4. Multinomial (Softmax) Regression and Gradient Descent

Daniel suggests that instead of building ten models, we can expand a single logistic regression model into a multinomial regression and solve it with similar gradient descent algorithm.

The main function which you will call to run the code you will implement in this section is `run_softmax_on_MNIST` in main.py (already implemented). In the appendix at the bottom of this page, we describe a number of the methods that are already implemented for you in `softmax.py` that will be useful.

In order for the regression to work, you will need to implement three methods. Below we describe what the functions should do. We have included some test cases in `test.py` to help you verify that the methods you have implemented are behaving sensibly.

You will be working in the file `part1/softmax.py` in this problem

Computing Probabilities for Softmax

5.0/5.0 points (graded)

Write a function `compute_probabilities` that computes, for each data point $x^{(i)}$, the probability that $x^{(i)}$ is labeled as j for $j = 0, 1, \ldots, k-1$

The softmax function h for a particular vector x requires computing

$$h(x) = \frac{1}{\sum_{j=1}^{k} e^{\theta_j \cdot x/\tau}} \begin{bmatrix} e^{\theta_1 \cdot x/\tau} \\ e^{\theta_2 \cdot x/\tau} \\ \vdots \\ e^{\theta_k \cdot x/\tau} \end{bmatrix},$$

where $\tau > 0$ is the **temperature parameter** . When computing the output probabilities (they should always be in the range $[0, 1]$), the terms $e^{\theta_j \cdot x/\tau}$ may be very large or very small, due to the use of the exponential function. This can cause numerical or overflow errors. To deal with this, we can simply subtract some fixed amount c from each exponent to keep the resulting number from getting too large. Since

$$h(x) = \frac{e^{-c}}{e^{-c}\sum_{j=1}^{k} e^{\theta_j \cdot x/\tau}} \begin{bmatrix} e^{\theta_1 \cdot x/\tau} \\ e^{\theta_2 \cdot x/\tau} \\ \vdots \\ e^{\theta_k \cdot x/\tau} \end{bmatrix}$$
$$= \frac{1}{\sum_{j=1}^{k} e^{[\theta_j \cdot x/\tau] - c}} \begin{bmatrix} e^{[\theta_1 \cdot x/\tau] - c} \\ e^{[\theta_2 \cdot x/\tau] - c} \\ \vdots \\ e^{[\theta_k \cdot x/\tau] - c} \end{bmatrix},$$

subtracting some fixed amount c from each exponent will not change the final probabilities. A suitable choice for this fixed amount is $c = \max_j \theta_j \cdot x/\tau$.

Reminder: You can implement this function locally first, and run `python test.py` in your project1 directory to validate basic functionality before checking against the online grader here.

Available Functions: You have access to the NumPy python library as `np` ; No need to import anything.

```
9       temp_parameter - the temperature parameter of softmax function (scalar)
10  Returns:
11      H - (k, n) NumPy array, where each entry H[j][i] is the probability that X[i] is labeled as j
12  """
13  #YOUR CODE HERE
14  n, d = X.shape
15  k, d = theta.shape
16  H = np.zeros((k, n))
17  for i in range(n):
18      c = max([theta[j,:]@X[i,:]/temp_parameter for j in range(k)])
19      z = sum([np.exp(theta[j,:]@X[i,:]/temp_parameter-c) for j in range(k)])
20      for j in range(k):
21          H[j, i] = np.exp(theta[j,:]@X[i,:]/temp_parameter-c) / z
22  return H
23
```

Press ESC then TAB or click outside of the code editor to exit

Correct

Test results

Hide output

CORRECT

Test: all pos

Testing x random, theta positive random and temp_parameter = 1

Output:

```
[[1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]]
Test completed
```

Test: all pos with fractional temp

Testing x random, theta positive random and 0 < temp_parameter < 1

Output:

```
[[0.45033808 0.44818748 0.67836741 0.29876617 0.37386131 0.55232439
  0.67048819 0.52337011 0.65615931 0.58764754]
 [0.54966192 0.55181252 0.32163259 0.70123383 0.62613869 0.44767561
  0.32951181 0.47662989 0.34384069 0.41235246]]
Test completed
```

Test: all pos with larger temp

Testing x random, theta positive random and temp_parameter > 1

Output:

```
[[0.56316173 0.65069067 0.56389467 0.64921522 0.57911735 0.58689194
  0.57608399 0.56631399 0.60254964 0.61195191]
 [0.43683827 0.34930933 0.43610533 0.35078478 0.42088265 0.41310806
  0.42391601 0.43368601 0.39745036 0.38804809]]
Test completed
```

