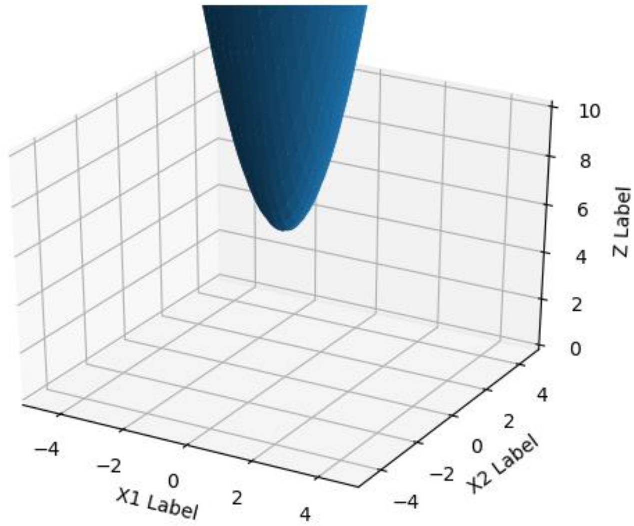
Name: Nattapat Juthaprachakul, Student ID: 301350117

CMPT-726 Machine Learning: Assignment 2

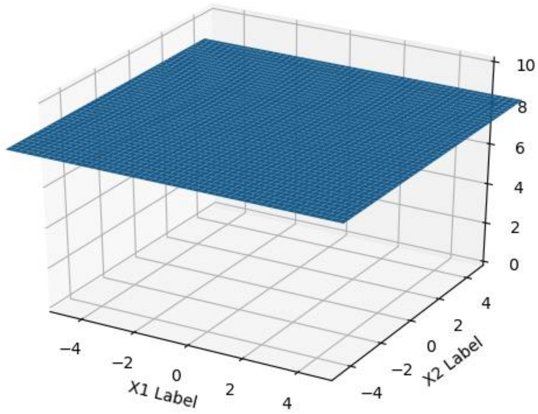
1. **Softmax for multi-class classification**
   1.1 ANS: The probability at the green point for each class is equally likely to be any of these 3 classes (0.33 percent for each class).
   1.2 ANS: The probability of input (green point) depends on which direction the green dot heads to (moving along the line in this case). For example, if the green dot moves downward along the red line, the probability of input x being class 1 and 3 (region 1 and 3) is more likely than class 2 (region 2). Also, the more the green dot moves downward, the more unlikely the input is classified as class 2 (region 2). In sum, when green dot moves along red line downward, the probability of input classified to be either class 1 or 3 is equally likely but unlikely to be class 2. This logic applies to both moving leftward and rightward as well. (Moving leftward probability of input being classified to be either class 2 and 3 is equally likely but unlikely to be class 1 and moving rightward probability of input being classified to be either class 1 and 2 is equally likely but unlikely to be class 3.)
   1.3 ANS: The probability of inputs being classified as the following class depends on the region that the green dot is in. For example, if the green dot is in region 1, the input is likely to be classified as class 1 than classified as class 2 and 3.

2. Generalized Linear model for classification

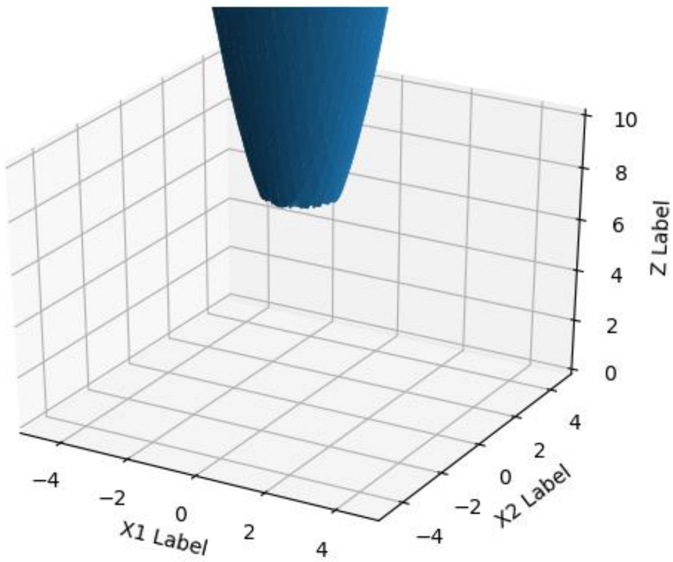
2.1



2.2



2.3



3. 3.1)ANS:

$$a_2^{(2)} = w_{21}^{(1)}x_1 + w_{22}^{(1)}x_2 + w_{23}^{(1)}x_3 \quad , \quad z_2^{(2)} = h(a_2^{(2)})$$

$$E_n(w) = \frac{1}{2}(y(x_n,w) - t_n)^2 = \frac{1}{2}(z^{(4)} - t_n)^2$$

$$\rightarrow \frac{\partial E_n(w)}{\partial a_1^{(4)}} = \frac{\partial E_n(w)}{\partial z^{(4)}}\frac{\partial z^{(4)}}{\partial a_1^{(4)}} = \underbrace{(z^{(4)} - t_n)[\cancel{h'(a_1^{(4)})}\,1]}_{\square} \#$$

$$\rightarrow \frac{\partial E_n(w)}{\partial w_{12}^{(3)}} = \frac{\partial E_n(w)}{\partial z^{(4)}}\frac{\partial z^{(4)}}{\partial a_1^{(4)}}\frac{\partial a_1^{(4)}}{\partial w_{12}^{(3)}} = \square \times \frac{\partial a_1^{(4)}}{\partial w_{12}^{(3)}}$$

$$\text{since } \frac{\partial a_1^{(4)}}{\partial w_{12}^{(3)}} = w_{11}^{(3)}z_{11}^{(3)} + w_{12}^{(3)}z_{12}^{(3)} + w_{13}^{(3)}z_{13}^{(3)}$$

$$= 0 + z_{12}^{(3)} + 0 = z_{12}^{(3)}$$

$$\frac{\partial E_n(w)}{\partial w_{12}^{(3)}} = (z^{(4)} - t_n)[\cancel{h'(a_1^{(4)})}\,1](z_{12}^{(3)}) \#$$

3.2)ANS:

$$\rightarrow \frac{\partial E_n(w)}{\partial a_1^{(3)}} = \frac{\partial E_n(w)}{\partial z^{(4)}}\frac{\partial z^{(4)}}{\partial a_1^{(4)}}\frac{\partial a_1^{(4)}}{\partial z_1^{(3)}}\frac{\partial z_1^{(3)}}{\partial a_1^{(3)}}$$

$$= \delta_1^{(4)}\,\cancel{h'(a_1^{(4)})}\,1\,[w_{11}^{(3)}]\,h'(a_1^{(3)}) \#$$

$$\rightarrow \frac{\partial E_n(w)}{\partial w_{11}^{(2)}} = \delta_1^{(4)} h'(a_1^{(4)})\, w_{11}^{(3)} h'(a_1^{(3)}) \frac{\partial a_1^{(3)}}{\partial w_{11}^{(2)}}$$

$$= \delta_1^{(4)}\,\cancel{h'(a_1^{(4)})}\,1\, w_{11}^{(3)} h'(a_1^{(3)})\, z_{11}^{(2)} \#$$

3.3)ANS:

$$\rightarrow \frac{\partial E_n(w)}{\partial a_3^{(2)}} = \sum_k \frac{\partial E_n}{\partial a_k}\frac{\partial a_k}{\partial a_j} \qquad \text{recall } \delta_j = \sum_k \underbrace{\frac{\partial E_n}{\partial a_k}}_{\delta_k}\frac{\partial a_k}{\partial a_j}$$

$$= \delta_1^{(3)}\frac{\partial a_1^{(3)}}{\partial a_3^{(2)}} + \delta_2^{(3)}\frac{\partial a_2^{(3)}}{\partial a_3^{(2)}} + \delta_3^{(3)}\frac{\partial a_3^{(3)}}{\partial a_3^{(2)}}$$

$$= \delta_1^{(3)}\frac{\partial a_1^{(3)}}{\partial z_3^{(2)}}\frac{\partial z_3^{(2)}}{\partial a_3^{(2)}} + \delta_2^{(3)}\frac{\partial a_2^{(3)}}{\partial z_3^{(2)}}\frac{\partial z_3^{(2)}}{\partial a_3^{(2)}} + \delta_3^{(3)}\frac{\partial a_3^{(3)}}{\partial z_3^{(2)}}\frac{\partial z_3^{(2)}}{\partial a_3^{(2)}}$$

$$= \delta_1^{(3)} w_{13}^{(2)} h'(a_3^{(2)}) + \delta_2^{(3)} w_{23}^{(2)} h'(a_3^{(2)}) + \delta_3^{(3)} w_{13}^{(2)} h'(a_3^{(2)})$$

$$\#$$

$$\rightarrow \frac{\partial E_n(w)}{\partial w_{11}^{(1)}} = \frac{\partial E_n(w)}{\partial a_1^{(2)}}\frac{\partial a_1^{(2)}}{\partial w_{11}^{(1)}}$$

