Hand Written Digit Recognition Using ANN

Sourav Karmakar

souravkarmakar29@gmail.com

Basics: Digital Image

The digital images are nothing but discretized intensity values associated with discretized spatial co-ordinates

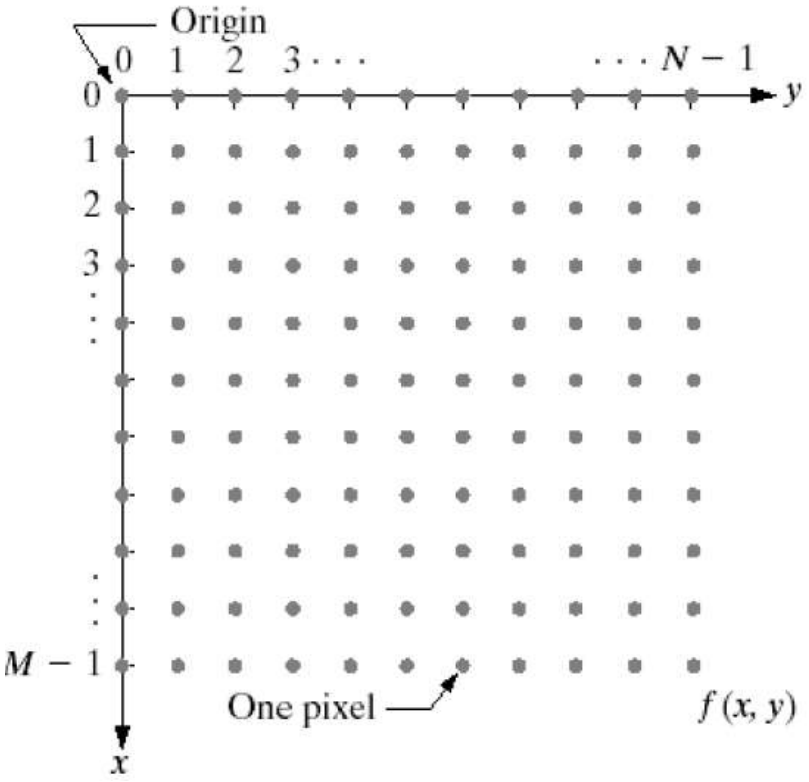
The digital images are of two types:

- **The Grayscale Images**

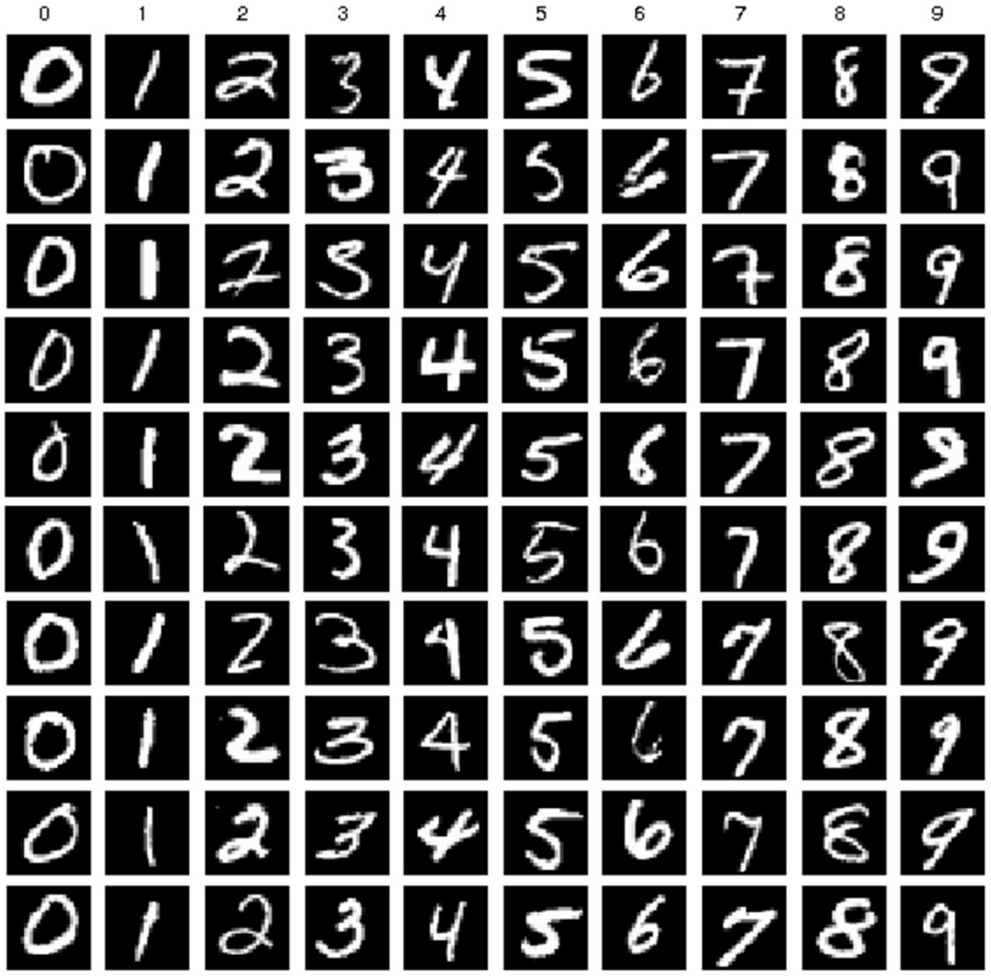
- Each pixel values are integers and lies between 0 to 255
- Extreme black is denoted as 0 and Extreme white is denoted as 255

- **The Colour Images**

- There are three values associated with each pixel. **R**,**G**,**B**. Each values are integers bet. 0 - 255
- 0 denotes absence of that colour, 255 denotes fully presence of that colour.