Test: all zero

Testing x, theta = 0 and temp_parameter = 1

Output:

```
[[0.33333333 0.33333333]
 [0.33333333 0.33333333]
 [0.33333333 0.33333333]]
Test completed
```

Test: large magnitudes

Testing x large random, theta large positive random and 0.5 < temp_parameter < 1.5

Output:

```
[[1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]]
Test completed
```

Test: large temp

Testing x random, theta positive random and 100.0 < temp_parameter < 200.0

Output:

```
[[1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]]
Test completed
```

Test: negative theta

Testing x random, theta negative random and temp_parameter = 1

Output:

```
[[1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]]
Test completed
```

Test: small magnitudes

Testing x large random, theta large negative random and 0.5 < temp_parameter < 1.5

Output:

```
[[1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]]
Test completed
```

Test: theta zero

Testing x random, theta = 0 and temp_parameter = 1

Output:

```
[[1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]]
Test completed
```

Hide output

Submit You have used 1 of 20 attempts

Cost Function

5/5 points (graded)

Write a function `compute_cost_function` that computes the total cost over every data point.

The cost function $J(\theta)$ is given by: (Use natural log)

$$J(\theta) = -\frac{1}{n}\left[\sum_{i=1}^{n}\sum_{j=1}^{k}[[y^{(i)} == j]]\log\frac{e^{\theta_j \cdot x^{(i)}/\tau}}{\sum_{l=1}^{k} e^{\theta_l \cdot x^{(i)}/\tau}}\right] + \frac{\lambda}{2}\sum_{i=1}^{k}\sum_{j=0}^{d-1}\theta_{ij}^2$$

Available Functions: You have access to the NumPy python library as `np` and the previous function as `compute_probabilities`

```
15        c - the cost value (scalar)
16    """
17    #YOUR CODE HERE
18    n, d = X.shape
19    k, d = theta.shape
20    probs = compute_probabilities(X, theta, temp_parameter)
21    #probs = np.clip(compute_probabilities(X, theta, temp_parameter), 1e-15, 1-1e-15)
22    #print([np.sum([probs[j,i] for j in range(k)]) for i in range(n)])
23    J = 0
24    for i in range(n):
25        for j in range(k):
```

```
            J += (Y[i] == j) * (0 if probs[j, i] < 1e-15 else np.log(probs[j, i]))
    J *= (-1/n)
    J += (lambda_factor / 2)* np.sum(theta**2)
    return J

```

Press ESC then TAB or click outside of the code editor to exit

Correct

Test results

Hide output

CORRECT

Test: cost random data samples

Test cost function on random data with zero'd theta

Output:

```
Input: x:  [[ 1. 34. 85. 70. 73. 79. 82. 35. 96. 69.  8.]
 [ 1. 62. 88. 89. 98. 35. 46. 17. 80. 15. 75.]
 [ 1. 19. 90. 10. 83.  8.  1. 44. 69.  9.  5.]
 [ 1. 61. 73. 56. 46. 82. 60. 39. 46. 71. 50.]
 [ 1. 58. 44. 32. 51. 39. 40. 79. 74. 94. 26.]
 [ 1.  4. 60.  9. 90. 85.  9. 11. 65. 18. 60.]
 [ 1. 42. 88. 12. 76. 45. 35. 72. 50. 46. 68.]
 [ 1. 17. 13. 24. 87. 25.  1. 13. 27. 93. 98.]
 [ 1.  3. 65. 63. 25. 48. 66. 14. 87.  4. 98.]
 [ 1. 79. 69. 71. 78. 59. 47. 34. 54. 68. 31.]]  theta:  [[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]  temp_parameter:  1.0  lambda_factor:  0.0001
Output:  [1 1 1 1 1 1 1 1 1 1]
Cost:
2.302585
Test completed
```

Test: integration random data samples

Test cost function computation on random data points across two full iterations of softmax regression

