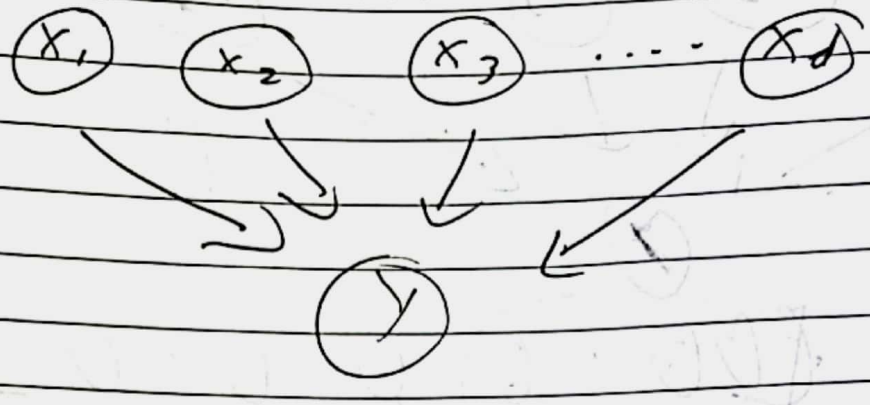
H.W.5

5.1 Gradient based learning.



a) $\mathcal{L} = \sum_t \log(P_{y_t} | \vec{x_t})$

$$\frac{d\mathcal{L}}{dw_i} = \frac{d}{dw_i}\left(\sum_t \log P(y_t | \vec{x_t})\right)$$

$$= \frac{d}{dw_i}\left(\right.$$

Now,

when $y = 1$, $\log P(Y=1|\vec{x}) = p_t$

& $y = 0$ is $1 - p_t$

$$\therefore \frac{d\mathcal{L}}{dw_i} = \frac{d}{dw_i} \sum_t \log\left(p_t^{y_t} (1-p_t)^{1-y_t}\right)$$

$$= \sum_t \frac{d}{dw_i}\left(\log\left(p_t^{y_t} (1-p_t)^{1-y_t}\right)\right)$$

$$= \sum_t \frac{d}{dw_i}\left(\log p_t^{y_t} + \log(1-p_t)^{1-y_t}\right)$$

$$= \sum_t \left[\frac{1}{p_t} \times y_t \times \frac{dp_t}{dw_i} + \frac{1-y_t}{1-p_t} \times \frac{\partial -p_t}{\partial w_i} \right]$$

$$= \sum_t \left[\frac{y_t}{p_t} - \frac{1-y_t}{1-p_t} \right] \frac{\partial p_t}{\partial w_i}$$

$$= \sum_t \left[\frac{y_t - y_t p_t - p_t + y_t p_t}{p_t (1-p_t)} \right] \frac{\partial}{\partial w_i} g(\vec{w} \cdot \vec{x_t})$$

$$= \sum_t \left[\frac{y_t - p_t}{p_t (1-p_t)} \right] g'(\vec{w} \cdot \vec{x_t}) \times x_{it}$$

$$= \sum_{t=1}^{T} \left[\frac{g'(\vec{w} \cdot \vec{x_t})}{p_t (1-p_t)} \right] (y_t - p_t) \times x_{it}$$

//

i) When $g(z) = [1 + e^{-z}]^{-1}$

$$\therefore g'(z) = [1+e^{-z}]^{-2} \times e^{-z}$$

$$= [1+e^{-z}]^{-1} [1+e^{-z}]^{-1} e^{-z}$$

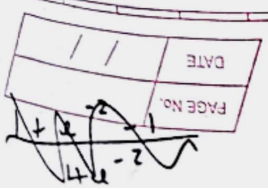
$$= g(z) \times \frac{e^{-z}}{1+e^{-z}}$$

$$= g(z) \times (1 - g(z))$$

∴ From part (a) :-

$$\frac{\partial L}{\partial w_i} = \sum_{t=1}^{T} \frac{g(\vec{w} \vec{x_t}) \times 1 - g(\vec{w} \vec{x_t}) \times (y_t - p_t)}{p_t (1-p_t)} \times x_{it}$$

P.T.O.



