ESIEE Paris

Deep Learning

Project

Designing a Deep Neural Network predictor with SGD Algorithm

Sahar Hosseini

January 2020

Libraries

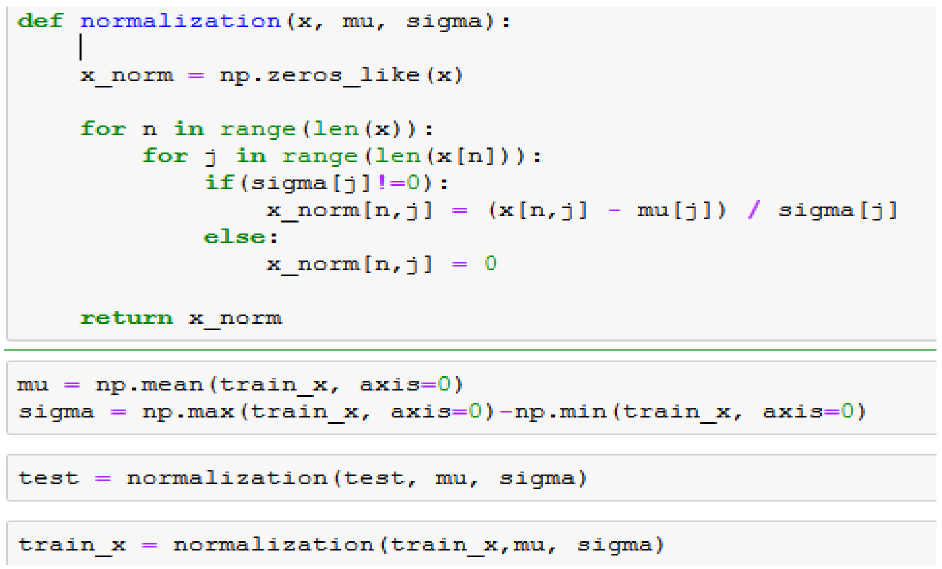
Python libraries: Matplotlib, panda, numpy

Dataset

The dataset of Digit Recognizer Competition is the famous **MNIST**. This dataset consist of gray-scale images of hand-drawn digits, from zero through nine with shape 28x28 and position in the center. In the competition is divided of the dataset in the two parts, train-set (train.csv), test-set (test.csv). In the train-set contains 42000 examples and in the test-set contain 28000 images to classify.

Normalization:

To avoid uses a lot of sum, normalize of the data with mean zero and a small variance. For image exists a very easy normalization, how the values of an image is between 0 - 255, we just divide by 255. With this normalization the values of the images are between 0 - 1.



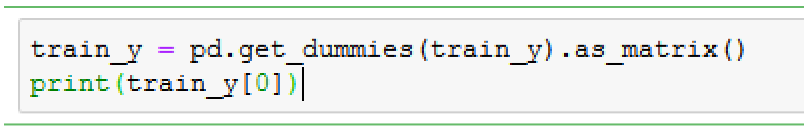
```
def normalization(x, mu, sigma):
    
    x_norm = np.zeros_like(x)
    
    for n in range(len(x)):
        for j in range(len(x[n])):
            if(sigma[j]!=0):
                x_norm[n,j] = (x[n,j] - mu[j]) / sigma[j]
            else:
                x_norm[n,j] = 0
                
    return x_norm
```

```
mu = np.mean(train_x, axis=0)
sigma = np.max(train_x, axis=0)-np.min(train_x, axis=0)
```

```
test = normalization(test, mu, sigma)
```

```
train_x = normalization(train_x,mu, sigma)
```

One Hot Encoding or Dummies is a group of the digits when each digit corresponds the probability of the class is True.



```
train_y = pd.get_dummies(train_y).as_matrix()
print(train_y[0])
```

Design a Model

The NN consist of the connection of artificial neurons divide in layers. The most famous neuron is the perceptron. The artificial neuron consist in simulate the function of the biological neuron.

