CS589 HW3

Aryan Tipnis

May 7, 2024

Correctness Verification :

To print the output of the file *backprop_example1.txt* run the file called *neuralNetworks.py* using the command *python neuralNetworks.py backprop_example1.txt*. The results will be printed in the terminal.

```
Training instance 1:
a1:             1.0             0.13

a2:             1.0             0.60181         0.58079

a3:             0.79403

Predicted Output for instance 1: 0.79403
Expected Output for instance 1: 0.9

Training instance 2:
a1:             1.0             0.42

a2:             1.0             0.60874         0.59484

a3:             0.79597

Predicted Output for instance 2: 0.79597
Expected Output for instance 2: 0.23

Cost Function:
        Cost, J, associated with instance 1: 0.366
        Cost, J, associated with instance 2: 1.276
        Final (regularized) cost, J, based on the complete training set: 0.82098
```

```
Running Backpropogation:

        Computing gradients based on training instance 1:
                delta3: -0.10597
                delta2: -0.0127 -0.01548

                Gradients of Theta2 based on training instance 1:
                -0.10597   -0.06378   -0.06155

                Gradients of Theta1 based on training instance 1:
                -0.01270   -0.00165
                -0.01548   -0.00201

        Computing gradients based on training instance 2:
                delta3: 0.56597
                delta2: 0.0674 0.08184

                Gradients of Theta2 based on training instance 2:
                0.56597   0.34452   0.33666

                Gradients of Theta1 based on training instance 2:
                0.06740   0.02831
                0.08184   0.03437

The entire training set has been processed. Computing the average (regularized) gradients:

        Final regularized gradients of Theta1:
                0.02735   0.01333
                0.03318   0.01618

        Final regularized gradients of Theta2:
                0.23000   0.14037   0.13756
```

To print the output of the file *backprop_example2.txt* run the file called *neuralNetworks.py* using the command *python neuralNetworks.py backprop_example2.txt*. The results will be printed in the terminal.

```
Training instance 1:
a1:             1.0             0.32            0.68

a2:             1.0             0.677           0.75384         0.58817         0.70566

a3:             1.0             0.87519         0.89296         0.8148

a4:             0.83318         0.84132

Predicted Output for instance 1: 0.83318 0.84132
Expected Output for instance 1: [0.75 0.98]

Training instance 2:
a1:             1.0             0.83            0.02

a2:             1.0             0.63472         0.69292         0.54391         0.64659

a3:             1.0             0.8602          0.88336         0.79791

a4:             0.82953         0.83832

Predicted Output for instance 2: 0.82953 0.83832
Expected Output for instance 2: [0.75 0.28]

Cost Function:
        Cost, J, associated with instance 1: 0.791
        Cost, J, associated with instance 2: 1.944
        Final (regularized) cost, J, based on the complete training set: 1.90351

Running Backpropogation:

        Computing gradients based on training instance 1:
                delta4: 0.08318 -0.13868
                delta3: 0.00639 -0.00925 -0.00779
                delta2: -0.00087 -0.00133 -0.00053 -0.0007

                Gradients of Theta3 based on training instance 1:
                0.08318   0.07280   0.07427   0.06777
                -0.13868   -0.12138   -0.12384   -0.11300

                Gradients of Theta2 based on training instance 1:
                0.00639   0.00433   0.00482   0.00376   0.00451
                -0.00925   -0.00626   -0.00698   -0.00544   -0.00653
                -0.00779   -0.00527   -0.00587   -0.00458   -0.00550

                Gradients of Theta1 based on training instance 1:
                -0.00087   -0.00028   -0.00059
                -0.00133   -0.00043   -0.00091
                -0.00053   -0.00017   -0.00036
                -0.00070   -0.00022   -0.00048

        Computing gradients based on training instance 2:
                delta4: 0.07953 0.55832
                delta3: 0.01503 0.05809 0.06892
                delta2: 0.01694 0.01465 0.01999 0.01622

                Gradients of Theta3 based on training instance 2:
                0.07953   0.06841   0.07025   0.06346
                0.55832   0.48027   0.49320   0.44549

                Gradients of Theta2 based on training instance 2:
                0.01503   0.00954   0.01042   0.00818   0.00972
                0.05809   0.03687   0.04025   0.03160   0.03756
                0.06892   0.04374   0.04775   0.03748   0.04456

                Gradients of Theta1 based on training instance 2:
                0.01694   0.01406   0.00034
                0.01465   0.01216   0.00029
                0.01999   0.01659   0.00040
                0.01622   0.01346   0.00032

The entire training set has been processed. Computing the average (regularized) gradients:

        Final regularized gradients of Theta1:
                0.00804   0.02564   0.04987
                0.00666   0.01837   0.06719
                0.00973   0.03196   0.05252
                0.00776   0.05037   0.08492

        Final regularized gradients of Theta2:
                0.01071   0.09068   0.02512   0.12597   0.11586
                0.02442   0.06780   0.04164   0.05308   0.12677
                0.03056   0.08924   0.12094   0.10270   0.03078

        Final regularized gradients of Theta3:
                0.08135   0.17935   0.12476   0.13186
                0.20982   0.19195   0.30343   0.25249
```