$$\therefore \frac{\partial L}{\partial w_i} = \frac{p_t \times (1-p_t)}{p_t \times (1-p_t)} \times (y_t - p_t) x_{it}$$

$$= (y_t - p_t) x_{it}$$

Q5.)2)

$$L = \sum_{t=1}^{T} \log P(y_t | \vec{x_t})$$

$$L = \sum_t \log P(y=0|\vec{x_t})^{y_{0t}} \times \cdots P(y=k|\vec{x_t})^{y_{kt}}$$

$$= \sum_t \log \prod_{i=1}^{k} P(y=i|\vec{x_t})^{y_{it}}$$

$$= \sum_t \sum_i y_{it} \log P(y=i|\vec{x_t})$$

$$= \sum_{t=1}^{T} \sum_{i=1}^{k} y_{it} \log p_{it} \quad \text{i.e.}$$

$$= \sum_t \sum_i y_{it} \log \frac{e^{\vec{w_i}\vec{x_t}}}{\sum_{j=1}^{k} e^{\vec{w_j}\cdot\vec{x_t}}}$$

$$= \sum_{t=1}^{T} \sum_{i=1}^{k} \left(y_{it} \log e^{\vec{w_i}\vec{x_t}} - \log \sum_{j=1}^{k} e^{\vec{w_j}\cdot\vec{x_t}} \right)$$

$$\therefore \frac{\partial L}{\partial w_i} = \frac{\partial}{\partial w_i} \sum_{t=1}^{T} \left(y_{it} e^{\vec{w_i}\vec{x_t}} - \log \sum_{j=1}^{k} e^{\vec{w_j}\cdot\vec{x_t}} \right)$$

$$= \sum_t \left(y_{it}\vec{x_t} - \frac{e^{\vec{w_i}\vec{x_t}}}{\sum_{j=1}^{k} e^{\vec{w_j}\vec{x_t}}} \right)$$

$$= \sum_t \left(y_{it}\vec{x_t} - p_{it}\vec{x_t} \right)$$

$$\frac{\partial L}{\partial w_i} = \sum_{t=1}^{T} (y_{it} - p_{it})\vec{x_t}$$

HP

Q.5.3)

given,

$$x_{n+1} = x_n - \eta g'(x_n)$$

& $$g(x) = \frac{\alpha}{2}(x - x_*)^2$$

$$\therefore g'(x) = \frac{\alpha}{2} \times 2(x - x_*) \times 1$$

a) Now,

$$\varepsilon_n = x_n - x_*$$

$$\varepsilon_{n+1} = x_{n+1} - x_*$$

$$\varepsilon_{n+1} = x_n - \eta g'(x_n) - x_*$$

$$= x_n - \eta \alpha (x_n - x_*) - x_*$$

$$= x_n - x_* - \eta \alpha x_n + \eta \alpha x_*$$

$$= (x_n - x_*) - \eta \alpha (x_n - x_*)$$

$$= (1 - \eta \alpha)(x_n - x_*)$$

$$= \varepsilon_n (1 - \eta \alpha) \quad //$$

$$\therefore \varepsilon_{n+1} = \varepsilon_n (1 - \eta \alpha)$$

Ily,

$$\varepsilon_n = \varepsilon_{n-1}(1-\eta\alpha)$$

$$= \varepsilon_{n-2}(1-\eta\alpha)(1-\eta\alpha)$$

$$\vdots$$

$$= \varepsilon_0 (1-\eta\alpha)^n$$

//

(b) For faster convergence, error should = 0.

$$\therefore \varepsilon_n = \varepsilon_0(1-\eta\alpha)^n \to 0$$

$$\therefore \varepsilon_0(1-\eta\alpha)^n \to 0$$

$$(1-\eta\alpha)^n \to 0$$

For it to be tending to 0 when we are multiplying it with itself,

$$-1 < (1-\eta\alpha) < 1$$

$$-2 < -\eta\alpha < 0$$

$$2 > \eta\alpha > 0$$

$$\therefore 0 < \eta < \frac{2}{\alpha}$$

$\therefore \eta$ can be anything between 0 & $\frac{2}{\alpha}$ $\therefore \eta \in (0, \frac{2}{\alpha})$

