Assignment 2

CMSC 678 — Introduction to Machine Learning

Nisha Pillai

References :

Question 4 : I learnt the concept ‘stratified sampling’ from
http://stattrek.com/statistics/dictionary.aspx?definition=stratified%20sampling

Question 7 : I referred multiple sites to learn about neural network architecture and implementation. I compared other scripts with mine when I hit ‘overflow’ error and tried to understand where am I making a mistake.

Here is the list of posts/scripts/videos that I referred for the same.

https://www.youtube.com/watch?v=An5z8lR8asY
https://www.youtube.com/watch?v=q555kfIFUCM
https://www.youtube.com/watch?v=aVId8KMsdUU
https://www.youtube.com/watch?v=xfPz92B0rv8
http://rohanvarma.me/Neural-Net/
http://www.wildml.com/2015/09/implementing-a-neural-network-from-scratch/
http://neuralnetworksanddeeplearning.com/chap1.html
https://databoys.github.io/ImprovingNN/
https://github.com/kdexd/digit-classifier

Answers:

Qn 4 :

a) getDS function extracts MNIST train/test data from the pickle provided in the assignment.

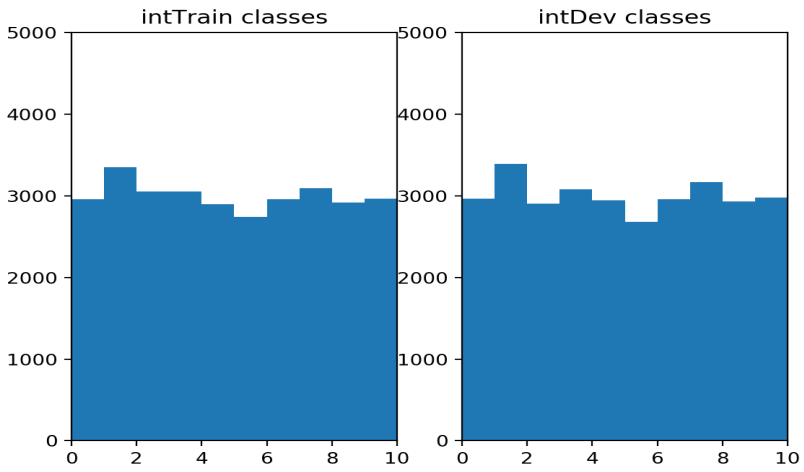
b) Implemented four types of train/dev split

i) Split images_train **sequentially with a percent**: If we provide percent as '0.6', then the first 60% instances would be 'int-train' and next 40% would be 'int-dev'

Results with 0.5 percent split

	Number of instances per classes									
	0	1	2	3	4	5	6	7	8	9
intTrain	2959	3353	3053	3052	2897	2742	2961	3096	2919	2968
intDev	2964	3389	2905	3079	2945	2679	2957	3169	2932	2981
Full train set	5923	6742	5958	6131	5842	5421	5918	6265	5851	5949

Histogram



ii) **sequential split with specified intTrain and intDev instance counts**

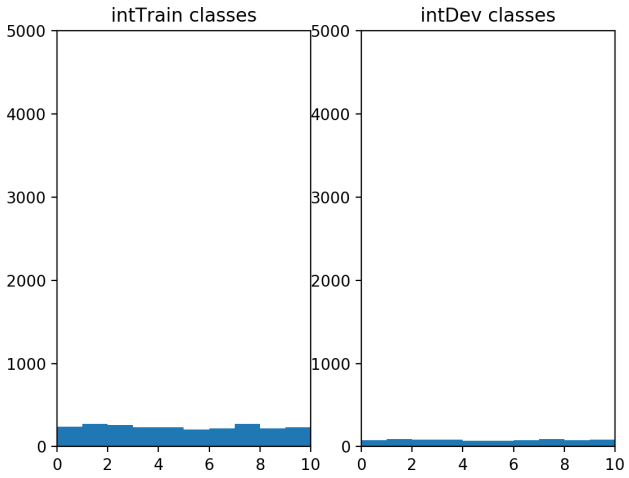
If we specify 2400,1600 as input parameters, it would take first 2400 instances for intTrain and next 1600 for intDev

Split results with (2400 intTrain and 800 intDev instances)