$$= \left[\left(\delta_1^{(3)} w_{11}^{(2)} + \delta_2^{(3)} w_{21}^{(2)} + \delta_3^{(3)} w_{31}^{(2)}\right] h'(a_1^{(2)})\right] x_1$$

$$\#$$

4. 4.1)ANS

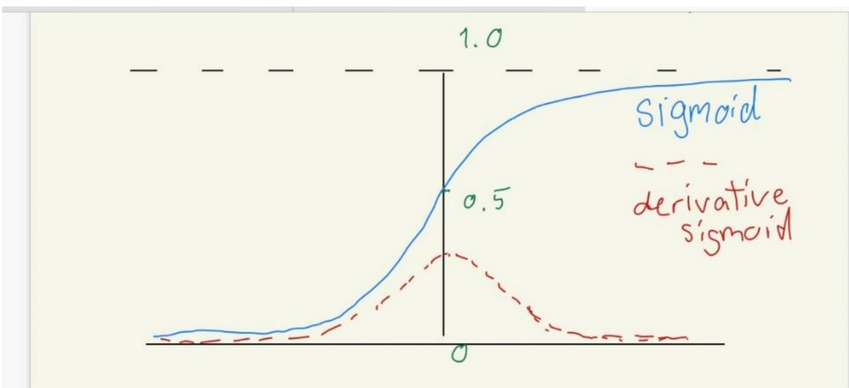
Q4)

4.1)
$$\frac{\partial E_n(w)}{\partial w_{11}^{(1)}} = \frac{\partial E_n(w)}{\partial z_1^{(153)}}\frac{\partial z_1^{(153)}}{\partial a_1^{(153)}}\frac{\partial a_1^{(153)}}{\partial w_{11}^{(152)}}\frac{\partial w_{11}^{(152)}}{\partial z_1^{(152)}}\frac{\partial z_1^{(152)}}{\partial a_1^{(152)}}\frac{\partial a_1^{(152)}}{\partial w_{11}^{(151)}}\cdots\frac{\partial a_1^{(2)}}{\partial w_{11}^{(1)}}$$

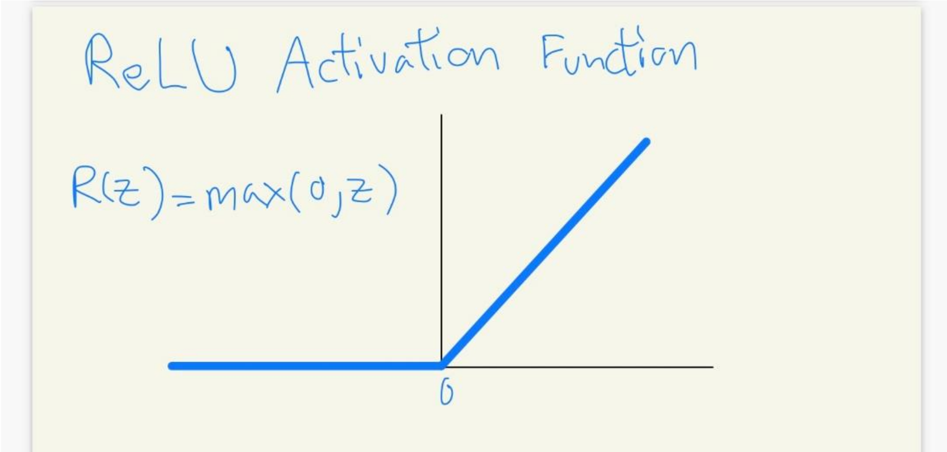
$$= \frac{\partial E_n(w)}{\partial z_1^{(153)}}\left[\prod_{i=2}^{153}\frac{\partial z_1^{(i)}}{\partial a_1^{(i)}}\frac{\partial a_1^{(i)}}{\partial w_{11}^{(i-1)}}\right]$$

4.2)ANS: The learning will be very slow if the learning rate is very small and the area of the update is in the top and bottom curve (very flat area). The update is slow because the derivative is very small (small rate of change/update). For the softmax function, the gradient of weight could be small or zero when we do backpropagate to modify the weight to minimize the cost function through many layers and many connected nodes as the majority of derivative value of sigmoid lies between 0 and 0.25 (dotted red curve in the graph below), the multiplication of number between 0 and 1 many times will make the values smaller over time and become zero in some connection.

Sol[n]
$$\frac{\partial \sigma(a)}{\partial a} = \frac{\partial}{\partial a}\left[\frac{1}{1+e^{-a}}\right] = \frac{\partial}{\partial a}\left[(1+e^{-a})^{-1}\right]$$
$$= -(1+e^{-a})^{-2}(-e^{-a})$$
$$= \frac{e^{-a}}{(1+e^{-a})^2} = \frac{1}{1+e^{-a}}\cdot\frac{e^{-a}}{1+e^{-a}}$$
$$= \frac{1}{1+e^{-a}} \times \frac{(1+e^{-a})-1}{1+e^{-a}}$$
$$= \frac{1}{1+e^{-a}}\left[\frac{1+e^{-a}}{1+e^{-a}} - \frac{1}{1+e^{-a}}\right]$$
$$= \left(\frac{1}{1+e^{-a}}\right)\left(1 - \frac{1}{1+e^{-a}}\right)$$
$$= \sigma(a)(1-\sigma(a))$$



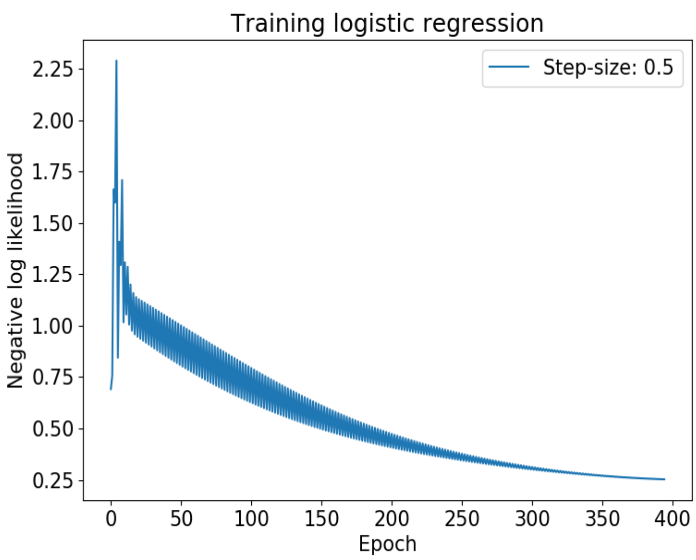
4.3)ANS: When the inputs to ReLU is equal or smaller than zero (the large negative bias term is learned which makes weighted sum of inputs becomes zero or negative), the output of ReLU will be zero, making the derivative become zero as well.

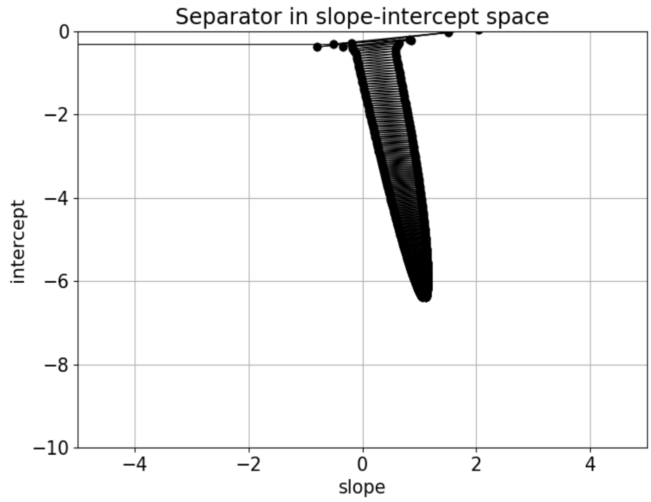


4.4)ANS: The gradient of weight could be zero when we do update/backpropagate to modify the weight to minimize the cost function through many layers and many connected nodes (bipartite). This might be the result when the majority of output from ReLU could become zero as the summation of weight term and bias is zero or negative. The function gradient at zero becomes zero as well.(the gradient descent learning will not alter the weights) This situation is called ‘dead’ ReLU.

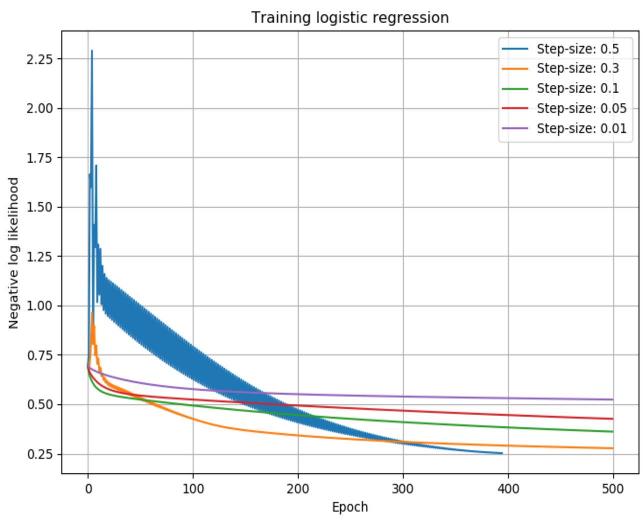
5. 5.1)ANS: When learning rate is too large, it causes drastic weight updates which lead to divergent behaviors (the oscillating curves) observed in the left and right pictures.