But fastest, $(1-\eta\alpha)^n$ will reach 0 is when,

$$1-\eta\alpha = 0$$

$$\therefore \eta\alpha = 1$$

$$\therefore \eta = \frac{1}{\alpha}$$ //

Also $g''(x) = \frac{d}{dx}\alpha(x - x^*)$

$= \alpha - 0$

$= \alpha$

$\therefore \eta = \frac{1}{\alpha} = \frac{1}{g''(x)}$ //

c) $\varepsilon_{n+1} = x_{n+1} - x^*$

$= x_n - \eta g'(x_n) + \beta(x_n - x_{n-1}) - x^*$

$= x_n - \eta \frac{\alpha}{1}(x_n - x^*) + \beta(x_n - x_{n-1}) - x^*$

$= x_n - \eta\alpha x_n + \eta\alpha x^* + \beta x_n - \beta x_{n-1} - x^*$

$= x_n - x^* + \eta\alpha x^* - \eta\alpha x_n + \beta x_n - \beta x_{n-1}$

$= (x_n - x^*) - \eta\alpha(x_n - x^*) + \beta x_n - \beta x^* + \beta x^* - \beta x_{n-1}$

$= \varepsilon_n - \eta\alpha\varepsilon_n + \beta(x_n - x^*) - \beta(x_{n-1} - x^*)$

$= \varepsilon_n - \eta\alpha\varepsilon_n + \beta\varepsilon_n - \beta\varepsilon_{n-1}$

$= \varepsilon_n(1 - \eta\alpha + \beta) - \beta\varepsilon_{n-1}$

//

d) Now,

$$\varepsilon_{n+1} = \varepsilon_n(1 - \eta\alpha + \beta) - \beta\varepsilon_{n-1}$$

$$\varepsilon_n = \varepsilon_{n-1}(1 - n\alpha + \beta) - \beta\varepsilon_{n-2}$$

$$= \varepsilon_{n-1}\left(1 - \frac{4}{9}x1 + \frac{1}{9}\right) - \frac{\varepsilon_{n-2}}{9}$$

$$= \varepsilon_{n-1} \times \frac{6}{9} - \frac{\varepsilon_{n-2}}{9}$$

$$\varepsilon_n = \frac{1}{9}(6\varepsilon_{n-1} - \varepsilon_{n-2})$$

$$= \frac{1}{9}\left(6 \times \left[\frac{1}{9}(6\varepsilon_{n-2} - \varepsilon_{n-3})\right] - \varepsilon_{n-2}\right)$$

Now, If

$$\varepsilon_n = \lambda^n \varepsilon_0$$

$$\varepsilon_{n-1} = \lambda^{n-1}\varepsilon_0$$

$$\varepsilon_{n-2} = \lambda^{n-2}\varepsilon_0$$

$$\therefore \varepsilon_n = \frac{1}{9}(6\varepsilon_{n-1} - \varepsilon_{n-2})$$

$$\therefore \lambda^n \varepsilon_0 = \frac{1}{9}(6 \times \lambda^{n-1}\varepsilon_0 - \lambda^{n-2}\varepsilon_0)$$

$$\therefore \lambda^n = \left(\frac{6\lambda^{n-1}}{9} - \frac{\lambda^{n-2}}{9}\right)$$

$$\div \text{by } \lambda^n$$

$$1 = \frac{6}{9}\lambda^{-1} - \frac{\lambda^{-2}}{9}$$

$$= \frac{1}{\lambda}\left(\frac{6}{9} - \frac{1}{9\lambda}\right)$$

PTO.

$$\therefore \lambda^{-2} - 6\lambda^{-1} + 9 = 0$$

$$\lambda^{-2} - 3\lambda^{-1} - 3\lambda^{-1} + 9 = 0$$

$$\lambda^{-1}(\lambda^{-1} - 3) - 3(\lambda^{-1} - 3) = 0$$

$$\therefore (\lambda^{-1} - 3)(\lambda^{-1} - 3) = 0$$

$$\therefore \lambda^{-1} = 3$$

$$\lambda = \frac{1}{3}$$

$\because \lambda$ exists,

$$\varepsilon_n = \lambda^n \varepsilon_0$$

is a valid solution.

