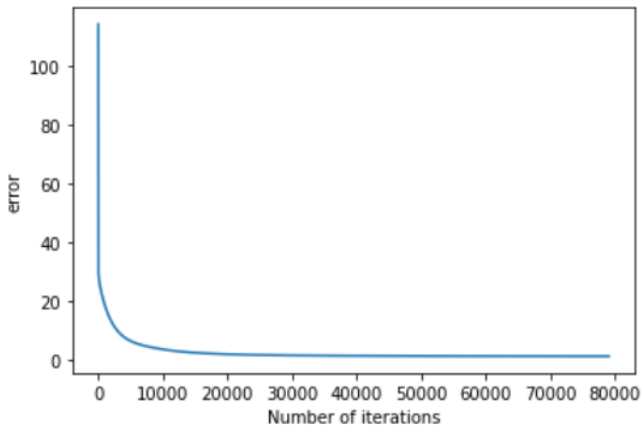

FIGURE 5. Plot of cost function in iteration

	0	1	2	3	4	5	6	7	8
0	15	7	23	20	15	5	85	B+	85
1	15	10	27	20	15	4	91	A0	90
2	2	3	7	13	9	1	35	F	34
3	13	7	13	18	15	5	71	C0	70
4	14	3	16	20	15	3	71	C0	71

FIGURE 6. Network's output before final touch

The final parameters we obtained are

(3)
$$W_1 = \texttt{diag}(20.38345244, 4.94426245, 4.28029794, 6.20767928, 7.11592338)$$

(4)
$$b_1 = (-30.85224534, -23.61155618, -33.72601566, -37.9907464, \\ -20.03404771)^T$$

Since $a_{1i} = w_i x_i + b_{1i}$ with positive w_i, the order-preserving condition will be satisfied.

Recall that a_{1i} are supposed adjustments of field values, and hence we have to take the integral parts of them at the final. It contribute to more error in the summations $\sum_i a_{1i}$ with target values. The modified parameters updates (following diagonal directions only) might get in the way to the targets, too. In short, we need more adjustment for the network's output to match up the targets as the Figure 6 shows.

At rows 1,2,3, the rounded network's output sums (column 6) do not match up with the actual totals (written in column 8). To make it correct, we add to or subtract from each of five fields (columns 0 to 4) in turn, if in bounds $0 \le a_{1i} \le M_i$, until the five fields sum up to the total. This completes the modification we want as given in Figure 2. Note that this final touch might cause to break the order-preserving property.

The actual Python code in Jupyter notebook with the example data can be downloaded from [5]. The coding is done in Python from the scratch only using numpy and pandas. For an introductory book of Python, we recommend [3]. For a quick browse, we include the coding for our neural network in compact form with no comments.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.special import expit as sigmoid

def d_sigmoid(x):
    return sigmoid(x)*(1-sigmoid(x))

### Preparation of Dataframe
filename = 'alg_data.xlsx'
mycut = [83,75,65,58,55,47,40,33]
df = pd.read_excel(filename, header=1,encoding = 'cp949')
N = df.shape[0]
N_cols = df.shape[1]
orig_col_names = [col for col in df.columns]
new_col_names = list(range(len(orig_col_names)))
df.columns = new_col_names
Mi = np.array([15,15,30,20,5]).reshape(-1,1)
df[6] = sum(df[i] for i in range(6))
cut = [max(df[6])+1] + mycut + [0]
NORM_CUT = [100,95,90,85,80,75,70,65,60,0]
GRADES = ['A+','A0','B+','B0','C+','C0','D+','D0','F']
def grade(x, cut):
    for i in range(9):
        if x >= cut[i+1]:
            return GRADES[i]
for i in range(N):
    df.at[i,7] = grade(df.at[i,6], cut)
def normalize_total(x):
    for i in range(9):
        if cut[i+1]<= x <cut[i]:
            slope = (NORM_CUT[i]-NORM_CUT[i+1])
            /(cut[i]-cut[i+1])
            res = NORM_CUT[i+1] + slope*(x-cut[i+1])
            return int(res)
for i in range(N):
    df.at[i,8] = normalize_total(df.at[i,6])
df[8] = df[8].astype(int)
Xw = np.array(df[list(range(6))]).T
Yw = np.array(df[8]).reshape(1,-1).astype(float)
```