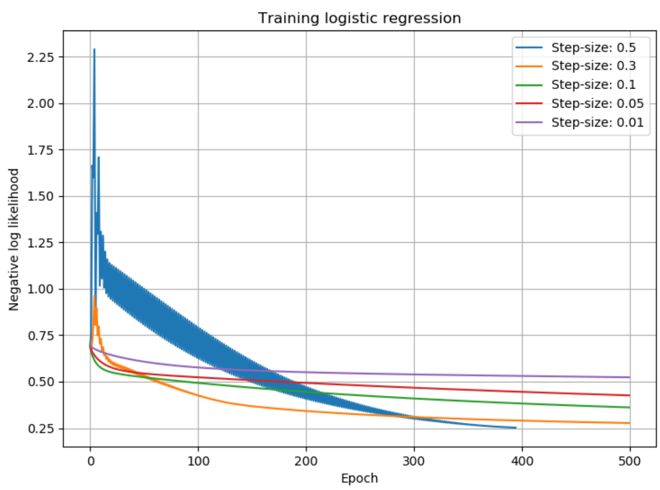
The performance of the model (such as its loss on the training dataset) will oscillate over training epochs. Oscillating performance is caused by weights that diverge.



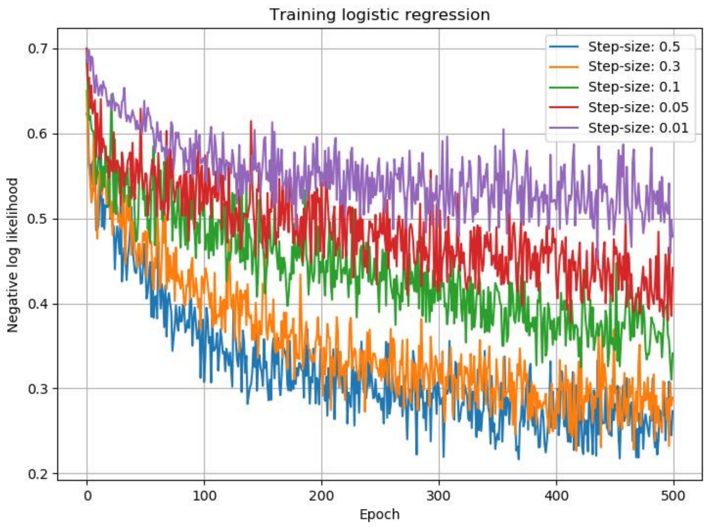


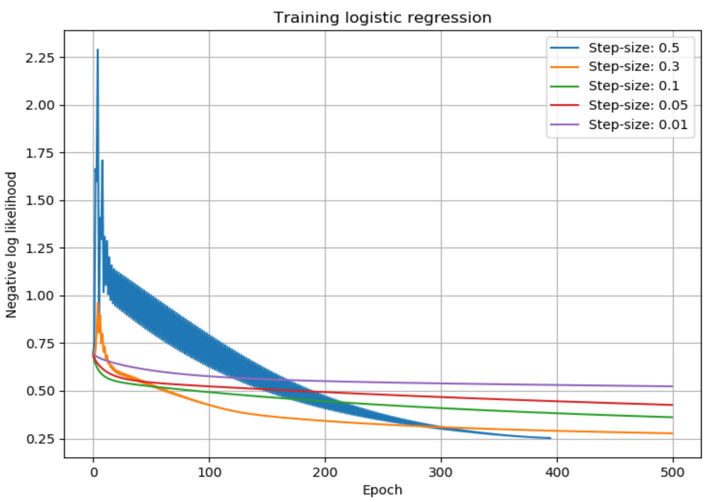
5.2)ANS: in our case, the 0.5 learning rate finds the lowest training error at the fewest number of epoch. As we can see the larger the learning rate is, the faster the training becomes (given the same training error).

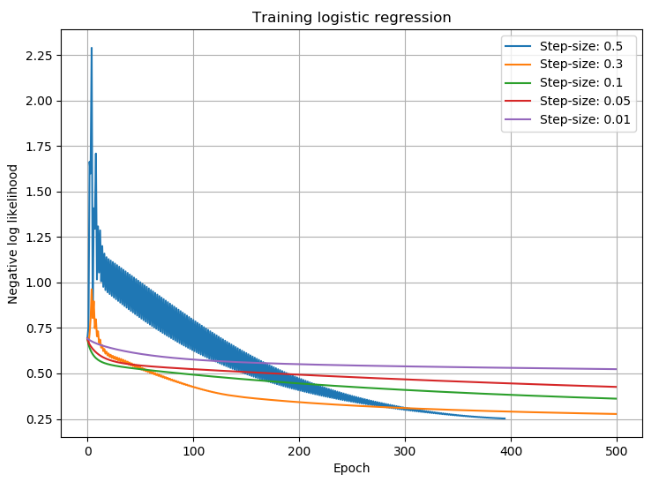




5.3)ANS: in our example, it is not so obvious that which update technique is better than another in term of speed(number of epoch) and performance(error rate). We can see that for the SGD graph is very oscillating since it calculates one data point then makes an update. However, in the setting with big data sets, Gradient Descent takes time to calculate cost or gradient as it needs to sum over all data points. Nonetheless, we do not need to have exact gradient to minimize the cost function in a given iteration. The approximation of gradient is enough; therefore, the Stochastic Gradient descent approximates the gradient using just only one data point at a time which in turn saves lots of time compared to summing over all data.







6. **Fine-Tuning a Pre-trained Network**

Settings: gaming laptop with a single Nvidia GTX 1050Ti

6.1) Main task that I do:

-Write a Python function to be used at the end of training that generates HTML output showing each test image and its classification scores. You could produce an HTML table output for example. (You can convert the HTML output to PDF or use screen shots.)

6.2) Other tasks:

-Try applying L2 regularization to the coefficients in the small networks we added.

-Try modifying the structure of the new layers that were added on top of ResNet20.

The main code is to evaluate the models and to save loss and classification score (max value from softmax output) for each test image. The next code generates each test image with its classification score and saves them all in our specified folder. The final code is to generate HTML table and convert it into PDF. I attached the filename: image_score_html.html and test_image.pdf.

I also created new additional layers (modifying the original assignment code of just one layer) on top of ResNet20 as follow: [feed forward layer -> batch normalization -> feed forward layer -> softmax output] with dropout technique, L2 regularization(weight decay) and specifiable number of hidden nodes.

Results: different epochs on training set of 50k inputs and 1 epoch on test set of 10k inputs.

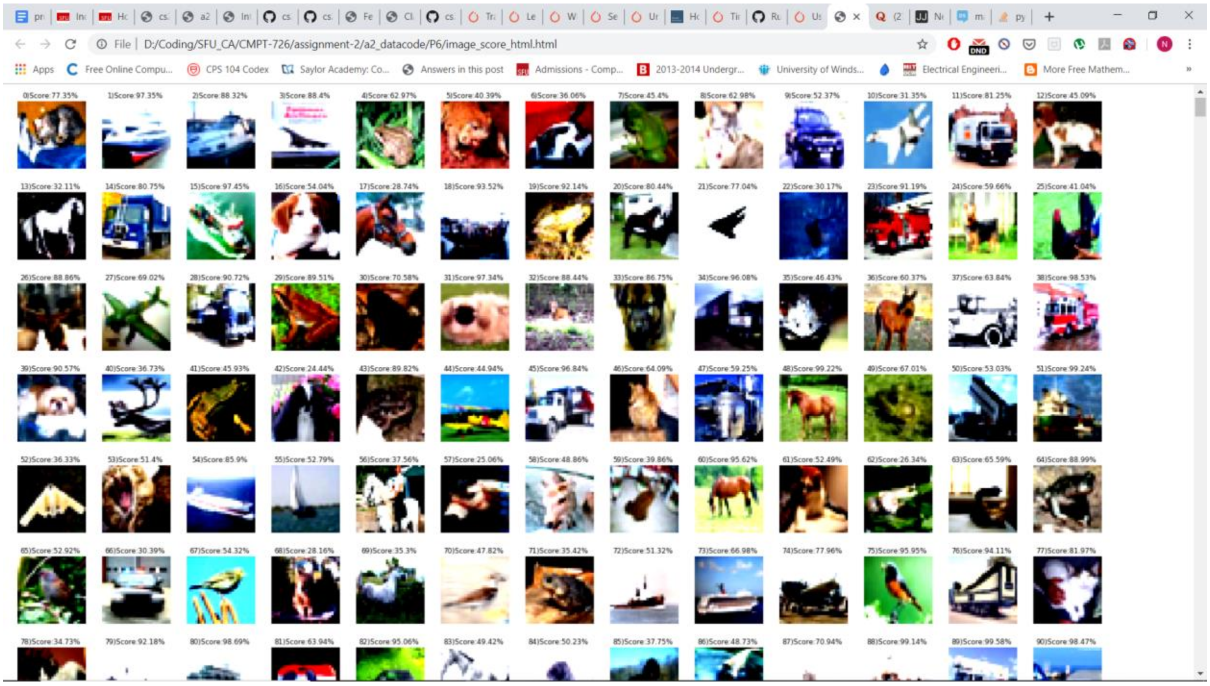
Note: Original feed-forward layer(64 nodes) denotes as fc(64)

