CSE 574: Introduction to Machine Learning

Assignment 2

Team: Sphoorthi Keshannagari (50372699) skeshann@buffalo.edu
Bharath Reddy Nallu (50376364) bnallu@buffalo.edu
Ishita Aggarwal (50431309) iaggarwa@buffalo.edu

Neural Networks

In this project we implemented Neural Network (forward pass and back propagation).

1.1 NNScript:

The functions in the script are implemented as below:

1.1.1 Sigmoid(z):

We implemented the below mentioned formula in this function:

$$f(z) = \frac{1}{1 + exp(-z)}$$

Note: The implementation works with z as a scalar/vector or a matrix.

1.1.2 Preprocess():

In this function, we first load the mnist_all.mat file as a dictionary. And then we divide the train data provided into two parts: train data (containing 50k values) and validation data (containing 10k values). We use the test data provided in the .mat file.

Feature Selection:
As a part of feature selection, we remove the redundant data from the training, test and validation data. We validate by checking every column and remove the indexes of the columns which have the same training points. The indices obtained after performing the above logic are:

Selected Feature Indices:
[12, 13, 14, 15, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324, 325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 338, 339, 340, 341, 342,

343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 399, 400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 428, 429, 430, 431, 432, 433, 434, 435, 436, 437, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 477, 478, 479, 480, 481, 482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 510, 511, 512, 513, 514, 515, 516, 517, 518, 519, 520, 521, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558, 559, 561, 562, 563, 564, 565, 566, 567, 568, 569, 570, 571, 572, 573, 574, 575, 576, 577, 578, 579, 580, 581, 582, 583, 584, 585, 586, 587, 588, 589, 590, 591, 592, 593, 594, 595, 596, 597, 598, 599, 600, 601, 602, 603, 604, 605, 606, 607, 608, 609, 610, 611, 612, 613, 614, 615, 616, 617, 618, 619, 620, 621, 622, 623, 624, 625, 626, 627, 628, 629, 630, 631, 632, 633, 634, 635, 636, 637, 638, 639, 640, 641, 642, 643, 646, 647, 648, 649, 650, 651, 652, 653, 654, 655, 656, 657, 658, 659, 660, 661, 662, 663, 664, 665, 666, 667, 668, 669, 670, 674, 675, 676, 677, 678, 679, 680, 681, 682, 683, 684, 685, 686, 687, 688, 689, 690, 691, 692, 693, 694, 695, 696, 697, 698, 702, 703, 704, 705, 706, 707, 708, 709, 710, 711, 712, 713, 714, 715, 716, 717, 718, 719, 720, 721, 722, 723, 724, 725, 726, 731, 732, 733, 734, 735, 736, 737, 738, 739, 740, 741, 742, 743, 744, 745, 746, 747, 748, 749, 750, 751, 752, 753, 760, 761, 762, 763, 764, 765, 766, 767, 768, 769, 770, 771, 772, 773, 774, 775, 776, 777, 778, 779]

Train shape: 50000 717
Validation shape: 10000 717
Test shape: 10000 717

Number of features selected: 717

We pass these features onto the params.pickle file.

1.1.3 nnObjFunction(params, *args):

We pass the weights of matrices w1 (weight of connection from input layer to hidden layer) and w2 (weight of connection from hidden layer to output layer) in a vector form.
As a part of args we pass the following values – n_input, n_hidden, n_class, training_data, training_label and lambda.
The feed forward pass and back propagation logics are implemented within this function.
Inorder to avoid the overfitting problem, we perform regularization on the data.
λ (regularization term) and m (number of hidden units) are the hyper parameters.

1. λ value ranges from 10 – 60 with increments of 10.
2. The Lambda Vector: [0 10 20 30 40 50 60].
3. The m value ranges from 4 – 20 with increments of 4.
4. The Hidden Node Vector: [0 10 20 30 40 50 60].

We use the values of objective function to identify optimal λ, the value of λ at which the objective function is minimized is chosen as optimal. Each node in the hidden layer produces a different output for the same input after training, therefore considering multiple hidden nodes gives a more accurate result.
We pass the w1, w2, optimal λ and optimal m values to the params.pickle file.

1.1.4 nnPredict(w1,w2,data):

This function takes the learned weights w1 & w2 corresponding to the hidden layer and output layer and computes the predicted class for each of the input data points.