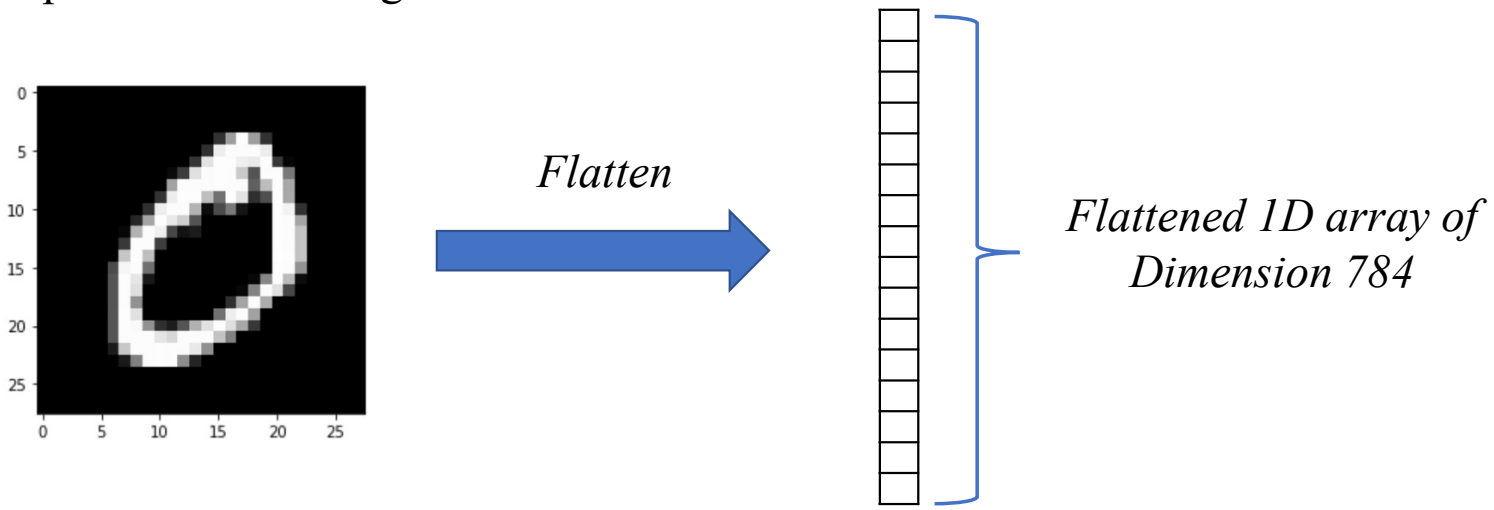
MNIST: Hand Written Digit Dataset



- Contains 60000 training and 10000 test samples.
- Each image is grayscale image of size 28×28.
- Each training and test sample also supplied with corresponding label.
- There are total 10 classes. Digit-0 belongs to Class-0, Digit-1 belongs to Class-1,…, Digit-9 belongs to Class-9
- Next we shall discuss about data pre-processing steps before discussing the architecture of the MLP for Hand Written Digit Recognition.

Data Preprocessing

- **Flattening the images:**
  - Originally the training dataset is of size $60000 \times 28 \times 28$. As each image is of size 28×28. So there are total 784 pixels in each image.



  - After flattening each image will become a one dimensional array of size 784 and the training data set shall become of size 60000×784.
  - Similarly we flatten test dataset also. The $10000 \times 28 \times 28$ sized test dataset shall be of size 10000×784 after flattening.

Data Preprocessing

- **Normalizing the data:**
  - The digital image contains integer values between 0 – 255. First we have to convert the data type of the image from integer to float.
  - Then we have to divide the values in each pixel by 255.
  - This will ensure that each pixel will contain value between 0 to 1.
  - Normalization will help to stabilize the gradient descent algorithm.
  - Note that both training and test data have to be normalized.

Data Preprocessing

- **One Hot Encoding of the labels:**
  - For Multiclass classification from each class labels / target values should be converted to **one hot encoded vectors**. Each one hot encoded vectors should have dimension = No. of Classes.

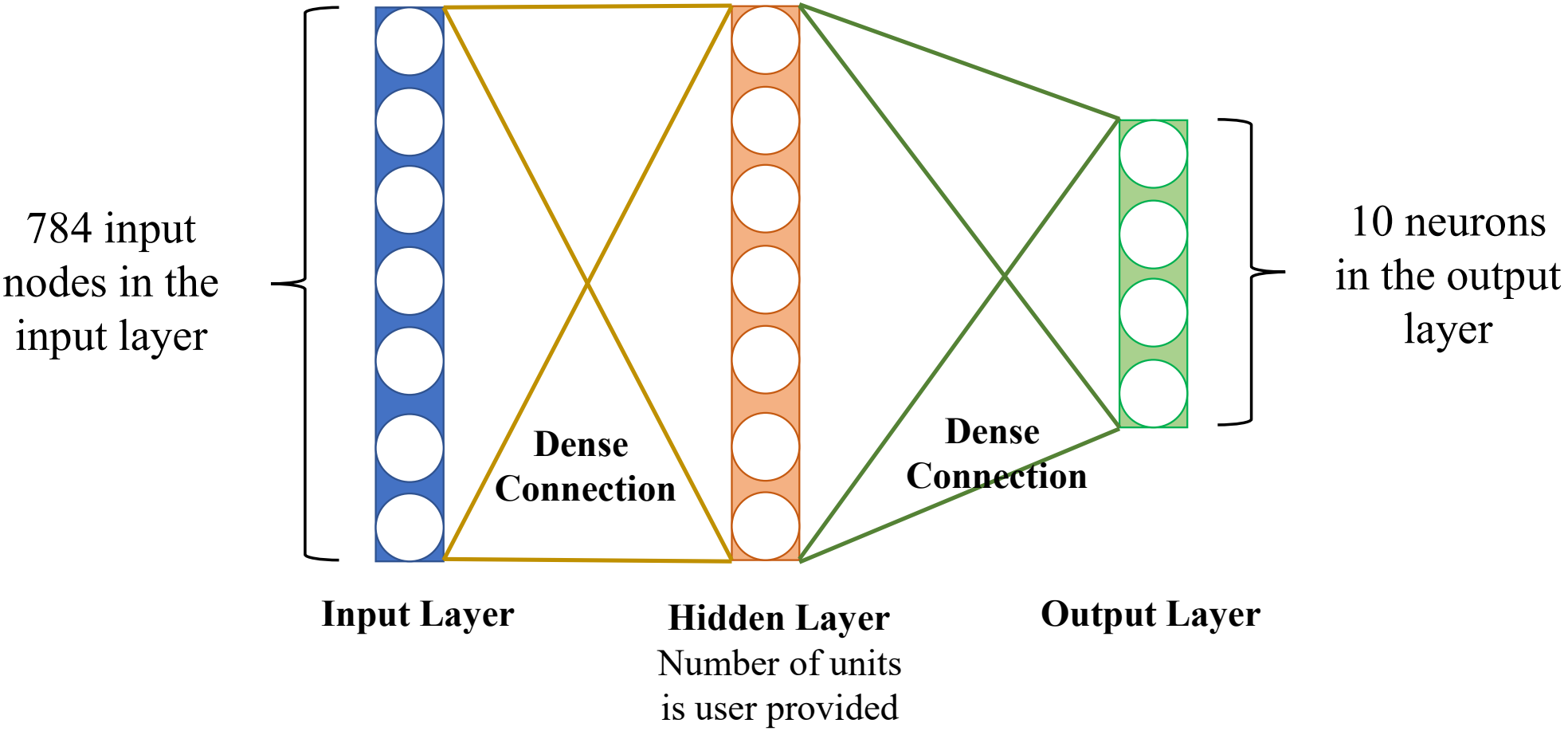
Images	Class Labels	One-Hot Encoded Vector
	0	$[1, 0, 0, 0, 0, 0, 0, 0, 0, 0]^T$
	1	$[0, 1, 0, 0, 0, 0, 0, 0, 0, 0]^T$
	5	$[0, 0, 0, 0, 0, 1, 0, 0, 0, 0]^T$

And similarly for other classes

  - As the output of the MLP will provide a probability distribution function hence we have to one-hot-encode the target variables.
  - This is optional in PyTorch because the "cross-entropy" loss in PyTorch takes logit inputs and the labels in integer format.

ANN (MLP) Architecture

- **Single Hidden Layer Architecture:**



ANN (MLP) Training

- **Batch Size:**

  Batch size is limited by the physical memory availability in the system and size of the each of the samples. If the available memory in the system is small choose smaller batch size (maybe 16 or 32). One can choose larger batch size if the available memory is more.

- **Number of Epochs:**

  Number of epochs should be reasonable. For less complex dataset smaller epochs are sufficient. However higher epochs will led to longer computation time.

- **Loss Function:**

  As we are using one-hot-encoded vector and in the output layer we are expecting to have a probability distribution, hence we shall use categorical cross entropy loss.

- **Optimizer:**

  We shall be using *Adam* optimizer. Optimizer ensures that while learning through backpropagation the network doesn't halt in local minima. There are other types of optimizers as well.

Few Practical Considerations

- **Activation function:**

  ReLU activation function in hidden units, and softmax at output layer.

- **Batch Normalization (optional):**

  Batch normalization helps to obtain faster convergence.

- **Dropout (optional):**

  Dropout helps to regularize the neural network and reduce overfitting.

- **Initialization (optional):**

  Weights should be initialized with small random numbers. Bias can be initialized with zero.

- **Check for convergence:**

  Plot the loss function vs epoch. If it is decreasing with acceptable rate then the network is converging. If it doesn't decrease or very slowly decrease then we have to modify the network hypermeters until it shows convergence.

Thank You