The biological neural learn with its connections, if the connections is enough strong the signal is activate. In artificial neurons this feature is representation with variable Weight (wi), that they are changed in the training. These Weights represent the knowledge of the Neural Networks. Mathematically an artificial neuron can be represented by the sum of product between Input values and its respective weight, as equation bellow.

$$Z = \sum_{i=1}^{n} X_i \cdot W_i + b_i$$

In the artificial neuron contain the activation function, when the Z is apply. The most famous activation functions are Sigmoid, Tahn, ReLu, LReLu and PReLu.

The derivative of the Z will be important for train the model.

$$\frac{dZ}{dW} = X_i$$

Activation functions:

The **ReLu** (Rectifier Linear Function), this function is similar to the functioning of the biological neuron, and allow us to run the model more fast.

The **Softmax** function allow us predict the model, because it normalize the data in one hot encoding.

The derivative of the σ(x) will be important for train the model.

```
def Softmax(x):
    x -= np.max(x)
    sm = (np.exp(x).T / np.sum(np.exp(x),axis=1)).T
    return sm
```

Creating weights:
To simplify and helping the calculus, we are use the matrix representation for representation the weights, inputs and the outputs. In the method below will create the weights and the biases with

```
def ReLu(x, derivative=False):
    if(derivative==False):
        return x*(x > 0)
    else:
        return 1*(x > 0)
```

mean zero and the variance proportional to root of the numbers of inputs.

$$y = \sigma(x) = \frac{e^x}{\sum_{i=1}^{N} e^x}$$

```
def CreateWeights():

    ninputs = 784
    wl1 = 500 ##Number of neurons in the first layer
    wl2 = 300 ##Number of neurons in the second layer
    nclass = 10 ##Numer of the class, in this case it is the number of the digits.

    #layer1
    w1 = np.random.normal(0, ninputs**-0.5, [ninputs,wl1])
    b1 = np.random.normal(0, ninputs**-0.5, [1,wl1])

    #Layer2
    w2 = np.random.normal(0, wl1**-0.5, [wl1,wl2])
    b2 = np.random.normal(0, wl1**-0.5, [1,wl2])

    #Layer3
    w3 = np.random.normal(0, wl2**-0.5, [wl2,nclass])
    b3 = np.random.normal(0, wl2**-0.5, [1,nclass])

    return [w1,w2,w3,b1,b2,b3]
```

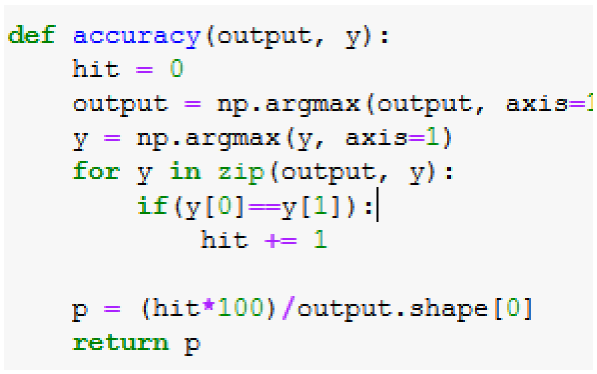
Dropout:
Dropout is a simple technique for prevent over fitting. This technique consist in randomly dropping neurons, for that layer not to depend of these neurons, to predict right.

```
def Dropout(x, dropout_percent):
    mask = np.random.binomial([np.ones_like(x)],(1-dropout_percent))[0]  / (1-dropout_percent)
    return x*mask
```

Predict:
The model is a Neural Network with 2 layers Fully Connected with respectively, 500 and 300 neurons, the both layers with Activation Function ReLu. The last layer contain 10 neurons and used the Activation Function Softmax. Between the layer we will use the Dropout for prevent over fitting

```
def predict(weights, x, dropout_percent=0):

    w1,w2,w3,b1,b2,b3  = weights

    #1-Hidden Layer
    first = ReLu(x@w1+b1)
    first = Dropout(first, dropout_percent)

    #2-Hidden Layer
    second = ReLu(first@w2+b2)
    second = Dropout(second, dropout_percent)

    #Output Layer
    return [first, second, Softmax(second@w3+b3)]
```