1.1.5 Final Results:

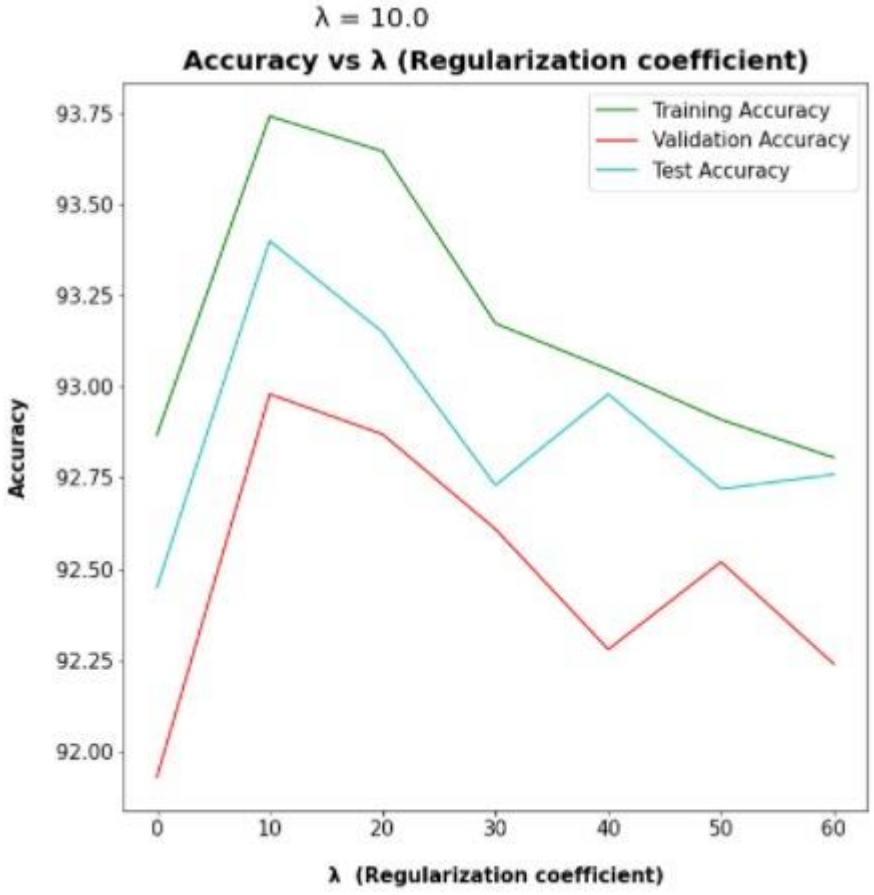
The optimal lambda obtained: 10
The optimal m (nodes) obtained: 20
Highlighted below are the Accuracies (in %) and Training Times (in seconds) obtained at optimal lambda and m values:

	λ	m	Training Accuracy	Validation Accuracy	Test Accuracy	Training Time
0	0	4	68.568	67.69	68.1	57.726262
1	0	8	79.104	78.81	79.1	49.700812
2	0	12	91.924	91.39	91.5	57.69129
3	0	16	92.45	91.9	92.38	54.652939
4	0	20	92.868	91.93	92.45	68.25402
5	10	4	59.414	58.94	58.5	45.499745
6	10	8	89.048	87.96	88.48	52.16306
7	10	12	92.226	91.2	92.02	56.506511
8	10	16	92.946	92.45	92.69	57.988639
9	10	20	93.742	92.98	93.4	63.616954
10	20	4	77.912	77.59	77.25	47.589622
11	20	8	86.424	85.69	85.9	48.084298
12	20	12	91.172	90.53	91.11	58.443866
13	20	16	93.028	92.19	93.17	61.990626
14	20	20	93.646	92.87	93.15	64.386033
15	30	4	73.052	73.21	72.82	50.97528
16	30	8	90.442	89.61	90.45	51.388974
17	30	12	92.144	91.64	92.54	58.476915
18	30	16	92.41	91.93	92.2	56.885042
19	30	20	93.174	92.61	92.73	60.513566
20	40	4	61.83	61.35	61.89	48.070034
21	40	8	88.862	88.45	88.6	52.886247
22	40	12	90.97	90.62	90.99	55.385645
23	40	16	92.328	91.61	92.23	55.03998
24	40	20	93.048	92.28	92.98	60.322332
25	50	4	68.866	68.67	70.07	51.590235
26	50	8	90.672	89.69	90.44	52.7573
27	50	12	92.306	91.73	92.25	54.629051
28	50	16	92.506	91.68	92.44	55.441307
29	50	20	92.91	92.52	92.72	61.112334
30	60	4	76.344	75.24	75.59	46.140421
31	60	8	81.468	80.69	81.59	55.324739
32	60	12	91.774	91.26	91.9	57.463124