Wine Dataset :

3.

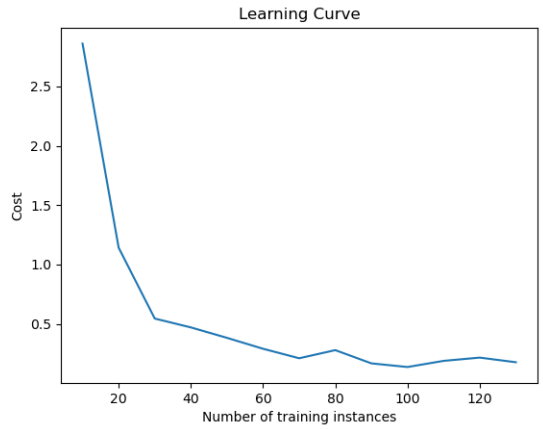
Layers	Neurons per Layer	Regularization Param	Accuracy	F1-score
1	4	0	97.30%	97.39%
1	4	0.01	98.31%	98.32%
1	4	0.2	96.77%	96.80%
1	8	0.01	96.66%	96.69%
2	2, 2	0.01	96.66%	96.75%
2	2, 2	0.5	94.42%	94.40%

For each of these architectures, the step-size of $\alpha = 0.9$ and first stopping criterion of $\epsilon = 0.1$ is used.

4. After comparing a number of hyper-parameters and architectures for Wine Dataset, it can be seen that a single hidden layer and a fewer number of neurons (like 4 or 8) result in higher accuracies and F1-score. I believe that changing the regularization parameters resulted the most in improving performance. Smaller values of λ especially values close to 0 like 0.01 or 0.0001 resulted in better performance. This could be because the wine dataset may be simple enough and can be solved using a small network, due to which the best accuracy may be achieved by using $\lambda = 0$. The α is set to 0.9 because having a large alpha value allows the weights to be updated fairly quickly and converge to decent local minimum.
Further, increasing the number of layers or neurons did not result in significant improvement of the model. This is likely due to the risk of overfitting, where the model starts to memorize noise in the training data rather than learning meaningful patterns.

5. Based on the above findings, I would choose a neural network with a single hidden layer of 4 neurons and a regularization parameter $\lambda = 0.01$, step-size of $\alpha = 0.9$ and the first stopping criterion of $\epsilon = 0.1$. This is because the above network results in an accuracy of 98.31% and a F1-score of 98.32% which is higher than any other model. With fewer parameters to learn, the model is less prone to overfitting and can capture the essential patterns in the data efficiently. Moreover, thus network is fairly simple and takes less time to train as compared to other networks with different hyper-parameters. This configuration strikes an optimal balance between model complexity and performance, leading to superior accuracy and generalization on the Wine Dataset.

6.



The step size value used is 0.9

US House Votes Dataset :

3.

Layers	Neurons per Layer	Regularization Param	Learning Rate	Accuracy	F1-score
1	4	0.01	0.8	95.86%	95.64%
1	8	0.01	0.8	96.32%	96.12%
1	8	0.1	0.8	96.08%	95.86%
2	8	0.3	0.8	91.25%	90.82%
2	16, 8	0.01	0.8	95.63%	95.41%
3	4, 4, 4	0.01	0.8	94.71%	94.44%

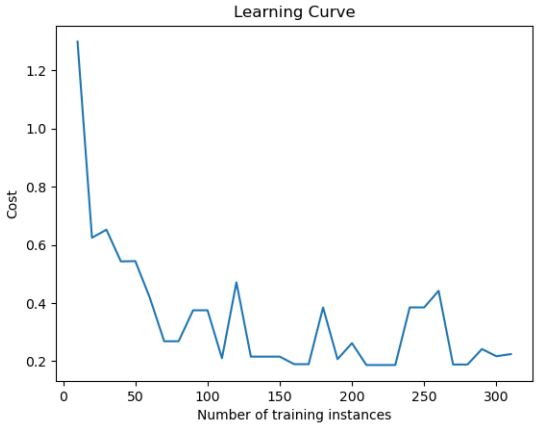
For each of these architectures, the stopping criterion of $\epsilon = 0.1$ is used.