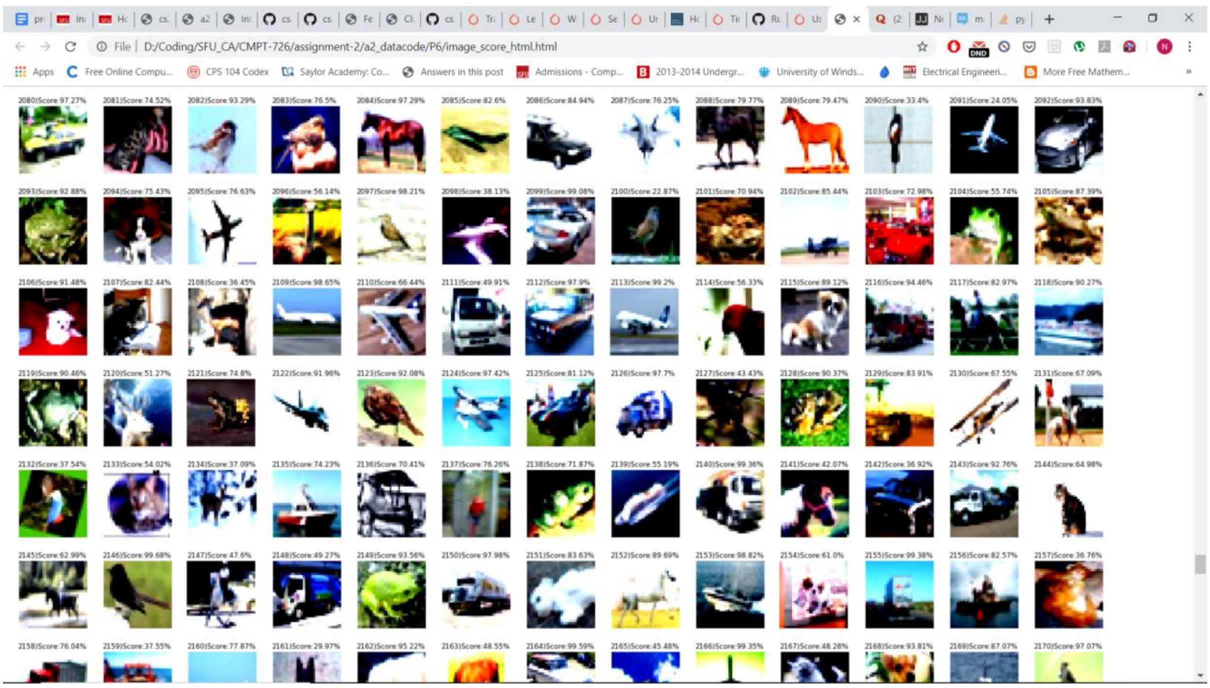
Additional feed-forward layer(256 nodes) denotes as fc(256)

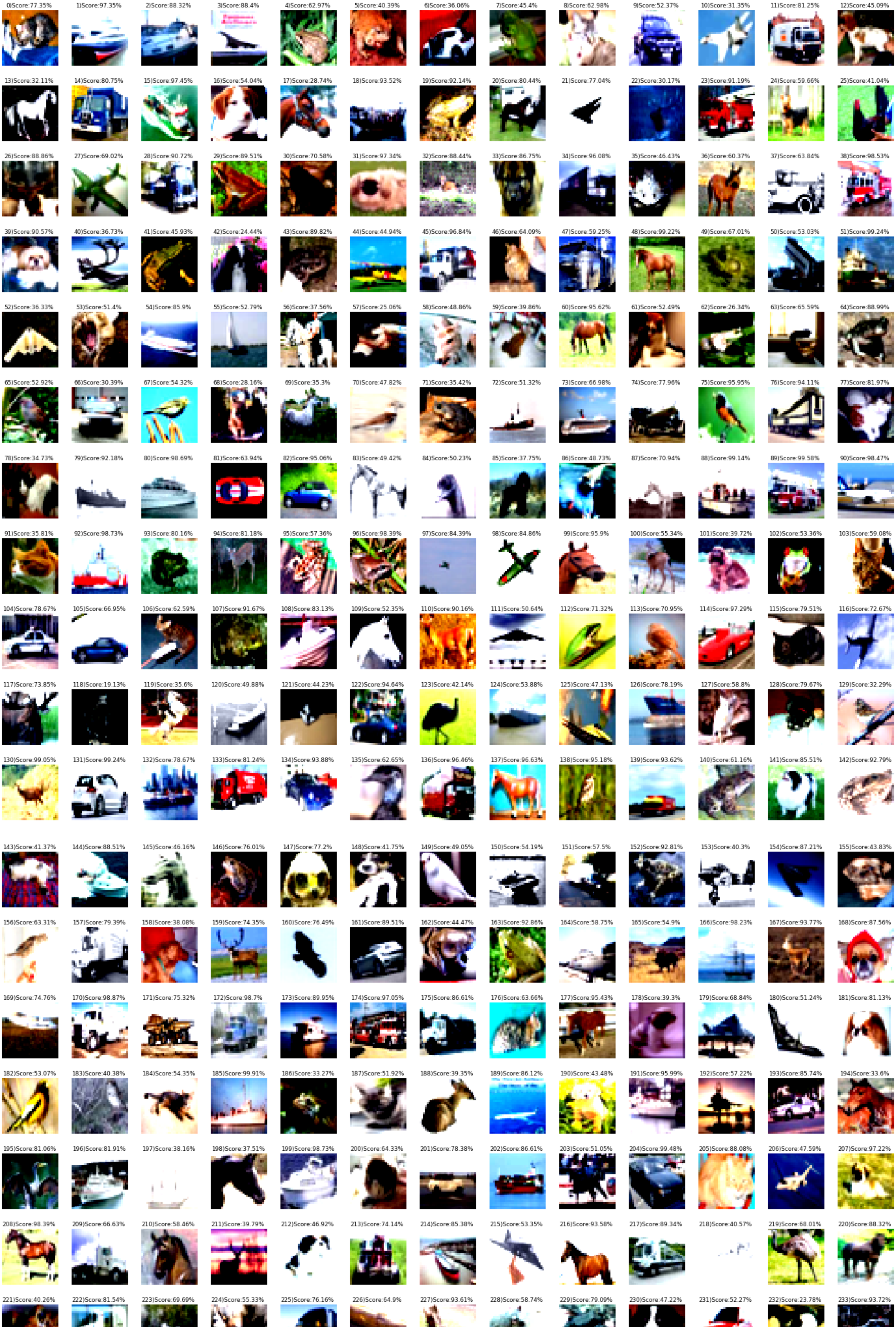
1. 1 epoch training, ResNet20 + fc(64)
   Running optimization on: fc(64)
   Accuracy: 65.10%
2. 10 epoch training, ResNet20 + fc(64)
   Running optimization on: fc(64)
   Accuracy: 68.55%
3. 10 epoch training, ResNet20 + fc(64), L2 regularization(0.001)
   Running optimization on: fc(64)
   Accuracy: 68.65%
4. 10 epoch training, ResNet20 + fc(64) + fc(256), L2 regularization(0.001)
   Running optimization on: fc(256)
   Accuracy: 68.52%
5. 10 epoch training, ResNet20 + fc(64)+ fc(256)
   Running optimization on: ResNet20 + fc(256)
   Accuracy: 84.46%
6. 10 epoch training, ResNet20 + fc(64)
   Running optimization on: ResNet20 + fc(64)