33	60	16	92.1	91.63	92.2	55.580449
34	60	20	92.806	92.24	92.76	60.685109

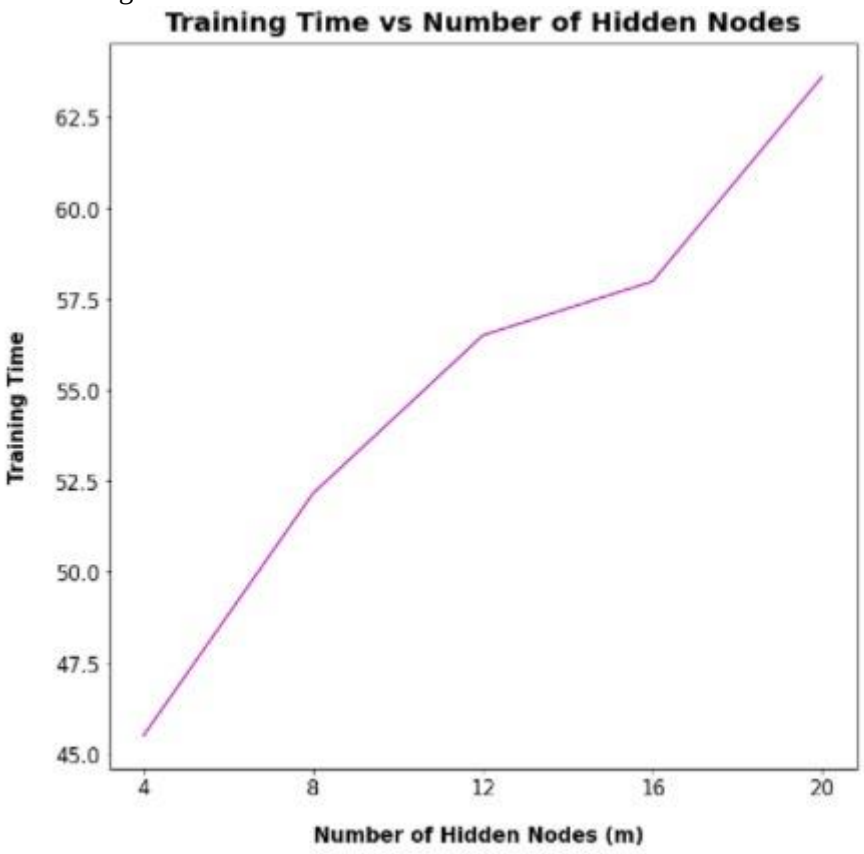
The accuracy obtained at optimal λ and optimal m: 93.742

Plot for Accuracy vs Lambda λ:



From the above graph, we observe that after the optimal λ the data starts to overfit.

Plot for Training Time vs Number of Hidden Units m:



From the above graph, we observe that the training time increases with an increase in number of hidden units.

Accuracy of classification method on the handwritten digits test data -
Train Accuracy: 93.742
Test Accuracy: 93.4
Validation Accuracy: 92.98

1.2 **FacesNNScript:**

1.2.1 Code Explained:

We added our codes for sigmoid(z), nnObjFunction(params, *args) and nnPredict(w1,w2,data) functions from the nnScript file.