Output:

```
Input: x:  [[ 1. 45. 55. 92. 28. 40. 18. 14. 22. 83. 93.]
 [ 1. 16. 53. 44. 56. 90. 21. 53.  6. 41. 92.]
 [ 1. 55. 23. 82. 91. 64. 66. 45.  2.  7. 82.]
 [ 1.  3. 41. 27. 38.  2. 28. 58.  4. 17. 78.]
 [ 1. 10. 10. 27. 69. 41. 37. 30. 70. 73. 36.]
 [ 1. 42. 84. 43. 56. 16. 90. 61.  9. 92. 84.]
 [ 1. 56. 89.  9.  2. 14. 63. 10. 83. 89. 26.]
 [ 1. 14.  5.  3. 89.  5. 19. 72. 61. 11. 49.]
 [ 1. 53. 18. 60. 60. 49. 60. 27. 73. 50.  8.]
 [ 1. 49. 78. 19.  9. 78. 46. 16. 50. 18. 72.]]  theta:  [[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]  temp_parameter:  1.0  lambda_factor:  0.0001
Output:  [1 1 1 1 1 1 1 1 1 1]
Costs across two iterations:
2.302585
0.081307
Test completed
```

Test: integration temperature

Test cost function computation on random data points across two iterations with non 1.0 temperature

Output:

```
Input: x:  [[ 1. 16. 47. 54.  1. 58. 42. 45. 39. 92. 47.]
 [ 1. 72. 60. 81. 26. 10. 85. 65. 39. 36. 65.]
 [ 1. 77. 99. 16. 79. 94. 87. 93. 18. 17. 76.]
 [ 1. 99. 43. 77. 67. 66. 77. 36. 48. 77. 77.]
 [ 1. 51. 90. 72. 30. 48. 36.  2.  7. 66. 36.]
 [ 1. 94. 90. 13. 72. 81. 40. 32. 15. 62. 34.]
 [ 1. 70. 11. 47. 83.  7. 23. 25. 22. 82. 54.]
 [ 1. 54. 16.  4. 21. 48. 81. 55. 51. 20. 39.]
 [ 1. 39. 68. 50. 82. 42. 20. 64. 24. 12. 65.]
 [ 1. 13. 26. 62. 92. 50. 49. 68. 26. 70. 56.]]  theta:  [[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]  temp_parameter:  0.5  lambda_factor:  0.0001
Output:  [1 1 1 1 1 1 1 1 1 1]
Costs across two iterations:
2.302585
2.302585
Test completed
```

Hide output

Submit You have used 6 of 20 attempts

✔ Correct (5/5 points)

Gradient Descent

5/5 points (graded)

Write a function `run_gradient_descent_iteration` that runs one step of batch gradient descent.

The partial derivative of $J(\theta)$ wrt a particular θ_j is given by:

$$\nabla_{\Theta_j} J(\theta) = -\frac{1}{\tau n} \sum_{i=1}^{n} [x^{(i)} ([[y^{(i)} == j]] - p(y^{(i)} = j | x^{(i)}, \theta))] + \lambda \theta_j$$

Available Functions: You have access to the NumPy python library as `np`, `computer_probabilities` which you previously implemented and `scipy.sparse` as `sparse`

You should use `sparse.coo_martix` so that your function can handle larger matrices efficiently (and not time out for the online graders). The sparse matrix representation can handle sparse matrices efficiently.

Hint

Show

```
        model for label j
    alpha - the learning rate (scalar)
    lambda_factor - the regularization constant (scalar)
    temp_parameter - the temperature parameter of softmax function (scalar)

Returns:
    theta - (k, d) NumPy array that is the final value of parameters theta
"""
#YOUR CODE HERE
n, d = X.shape
k, d = theta.shape
probs = compute_probabilities(X, theta, temp_parameter)
for j in range(k):
    theta[j,:] -= alpha*(-sum([X[i,:]*((Y[i] == j) - probs[j, i]) for i in range(n)]) / (temp_parameter*n) + lambda_factor*theta[j
return theta
```

Press ESC then TAB or click outside of the code editor to exit

Correct

Test results

Hide output

CORRECT

Test: integration random data samples

Test cost function computation on random data points across two full iterations of softmax regression

Output:

```
Input: x:  [[ 1. 52. 86. 72. 74. 91. 45. 25.  2. 93. 73.]
 [ 1. 80.  8.  7. 64. 11. 62. 83. 17. 65. 99.]
 [ 1. 95. 38. 22. 39. 18.  2. 86. 13. 73. 56.]
 [ 1. 25.  9. 65. 34. 68. 90. 73. 80. 99. 19.]
 [ 1. 93. 55. 88. 86. 61.  2. 58. 12. 58. 69.]
 [ 1. 10.  8. 67. 12. 78. 37. 41. 17. 25. 88.]
 [ 1. 69. 49. 98.  6. 82. 44. 49. 73. 18. 30.]
 [ 1. 47. 69. 27. 71.  7. 61. 92. 11. 34. 61.]
 [ 1. 52. 88. 15. 87. 98.  2. 54.  5. 10. 28.]
 [ 1. 75. 60. 28. 29. 43. 14. 61. 42. 78. 70.]]  theta:  [[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]  temp_parameter:  1.0  lambda_factor:  0.0001
Output:  [1 1 1 1 1 1 1 1 1 1]
Costs across two iterations:
2.302585
0.106401
Test completed
```