Now, when $\beta = 0$,

$$\text{Wkt, } \varepsilon_n = \varepsilon_0 (1 - \eta \alpha_i)^n$$

$$\therefore \lambda^n \varepsilon_0 = \varepsilon_0 \left(1 - \frac{4}{9} \times 1\right)^n$$

$$\therefore \lambda^n = \frac{5}{9}^n$$

$$\therefore \lambda = \frac{5}{9}$$

$\therefore$ When $\beta = \frac{1}{9}$, $\lambda = \frac{1}{3}$

& when $\beta = 0$, $\lambda = \frac{5}{9}$

CSE250A_Hw5_A59017531

November 4, 2022

[282]:
```
import numpy as np
import matplotlib.pyplot as plt
from prettytable import PrettyTable
```

0.1 Importing dataset

[283]:
```
train3 = []
with open('/content/train3.txt') as f:
  for line in f.readlines():
    train3.append(line.split())

train3 = np.array(train3,dtype=float)
print(train3.shape)
```

```
(700, 64)
```

[284]:
```
train5 = []
with open('/content/train5.txt') as f:
  for line in f.readlines():
    train5.append(line.split())

train5 = np.array(train5,dtype=float)
print(train5.shape)
```

```
(700, 64)
```

0.2 Creating Labels to shuffle the data

0.2.1 0: image 3

[285]:
```
y3 = np.array(([0]*train3.shape[0])).reshape(-1,1)
full3 = np.hstack((train3,y3))
print(full3[0])
```

```
[0. 0. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 0. 1. 1. 1. 0. 1. 0. 1. 1. 1. 0. 0.
 0. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0. 1. 1. 0. 0. 0.
 0. 1. 1. 1. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0.]
```

0.2.2 1: image 5

[286]:
```
y5 = np.array(([1]*train5.shape[0])).reshape(-1,1)
full5 = np.hstack((train5,y5))
print(full5[0])
```

```
[0. 1. 1. 1. 1. 1. 1. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1. 0. 0. 0.
 0. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0. 0. 0. 1. 1.
 1. 1. 0. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1.]
```

[287]:
```
dataset = np.vstack((full3,full5))
np.random.shuffle(dataset)
train35x = dataset[:,:-1]
train35y = dataset[:,-1].reshape(-1,1)
```

0.3 Gradient Ascent

0.3.1 Init

[288]:
```
def initWeights():
  w = 100 * np.random.random_sample((train3.shape[1],1)) - 50
  w/=100
  return(w)
```

[289]:
```
def sigmoid(x):
  return 1/(1 + np.exp(-x))
def calcProb(wts):
  return sigmoid(np.dot(train35x,wts))
```

[290]:
```
def lrate():
  neuLR = np.random.random_sample()/1400
  return neuLR
```

0.3.2 Algo used Gradient Ascent:

[291]:
```
N = 50000
finalW = []
minErr = float("inf")
neuLR = 0.2/train35x.shape[0]
wts = initWeights()
lwlis = []
errl = []
print("USING Learning rate as: ",neuLR)
for i in range(N):
    prob = calcProb(wts)
    t1 = np.log(prob)*train35y
    t2 = np.log(1-prob) * (1-train35y)
    lw = t1+t2
    # print(lw)
```

```
    # break
    # lwlis.append(np.sum(lw.reshape(1,-1)[0]))

    llh = np.sum(lw,axis = 0)

    lwlis.append(llh)
    yPred = np.where(prob > 0.5,1,0)
    err = np.sum(np.absolute(train35y - yPred),axis = 0) / train35x.shape[0]
    errl.append(err[0])

    if err[0] < minErr:
        minErr = err[0]
        finalW = wts

    # Next vars
    gradient = np.sum(((train35y - prob) * train35x) ,axis = 0).reshape(-1,1)
    wts = wts + (neuLR * gradient)
```