Metrics
In order to evaluate the model for measure how well it is going, we can use the accuracy metric and / or loss metric. The accuracy is measure by the percent of the hit of the model did.



```
def accuracy(output, y):
    hit = 0
    output = np.argmax(output, axis=1
    y = np.argmax(y, axis=1)
    for y in zip(output, y):
        if(y[0]==y[1]):
            hit += 1

    p = (hit*100)/output.shape[0]
    return p
```

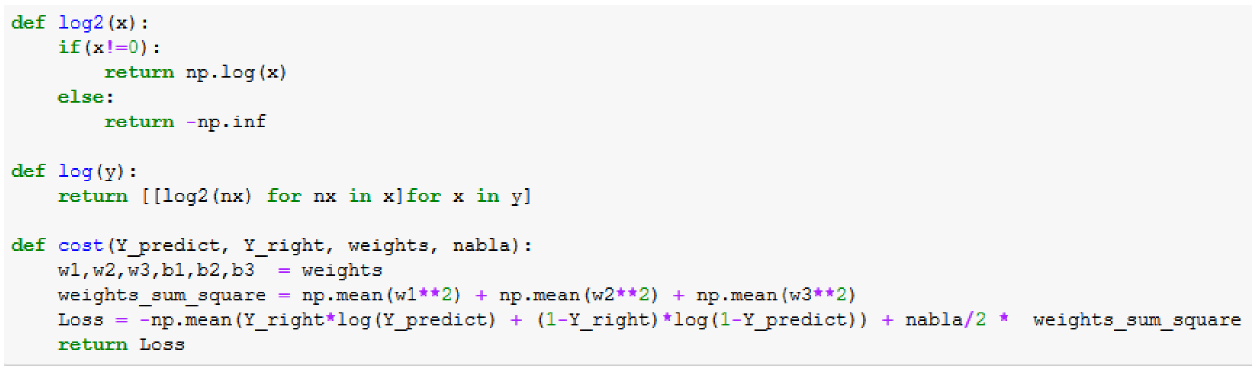
The variance of the numbers of inputs are small, the changes will be small also, so choose the cross-entropy as loss function, because it has high sensibility by small changes. The cross-entropy is used with probability, and compare the outputs of the model with the true results and return the distorting of the difference.

$$L(t|y) = -\frac{1}{N} \cdot \sum_{i=1}^{N} [t \cdot log(y) + (1-t) \cdot log(1-y)]$$

N is the number of the exemples, **t** is the right labels ,**y** is the predict labels.

The log of the number zeros isn't exist so we consider **log(0)** as the lowest value -inf after use the np.nan_to_num().The derivative of the L(t|y) is important for train the model.