Test: run gradient descent

Running one iteration of gradient descent

Output:

```
Input: x:  [[ 1. 84.  6. 98.  8. 95. 48. 11. 55. 26. 45.]
 [ 1. 67. 76. 86. 50. 62. 93. 11. 14. 55. 78.]
 [ 1. 35. 86. 45. 41. 64. 76. 82. 53. 70. 54.]
 [ 1. 18. 27. 72. 54. 19.  7. 48. 73. 76. 84.]
 [ 1.  7. 79. 51. 60. 22. 41. 73. 28.  4. 22.]
 [ 1. 91. 21. 90. 52. 79. 60. 76. 36. 49.  2.]
 [ 1. 27. 50. 76. 92. 36. 61. 51. 72. 14. 95.]
 [ 1. 39. 16. 77.  3. 91. 76. 38. 15. 30. 35.]
 [ 1. 27. 28. 20. 83. 87. 32. 14. 35. 96. 46.]
 [ 1. 36. 36. 86. 68.  9. 81. 23. 14. 41. 95.]]  theta:  [[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]  temp_parameter:  1.0  lambda_factor:  0.0001
Output:  [1 1 1 1 1 1 1 1 1 1]
Theta after gradient descent update:
[['-0.029999' '-1.292961' '-1.274962' '-2.102937' '-1.532954' '-1.691949'
  '-1.724948' '-1.280962' '-1.184964' '-1.382959' '-1.667950']
 ['0.269992' '11.636651' '11.474656' '18.926432' '13.796586' '15.227543'
  '15.524534' '11.528654' '10.664680' '12.446627' '15.011550']
 ['-0.029999' '-1.292961' '-1.274962' '-2.102937' '-1.532954' '-1.691949'
  '-1.724948' '-1.280962' '-1.184964' '-1.382959' '-1.667950']
 ['-0.029999' '-1.292961' '-1.274962' '-2.102937' '-1.532954' '-1.691949'
  '-1.724948' '-1.280962' '-1.184964' '-1.382959' '-1.667950']
 ['-0.029999' '-1.292961' '-1.274962' '-2.102937' '-1.532954' '-1.691949'
  '-1.724948' '-1.280962' '-1.184964' '-1.382959' '-1.667950']
 ['-0.029999' '-1.292961' '-1.274962' '-2.102937' '-1.532954' '-1.691949'
  '-1.724948' '-1.280962' '-1.184964' '-1.382959' '-1.667950']
 ['-0.029999' '-1.292961' '-1.274962' '-2.102937' '-1.532954' '-1.691949'
  '-1.724948' '-1.280962' '-1.184964' '-1.382959' '-1.667950']
 ['-0.029999' '-1.292961' '-1.274962' '-2.102937' '-1.532954' '-1.691949'
  '-1.724948' '-1.280962' '-1.184964' '-1.382959' '-1.667950']
 ['-0.029999' '-1.292961' '-1.274962' '-2.102937' '-1.532954' '-1.691949'
  '-1.724948' '-1.280962' '-1.184964' '-1.382959' '-1.667950']
 ['-0.029999' '-1.292961' '-1.274962' '-2.102937' '-1.532954' '-1.691949'
  '-1.724948' '-1.280962' '-1.184964' '-1.382959' '-1.667950']]
Test completed
```