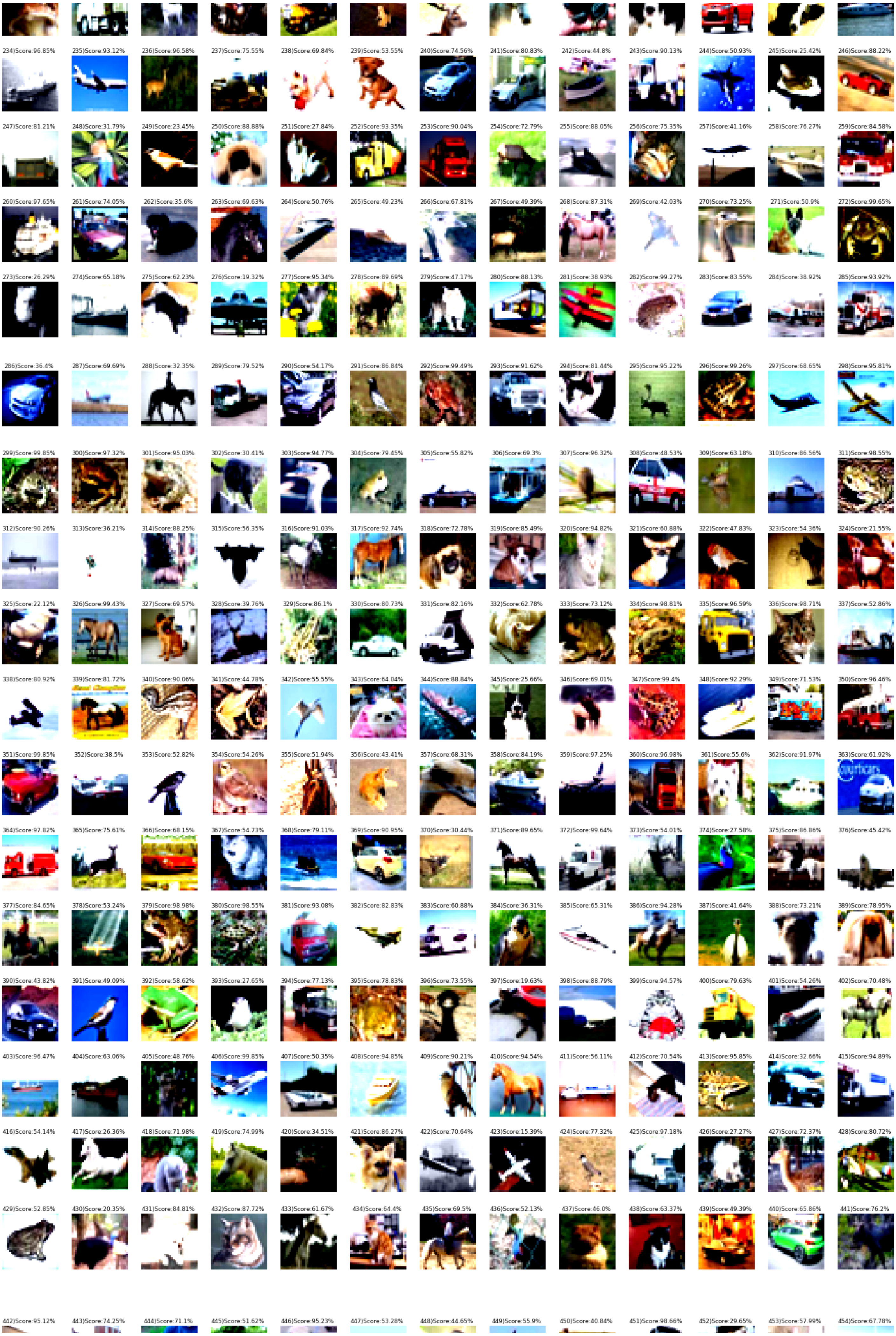
Accuracy: 84.44%

7. 10 epoch training, ResNet20 + fc(64)
   Running optimization on: ResNet20
   Accuracy: 80.33%
8. 10 epoch training, ResNet20 + fc(64), dropout at 0.2
   Running optimization on: ResNet20 + fc(64)
   Accuracy: 84.55%






0)Score:77.35%
1)Score:97.35%
2)Score:88.32%
3)Score:88.4%
4)Score:62.97%
5)Score:40.39%
6)Score:36.06%
7)Score:45.4%
8)Score:62.98%
9)Score:52.37%
10)Score:31.35%
11)Score:81.25%
12)Score:45.09%
13)Score:32.11%
14)Score:80.75%
15)Score:97.45%
16)Score:54.04%
17)Score:28.74%
18)Score:93.52%
19)Score:92.14%
20)Score:80.44%
21)Score:77.04%
22)Score:30.17%
23)Score:91.19%
24)Score:59.66%
25)Score:41.04%
26)Score:88.86%
27)Score:69.02%
28)Score:90.72%
29)Score:89.51%
30)Score:70.58%
31)Score:97.34%
32)Score:88.44%
33)Score:86.75%
34)Score:96.08%
35)Score:46.43%
36)Score:60.37%
37)Score:63.84%
38)Score:98.53%
39)Score:90.57%
40)Score:36.73%
41)Score:45.93%
42)Score:24.44%
43)Score:89.82%
44)Score:44.94%
45)Score:96.84%
46)Score:64.09%
47)Score:59.25%
48)Score:99.22%
49)Score:67.01%
50)Score:53.03%
51)Score:99.24%
52)Score:36.33%
53)Score:51.4%
54)Score:85.9%
55)Score:52.79%
56)Score:37.56%
57)Score:25.06%
58)Score:48.86%
59)Score:39.86%
60)Score:95.62%
61)Score:52.49%
62)Score:26.34%
63)Score:65.59%
64)Score:88.99%
65)Score:52.92%
66)Score:30.39%
67)Score:54.32%
68)Score:28.16%
69)Score:35.3%
70)Score:47.82%
71)Score:35.42%
72)Score:51.32%
73)Score:66.98%
74)Score:77.96%
75)Score:95.95%
76)Score:94.11%
77)Score:81.97%
78)Score:34.73%
79)Score:92.18%
80)Score:98.69%
81)Score:63.94%
82)Score:95.06%
83)Score:49.42%
84)Score:50.23%
85)Score:37.75%
86)Score:48.73%
87)Score:70.94%
88)Score:99.14%
89)Score:99.58%
90)Score:98.47%
91)Score:35.81%
92)Score:98.73%
93)Score:80.16%
94)Score:81.18%
95)Score:57.36%
96)Score:98.39%
97)Score:84.39%
98)Score:84.86%
99)Score:95.9%
100)Score:55.34%
101)Score:39.72%
102)Score:53.36%
103)Score:59.08%
104)Score:78.67%
105)Score:66.95%
106)Score:62.59%
107)Score:91.67%
108)Score:83.13%
109)Score:52.35%
110)Score:90.16%
111)Score:50.64%
112)Score:71.32%
113)Score:70.95%
114)Score:97.29%
115)Score:79.51%
116)Score:72.67%
117)Score:73.85%
118)Score:19.13%
119)Score:35.6%
120)Score:49.88%
121)Score:44.23%
122)Score:94.64%
123)Score:42.14%
124)Score:53.88%
125)Score:47.13%
126)Score:78.19%
127)Score:58.8%
128)Score:79.67%
129)Score:32.29%
130)Score:99.05%
131)Score:99.24%
132)Score:78.67%
133)Score:81.24%
134)Score:93.88%
135)Score:62.65%
136)Score:96.46%
137)Score:96.63%
138)Score:95.18%
139)Score:93.62%
140)Score:61.16%
141)Score:85.51%
142)Score:92.79%
143)Score:41.37%
144)Score:88.51%
145)Score:46.16%
146)Score:76.01%
147)Score:77.2%
148)Score:41.75%
149)Score:49.05%
150)Score:54.19%
151)Score:57.5%
152)Score:92.81%
153)Score:40.3%
154)Score:87.21%
155)Score:43.83%
156)Score:63.31%
157)Score:79.39%
158)Score:38.08%
159)Score:74.35%
160)Score:76.49%
161)Score:89.51%
162)Score:44.47%
163)Score:92.86%
164)Score:58.75%
165)Score:54.9%
166)Score:98.23%
167)Score:93.77%
168)Score:87.56%
169)Score:74.76%
170)Score:98.87%
171)Score:75.32%
172)Score:98.7%
173)Score:89.95%
174)Score:97.05%
175)Score:86.61%
176)Score:63.66%
177)Score:95.43%
178)Score:39.3%
179)Score:68.84%
180)Score:51.24%
181)Score:81.13%
182)Score:53.07%
183)Score:40.38%
184)Score:54.35%
185)Score:99.91%
186)Score:33.27%
187)Score:51.92%
188)Score:39.35%
189)Score:86.12%
190)Score:43.48%
191)Score:95.99%
192)Score:57.22%
193)Score:85.74%
194)Score:33.6%
195)Score:81.06%
196)Score:81.91%
197)Score:38.16%
198)Score:37.51%
199)Score:98.73%
200)Score:64.33%
201)Score:78.38%
202)Score:86.61%
203)Score:51.05%
204)Score:99.48%
205)Score:88.08%
206)Score:47.59%
207)Score:97.22%
208)Score:98.39%
209)Score:66.63%
210)Score:58.46%
211)Score:39.79%
212)Score:46.92%
213)Score:74.14%
214)Score:85.38%
215)Score:53.35%
216)Score:93.58%
217)Score:89.34%
218)Score:40.57%
219)Score:68.01%
220)Score:88.32%
221)Score:40.26%
222)Score:81.54%
223)Score:69.69%
224)Score:55.33%
225)Score:76.16%
226)Score:64.9%
227)Score:93.61%
228)Score:58.74%
229)Score:79.09%
230)Score:47.22%
231)Score:52.27%
232)Score:23.78%
233)Score:93.72%


234)Score:96.85% 235)Score:93.12% 236)Score:96.58% 237)Score:75.55% 238)Score:69.84% 239)Score:53.55% 240)Score:74.56% 241)Score:80.83% 242)Score:44.8% 243)Score:90.13% 244)Score:50.93% 245)Score:25.42% 246)Score:88.22%
247)Score:81.21% 248)Score:31.79% 249)Score:23.45% 250)Score:88.88% 251)Score:27.84% 252)Score:93.35% 253)Score:90.04% 254)Score:72.79% 255)Score:88.05% 256)Score:75.35% 257)Score:41.16% 258)Score:76.27% 259)Score:84.58%
260)Score:97.65% 261)Score:74.05% 262)Score:35.6% 263)Score:69.63% 264)Score:50.76% 265)Score:49.23% 266)Score:67.81% 267)Score:49.39% 268)Score:87.31% 269)Score:42.03% 270)Score:73.25% 271)Score:50.9% 272)Score:99.65%
273)Score:26.29% 274)Score:65.18% 275)Score:62.23% 276)Score:19.32% 277)Score:95.34% 278)Score:89.69% 279)Score:47.17% 280)Score:88.13% 281)Score:38.93% 282)Score:99.27% 283)Score:83.55% 284)Score:38.92% 285)Score:93.92%
286)Score:36.4% 287)Score:69.69% 288)Score:32.35% 289)Score:79.52% 290)Score:54.17% 291)Score:86.84% 292)Score:99.49% 293)Score:91.62% 294)Score:81.44% 295)Score:95.22% 296)Score:99.26% 297)Score:68.65% 298)Score:95.81%
299)Score:99.85% 300)Score:97.32% 301)Score:95.03% 302)Score:30.41% 303)Score:94.77% 304)Score:79.45% 305)Score:55.82% 306)Score:69.3% 307)Score:96.32% 308)Score:48.53% 309)Score:63.18% 310)Score:86.56% 311)Score:98.55%
312)Score:90.26% 313)Score:36.21% 314)Score:88.25% 315)Score:56.35% 316)Score:91.03% 317)Score:92.74% 318)Score:72.78% 319)Score:85.49% 320)Score:94.82% 321)Score:60.88% 322)Score:47.83% 323)Score:54.36% 324)Score:21.55%
325)Score:22.12% 326)Score:99.43% 327)Score:69.57% 328)Score:39.76% 329)Score:86.1% 330)Score:80.73% 331)Score:82.16% 332)Score:62.78% 333)Score:73.12% 334)Score:98.81% 335)Score:96.59% 336)Score:98.71% 337)Score:52.86%
338)Score:80.92% 339)Score:81.72% 340)Score:90.06% 341)Score:44.78% 342)Score:55.55% 343)Score:64.04% 344)Score:88.84% 345)Score:25.66% 346)Score:69.01% 347)Score:99.4% 348)Score:92.29% 349)Score:71.53% 350)Score:96.46%
351)Score:99.85% 352)Score:38.5% 353)Score:52.82% 354)Score:54.26% 355)Score:51.94% 356)Score:43.41% 357)Score:68.31% 358)Score:84.19% 359)Score:97.25% 360)Score:96.98% 361)Score:55.6% 362)Score:91.97% 363)Score:61.92%
364)Score:97.82% 365)Score:75.61% 366)Score:68.15% 367)Score:54.73% 368)Score:79.11% 369)Score:90.95% 370)Score:30.44% 371)Score:89.65% 372)Score:99.64% 373)Score:54.01% 374)Score:27.58% 375)Score:86.86% 376)Score:45.42%
377)Score:84.65% 378)Score:53.24% 379)Score:98.98% 380)Score:98.55% 381)Score:93.08% 382)Score:82.83% 383)Score:60.88% 384)Score:36.31% 385)Score:65.31% 386)Score:94.28% 387)Score:41.64% 388)Score:73.21% 389)Score:78.95%
390)Score:43.82% 391)Score:49.09% 392)Score:58.62% 393)Score:27.65% 394)Score:77.13% 395)Score:78.83% 396)Score:73.55% 397)Score:19.63% 398)Score:88.79% 399)Score:94.57% 400)Score:79.63% 401)Score:54.26% 402)Score:70.48%
403)Score:96.47% 404)Score:63.06% 405)Score:48.76% 406)Score:99.85% 407)Score:50.35% 408)Score:94.85% 409)Score:90.21% 410)Score:94.54% 411)Score:56.11% 412)Score:70.54% 413)Score:95.85% 414)Score:32.66% 415)Score:94.89%
416)Score:54.14% 417)Score:26.36% 418)Score:71.98% 419)Score:74.99% 420)Score:34.51% 421)Score:86.27% 422)Score:70.64% 423)Score:15.39% 424)Score:77.32% 425)Score:97.18% 426)Score:27.27% 427)Score:72.37% 428)Score:80.72%
429)Score:52.85% 430)Score:20.35% 431)Score:84.81% 432)Score:87.72% 433)Score:61.67% 434)Score:64.4% 435)Score:69.5% 436)Score:52.13% 437)Score:46.0% 438)Score:63.37% 439)Score:49.39% 440)Score:65.86% 441)Score:76.2%
442)Score:95.12% 443)Score:74.25% 444)Score:71.1% 445)Score:51.62% 446)Score:95.23% 447)Score:53.28% 448)Score:44.65% 449)Score:55.9% 450)Score:40.84% 451)Score:98.66% 452)Score:29.65% 453)Score:57.99% 454)Score:67.78%