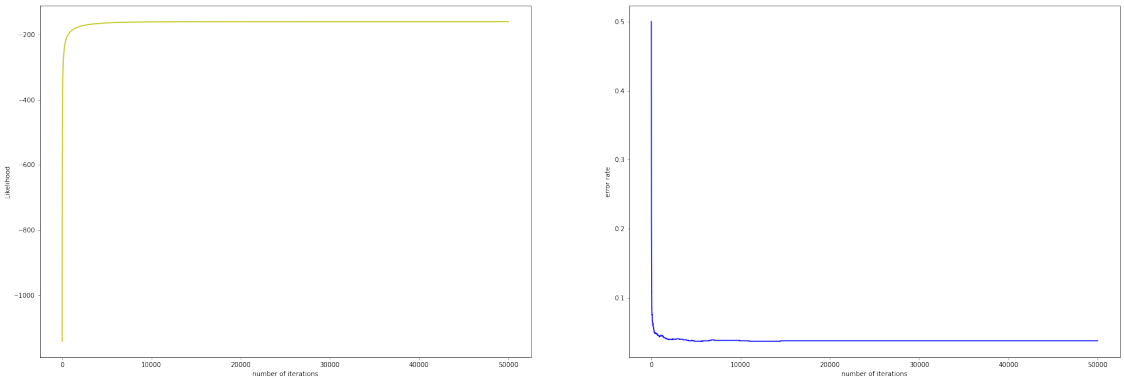
```
USING Learning rate as:  0.00014285714285714287
```

[292]:
```
minErr
```

[292]:
```
0.037142857142857144
```

0.3.3 Plot shows convergence!

[293]:
```
x = np.linspace(0,N,N)
fig=plt.figure(figsize=(30,10))
fig.add_subplot(1,2,1)
plt.plot(x,lwlis,'y')
plt.xlabel('number of iterations')
plt.ylabel('Likelihood')
fig.add_subplot(1,2,2)
plt.plot(x,errl,'b')
plt.xlabel('number of iterations')
plt.ylabel('error rate')
plt.show()
```



0.3.4 8x8 Weights matrix, in table and matrix form:

[294]:
```
tmp = finalW.reshape(8,8)
cnt = 0
for i in range(len(tmp)):
  for j in range(len(tmp[0])):
    print("W",cnt,": ",round(tmp[i][j],4),end="\t")
    cnt+=1
  print()
```

```
W 0 :  -0.7931  W 1 :  -1.4542  W 2 :  -1.1795  W 3 :  -1.0558  W 4 :  -0.7595
W 5 :  -0.7479  W 6 :  0.7966   W 7 :  1.7145
W 8 :  0.0449   W 9 :  -0.0231  W 10 :  0.1844  W 11 :  -0.0837 W 12 :  -0.3198
W 13 :  0.6953  W 14 :  -1.2356 W 15 :  -1.2506
W 16 :  3.1376  W 17 :  1.377   W 18 :  1.3376  W 19 :  0.2213  W 20 :  0.5923
W 21 :  -1.8775 W 22 :  -2.39   W 23 :  -2.3893
W 24 :  0.8289  W 25 :  0.3875  W 26 :  0.5412  W 27 :  -0.2347 W 28 :  -0.54
W 29 :  -2.1068 W 30 :  0.3277  W 31 :  -0.0442
W 32 :  0.4428  W 33 :  1.0352  W 34 :  0.0408  W 35 :  -0.3249 W 36 :  -0.6073
W 37 :  -0.2293 W 38 :  -0.3799 W 39 :  -0.306
W 40 :  1.1113  W 41 :  -0.1957 W 42 :  -0.2851 W 43 :  -0.0488 W 44 :  0.0733
W 45 :  -0.7715 W 46 :  0.7316  W 47 :  -1.4308
W 48 :  1.3597  W 49 :  -0.5857 W 50 :  1.24    W 51 :  0.5435  W 52 :  0.3816
W 53 :  -0.2663 W 54 :  0.2263  W 55 :  -1.0626
W 56 :  0.5138  W 57 :  0.264   W 58 :  0.8857  W 59 :  1.6772  W 60 :  0.423
W 61 :  0.6365  W 62 :  0.4944  W 63 :  -0.4671
```

[295]:
```
print("FINAL WEIGHTS")
x = PrettyTable()
x.add_column("Weights", range(64))
x.add_column("Value", finalW[:,0])
print(x)
```

```
FINAL WEIGHTS
```