1.2.2 Accuracy of classification method on the CelebA dataset:

With $\lambda = 10$:

```
!python "facennScript".py

Training set Accuracy:85.11848341232228%

Validation set Accuracy:84.09005628517824%

Test set Accuracy:85.46555639666919%
```

With λ = 20:

```
Training set Accuracy:86.31279620853081%

Validation set Accuracy:85.51594746716698%

Test set Accuracy:86.63890991672974%
```

With λ = 30:

```
Training set Accuracy:84.66350710900474%

Validation set Accuracy:82.77673545966229%

Test set Accuracy:84.78425435276306%
```

2.0 Deep Neural Network:

We modified the deepnnScript provided to validate against multiple hidden layers, the results obtained from the same are as below:

# of Hidden Layers	Time taken(in seconds)	Accuracy	Accuracy (in %)
1	167	0.827782	82.78
2	197	0.81302	81.31
3	218	0.785011	78.51
5	311	0.776684	77.67
7	324	0.774035	77.41
12	502	0.753974	75.4
15	565	0.715746	71.58
20	734	0.5	50
30	1099	0.5	50

We observed that the increase in hidden layers increased the time taken but reduced the accuracy. 10^{-3} is the optimal value for learning rate. We observe that decreasing the number of batches from 100 to 10 with 3 hidden layers increases the accuracy from 78.51 to 81.68, but it also significantly increases the training time.

# of Hidden Layers	Accuracy (in %)
3	81.68
5	81.6
7	81.79

2.1 Comparison of accuracy and training time of deep neural network to neural network:

# of Hidden Layers	Time taken(in seconds)	Accuracy (in %)	Model
1	63.616954	93.742	Neural Network
3	218	78.51	Deep Neural Network
5	311	77.67	
7	324	77.41	

3: Convolution Neural Network:

3.1 Results:

We ran the cnnscript.py in Google Colab and obtained the following results:

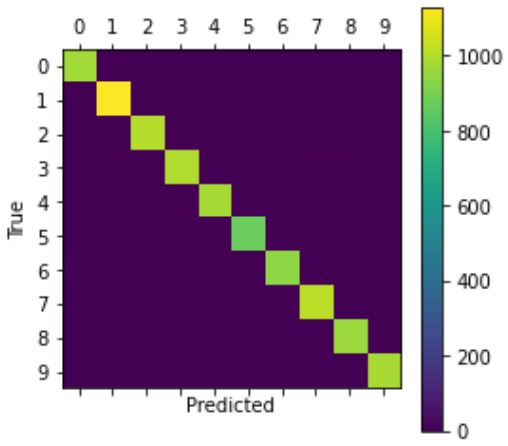
Iterations	Training Time (in seconds)	Accuracy (in %)
0	0	9.1
1	0	9.4
100	7	62.8
1000	65	93.1
10000	656	98.6

From the above results, we can derive that the accuracy increases with the increase in number of iterations, but the training time significantly increases as well.

The final confusion matrix obtained after 10000 iterations:

```
[[ 976    0    0    0    0    0    0    1    3    0]
 [   0 1131    1    0    0    0    1    1    1    0]
 [   4    5 1005    3    2    0    0    7    6    0]
 [   1    0    1  998    0    1    0    4    3    2]
 [   0    0    0    0  977    0    1    2    0    2]
 [   2    0    0    8    0  871    4    2    3    2]
 [   6    2    0    1    3    3  941    0    2    0]
 [   0    4    4    1    0    0    0 1015    2    2]
 [   3    1    2    1    2    1    0    2  960    2]
 [   4    5    0    1    8    2    0    4    2  983]]
```

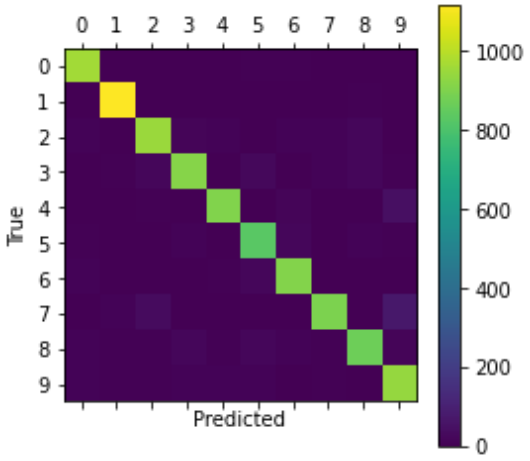
The true value vs predicted value graph after 10000 iterations:



The final confusion matrix obtained after 1000 iterations:

```
[[ 964    0    1    1    0    6    6    1    1    0]
 [   0 1118    4    2    0    1    3    0    7    0]
 [  11    2  950   14   12    0   12   11   19    1]
 [   4    6   20  916    0   26    1   10   19    8]
 [   1    3    5    1  906    0   15    1    2   48]
 [   7    1    2   11    2  833   20    1    9    6]
 [  10    4    2    1    8   19  912    0    2    0]
 [   2   12   34    4    4    2    0  898    2   70]
 [   9    5    5   19    8   21   10    7  873   17]
 [  11    7    6    9   13    9    0    7    4  943]]
```

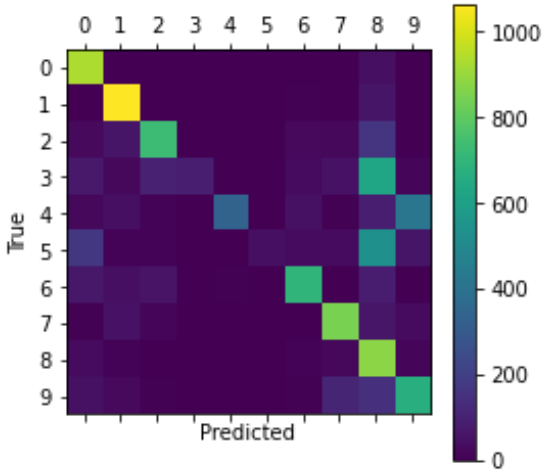
The true value vs predicted value graph after 1000 iterations:



The final confusion matrix obtained after 100 iterations:

```
[[ 935    0    0    0    0    0    1    2   42    0]
 [   0 1064    3    0    0    0    6    0   62    0]
 [  28   60  728    0    3    0   28   22  162    1]
 [  72   21   96   88    0    1   31   51  630   20]
 [  24   44   10    0  337    2   49    8   91  417]
 [ 172    9   11    0    0   43   30   32  536   59]
 [  68   45   56    0    8    0  695    0   85    1]
 [   8   52   19    0    0    0    3  849   64   33]
 [  31   10    3    0    0    0   11   23  881   15]
 [  49   25    6    0    1    0    5  112  146  665]]
```

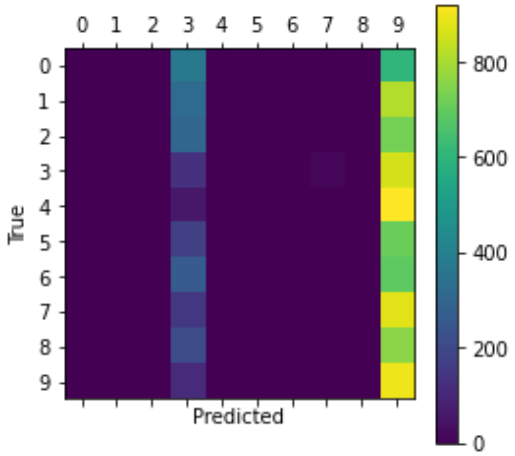
The true value vs predicted value graph after 100 iterations:



The final confusion matrix obtained after 1 iteration:

```
[[   0    0    0  374    0    0    0    0    0  606]
 [   0    0    0  321    0    0    0    0    0  814]
 [   0    0    0  305    0    0    0    0    0  727]
 [   0    0    0  132    0    0    0   16    0  862]
 [   0    0    0   61    0    0    0    0    0  921]
 [   0    0    0  175    0    0    0    2    0  715]
 [   0    0    0  265    0    0    0    0    0  693]
 [   0    0    0  150    0    0    0    0    0  878]
 [   0    0    0  211    0    0    0    1    0  762]
 [   0    0    0  111    0    0    0    1    0  897]]
```

The true value vs predicted value graph after 1 iteration:



The final confusion matrix obtained after 0 iteration:

```
[[   0    0    0  668    0    0    0    0    0  312]
 [   0    0    0  702    0    0    0    0    0  433]
 [   0    0    0  617    0    0    0    0    0  415]
 [   0    0    0  442    0    0    0    0    0  568]
 [   0    0    0  338    0    0    0    0    0  644]
 [   0    0    0  435    0    0    0    0    0  457]
 [   0    0    0  527    0    0    0    0    0  431]
 [   0    0    0  602    0    0    0    0    0  426]
 [   0    0    0  563    0    0    0    0    0  411]
 [   0    0    0  540    0    0    0    0    0  469]]
```

The true value vs predicted value graph after 0 iteration: