Data Normalization Explained

No: Date:

Data normalization / Scaling

⇒ Data Normalization is ~~setting~~ scaling tech so that values are between 0 and 1

Why ?

1.

x_1 and x_2 are datas

$w_1 x_1^{(1)} + w_1 x_2^{(1)} \cdots (1)$

$w_2 x_2^{(1)} + w_2 x_2^{(2)} \cdots (2)$

In (1), Sometimes x_2 is very big compared to x_1 and this creates a large variance σ^2. Large variance means large spread of data and is very hard for neural network or CNN to converge during optimisation.

~~How?~~

Optimisation (Gradient Descend)

$$\theta_j = \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

The value α in (1) will affect the step size of gradient descend. The difference in ranges of features (x_i) will cause different ~~of~~ step sizes for gradient descend

How

transforms.normalize ((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

image = (image - mean) / std

Neural Network Explained

No. Date.

Neural Network Model

In[16]

self.fc1 (28×28)

- Gets the array of input (28×28), and output is 64 in length
- The __init__ is for initialization

def forward()

=> RELU

Relu activation function gives output of 0 or 1, any negative values are converted to 0.

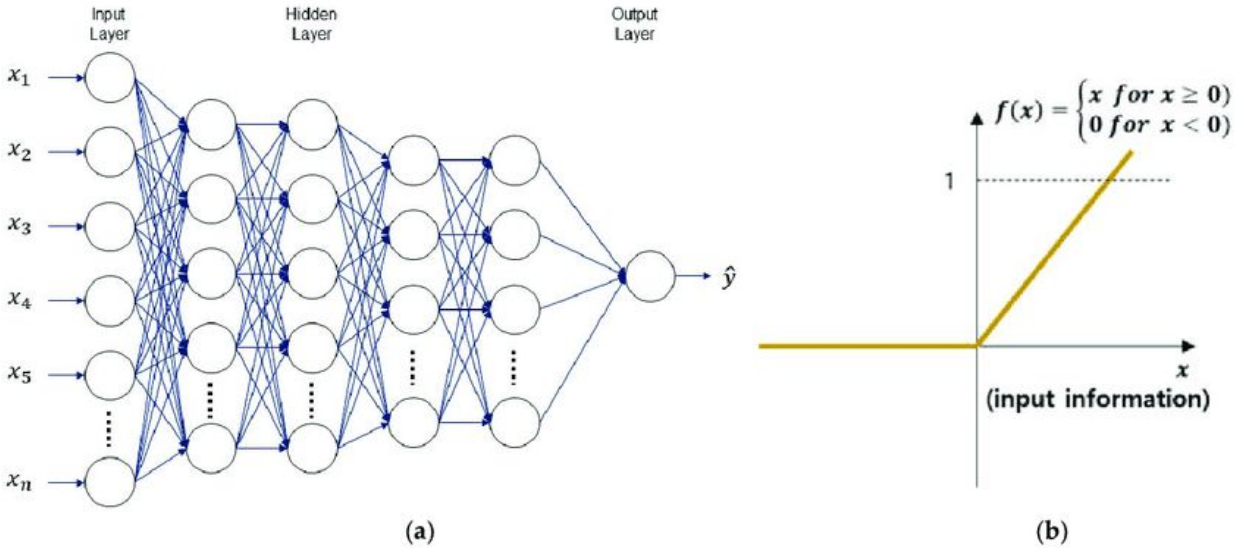
This is what happens in the line x = F.relu(self.fc1(x))