PROBLEM 2

```
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import random

def problem1(x1, x2):
    return 6+2*(x1**2) + 2*(x2**2)
def problem2(x1,x2):
    return 8
def problem3_max_graph(prob1,prob2):
    return [max(l1, l2) for l1, l2 in zip(prob1, prob2)]
def draw():
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    x1 = x2 = np.arange(-6.0, 6.0, 0.05)
    # x1 = x2 = np.arange(-3.0, 3.0, 0.05)
    X1, X2 = np.meshgrid(x1, x2)

    ##### PROBLEM 2.1
    zs1 = np.array([problem1(x1,x2) for x1,x2 in zip(np.ravel(X1), np.ravel(X2))])
    Z1 = zs1.reshape(X1.shape)
    ax.plot_surface(X1, X2, Z1) # uncomment to plot

    ##### PROBLEM 2.2
    zs2 = np.array([problem2(x1,x2) for x1,x2 in zip(np.ravel(X1), np.ravel(X2))])
    Z2 = zs2.reshape(X1.shape)
    # ax.plot_surface(X1, X2, Z2) # uncomment to plot

    ##### PROBLEM 2.3
    zs3 = problem3_max_graph(zs1,zs2)
    zs3 = np.asarray(zs3)
    Z3 = zs3.reshape(X1.shape)
    # ax.plot_surface(X1, X2, Z3) # uncomment to plot

    ax.set_xlabel('X1 Label')
    ax.set_ylabel('X2 Label')
    ax.set_zlabel('Z Label')

    ax.set_xlim3d(-5, 5)
    ax.set_ylim3d(-5,5)
    ax.set_zlim3d(0,10)
    plt.savefig('problem.png')
    plt.show()

if __name__ == '__main__':
    draw()
```

```
##### PROBLEM 5.1 logistic_regression.py #####################

#!/usr/bin/env python

# Run logistic regression training.

import numpy as np
import scipy.special as sps
import matplotlib.pyplot as plt
import assignment2 as a2


# Maximum number of iterations.  Continue until this limit, or when error change is below tol.
max_iter = 500
# max_iter = 1000
tol = 0.00001

# Step size for gradient descent.
eta = 0.5

# etas = []

# Load data.
data = np.genfromtxt('data.txt')

# Data matrix, with column of ones at end.
X = data[:, 0:3]

# Target values, 0 for class 1, 1 for class 2.
t = data[:, 3]

# For plotting data
class1 = np.where(t == 0)
X1 = X[class1]
class2 = np.where(t == 1)
X2 = X[class2]


# Initialize w.
w = np.array([0.1, 0, 0])

# Error values over all iterations.
e_all = []

DATA_FIG = 1

# Set up the slope-intercept figure
SI_FIG = 2
plt.figure(SI_FIG, figsize=(8.5, 6))
plt.rcParams.update({'font.size': 15})
plt.title('Separator in slope-intercept space')
plt.xlabel('slope')
```

```
plt.ylabel('intercept')
plt.axis([-5, 5, -10, 0])


for iter in range(0, max_iter):

  # Compute output using current w on all data X.
  y = sps.expit(np.dot(X, w))

  # e is the error, negative log-likelihood (Eqn 4.90)
  e = -np.mean(np.multiply(t, np.log(y)) + np.multiply((1-t), np.log(1-y)))

  # Add this error to the end of error vector.
  e_all.append(e)

  # Gradient of the error, using Eqn 4.91
  grad_e = np.mean(np.multiply((y - t), X.T), axis=1)

  # Update w, *subtracting* a step in the error derivative since we're minimizing
  w_old = w
  w = w - eta*grad_e
  #print("\nnew w:",w," w old:",w_old," grad_e:",grad_e)
  # Plot current separator and data.  Useful for interactive mode / debugging.
  # plt.figure(DATA_FIG)
  # plt.clf()
  # plt.plot(X1[:,0],X1[:,1],'b.')
  # plt.plot(X2[:,0],X2[:,1],'g.')
  # a2.draw_sep(w)
  # plt.axis([-5, 15, -10, 10])

  # Add next step of separator in m-b space.
  plt.figure(SI_FIG)
  a2.plot_mb(w, w_old)

  # Print some information.
  print('epoch {0:d}, negative log-likelihood {1:.4f}, w={2}'.format(iter, e, w.T))

  # Stop iterating if error doesn't change more than tol.
  if iter > 0:
    if np.absolute(e-e_all[iter-1]) < tol:
      break


# Plot error over iterations
TRAIN_FIG = 3
plt.figure(TRAIN_FIG, figsize=(8.5, 6))
plt.plot(e_all)
plt.legend(['Step-size: '+str(eta)],loc='upper right')
plt.ylabel('Negative log likelihood')
plt.title('Training logistic regression')
plt.xlabel('Epoch')
plt.grid()
```

```
plt.show()
# plt.savefig('0-5plot_neg_log_likelihood_over_epoch1.png')
```

```
######## PROBLEM 5.2 logistic_regression_mod.py ######

#!/usr/bin/env python

# Run logistic regression training.

import numpy as np
import scipy.special as sps
import matplotlib.pyplot as plt
import assignment2 as a2

# Maximum number of iterations.  Continue until this limit, or when error change is below tol.
max_iter = 500
tol = 0.00001
# Step size for gradient descent.
etas = [0.5,0.3,0.1,0.05,0.01]
# Load data.
data = np.genfromtxt('data.txt')

# Data matrix, with column of ones at end.
X = data[:, 0:3]

# Target values, 0 for class 1, 1 for class 2.
t = data[:, 3]

# For plotting data
class1 = np.where(t == 0)
X1 = X[class1]
class2 = np.where(t == 1)
X2 = X[class2]


# Initialize w.
w = np.array([0.1, 0, 0])

# Error values over all iterations.
e_all = []

DATA_FIG = 1

'''
# Set up the slope-intercept figure
SI_FIG = 2
plt.figure(SI_FIG, figsize=(8.5, 6))
plt.rcParams.update({'font.size': 15})
plt.title('Separator in slope-intercept space')
plt.xlabel('slope')
plt.ylabel('intercept')
plt.axis([-5, 5, -10, 0])
'''
e_all_list = []
for each_eta in etas:
```

```
    e_all =[]
    w = np.array([0.1, 0, 0])
    for iter in range(0, max_iter):

        # Compute output using current w on all data X.
        y = sps.expit(np.dot(X, w))

        # e is the error, negative log-likelihood (Eqn 4.90)
        e = -np.mean(np.multiply(t, np.log(y)) + np.multiply((1-t), np.log(1-y)))

        # Add this error to the end of error vector.
        e_all.append(e)

        # Gradient of the error, using Eqn 4.91
        grad_e = np.mean(np.multiply((y - t), X.T), axis=1)

        # Update w, *subtracting* a step in the error derivative since we're minimizing
        w_old = w
        w = w - each_eta*grad_e
        # Plot current separator and data.  Useful for interactive mode / debugging.
        # plt.figure(DATA_FIG)
        # plt.clf()
        # plt.plot(X1[:,0],X1[:,1],'b.')
        # plt.plot(X2[:,0],X2[:,1],'g.')
        # a2.draw_sep(w)
        # plt.axis([-5, 15, -10, 10])
        '''
        # Add next step of separator in m-b space.
        plt.figure(SI_FIG)
        a2.plot_mb(w, w_old)
        '''
        # Print some information.
        print('epoch {0:d}, negative log-likelihood {1:.4f}, w={2}'.format(iter, e, w.T))

        # Stop iterating if error doesn't change more than tol.
        if iter > 0:
          if np.absolute(e-e_all[iter-1]) < tol:
            break
    e_all_list.append(e_all)

# print("length e_all list:",len(e_all_list))
# Plot error over iterations
TRAIN_FIG = 3
plt.figure(TRAIN_FIG, figsize=(8.5, 6))

index = 0
legend_list = []
for each_eall in e_all_list:
    plt.plot(each_eall)
    legend_list.append('Step-size: '+str(etas[index]))
    index += 1
plt.legend(legend_list,loc='upper right')
```

```
plt.ylabel('Negative log likelihood')
plt.title('Training logistic regression')
plt.xlabel('Epoch')
plt.grid()
# plt.savefig('0-5plot_of_neg_log_likelihood_over_epoch.png')
plt.show()
```

