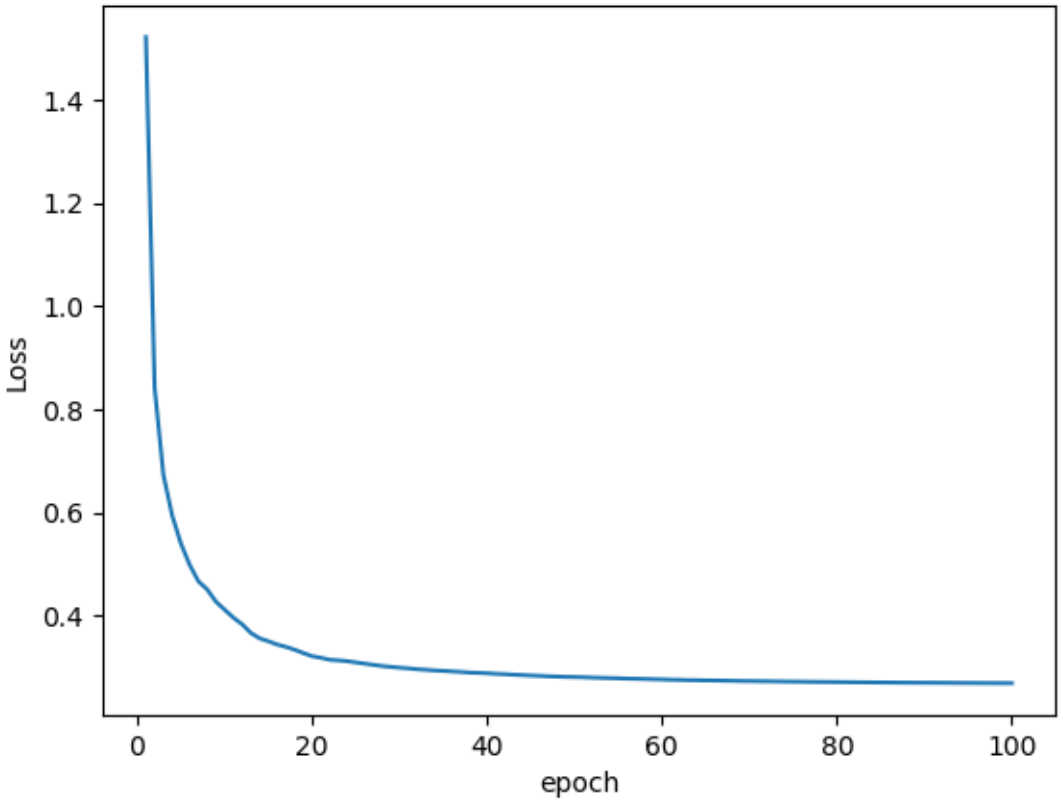
**scipy implementation**:

Result of checkNumericalGradient.py:

```
[[38. 38.]
 [12. 12.]]
The above two columns you get should be very similar.
Left-Your Numerical Gradient, Right-Analytical Gradient.

Norm of the difference: 2.1452381569477388e-12

Norm of the difference between numerical and analytical gradient
(should be < 1e-9)

As observed above, the norm of the difference between the two
solutions is sufficiently small. Hence, computeNumericalGradient.py is
sufficiently accurate.

Sample of visual comparison of computing gradient numerically(left)
vs. computing gradient using softmaxCost(right):

 [-0.00163356 -0.00163356]
 [-0.02236962 -0.02236962]
 [ 0.01252819  0.01252819]
 [-0.04757691 -0.04757691]
 [-0.07557783 -0.07557783]
 [-0.00957223 -0.00957223]
 [ 0.01762877  0.01762877]
 [-0.02603341 -0.02603341]
 [ 0.03862182  0.03862182]
 [ 0.01656232  0.01656232]
 [ 0.03731156  0.03731156]
 [-0.03321425 -0.03321425]
 [ 0.00067145  0.00067145]
 [ 0.00973924  0.00973924]
 [-0.01011385 -0.01011385]
 [-0.02362957 -0.02362957]
 [-0.00820524 -0.00820524]
 [ 0.01271434  0.01271434]
 [-0.03087598 -0.03087598]
 [ 0.03145277  0.03145277]

Norm of difference: 6.001464734720439e-10

As seen above, the norm of the difference between the two solutions is
sufficiently small, such that the softmaxCost function computes the
gradient with sufficient accuracy.

Plot of Loss vs. number of epochs (saved in Loss_vs_epoch.png):
```




Accuracy of softmax implementation: 92.520%.

The softmax implementation is acceptably accurate for this assignment.

Confusion matrix:

```
[[ 962    0    0    2    0    3    6    4    3    0]
 [   0 1113    3    2    0    1    4    2   10    0]
 [   5   10  925   14    8    3   14   10   40    3]
 [   3    1   21  916    0   27    4    9   22    7]
 [   1    1    4    3  919    0   10    3    9   32]
 [   9    3    2   36    9  772   12    8   34    7]
 [   9    3    5    2    8   18  910    2    1    0]
 [   1    7   22    6    6    1    0  953    3   29]
 [   8    9    6   25    8   26   11   11  862    8]
 [  10    7    1    7   24    9    0   23    8  920]]
```

As observed above, the confusion matrix is heavily concentrated along the diagonal, meaning the vast majority (92.52%) of the predicted labels are accurate. There is, however, a much smaller spread of mispredicted labels on either side of the diagonal.

**scikit implementaion**:

Accuracy of softmax implementation: 92.620%.

The softmax implementation is acceptably accurate for this assignment

Confusion matrix:

```
[[ 957    0    0    3    1    9    5    4    1    0]
 [   0 1111    4    2    0    2    3    2   11    0]
 [   5    7  931   15    8    3   16    8   36    3]
 [   4    1   18  917    1   24    4   11   22    8]
 [   1    2    8    3  914    0   10    6    8   30]
 [  10    3    3   36    8  781   13    6   27    5]
 [   9    3    6    3    6   16  912    2    1    0]
 [   1    8   21    5    6    2    0  954    3   28]
 [   9   11    8   25    7   25   12    6  859   12]
 [   9    8    0   10   22    7    0   20    7  926]]
```

As observed above, the confusion matrix is heavily concentrated along the diagonal, meaning the vast majority (92.62%) of the predicted labels are accurate. There is, however, a much smaller spread of mispredicted labels on either side of the diagonal.