Instances per class

Int Train : 240 276 262 233 234 209 219 274 221 232

Int Dev : 77 90 81 81 69 73 78 90 74 87



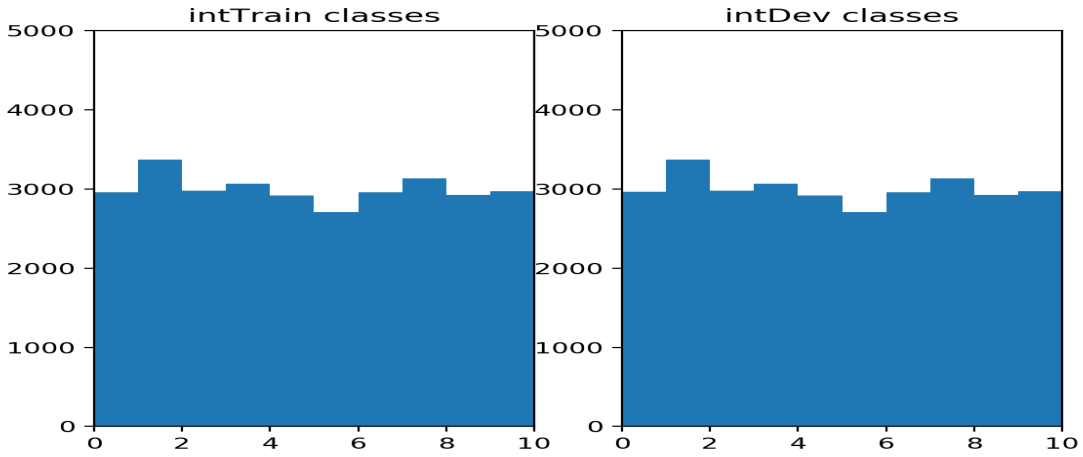
iii) **Split by stratified sequential sampling with percent as input**

If we provide 0.6 as the percent, the code would first group instances classwise, and then split the instances in sequential order.

Results with 0.5 split

Int Train : 2961 3371 2979 3065 2921 2710 2959 3132 2925 2974

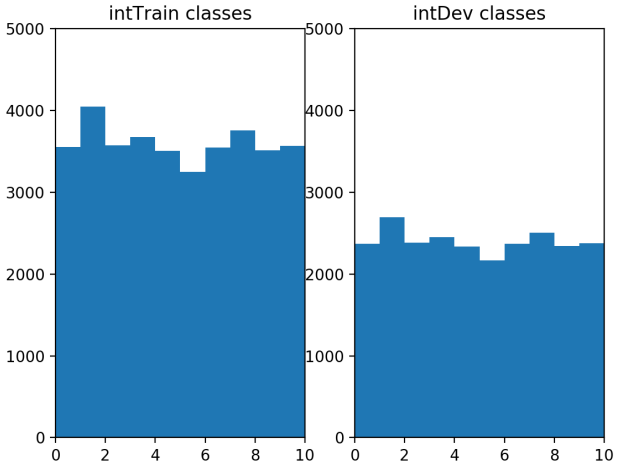
Int Dev : 2962 3371 2979 3066 2921 2711 2959 3133 2926 2975



iv) Split by stratified random sampling with percent as input

If we provide 0.6 as the percent, the code would first group instances classwise, and then split the instances in random order.

Results with 0.6 split



Instances per class

Int Train : 3553 4045 3574 3678 3505 3252 3550 3759 3510 3569

Int Dev : 2370 2697 2384 2453 2337 2169 2368 2506 2341 2380

Qn 5

I implemented a multiclass perceptron for this question. I experimented the implementation with different kinds of configuration.

i) With a fixed rate learning rate

ii) With a Robbins-Monro scalar learning rate.

iii) With weights and biases starting with value 0.

iv) With weights and biases starting with samples from uniform distribution over interval -1.0 – 1.0

Here I used a train/dev dataset split using stratified random sampling with 0.8 percent.

Accuracy was between **0.85 - 0.89** in these cases.

Qn 7

Initially, I started implementing NN in online mode. But it resulted in 'overflow' error due to the rapid increase of weights[large negative values]. Even with batch mode implementation, I hit the same 'overflow' error. Mini-batch implementation with 100 instances worked fine for my model. In the mini-batch mode, I list down some of the configurations that I tried.

i) NN with 1 hidden layer, non-linear function 'sigmoid' as activation function in both 'input to hidden' and 'hidden to output' layer, constant(0.1) learning rate, and 30 hidden layer nodes

Accuracy : 0.89 - 0.91

2) Multi-layer NN, non-linear function 'sigmoid' as activation function in both 'input to hidden' and 'hidden to output' layer, constant(0.1) learning rate.
Maximum accuracy that I could get: 0.89

3) NN with 1 hidden layer, non-linear function 'tanh' as activation function in both 'input to hidden' and 'hidden to output' layer, constant(0.01) learning rate
Maximum accuracy that I could get: 0.64
4) NN with Adagrad learning rate : 1 hidden layer, ‘sigmoid’ activation function in both layers. , $\eta = 0.1$ and eps = $1 \times 10{-}8$

Maximum accuracy : 0.51

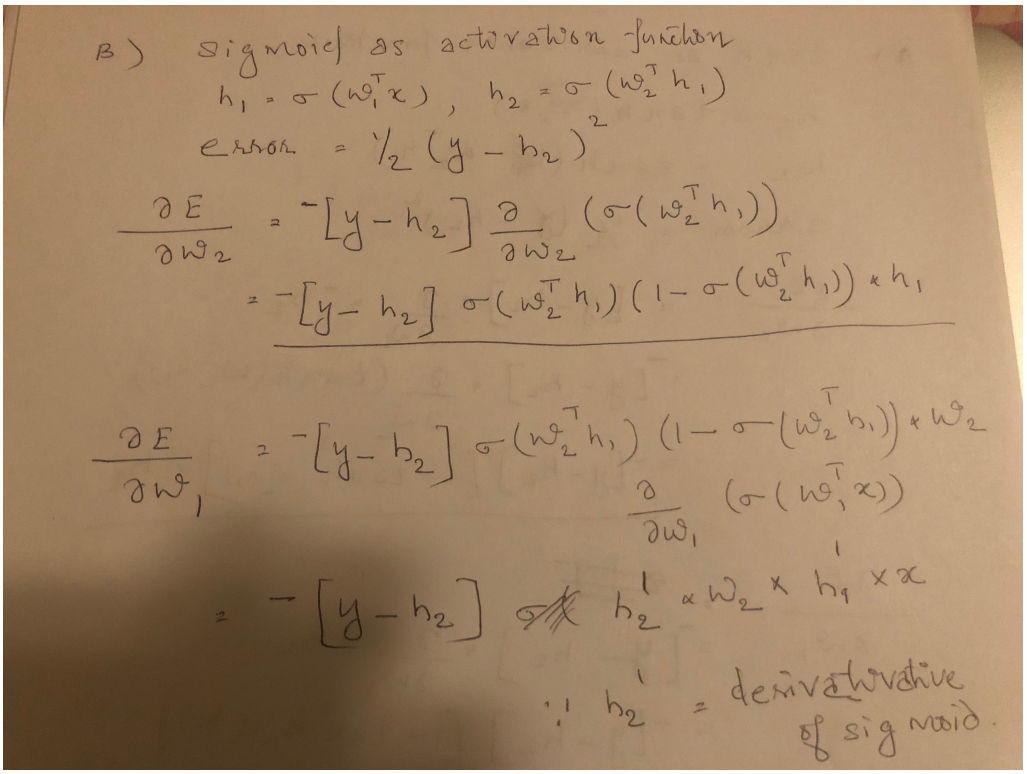
Qn 8

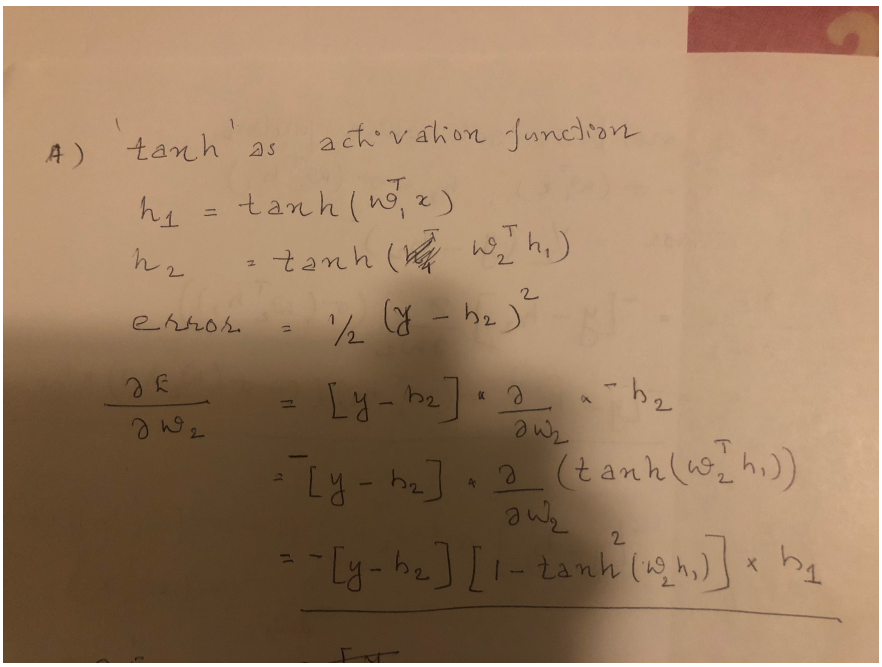
Trained the below models on original training and test data of MNIST.

1. Multi-class perceptron with 0.01 fixed learning rate. Accuracy was 0.87 for this model while testing on original images_test data.
2. Neural network with 1 hidden layer, fixed learning rate - 0.1, hidden layer nodes - 30.
   Accuracy was 0.91 in this case.

With both models, my results are same compared to previous int-dev tests. I sense a bug in my implementation as I could not increase the accuracy more than 0.91 even with fixed/agadrad learning rates.

Gradient Calculation(hand-notes)





$$= -[y - h_2][1 - \tanh(w_2 h_1)] \cdot 1$$

$$\frac{\partial E}{\partial w_1} = -[y - h_2] \times \frac{\partial}{\partial w_1} \times h_2$$

$$= -[y - h_2]\left[1 - \tanh^2(w_2 h_1)\right] \times w_2 \frac{\partial}{\partial w_1}\left[\tanh(w_1^T x)\right]$$

$$= -[y - h_2]\left[1 - \tanh^2(w_2 h_1)\right] \times w_2 \left[1 - \tanh^2(w_1^T x)\right] \times x$$