Test: run gradient descent temp

Running one iteration of gradient descent with temperature != 1

Output:

```
Input: x:  [[ 1. 38. 43. 61. 27.  8.  5. 62. 53. 24. 97.]
 [ 1.  1.  8. 62. 81. 15. 66.  3. 66. 94. 79.]
 [ 1. 43. 52. 20. 93. 69. 49. 36.  2. 79. 28.]
 [ 1. 29. 36. 15. 52. 91. 47. 56. 97. 62. 23.]
 [ 1. 81. 74. 59. 48. 66. 20. 87. 64. 99. 41.]
 [ 1. 86. 62. 37. 82. 19. 20. 65. 20. 25. 12.]
 [ 1. 37. 87. 59. 26. 27. 48. 95. 21. 25. 41.]
 [ 1.  7. 85. 71. 60. 80. 70. 80. 94. 88. 89.]
 [ 1. 56. 16. 50.  1. 51. 30. 25. 16. 45. 73.]
 [ 1. 78. 96. 73. 77. 95. 80. 51. 39. 45. 60.]]  theta:  [[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]  temp_parameter:  0.5  lambda_factor:  0.0001
Output:  [1 1 1 1 1 1 1 1 1 1]
Theta after gradient descent update:
[['-0.059998' '-2.735918' '-3.353899' '-3.041909' '-3.281902' '-3.125906'
  '-2.609922' '-3.359899' '-2.831915' '-3.515895' '-3.257902']
 ['0.539984' '24.623261' '30.185094' '27.377179' '29.537114' '28.133156'
  '23.489295' '30.239093' '25.487235' '31.643051' '29.321120']
 ['-0.059998' '-2.735918' '-3.353899' '-3.041909' '-3.281902' '-3.125906'
  '-2.609922' '-3.359899' '-2.831915' '-3.515895' '-3.257902']
 ['-0.059998' '-2.735918' '-3.353899' '-3.041909' '-3.281902' '-3.125906'
  '-2.609922' '-3.359899' '-2.831915' '-3.515895' '-3.257902']
 ['-0.059998' '-2.735918' '-3.353899' '-3.041909' '-3.281902' '-3.125906'
  '-2.609922' '-3.359899' '-2.831915' '-3.515895' '-3.257902']
 ['-0.059998' '-2.735918' '-3.353899' '-3.041909' '-3.281902' '-3.125906'
  '-2.609922' '-3.359899' '-2.831915' '-3.515895' '-3.257902']
 ['-0.059998' '-2.735918' '-3.353899' '-3.041909' '-3.281902' '-3.125906'
  '-2.609922' '-3.359899' '-2.831915' '-3.515895' '-3.257902']
 ['-0.059998' '-2.735918' '-3.353899' '-3.041909' '-3.281902' '-3.125906'
  '-2.609922' '-3.359899' '-2.831915' '-3.515895' '-3.257902']
 ['-0.059998' '-2.735918' '-3.353899' '-3.041909' '-3.281902' '-3.125906'
  '-2.609922' '-3.359899' '-2.831915' '-3.515895' '-3.257902']
 ['-0.059998' '-2.735918' '-3.353899' '-3.041909' '-3.281902' '-3.125906'
  '-2.609922' '-3.359899' '-2.831915' '-3.515895' '-3.257902']]
Test completed
```

Hide output

Submit

You have used 2 of 20 attempts

✔ Correct (5/5 points)

Test Error on Softmax Regression

1/1 point (graded)

Finally, report the final test error by running the `main.py` file, using the temperature parameter $\tau = 1$. If you have implemented everything correctly, the error on the test set should be around 0.1, which implies the linear softmax regression model is able to recognize MNIST digits with around 90 percent accuracy.

Note: For this project we will be looking at the error rate defined as the fraction of labels that don't match the target labels, also known as the "gold labels" or ground truth. In other contexts, you might want to consider other performance measures we've learned about in this class, including precision and recall.

Please enter the **test error** of your Softmax algorithm (copy the output from the `main.py` run).

0.10050000000000003 ✔

Submit You have used 1 of 20 attempts

✔ Correct (1/1 point)

Discussion

Hide Discussion

Topic: Unit 2 Nonlinear Classification, Linear regression, Collaborative Filtering (2 weeks):Project 2: Digit recognition (Part 1) / 4. Multinomial (Softmax) Regression and Gradient Descent

Add a Post

All Posts

How long does softmax regression take to run?

question posted about 17 hours ago by **KyawSint**

Hi, I completed and got the gradient descent marked correctly. For the test error, based on printing the iteration in softmax_regression(), I'm estimating it would take 20 minutes to complete. Is this normal?

This post is visible to everyone.

Add a Response

1 response

ieyasu2014
about 11 hours ago

I evaluated it, too, and it took 1,660 seconds.

I used scipy.sparse, but did not become so fast.

Is how to use scipy.sparse bad?

Add a comment

Showing all responses

Add a response:

Preview

Submit

Learn About Verified Certificates