PROBLEM 5.3 logistic_regression_sgd.py

```
#!/usr/bin/env python

# Run logistic regression training.

import numpy as np
import scipy.special as sps
import matplotlib.pyplot as plt
import assignment2 as a2
import random
# Maximum number of iterations.  Continue until this limit, or when error change is below tol.
max_iter = 500
tol = 0.00001


# Step size for gradient descent.
etas = [0.5,0.3,0.1,0.05,0.01]

# Load data.
data = np.genfromtxt('data.txt')

# Data matrix, with column of ones at end.
X = data[:, 0:3]
NUM_INPUT = X.shape[0]
# Target values, 0 for class 1, 1 for class 2.
t = data[:, 3]

# For plotting data
class1 = np.where(t == 0)
X1 = X[class1]
class2 = np.where(t == 1)
X2 = X[class2]


# Initialize w.
w = np.array([0.1, 0, 0])

# Error values over all iterations.
e_all = []

DATA_FIG = 1

eall_list = []
for each_eta in etas:
    eall_one_stepsize =[]
    w = np.array([0.1, 0, 0])
    for iter in range(0, max_iter):
        e_per_iter =[]

        for each_input in range(0,NUM_INPUT):
            each_input = np.random.randint(0,NUM_INPUT)
```

```
            y = sps.expit(np.dot(X[each_input,:], w))
            grad_e = np.multiply((y - t[each_input]), X[each_input,:].T)
            grad_e = grad_e *(1/NUM_INPUT)

            w_old = w
            w = w - each_eta*grad_e
            e = -( np.multiply(t[each_input], np.log(y)) + np.multiply((1-t[each_input]),np.log(1-y)) )

            e_per_iter.append(e)
        e_avg_per_iter = np.average(e_per_iter)

        print('epoch {0:d}, negative log-likelihood {1:.4f}, w={2}'.format(iter, e_avg_per_iter, w.T))
        eall_one_stepsize.append(e_avg_per_iter)

        if iter > 0:
            if np.absolute(e_avg_per_iter-eall_one_stepsize[iter-1]) < tol:
                break
    eall_list.append(eall_one_stepsize)

# Plot error over iterations
TRAIN_FIG = 3
plt.figure(TRAIN_FIG, figsize=(8.5, 6))

index = 0
legend_list = []
for each_eall in eall_list:
    plt.plot(each_eall)
    legend_list.append('Step-size: '+str(etas[index]))
    index += 1
plt.legend(legend_list,loc='upper right')
plt.ylabel('Negative log likelihood')
plt.title('Training logistic regression')
plt.xlabel('Epoch')
plt.grid()
# plt.savefig('sgd_plot_of_neg_log_likelihood_over_epoch.png')
plt.show()
```

```
######### PROBLEM 6 cifar_finetune.py ################
'''
This is starter code for Assignment 2 Problem 6 of CMPT 726 Fall 2019.
The file is adapted from the repo https://github.com/chenyaofo/CIFAR-pretrained-models
'''

#######################################################
####### Do not modify the code above this line #######
#######################################################
import torch.nn.functional as F
USE_NEW_ARCHITECTURE = False

class cifar_resnet20(nn.Module):
    def __init__(self):
        super(cifar_resnet20, self).__init__()
        ResNet20 = CifarResNet(BasicBlock, [3, 3, 3])
        url =
'https://github.com/chenyaofo/CIFAR-pretrained-models/releases/download/resnet/cifar100-resnet20-8412cc70.pth'
        ResNet20.load_state_dict(model_zoo.load_url(url))
        modules = list(ResNet20.children())[:-1]
        backbone = nn.Sequential(*modules) # * is to unpack argument lists
        self.backbone = nn.Sequential(*modules)

        if USE_NEW_ARCHITECTURE:
            hidden_nodes = 256
            self.fc1 = nn.Linear(in_features = 64, out_features = hidden_nodes) #
            self.fcbn1 = nn.BatchNorm1d(hidden_nodes)#
            self.fc = nn.Linear(in_features = hidden_nodes, out_features = 10)#
            self.dropout_rate =0.0  #To set dropout rate
        else:
            self.fc = nn.Linear(64, 10)

    def forward(self, x):
        out = self.backbone(x)
        out = out.view(out.shape[0], -1)

        if USE_NEW_ARCHITECTURE:
            out = F.dropout(F.relu(self.fcbn1(self.fc1(out))),
                            p=self.dropout_rate, training=self.training)
            out = self.fc(out)
            return F.log_softmax(out, dim=1)
        else:
            return F.log_softmax(self.fc(out),dim=1)
            # return self.fc(out)

def accuracy(out, labels):
    total =0
    for index in range(0,len(out)):
        if out[index] == labels[index]:
            total += 1
    return total, total/len(out)
```

```python
device = torch.device("cuda:0" if torch.cuda.is_available() else 'cpu')
# BATCH_SIZE = 128
BATCH_SIZE = 64


if __name__ == '__main__':
    model = cifar_resnet20()
    model.to(device) #.cuda() to use GPU

    transform = transforms.Compose([transforms.ToTensor(),
                                    transforms.Normalize(mean=(0.4914, 0.4822, 0.4465),
                                                         std=(0.2023, 0.1994, 0.2010))])
    trainset = datasets.CIFAR10('./data', download=True, transform=transform) #Number of datapoint = 50000
    trainloader = torch.utils.data.DataLoader(trainset, batch_size=BATCH_SIZE,
                                              shuffle=True, num_workers=0)#num_workers=2
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.SGD(list(model.fc.parameters()), lr=0.001, momentum=0.9,weight_decay=0.0)
    # optimizer = optim.SGD(list(model.backbone.parameters()), lr=0.001, momentum=0.9,weight_decay=0.0)
    # optimizer = optim.SGD(list(model.fc.parameters())+list(model.backbone.parameters()), lr=0.001,
momentum=0.9,weight_decay=0.0)

    ############################################################
    ### Do the training
    DO_TRAINING = True
    DO_TESTING = True
    NUM_EPOCH_TRAINING = 1

    if DO_TRAINING:
        model.train()
        for epoch in range(NUM_EPOCH_TRAINING):  # loop over the dataset multiple times
            running_loss = 0.0

            for i, data in enumerate(trainloader, 0):
                # get the inputs
                inputs, labels = data
                # zero the parameter gradients
                optimizer.zero_grad()
                # forward + backward + optimize
                outputs = model(inputs.to(device)) #.cuda() to use GPU
                loss = criterion(outputs.to(device), labels.to(device)) #.cuda() to use GPU
                loss.backward()
                optimizer.step()
                running_loss += loss.item()
                if i % 20 == 19:    # print every 20 mini-batches
                    print('[%d, %5d] loss: %.3f' %
                        (epoch + 1, i + 1, running_loss / 20))
                    running_loss = 0.0
        print('Finished Training')

    with torch.no_grad():
        TEST_BATCH_SIZE = 20
        SAVING_JSON = False #True
```

```
if DO_TESTING:
### DO TESTING/EVALUATION
    model.eval()
    testset = datasets.CIFAR10(root='./data', train=False,
    download=True, transform=transform)
    testloader = torch.utils.data.DataLoader(testset, batch_size=TEST_BATCH_SIZE,
    shuffle=False, num_workers=0)

    running_test_loss = 0
    match= running_match = 0
    total_test_loss,total_accuracy = 0,0
    score_dict = {}
    loss_dict ={}
    for i, test_data in enumerate(testloader, 0):
        # get the inputs
        test_inputs, test_labels = test_data
        test_outputs = model(test_inputs.to(device))
        test_loss = criterion(test_outputs.to(device), test_labels.to(device))
        pred = test_outputs.argmax(dim=1, keepdim=True) # get the index of the max log-probability

        running_test_loss += test_loss.item()
        total_test_loss += test_loss

        if TEST_BATCH_SIZE==1:
            loss_dict['image_loss'+str(i)] = test_loss.item()
            score = test_outputs[0][pred].to('cpu')
            score_dict['image_score'+str(i)] = score.item()

            if pred == test_labels.to(device) :
                match += 1
                total_accuracy += 1
                running_match +=1
                # print("found MATCH !!!!!!!!!!!!!!!")

        if TEST_BATCH_SIZE>1:
            temp_match , accuracy_batch = accuracy( pred.to('cpu'),test_labels.to('cpu') )
            match += temp_match
            running_match += temp_match
            total_accuracy += accuracy_batch

        if i % 20 == 19:    # print every 20 mini-batches
            print('[At %d] loss: %.3f, accuracy: %.3f, cumulative match: %i' %( i + 1, running_test_loss / 20,
running_match/20,match))
            running_match =0.0
            running_test_loss = 0.0

    if TEST_BATCH_SIZE==1 and SAVING_JSON:
        json.dump(loss_dict,open("test_img_loss1.json",'w'))
        json.dump(score_dict,open("test_img_score1.json",'w'))
        print('Save: 2 json files successfully!')

    print("total_accuracy:",total_accuracy)
```

```
print("total_test_loss:",total_test_loss)
print("Number of match found: ",match) #500 match for no train , 6500 match for 5 epoch train
print('Finished Testing')
```

######### PROBLEM 6 create_html_table.py #######

```
from tabulate import tabulate
import os
import natsort
FILE_NAME = 'image_score_html.html'
IMAGE_FOLDER = 'sample_test_image'

def create_table_html(input_filename_list):
    list = []
    NUM_IMAGE = len(input_filename_list)
    pic_per_row = 13 #14
    num_row = 200 #715

    count =0
    for i in range(0,num_row):
        list_per_row = []
        for j in range(0,pic_per_row):
            if count == NUM_IMAGE:
                break
            pic = "<img src="+IMAGE_FOLDER+"/"+input_filename_list[count]+">"
            list_per_row.append(pic)
            count += 1
        list.append(list_per_row)
    return list

if __name__ == '__main__':
    input_file_name_list = os.listdir(IMAGE_FOLDER)
    input_file_name_list=natsort.natsorted(input_file_name_list,reverse=False)
    table = create_table_html(input_file_name_list)
    html_output = tabulate(table, tablefmt='html')

    with open(FILE_NAME,'w') as file:
        file.write(html_output)
```

######### PROBLEM 6 create_image.py ##########

```
import numpy as np
import matplotlib.pyplot as plt
import torchvision
import torchvision.transforms as transforms
import torch
import os
import json
import math

def imshow(img):
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.axis('off')

    fig = plt.gcf()
    fig.set_size_inches(1.4, 1.4)

    plt.show()

def save_image(index,img,num,image_name,is_score=True):
    img = img / 2 + 0.5     # unnormalize
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.axis('off')

    fig = plt.gcf()
    fig.set_size_inches(1.4, 1.4)

    if is_score:
        name = "Score:"
    else:
        name = 'Loss:'
    fig.suptitle(str(index)+")"+name+str(num), fontsize=8)
    fig.savefig("sample_test_image/"+image_name, dpi=80,bbox_inches='tight')


TEST_BATCH_SIZE = 1
transform = transforms.Compose([transforms.ToTensor(),
                                transforms.Normalize(mean=(0.4914, 0.4822, 0.4465),
                                                     std=(0.2023, 0.1994, 0.2010))])
testset = torchvision.datasets.CIFAR10(root='./data', train=False,
download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=TEST_BATCH_SIZE,
shuffle=False, num_workers=0)

if __name__ == '__main__':
    classes = ('plane', 'car', 'bird', 'cat','deer', 'dog', 'frog', 'horse', 'ship', 'truck')

    SHOW_IMAGE = False
    SAVE_IMAGE = True
```

```
SOFTMAX_OUTPUT = True
USE_SCORE = True

if SHOW_IMAGE:
    # get some random training images
    testiter_show = iter(testloader)
    images, labels = testiter_show.next()
    # print labels
    print(' '.join('%5s' % classes[labels[j]] for j in range(TEST_BATCH_SIZE)))
    # show images
    imshow(torchvision.utils.make_grid(images))

if SAVE_IMAGE:
    NUM_IMAGE_SAVED = 10000
    #REQUIREMENTS
    # NEED TO LOAD JSON file for loss score of each test image (cifar_finetune.py)
    # NEED TEST_BATCH_SIZE = 1 before using this function
    if USE_SCORE:
        score_dict = json.load(open('test_img_score.json'))
        input_dict = score_dict
        key="image_score"
    else:
        loss_dict = json.load(open('test_img_loss.json'))
        input_dict = loss_dict
        key="image_loss"

    testiter = iter(testloader)
    for index in range(0,10000):
        test_image,test_label = testiter.next()

        image_torch = torchvision.utils.make_grid(test_image)
        single_class = classes[test_label]
        image_name = "test_image"+str(index)
        if SOFTMAX_OUTPUT and USE_SCORE:
            number = round(math.exp(input_dict[key+str(index)])*100,2)
            number = str(number)+"%"
        else:
            number = round(input_dict[key+str(index)],3)

        save_image(index,image_torch,number,image_name,is_score=USE_SCORE)
        if NUM_IMAGE_SAVED == index:
            break
        print("index: ",index)
```