Weights	Value
0	-0.7930971502761953
1	-1.4542176626113985
2	-1.179461941762627
3	-1.0558085734940366
4	-0.7595034789625994
5	-0.7478580076891667
6	0.7966308450532321
7	1.7144935529053704
8	0.044856012414151804
9	-0.023068917705294307
10	0.184432639776959
11	-0.08369501379328186
12	-0.31980200605467984
13	0.695308001099729
14	-1.2355579937600598
15	-1.2506010166308943
16	3.13762132638525
17	1.377002624226973
18	1.3375811520760787
19	0.22127818372553024
20	0.5923476145605169
21	-1.8775155200873153
22	-2.389994724425963
23	-2.3893362927912
24	0.8288722216870243
25	0.38751280908807934
26	0.5412212608428238
27	-0.234730547557729
28	-0.53999142163678
29	-2.106798321824953
30	0.3276566099364046
31	-0.04416086108299056
32	0.44282847104127987
33	1.035172070927948
34	0.04076469278698515
35	-0.32485890065781703
36	-0.6073199960535977
37	-0.22925816279423256
38	-0.3799340192676351
39	-0.30598899509349664
40	1.1113238735482975
41	-0.19573608796514153
42	-0.28505245414227726
43	-0.04876017278474009
44	0.07327446375341885

```
|    45   |  -0.7714622182289411  |
|    46   |   0.7316488355955569  |
|    47   |   -1.430775382189215  |
|    48   |   1.3597276668091318  |
|    49   |  -0.5856640305629807  |
|    50   |   1.2400132268949966  |
|    51   |   0.5435084708141752  |
|    52   |  0.38162507503740445  |
|    53   |  -0.26628236912329795 |
|    54   |  0.22631579669606525  |
|    55   |  -1.0626353203271255  |
|    56   |   0.5138333008748595  |
|    57   |   0.2640004323533073  |
|    58   |   0.8856856573589685  |
|    59   |   1.6771631090205692  |
|    60   |  0.42304963807716733  |
|    61   |   0.6364574209890034  |
|    62   |  0.49437534153936846  |
|    63   |  -0.46706621014570965 |
+---------+-----------------------+
```

0.4 Testing

[296]:
```
test3 = []
with open('/content/test3.txt') as f:
  for line in f.readlines():
    test3.append(line.split())

test3 = np.array(test3,dtype=float)
print(test3.shape)
```

```
(400, 64)
```

[297]:
```
test5 = []
with open('/content/test5.txt') as f:
  for line in f.readlines():
    test5.append(line.split())

test5 = np.array(test5,dtype=float)
print(test5.shape)
```

```
(400, 64)
```

[298]:
```
y_3 = np.array(([0]*test3.shape[0])).reshape(-1,1)
full3 = np.hstack((test3,y_3))
# print(full3[0])

y_5 = np.array(([1]*test5.shape[0])).reshape(-1,1)
```

```
full5 = np.hstack((test5,y_5))
# print(full5[0])

testset = np.vstack((full3,full5))
np.random.shuffle(testset)
test35x = testset[:,:-1]
test35y = testset[:,-1].reshape(-1,1)
```

0.5 Error rate for test3 and test5 combined

[299]:
```
def calcTest(wts):
  return sigmoid(np.dot(test35x,wts))
```

[300]:
```
prob = calcTest(finalW)
pred = np.where(prob > 0.5,1,0)
err = np.sum(np.absolute(test35y - pred),axis = 0) / test35x.shape[0]
print("Test Error:",str(err))
```

```
Test Error: [0.06]
```

0.6 Error rate for test3

[301]:
```
def calcTest(wts):
  return sigmoid(np.dot(test3,wts))
```

[302]:
```
prob = calcTest(finalW)
pred = np.where(prob > 0.5,1,0)
err = np.sum(np.absolute(y_3 - pred),axis = 0) / test3.shape[0]
print("Test Error:",str(err))
```

```
Test Error: [0.0675]
```

0.7 Error rate for test5

[303]:
```
def calcTest(wts):
  return sigmoid(np.dot(test5,wts))
```

[304]:
```
prob = calcTest(finalW)
pred = np.where(prob > 0.5,1,0)
err = np.sum(np.absolute(y_5 - pred),axis = 0) / test5.shape[0]
print("Test Error:",str(err))
```

```
Test Error: [0.0525]
```

[304]:
```

```