```
X = Xw[[0,1,2,3,5]]
Y = Yw - Xw[4]
dg = df.copy()

### Neural Network

W2 = np.ones((1,5))
def initialize():
    W1 = np.eye(5)
    b1 = np.zeros((5,1))
    params = {"W1":W1, "b1":b1}
    return params
def forward_propagation(X, params):
    W1 = params["W1"]
    b1 = params["b1"]
    Z1 = W1 @ X + b1
    A1 = Mi * sigmoid(Z1/Mi)
    Z2 = W2 @ A1
    A2 = Z2
    return A2, A1, Z1
def mse_loss(AL, Y):
    loss = np.mean( (AL-Y)**2 / 2)
    loss = np.squeeze(loss)
    return loss
def backward_propagation(X, Y, params):
    A2, A1, Z1 = forward_propagation(X,params)
    W1 = params["W1"]
    b1 = params["b1"]
    N = X.shape[1]  # number of data samples in X
    dZ2 = (A2 - Y)/N
    dA1 = W2.T @ dZ2
    dZ1 = d_sigmoid(Z1/Mi) * dA1
    dW1 = (dZ1 @ X.T)
    db1 = np.sum(dZ1, axis=1, keepdims=True)
    grads={"dW1":dW1,"db1":db1}
    return grads
def update_params(params, grads, lr):
    params["W1"] = params["W1"] - lr * grads["dW1"] * np.eye(5)
    params["b1"] = params["b1"] - lr * grads["db1"]
    return params
def neural_net(X, Y, params, nitr, lr):
```

```
    cost_history = []
    for i in range(nitr):
        A2, _, _ = forward_propagation(X, params)
        cost = mse_loss(A2, Y)
        cost_history.append(cost)
        grads = backward_propagation(X, Y, params)
        params = update_params(params, grads, lr)
        if i % (nitr // 10) == 0:
            print(f'After {i} iterations,
            Cost: {np.round_(cost_history[-1],5)}')
            #rint(params['W1'])
        if cost < 1:
            break
        return params, cost_history

### Iteration
params = initialize()
params, cost_history = neural_net(X,Y,params,100001,0.01)
plt.plot(cost_history)
print('final cost :', cost_history[-1])
print(params['W1'])
print(params['b1'])
A2, A1, Z1 = forward_propagation(X, params)
for j in range(5):
    dg[j] = (np.round_(A1.T)[:,j]).astype(int)
df[9] = np.round_(A2.reshape(-1,1)).astype(int)
dg[5] = df[4]
dg[6] = sum(dg[j] for j in range(6))
dg[5] = df[4]
dg[6] = sum(dg[j] for j in range(6))

### Final touch
for i in range(N):
    diff = dg[8][i] - dg[6][i]
    j = 0
    while diff != 0 :
        if 0 <= dg.at[i,j] + np.sign(diff) <= np.squeeze(Mi[j]):
        dg.at[i,j] += np.sign(diff)
        diff = diff - np.sign(diff)
    j = (j + 1) % 5
dg[6] = sum(dg[j] for j in range(6))
```

References

[1] M. Hagan, H. Demuth, M. Beale and O. Jesús, *Neural network design*, 2nd edition, ebook, https://hagan.okstate.edu/NNDesign.pdf
[2] S. Marsland, *Machine learning, an algorithmic perspective*, second edition, CRC press, 2015
[3] E. Matthes, *Python crash course*, no starch press, 2016
[4] A. Ng, *Machine learning lectures*, Youtube channel Artificial Intelligence - All in One, 2016
[5] Y.H. Park, *Jupyter notebook with a sample data for this article*, https://deepmath.kangwon.ac.kr/∼yhpark/pub/grading.zip, 2020
[6] Y.H. Park, *Derivatives in neural networks*, unpublished note, https://deepmath.kangwon.ac.kr/∼yhpark/pub/derivativesinNN.pdf, 2019

Young Ho Park
Department of Mathematics, Kangwon National University
Chuncheon 24341, Korea
E-mail: yhpark@kangwon.ac.kr