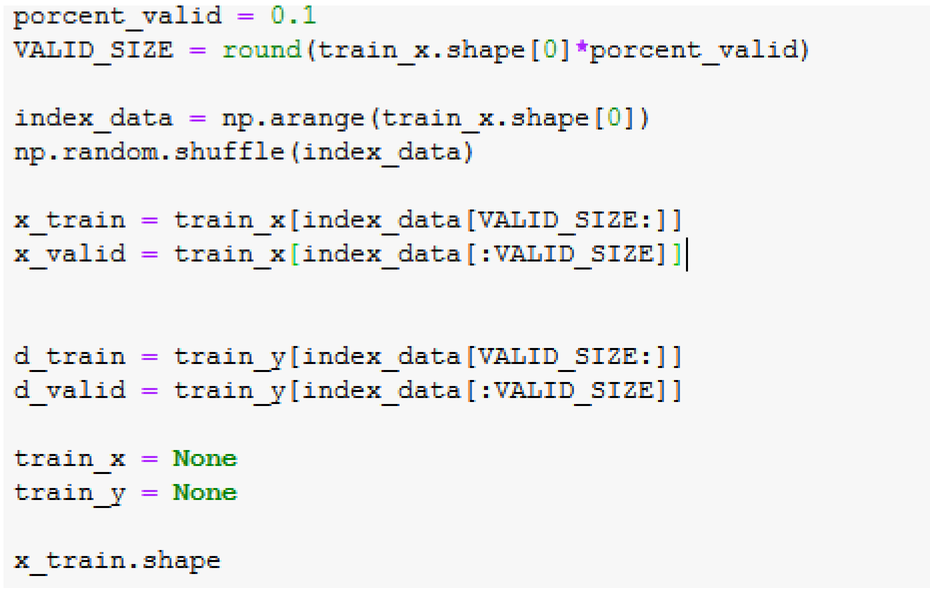
$$\begin{aligned} \frac{dL(t|y)}{dy} &= \frac{1}{N} \cdot \frac{t}{y} - \frac{(1-t)}{(1-y)} \\ &= \frac{1}{N} \cdot \frac{t.(1-y)-(1-t).y}{y.(1-y)} \\ &= \frac{1}{N} \cdot \frac{t-ty-y+ty}{y.(1-y)} \\ &= \frac{1}{N} \cdot \frac{t-y}{y.(1-y)} \end{aligned}$$



```
def log2(x):
    if(x!=0):
        return np.log(x)
    else:
        return -np.inf

def log(y):
    return [[log2(nx) for nx in x]for x in y]

def cost(Y_predict, Y_right, weights, nabla):
    w1,w2,w3,b1,b2,b3  = weights
    weights_sum_square = np.mean(w1**2) + np.mean(w2**2) + np.mean(w3**2)
    Loss = -np.mean(Y_right*log(Y_predict) + (1-Y_right)*log(1-Y_predict)) + nabla/2 *  weights_sum_square
    return Loss
```

Cross - validation

The cross-validation is a technique used for measure the accuracy and visualization over fitting. This technique consist in split the data in train_data and test_data, where the test_data consist of 10% - 30% of the all data.



```
porcent_valid = 0.1
VALID_SIZE = round(train_x.shape[0]*porcent_valid)

index_data = np.arange(train_x.shape[0])
np.random.shuffle(index_data)

x_train = train_x[index_data[VALID_SIZE:]]
x_valid = train_x[index_data[:VALID_SIZE]]


d_train = train_y[index_data[VALID_SIZE:]]
d_valid = train_y[index_data[:VALID_SIZE]]

train_x = None
train_y = None

x_train.shape
```

Training phase

SGD (Stochastic Gradient Descent):

SGD is an optimizer used for fit the neural network, this technique is based by Gradient Descent. In SGD is used the matrix representation, the equation for represent the update the weights is below.

$$V_{k+1} = V_k - \eta . \nabla L(W_{ij})$$

$$W = V_{k+1}$$

The η is the step size, the ∇L(W) is the gradient of the Loss.

The gradient can be solved using the chain rule of the derivate the Loss function by Weights. The equation for output layer is below.

$$\nabla L(W_{ij}) = \frac{dL(W_{ij})}{dW_{ij}}$$

$$= \frac{dL(t|y)}{dy} . \frac{dy}{dZ} . \frac{dZ}{dW_{ij}}$$

$$= \frac{1}{N} . \frac{(t-y)}{y.(1-y)} . y.(1-y). X_{ij}$$

$$= \frac{1}{N} . (t-y) X_{ij}$$

For hidden layer,

$$\nabla L(W_{ij}) = \frac{dL(W_{ij})}{dW_{ij}}$$

$$= \frac{dL(t|y_{ij})}{dy_{ij}} . \frac{dy_{ij}}{dZ_{ij}} . \frac{dZ_{ij}}{dy_{ij}} . \frac{dy_{ij}}{dZ_{ij}} . \frac{dZ_{ij}}{dW_{ij}}$$

$$= \frac{1}{N} . \frac{(t-y)}{y.(1-y)} . y.(1-y). W_{ij}. \sigma(x)'. X_{ij}$$

$$= \frac{1}{N} . (t-y). W_{ij}. \sigma(x)'. X_{ij}$$

Momentum:

The SGD in training have a lot of oscillations. The momentum term is used for soft the oscillations and accelerates of the convergence.

$$V_{i+1} = \gamma V_i + \eta . \nabla L(W)$$

$$W = W - V_{i+1}$$

L2 regulizer

The L2 regulizer is the technique which is used for penalize the higher weights,

$$L(t|y|w) = -\frac{1}{N} \cdot \sum_{i=1}^{N} [(t \cdot log(y) + (1-t) \cdot log(1-y)) + \frac{\lambda}{2} \cdot \sum_{j=1}^{nj} \sum_{i=1}^{ni} W_{ij}^{2}]$$

With this modification in the Loss function the equation of the update the weights is changed. For the output layer,

$$\nabla L(W_{ij}) = \frac{1}{N} \cdot [(t-y).\, X_{ij} + \lambda W_{ij}]$$

For the hidden layer,

$$\nabla L(W_{ij}) = \frac{1}{N} \cdot [(t-y).\, W_{ij}.\, \sigma(x)'.\, X_{ij} + \lambda W_{ij}]$$

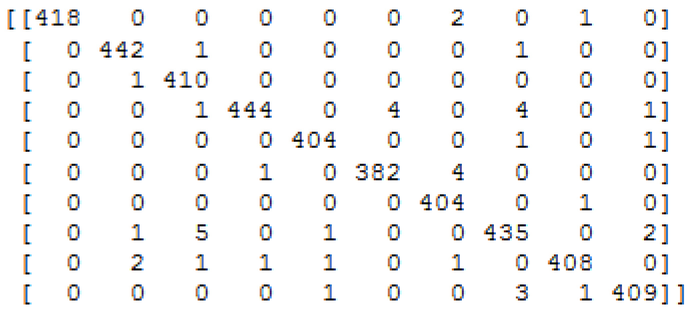
Training process:

In the training process is divided all the data in batchs, it's a process part because it's principle fundament of the stochastic process. These batchs need to be divide with randomlly for improved the accuracy of the model.

Results

Confusion Matrix:

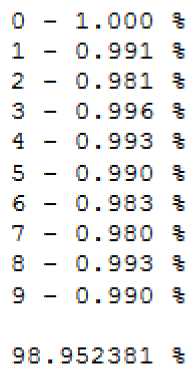
A confusion matrix is a table that is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known. Below image display the confusion matrix (10x10), the column values are true label and row values are predicted, we could interpret that for number 9, one time is predicted as 4, 3 times predicted as 7, 1 time predicted as 8 and 409 times predicted as 9.



```
[[418   0   0   0   0   0   2   0   1   0]
 [  0 442   1   0   0   0   0   1   0   0]
 [  0   1 410   0   0   0   0   0   0   0]
 [  0   0   1 444   0   4   0   4   0   1]
 [  0   0   0   0 404   0   0   1   0   1]
 [  0   0   0   1   0 382   4   0   0   0]
 [  0   0   0   0   0   0 404   0   1   0]
 [  0   1   5   0   1   0   0 435   0   2]
 [  0   2   1   1   1   0   1   0 408   0]
 [  0   0   0   0   1   0   0   3   1 409]]
```

Accuracy:

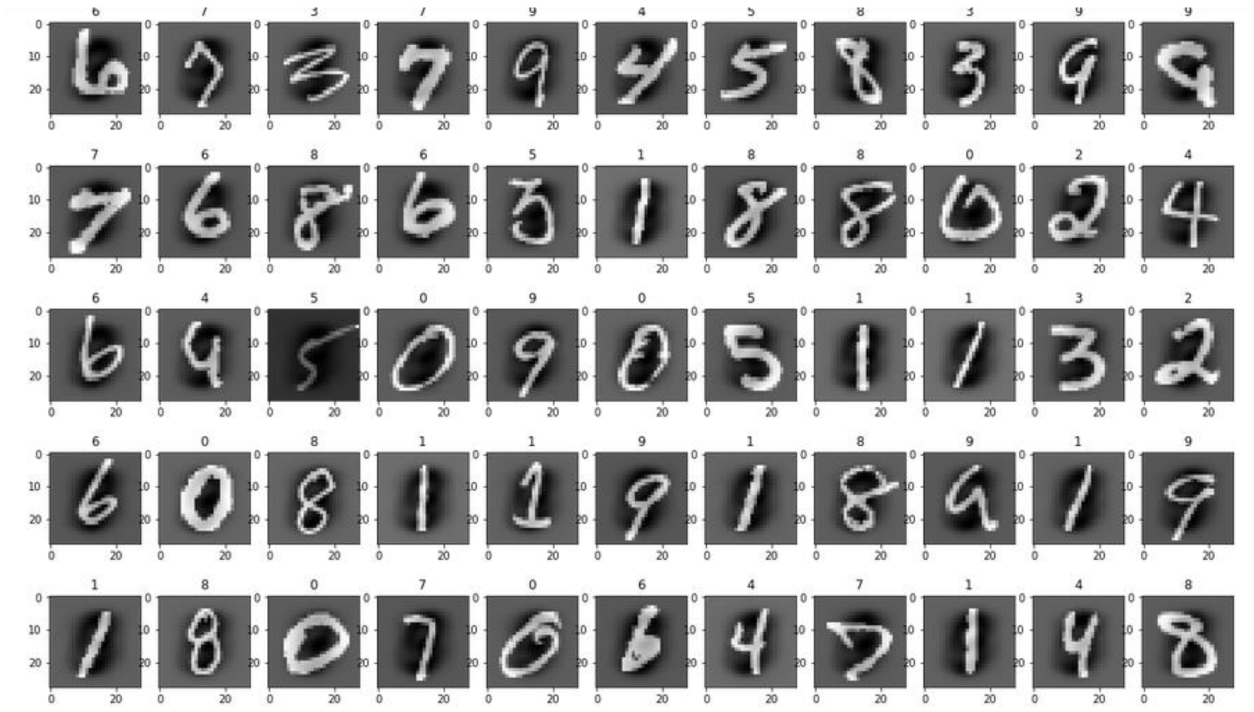
As a result, number zero has the most precision and the least precision is number 7. Total accuracy for all numbers is 98.95%.



```
0 - 1.000 %
1 - 0.991 %
2 - 0.981 %
3 - 0.996 %
4 - 0.993 %
5 - 0.990 %
6 - 0.983 %
7 - 0.980 %
8 - 0.993 %
9 - 0.990 %

98.952381 %
```

Prediction:

Below images demonstrate the visual prediction for different values. In first column first number the image digit is 6 and it also predicted as 6, but ninth column, row forth the image of digit is not really same as 9 but it consider as 9.



Conclusion

This project is about digit recognizer by using MINIS dataset. The train-set contains 42000 examples and the test-set contain 28000 images to classify. Data normalized to reduce some calculation in later process. Neural network model design by softmax and relu activation function and stochastic gradient descent algorithm utilize in learning phase to speed up this phase. After learning and training phase, confusion matrix compute. Number zero has the most precision and the least precision is number 7. Total accuracy for this model equal to 98.95%. at least I learnt how to design a simple model without built-in function which defined in keras, sklearn and etc.

Refrences

Image transformation retrieved from

https://gist.github.com/fmder/e28813c1e8721830ff9c

CHAPTER 1 Using neural nets to recognize handwritten digits retrieved from

http://neuralnetworksanddeeplearning.com/chap1.html?fbclid=IwAR3tGY6sku2cAsXfZegBpubpAkm5QcQ31Qamsyrd3EFkSlaVVkC6ESs7OUY on Dec 2019

Bishop - Pattern Recognition and Machine Learning - Springer 2006, retrieved from
http://users.isr.ist.utl.pt/~wurmd/Livros/school/Bishop%20-%20Pattern%20Recognition%20And%20Machine%20Learning%20-%20Springer%20%202006.pdf

Mini-Batch Gradient Descent with Python retrieved from
https://www.geeksforgeeks.org/ml-mini-batch-gradient-descent-with-python/ on No date

Simple guide to confusion matrix terminology retrieved from
https://www.dataschool.io/simple-guide-to-confusion-matrix-terminology/ on 25 March 2014