- Each image are represented by vectors

x_1 (image 0) w (weight) $\Sigma x w$ bias $\Sigma w x + b = \hat{y}$

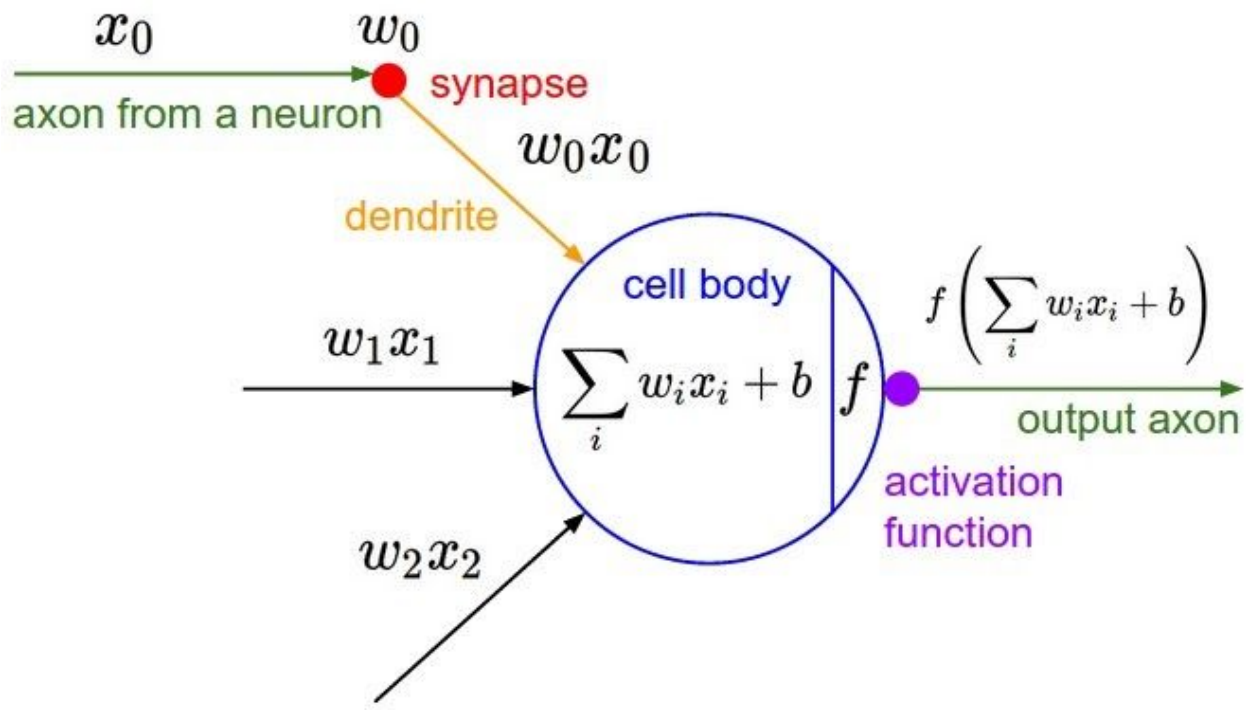
$$\begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} w_1 \\ w_2 \\ \vdots \end{bmatrix} = \begin{bmatrix} \vdots \\ \vdots \\ \vdots \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ \vdots \end{bmatrix} = \begin{bmatrix} \vdots \\ \vdots \\ \vdots \end{bmatrix}$$

Relu function $\begin{cases} x \geq 0 & 1 \\ x < 0 & 0 \end{cases}$ Relu($\hat{y}$)



a) Each Xi represents the datas and y is the output of the data

b) relu function



This is what happens in each line of the code in forward function.

In LN[21]

Adam Optimizer

Adaptive Moment Estimation is an algorithm for optimization technique for gradient descent. The method is really efficient when working with large problems involving a lot of data or parameters. It requires less memory and is efficient. Intuitively, it is a combination of the ‘gradient descent with momentum’ algorithm and the ‘RMSP’ algorithm.

Learning rate of 0.0001 is used in this neural network model.

In each epochs, the weight is adjusted to minimise the loss function , For further studies on nll loss function pleaser refer to the documentation
https://pytorch.org/docs/stable/generated/torch.nn.NLLLoss.html#torch.nn.NLLLoss

Once the model has trained with accuracy of 98% , this doesn't mean that it has successfully trained with accurate prediction. The model has to go through testing and validation for human to make sure the accuracy is indeed accurate.