4. After looking at this table, we can see that a lower regularization parameter like 0.01 or 0.1 gives a higher accuracy and F1-score values. As we started to increase the regularization parameter, there was a decrease in the metrics. Lower regularization parameters also result in faster training of the model. An architecture of single layer with 8 neurons resulted in a pretty high accuracy of 96.32% and F1-score of 96.12%. Further, increasing the neurons per layer or number of layers did not result in significant improvement of the dataset. This is likely due to the risk of overfitting, where the model starts to memorize noise in the training data rather than learning meaningful patterns. The a higher values of α is chosen so that the weights are updated fairly quickly.

5. I would choose a neural network with 1 hidden layer of 8 neurons with regularization param 0.01 and learning rate = 0.8 since it gave me fairly high accuracy. This architecture is relatively simple, thus striking a balance between complexity and computational efficiency. With only one hidden layer and a moderate number of neurons (8), the model is less prone to overfitting compared to deeper architectures. A lower regularization parameter leads to

better generalization performance on unseen data, which is crucial in real-life deployment scenarios where the model needs to perform well on new, unseen examples.

6.



The step size value used is 0.8

Extra Points 1:

Refer to the code in *neuralNetworks.py*. All algorithms are implemented using the vectorized form.

Extra Points 2: Breast Cancer Dataset

3.

Layers	Neurons per Layer	Regularization Param	Learning Rate	Accuracy	F1-score
1	4	0.01	0.9	95.7%	95.28%
1	8	0.01	0.9	96.13%	95.74%
1	16	0.01	2	95.79%	95.56%
1	2, 2	0.001	0.9	95.99%	95.59%
2	4, 4	0.001	0.9	95.84	95.41%
3	8, 8, 8	0.001	0.7	96.71%	96.37%

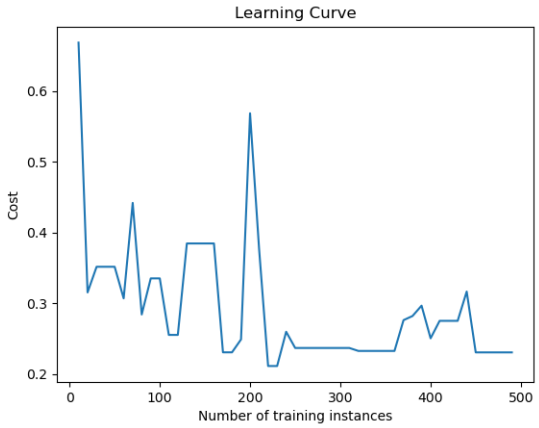
For each of these architectures, the stopping criterion of $\epsilon = 0.2$ is used.

4. After comparing a number of hyper-parameters and architectures for Cancer Dataset, it can be seen that increasing the number of layers and neurons per layer did not significantly improve the performance of the model. This is likely due to the risk of overfitting, where the model starts to memorize noise in the training data rather than learning meaningful patterns.I believe the stopping criteria ϵ would be the main factor that would improve the performance.

Smaller values of λ especially values close to 0 like 0.01 or 0.0001 resulted in better performance. The α can be seen to be varied in the examples above but having a large alpha value allows the weights to be updated fairly quickly and converge to decent local minimum.

5. Based on the above findings, I would choose a neural network with a single hidden layer of 8 neurons and a regularization parameter $\lambda = 0.01$, step-size of $\alpha = 0.9$ and the first stopping criterion. Due to the smaller stopping criteria, this will result in more accurate weight calculation which would result in higher accuracy and precision. With fewer parameters to learn, the model is less prone to overfitting and can capture the essential patterns in the data efficiently. Moreover, thus network is fairly simple and takes less time to train as compared to other networks with different hyper-parameters. This configuration strikes an optimal balance between model complexity and performance, leading to superior accuracy and generalization on the Cancer Dataset.

6.



The step size value used is 0.9