
**دانشکدهٔ مهندسی کامپیوتر**

# عنوان: گزارش پروژه شبکه های عصبی

**نام استاد: دکتر عبدی**

**نام دانشجو: امیرحسین اسلامی**

**شماره دانشجویی: ۴۰۰۴۱۱۰۹۹**

**بهار ۱۴۰۴**

.

# فهرست مطالب


**فصل 1 پیش پردازش:** **۹**

۱-۱- استاندارد سازی ..... ۱۰

۱-۲- تعریف مقادیر اولیهٔ وزن ها ..... ۱۱

۱-۳- پیاده سازی تابع زیان ..... ۱۱

۱-۴- تابع گرادیان ..... ۱۲

۱-۵- تابع بروزرسانی ضرایب ..... ۱۳

**فصل ۲: پیاده سازی** **۱۴**

۲-۱- پیاده سازی مدل ..... ۱۶

**فصل ۳: شبکه عصبی** **۲۳**

3-1- پیاده سازی feedforward ..... ۲۴

۳-۲- تابع محاسبه زیان ..... ۲۵

۳-۳- تابع رگوله سازی ..... ۲۶

۳-۴- مشتق تابع رلو ..... ۲۶

3-5- تابع Back Propagation1 ..... ۲۷

۳-۶- پیاده سازی تابع آموزش ..... ۲۷

۳-۶-۲- افزایش تعداد آموزش برای برطرف کردن مقدار دلخواه ..... ۲۸

۳-۶-۳- افزایش دوباره ی تعداد دفعات آموزش ..... ۲۹

۳-۶-۴- تغییر نرخ یادگیری و معماری شبکه ..... ۳۰

۳-۶-۵- تغییر نرخ یادگیری ..... ۳۱

۳-۶-۶- تغییر نرخ یادگیری به ۰.۱ ..... ۳۱

۳-۶-۷- تغییر معماری شبکه ..... ۳۲

۳-۶-۸- تغییر نرخ یادگیری ..... ۳۲

۳-۶-۹- تغییر معماری ..... ۳۳

۳-۶-۱۰- بازگشت به معماری اولیه و تغییر لامبدا ..... ۳۴

۳-۶-۱۱- رفع ارور ..... ۳۵

**فصل 4 پیاده سازی مدل روی دیتاست MNIST:** **۳۸**

۴-۱- آماده سازی دیتاست ..... ۳۹

۴-۲- طراحی تلفات با کراس آنتروپی ..... ۴۰

۴-۳- آموزش مدل ..... ۴۱

۴-۴- افزایش تعداد لایه های شبکه ..... ۵۱

۴-۵- چاپ معماری مدل ..... ۵۳

# فهرست مطالب

# فهرست اشکال


شکل (۱-۱) پیاده سازی استاندارد سازی ..... ۱۰
شکل (۱-۲) مقادیر اولیه ی وزن ها ..... ۱۱
شکل (۱-۳) پیاده سازی تابع زیان ..... ۱۱
شکل (1-4) پیاده سازی تابع گرادیان ..... ۱۲
شکل (۱-۵) پیاده سازی تابع بروزرسانی ضرایب ..... ۱۳
شکل (۲-۱) ساختار مدل ..... ۱۶
شکل (۲-۲) ارور مدل ..... ۱۷
شکل (۲-۳) نمودار تابع زیان ..... ۱۸
شکل (۲-۴) کد اصلاح شده ..... ۱۹
شکل (۲-۵) پاس شدن تست کیس مدل ..... ۲۰
شکل (۲-۶) نمودار مقدار واقعی خروجی ..... ۲۱
شکل (۲-۷) نمودار مقدار خروجی پیش بینی شده ..... ۲۲
شکل (1-3) تابع feed forward ..... ۲۴
شکل (۳-۲) تابع محاسبه زیان ..... ۲۵
شکل (۳-۳) تابع رگوله سازی ..... ۲۶
شکل (۳-۴) مشتق تابع رلو ..... ۲۶
شکل (۳-۵) تابع بازگشتی۱ ..... ۲۷
شکل (۳-۶) تابع آموزش ..... ۲۷
شکل (۳-۷) ارور آموزش مدل شبکه عصبی ..... ۲۸
شکل (۳-۸) افزایش دفعات آموزش ..... ۲۸
شکل (۳-۹) افزایش تعداد دفعات آموزش به مقدار نهایی ..... ۲۹
شکل (۳-۱۰) نتیجه با ۱۰۰۰۰ تعداد دفعات آموزش ..... ۲۹
شکل (۳-۱۱) تغییر نرخ یادگیری و معماری شبکه ..... ۳۰
شکل (12-3) نرخ تلفات ..... ۳۰
شکل (۳-۱۳) نرخ یادگیری ۰.۰۵ ..... ۳۱
شکل (۳-۱۴) نتیجه با نرخ یادگیری ۰.۱ ..... ۳۱
شکل (۳-۱۵) نتیجه با تغییر معماری شبکه ..... ۳۲
شکل (۳-۱۶) تلفات با نرخ ۰.۵ ..... ۳۲
شکل (۳-۱۷) نتیجه تلفات پس از تغییر معماری ..... ۳۳

# فهرست اشکال


شکل (۳-۱۸) نتیجه با لامبدای ۰.۰۵ ..... ۳۴
شکل (۳-۱۹) نتیجه با لامبدای ۰.۰۱ ..... ۳۴
شکل (۳-۲۰) اصلاح کد ..... ۳۵
شکل (۳-۲۱) تابع تلفات ..... ۳۶
شکل (3-22) تابع تلفات با ۴۰۰۰ تعداد Iteration ..... ۳۷
شکل (۳-۲۳) تلفات روی دیتاست اصلی ..... ۳۷
شکل (۴-۱) دریافت و اسکیل دیتاست ..... ۳۹
شکل (۴-۲) فرمول کراس آنتروپی ..... ۴۰
شکل (۴-۳) تابع تلفات کراس آنتروپی ..... ۴۰
شکل (4-4) خروجی آموزش اول Mnist ..... ۴۱
شکل (۴-۵) خروجی لایه آخر فید فوروارد ..... ۴۱
شکل (4-6) خروجی جدید Mnist ..... ۴۲
شکل (4-7) اصلاح تابع Initialization ..... ۴۲
شکل (۴-۸) تغییر پارامتر نرخ یادگیری ..... ۴۲
شکل (۴-۹) خروجی یادگیری ..... ۴۳
شکل (۴-۱۰) نوشتن کد برای ذخیره سازی و بارگزاری مجدد مدل برای ذخیره و از دست ندادن مدل ها .... ۴۴
شکل (۴-۱۱) آموزش مجدد با پارامتر های جدید ..... ۴۴
شکل (۴-۱۲) نتیجه آموزش ..... ۴۵
شکل (۴-۱۳) اصلاح معماری شبکه ..... ۴۵
شکل (۴-۱۴) خروجی آموزش دوباره مدل ..... ۴۶
شکل (۴-۱۵) تغییر تابع مدل برای ذخیره بهترین مدل با کمترین تلفات ..... ۴۶
شکل (۴-۱۶) تغییر پارامتر های مدل ..... ۴۶
شکل (۴-۱۷) خروجی آموزش مدل ..... ۴۷
شکل (۴-۱۸) کاهش نرخ یادگیری ..... ۴۷
شکل (۴-۱۹) تغییر مدل ..... ۴۷
شکل (۴-۲۰) نتیجه خروجی ..... ۴۸
شکل (۴-۲۱) کاهش نرخ یادگیری ..... ۴۸
شکل (۴-۲۲) اصلاح ذخیره سازی ..... ۴۸
شکل (۴-۲۳) خروجی آموزش ..... ۴۹

# فهرست اشکال

شکل (۴-۲۴) اصلاح تابع مدل ..... ۴۹
شکل (۴-۲۵) اصلاح پارامتر های مدل ..... ۴۹
شکل (۴-۲۶) نتیجه ..... ۵۰
شکل (۴-۲۷) گیر کردن در نقطه کمینه محلی ..... ۵۱
شکل (۴-۲۸) لایه های شبکه ..... ۵۱
شکل (۴-۲۹) خروجی مدل با ۱۰ لایه پنهان ..... ۵۲
شکل (۴-۳۰) معماری مدل ..... ۵۳

# فهرست جداول

جدول (۲-۱) نتیجه بررسی پرسش نامه ها در ارتباط با عوامل موثر ..... **Error! Bookmark not defined.**

# فصل ۱:

# پیش پردازش

## ۱-۱- استاندارد سازی

برای اینکه بتوانیم از داده ها بهترین استفاده و بهره وری را داشته باشیم به استاندارد سازی نیاز داریم چرا که باید تاثیر ویژگی های دیتاست در یک حد باشد اما وقتی محدوده ی داده ها تفاوت می کند و در مواردی بعضی ویژگی ها مقادیری چندین برابر ویژگی های دیگر دارند این باعث می شود دیگر ویژگی ها کم اهمیت تر شوند که دلخواه ما نیست و ما با استاندارد و نرمالیزه سازی کاری می کنیم که همه ی ویژگی ها به یک اندازه تاثیر گذار شوند.

```
    # START TODO ##################
    m = np.mean(X)
    standard = np.std(X)
    standardized_data = (X-m)/standard
    return standardized_data
    # END TODO ####################

    raise NotImplementedError()
```

```
# Sample test cases
np.random.seed(seed)
x = np.random.randint(0, 100, size = 10)
x = standardize(x)
assert np.allclose(x[1], -0.8183755)
print('Sample Test passed', '\U0001F44D')
```

```
Sample Test passed 👍
```

شکل (۱-۱) پیاده سازی استاندارد سازی

## ۲-۱- تعریف مقادیر اولیهٔ وزن ها

```
# START TODO ##################
n_x = X.shape[1]
m = X.shape[0]
n_y = Y.shape[1]

w = np.random.rand(n_x,n_y)
b = np.zeros((n_y,1))

return w,b
# END TODO ####################
```

شکل (۲-۱) مقادیر اولیه ی وزن ها

## ۳-۱- پیاده سازی تابع زیان

```
# START TODO ##################
diff = y_pred - y_true
diff_square = diff**2
sum = np.sum(diff_square)
mse = sum/(2*y_pred.shape[0])
return mse
# END TODO####################
```

شکل (۳-۱) پیاده سازی تابع زیان

## ۴-۱- تابع گرادیان

```python
def grads(X, y_pred, y_true):
    """
    Args:
    'X': The input matrix
    'y_pred': ndarray of shape (m,1) storing output predictions
    'y_true': ndarray of shape (m,1) storing true house prices
    Returns:
    'dW': gradients of weights
    'db': gradient of biases
    """

    # START TODO #################
    alpha = 0.1
    m = y_pred.shape[0]
    dw = np.dot((np.transpose(X)),(y_pred - y_true))/m
    db = np.sum(y_pred-y_true)/m
    return dw, db

    # END TODO####################
    raise NotImplementedError()
```

شکل (۴-۱) پیاده سازی تابع گرادیان

## ۵-۱- تابع بروزرسانی ضرایب

```python
def sgd(W, b, grads, alpha):
    """
    Args:
    'W' : (n, output_size) array
    'b' : (1,output_size)
    'grads': list of dW, db
    'alpha': learning rate
    Return:
    Updated parameters
    updated_w : updated value of 'W' using the formula given above
    updated_b : updated value of 'b' using the formula given above
    """

    # START TODO #################
    updated_w = W - alpha*grads[0]
    updated_b = b - alpha*grads[1]

    return updated_w, updated_b

    # END TODO ###################
    raise NotImplementedError()
```

شکل (۵-۱) پیاده سازی تابع بروزرسانی ضرایب

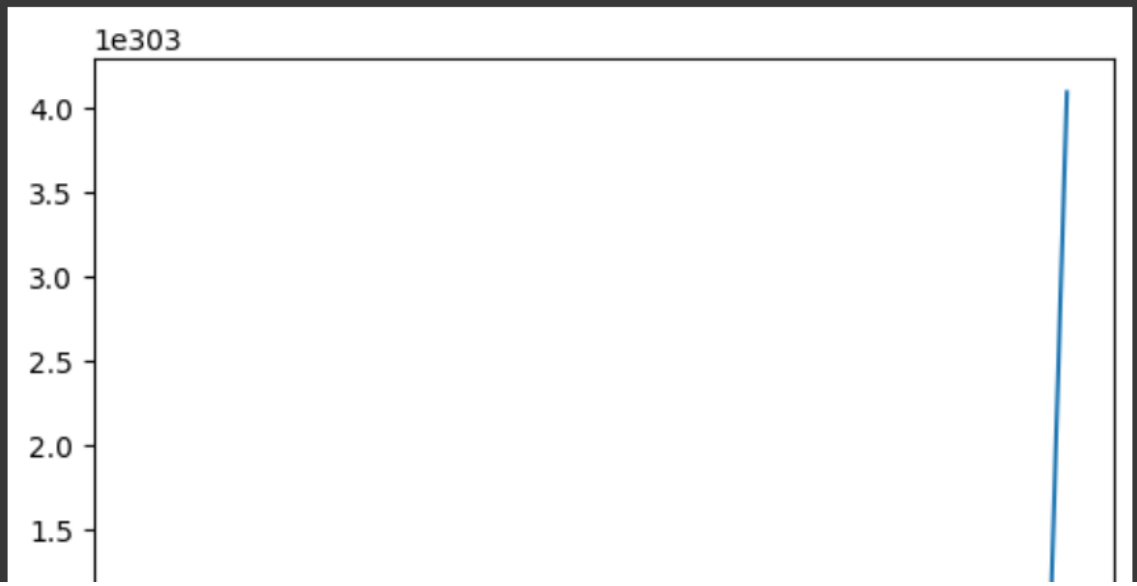
# فصل ۲: پیاده سازی

## ۱-۲- پیاده سازی مدل

```
m = X.shape[0]
n_x = X.shape[1]

if n_x != 8:
  print("N of features is not 8!")
  return

W,b = initialize_parameters(first_x, first_y)

for i in range(iterations):

  y_predicted = np.dot(X,W)+b

  los = loss(y_predicted,Y)
  losses.append(los)

  dW,db = grads(X,y_predicted, Y)

  W, b = sgd(W,b,[dW,db],learning_rate)

# plot loss curve
plt.plot(losses)
return [y_predicted, Y, losses, W, b]
```

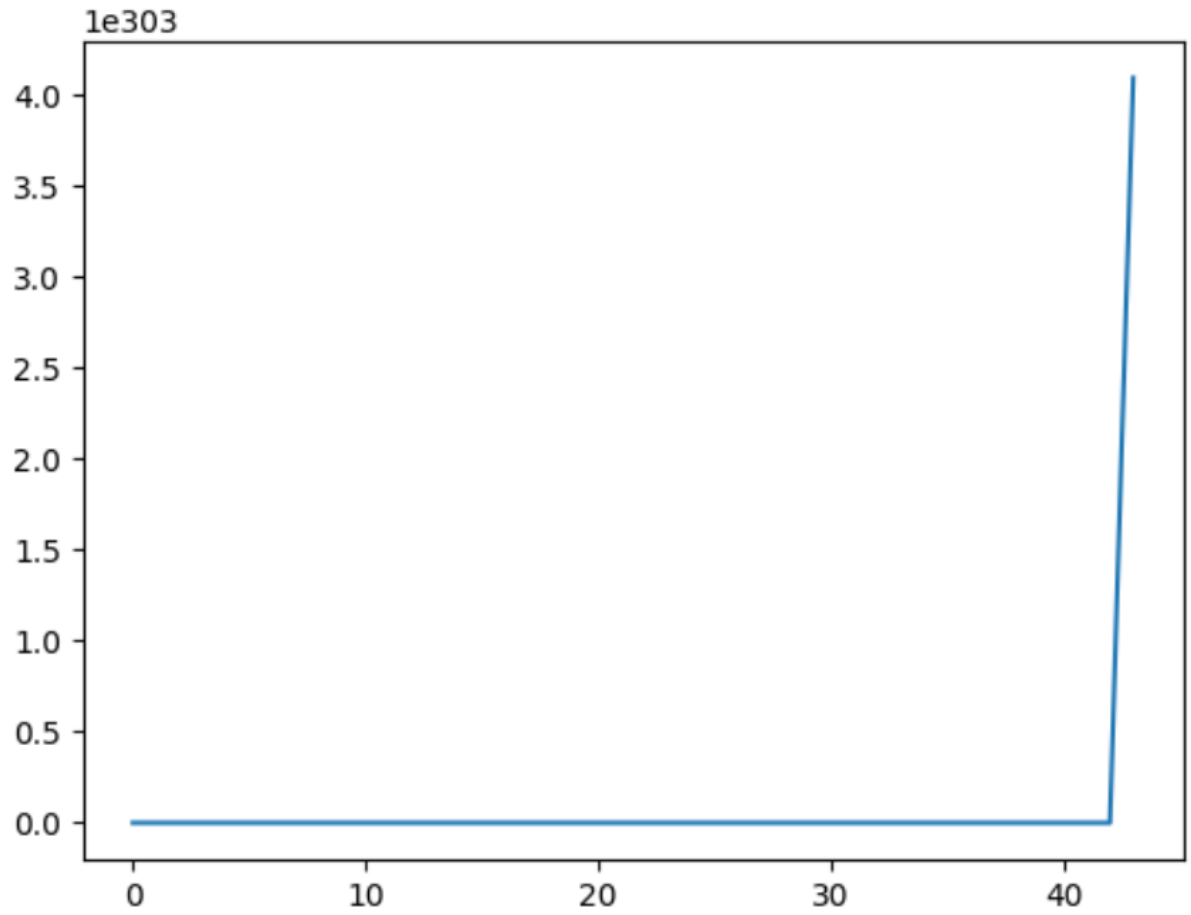
شکل (۱-۲) ساختار مدل

```
AssertionError                            Traceback (most recent
<ipython-input-90-863a51ff1765> in <cell line: 0>()
      2 np.random.seed(1)
      3 y_pred , y_true, losses , trained_w , trained_b = model(X, Y)
----> 4 assert np.allclose(losses[100], 0.3212234664254295)
      5 print('Sample Test passed', '\U0001F44D')

AssertionError:
```




شکل (۲-۲) ارور مدل

رسم تابع زیان:

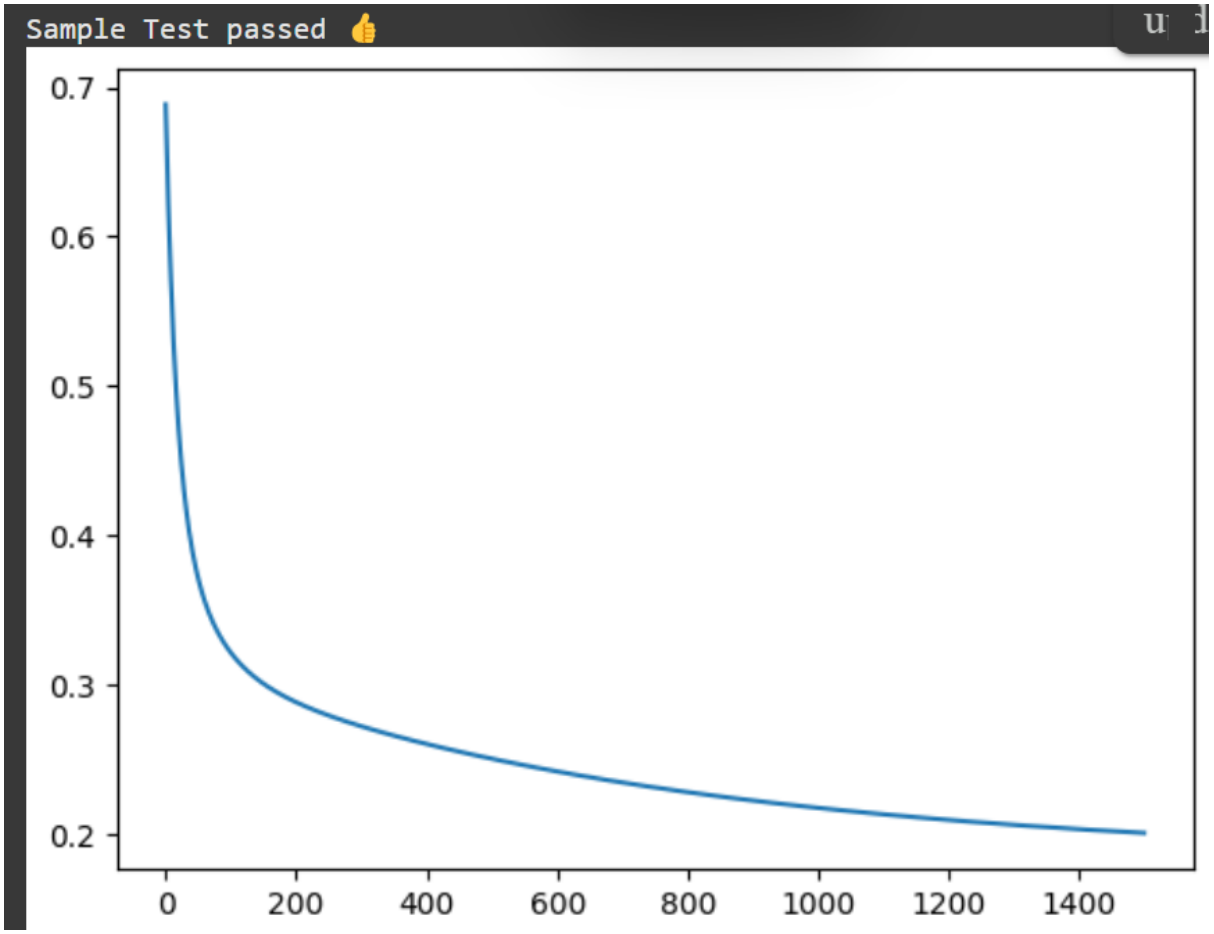



شکل (۳-۲) نمودار تابع زیان

```
def initialize_parameters(X, Y):
    """
    Args:
    'Y': ndarray of shape (m,output_size)
    'X': ndarray of shape (m, no. of features)
    Returns:
    'W', 'b': Wts. and biases
    'W' : ndarray of shape(no. of features, output_size)
    'b' : ndarray of shape(1, output_size)

    USE output_size = 1 since we are only predicting median_
    IF we predicting say house price and no of people that ca
    """
    # START TODO #################
    np.random.seed(seed)
    n_x = X.shape[1]
    n_y = Y.shape[1]

    W = np.random.rand(n_x,n_y)
    b = np.zeros((1,n_y))

    return W,b
    # END TODO ####################
```

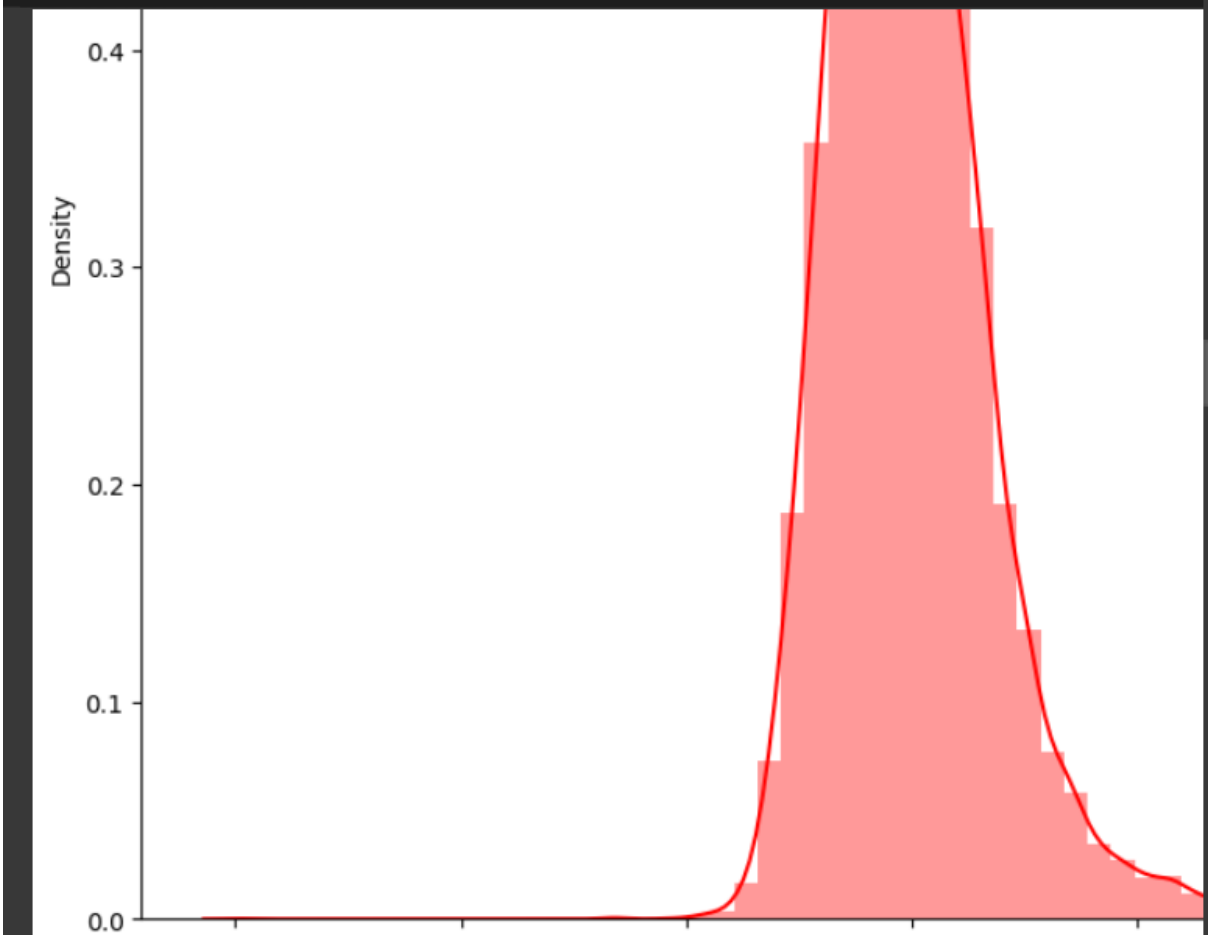
شکل (۴-۲) کد اصلاح شده

مشکل این بود که در اینجا سید و گروه تصادفی را گروه ۱ انتخاب کرده بودم که اشتباه بود اما اصلاح شد و مقداری که بالاتر تعریف شده بود قرار گرفت و تست کیس پاس شد.

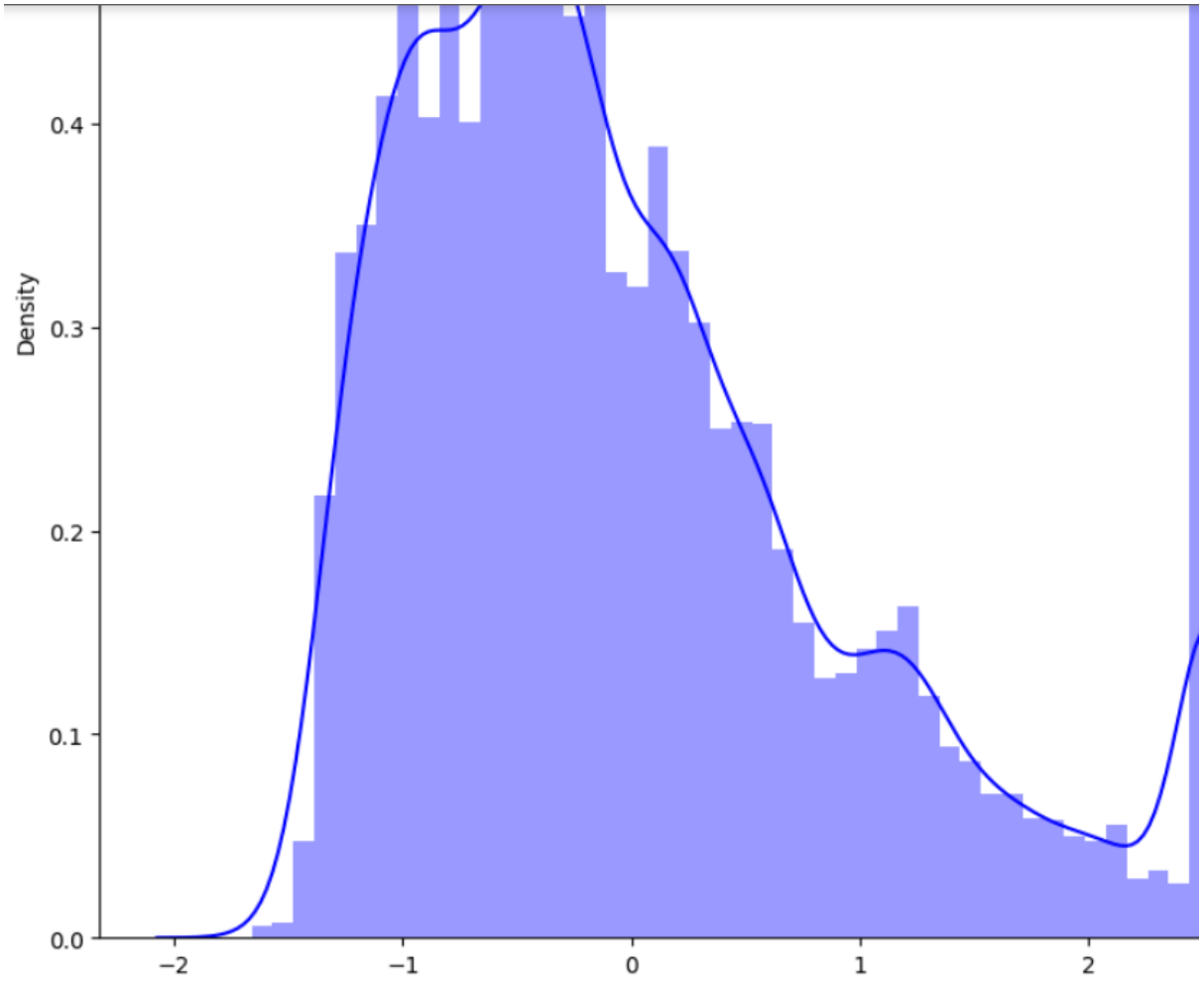



شکل (۵-۲) پاس شدن تست کیس مدل

```
distplot(y_pred, 1)
distplot(y_true, 2)
```




شکل (۶-۲) نمودار مقدار واقعی خروجی




شکل (۷-۲)  نمودار مقدار خروجی پیش بینی شده

# فصل ۳: شبکه عصبی

## ۱-۳- پیاده سازی feedforward

```
W1,b1,W2,b2,W3,b3,W4,b4,W5,b5 = params

m = X.shape[1]
z = np.dot(W1,X) + b1
a = relu(z)
l = [z,a]

z = np.dot(W2,a) + b2
a = relu(z)
l.extend([z,a])

z = np.dot(W3,a) + b3
a = relu(z)
l.extend([z,a])

z = np.dot(W4,a) + b4
a = relu(z)
l.extend([z,a])

z = np.dot(W5,a) + b5
y_out = z
l.append(z)
```

شکل (۱-۳) تابع feed forward

## ۲-۳- تابع محاسبه زیان

```
def loss_compute(y_pred, yd):
    '''
    Inputs:
    - y_pred: numpy array containing predicte
    - yd: numpy array containing desired valu
    dataset

    Outputs:
    - loss: Calculate and return the loss usi
    '''
    # START TODO ##################
    m = y_pred.shape[1]
    loss = np.sum((yd - y_pred)**2)/(2*m)
    return loss
    # END TODO ####################
    raise NotImplementedError()
```

شکل (۲-۳) تابع محاسبه زیان

## ۳-۳- تابع رگوله سازی

```python
def regularization_L2(lmbda, W1, W2, W3, W4, W5, m):
    '''
    Inputs:
    - lmbda: Regularization parameter
    - W1, W2, W3, W4, W5: numpy arrays containing weights of the model
    - m: no of examples

    Outputs:
    - total_reg_loss: sum of L2 regularization loss of each layer
    '''

    # START TODO ################
    total_reg_loss = lmbda*(np.sum(W1**2) + np.sum(W2**2) + np.sum(W3**2) + np.sum(W4**2) + np.sum(W5**
    return total_reg_loss
    # END TODO ##################
    raise NotImplementedError()
```

شکل (۳-۳) تابع رگوله سازی

## ۳-۴- مشتق تابع رلو

```python
#DERIVATIVE OF RELU
def drelu(x):

    # START TODO ##################
    return np.where(x>=0,1,0)
    # END TODO ####################
    raise NotImplementedError()
```

شکل (۳-۴) مشتق تابع رلو

## ۳-۵- تابع Back Propagation1

```
grad = {}
dA5 = -1*(yd - y_pred)

m = X.shape[1]
W1,b1,W2,b2,W3,b3,W4,b4,W5,b5 = parameters
z1,a1,z2,a2,z3,a3,z4,a4,z5 = l

# Layer 5
dz5, dW5, db5 = back_prop_linear(dA5, z5, a4, 'relu', m, lmbda, W5)
grad['dW5'], grad['db5'] = dW5, db5
dA4 = back_prop_actf(W5, dz5)

# Layer 4
dz4, dW4, db4 = back_prop_linear(dA4, z4, a3, 'relu', m, lmbda, W4)
grad['dW4'], grad['db4'] = dW4, db4
dA3 = back_prop_actf(W4, dz4)

# Layer 3
dz3, dW3, db3 = back_prop_linear(dA3, z3, a2, 'relu', m, lmbda, W3)
grad['dW3'], grad['db3'] = dW3, db3
dA2 = back_prop_actf(W3, dz3)
```

شکل (۳-۵) تابع بازگشتی۱

## ۳-۶- پیاده سازی تابع آموزش

```
# START TODO ################
W1,b1,W2,b2,W3,b3,W4,b4,W5,b5 = parameters
losses = []

for i in range(num_iters):
    y_pred, l = feed_forward(X, parameters)
    loss_1 = loss_compute(y_pred, yd)
    loss_2 = regularization_L2(lmbda, W1, W2, W3, W4, W5, X.shape[1])
    total_loss = loss_1 + loss_2
    losses.append(total_loss)
    grads = Backpropagation1(X, yd, l, y_pred, parameters, lmbda)
    W1,b1,W2,b2,W3,b3,W4,b4,W5,b5 = grads['dW1'], grads['db1'], grads['dW2'], grads['db2'], grads['

parameters = [W1,b1,W2,b2,W3,b3,W4,b4,W5,b5]
return losses, parameters
# END TODO ###################
raise NotImplementedError()
```

شکل (۳-۶) تابع آموزش

```
assert np.allclose(losses[200],0.018760000609514545)
print('Sample Test passed', '\U0001F44D')
#You can change num_iters to 10000, will take around 10-15 minutes to train
'''
You can also play around with the model hyper-parameters to see if you can improve on our results
Try changing:
    - The model architecture (number of layers, number of neurons in each layer)
    - The batch size
    - The learning rate

Make sure you set the parameters to their original values before submitting the assignment
'''
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-151-f2d78d004c89> in <cell line: 0>()
      3 parameters = Initialization(8,16,64,64,16,1)#ALL ACTIVATION RELU
      4 losses, parameters_final = training(x_train, y_actual,parameters, eta = 0.05, num_iters=300,
lmbda = 0.1)
----> 5 assert np.allclose(losses[200],0.018760000609514545)
      6 print('Sample Test passed', '\U0001F44D')
      7 #You can change num_iters to 10000, will take around 10-15 minutes to train

AssertionError:
```

شکل (۳-۷) ارور آموزش مدل شبکه عصبی

## ۳-۶-۲- افزایش تعداد آموزش برای برطرف کردن مقدار دلخواه

```
#Sample test case
np.random.seed(2)
parameters = Initialization(8,16,64,64,16,1)#ALL ACTIVATION RELU
losses, parameters_final = training(x_train, y_actual,parameters, eta = 0.05, num_iters=1000, lmbda = 0
print(losses[200])
assert np.allclose(losses[200],0.018760000609514545)
print('Sample Test passed', '\U0001F44D')
#You can change num_iters to 10000, will take around 10-15 minutes to train
'''
You can also play around with the model hyper-parameters to see if you can improve on our results
Try changing:
    - The model architecture (number of layers, number of neurons in each layer)
    - The batch size
    - The learning rate

Make sure you set the parameters to their original values before submitting the assignment
'''
```

شکل (۳-۸) افزایش دفعات آموزش

مشکل رفع نشد.

### ۳-۶-۳- افزایش دوباره ی تعداد دفعات آموزش

```
#Sample test case
np.random.seed(2)
parameters = Initialization(8,16,64,64,16,1)#ALL ACTIVATION RELU
losses, parameters_final = training(x_train, y_actual,parameters, eta = 0.05, num_iters=10000, lmbda = 
print(losses[200])
assert np.allclose(losses[200],0.018760000609514545)
print('Sample Test passed', '\U0001F44D')
#You can change num_iters to 10000, will take around 10-15 minutes to train
'''
You can also play around with the model hyper-parameters to see if you can improve on our results
Try changing:
    - The model architecture (number of layers, number of neurons in each layer)
    - The batch size
    - The learning rate

Make sure you set the parameters to their original values before submitting the assignment
'''
```

شکل (۹-۳) افزایش تعداد دفعات آموزش به مقدار نهایی

```
#Sample test case
np.random.seed(2)
parameters = Initialization(8,16,64,64,16,1)#ALL ACTIVATION RELU
losses, parameters_final = training(x_train, y_actual,parameters, eta = 0.05, num_iters=10000, lmbda =
print(losses[200])
assert np.allclose(losses[200],0.018760000609514545)
print('Sample Test passed', '\U0001F44D')
#You can change num_iters to 10000, will take around 10-15 minutes to train
'''
You can also play around with the model hyper-parameters to see if you can improve on our results
Try changing:
    - The model architecture (number of layers, number of neurons in each layer)
    - The batch size
    - The learning rate

Make sure you set the parameters to their original values before submitting the assignment
'''
```

```
0.025879440153021377
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-175-8c10f11140ab> in <cell line: 0>()
      4 losses, parameters_final = training(x_train, y_actual,parameters, eta = 0.05, num_iters=10000,
lmbda = 0.1)
```

شکل (۱۰-۳) نتیجه با ۱۰۰۰۰ تعداد دفعات آموزش

## ۴-۶-۳- تغییر نرخ یادگیری و معماری شبکه

```
#Sample test case
np.random.seed(2)
parameters = Initialization(8, 32, 64, 64, 32, 1)#ALL ACTIVATION RELU
losses, parameters_final = training(x_train, y_actual,parameters, eta = 0.02, num_iters=10000, lmbda = 
print(losses[200])
assert np.allclose(losses[200],0.018760000609514545)
print('Sample Test passed', '\U0001F44D')
#You can change num_iters to 10000, will take around 10-15 minutes to train
'''
You can also play around with the model hyper-parameters to see if you can improve on our results
Try changing:
    - The model architecture (number of layers, number of neurons in each layer)
    - The batch size
    - The learning rate

Make sure you set the parameters to their original values before submitting the assignment
'''
```

شکل (۱۱-۳) تغییر نرخ یادگیری و معماری شبکه

```
#Sample test case
np.random.seed(2)
parameters = Initialization(8, 32, 64, 64, 32, 1)#ALL ACTIVATION RELU
losses, parameters_final = training(x_train, y_actual,parameters, eta = 0.02, num_iters=5000, lmbda = 0.
print(losses[200])
assert np.allclose(losses[200],0.018760000609514545)
print('Sample Test passed', '\U0001F44D')
#You can change num_iters to 10000, will take around 10-15 minutes to train
'''
You can also play around with the model hyper-parameters to see if you can improve on our results
Try changing:
    - The model architecture (number of layers, number of neurons in each layer)
    - The batch size
    - The learning rate

Make sure you set the parameters to their original values before submitting the assignment
'''
2.32279077491795
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-177-07180c178d7a> in <cell line: 0>()
```

شکل (۱۲-۳) نرخ تلفات

### ۳-۶-۵- تغییر نرخ یادگیری

```
#Sample test case
np.random.seed(2)
parameters = Initialization(8, 16, 32, 32, 16, 1)
#ALL ACTIVATION RELU
losses, parameters_final = training(x_train, y_actual,parameters, eta = 0.05, num_iters=201, lmbda = 0.
print(losses[200])
assert np.allclose(losses[200],0.018760000609514545)
print('Sample Test passed', '\U0001F44D')
#You can change num_iters to 10000, will take around 10-15 minutes to train
'''
You can also play around with the model hyper-parameters to see if you can improve on our results
Try changing:
    - The model architecture (number of layers, number of neurons in each layer)
    - The batch size
    - The learning rate

Make sure you set the parameters to their original values before submitting the assignment
'''
```

```
0.05628153285817958
```

شکل (۱۳-۳) نرخ یادگیری ۰.۰۵

مشاهده می شود که با شبکه ی قبلی بهتر نتیجه می گرفتیم.

### ۳-۶-۶- تغییر نرخ یادگیری به ۰.۱

```
#Sample test case
np.random.seed(2)
parameters = Initialization(8, 16, 32, 32, 16, 1)
#ALL ACTIVATION RELU
losses, parameters_final = training(x_train, y_actual,parameters, eta = 0.1, num_iters=201, lmbda = 0.1
print(losses[200])
assert np.allclose(losses[200],0.018760000609514545)
print('Sample Test passed', '\U0001F44D')
#You can change num_iters to 10000, will take around 10-15 minutes to train
'''
You can also play around with the model hyper-parameters to see if you can improve on our results
Try changing:
    - The model architecture (number of layers, number of neurons in each layer)
    - The batch size
    - The learning rate

Make sure you set the parameters to their original values before submitting the assignment
'''
```

```
0.05628153285817958
```

شکل (۱۴-۳) نتیجه با نرخ یادگیری ۰.۱

### ۳-۶-۷- تغییر معماری شبکه

```
#Sample test case
np.random.seed(2)
parameters = Initialization(8, 8, 16, 16, 8, 1)
#ALL ACTIVATION RELU
losses, parameters_final = training(x_train, y_actual,parameters, eta = 0.1, num_iters=201, lmbda = 0.1)
print(losses[200])
assert np.allclose(losses[200],0.018760000609514545)
print('Sample Test passed', '\U0001F44D')
#You can change num_iters to 10000, will take around 10-15 minutes to train
'''
You can also play around with the model hyper-parameters to see if you can improve on our results
Try changing:
    - The model architecture (number of layers, number of neurons in each layer)
    - The batch size
    - The learning rate

Make sure you set the parameters to their original values before submitting the assignment
'''
```

```
0.10372333800348986
```

شکل (۱۵-۳) نتیجه با تغییر معماری شبکه

تلفات بیشتر شد.

### ۳-۶-۸- تغییر نرخ یادگیری

```
#Sample test case
np.random.seed(2)
parameters = Initialization(8, 8, 16, 16, 8, 1)
#ALL ACTIVATION RELU
losses, parameters_final = training(x_train, y_actual,parameters, eta = 0.05, num_iters=201, lmbda = 0.1)
print(losses[200])
assert np.allclose(losses[200],0.018760000609514545)
print('Sample Test passed', '\U0001F44D')
#You can change num_iters to 10000, will take around 10-15 minutes to train
'''
You can also play around with the model hyper-parameters to see if you can improve on our results
Try changing:
    - The model architecture (number of layers, number of neurons in each layer)
    - The batch size
    - The learning rate

Make sure you set the parameters to their original values before submitting the assignment
'''
```

```
0.10372333800348986
```

شکل (۱۶-۳) تلفات با نرخ ۰.۵

مشاهده می شود تفاوت چندانی به وجود نمی آید باید سراغ دیگر هایپر پارامتر ها برویم.

### ۹-۶-۳- تغییر معماری

```
#Sample test case
np.random.seed(2)
parameters = Initialization(8, 32, 128, 128, 32, 1)
#ALL ACTIVATION RELU
losses, parameters_final = training(x_train, y_actual,parameters, eta = 0.05, num_iters=201, lmbda = 0.1
print(losses[200])
assert np.allclose(losses[200],0.018760000609514545)
print('Sample Test passed', '\U0001F44D')
#You can change num_iters to 10000, will take around 10-15 minutes to train
'''
You can also play around with the model hyper-parameters to see if you can improve on our results
Try changing:
    - The model architecture (number of layers, number of neurons in each layer)
    - The batch size
    - The learning rate

Make sure you set the parameters to their original values before submitting the assignment
'''
```

```
53.258497609209165
```

شکل (۱۷-۳) نتیجه تلفات پس از تغییر معماری

این معماری اصلا مطرح نیست.

### ۱۰-۶-۳- بازگشت به معماری اولیه و تغییر لامبدا

```
e test case
dom.seed(2)
ters = Initialization(8, 16, 64, 64, 16, 1)
CTIVATION RELU
, parameters_final = training(x_train, y_actual,parameters, eta = 0.05, num_iters=201, lmbda = 0.05)
losses[200])
 np.allclose(losses[200],0.018760000609514545)
'Sample Test passed', '\U0001F44D')
an change num_iters to 10000, will take around 10-15 minutes to train

n also play around with the model hyper-parameters to see if you can improve on our results
anging:
The model architecture (number of layers, number of neurons in each layer)
The batch size
The learning rate

ure you set the parameters to their original values before submitting the assignment

0.025879431003314232
```

شکل (۱۸-۳) نتیجه با لامبدای ۰.۰۵

```
e test case
dom.seed(2)
ters = Initialization(8, 16, 64, 64, 16, 1)
CTIVATION RELU
, parameters_final = training(x_train, y_actual,parameters, eta = 0.05, num_iters=201, lmbda = 0.01)
losses[200])
 np.allclose(losses[200],0.018760000609514545)
'Sample Test passed', '\U0001F44D')
an change num_iters to 10000, will take around 10-15 minutes to train

n also play around with the model hyper-parameters to see if you can improve on our results
anging:
The model architecture (number of layers, number of neurons in each layer)
The batch size
The learning rate

ure you set the parameters to their original values before submitting the assignment

0.025879423684045083
```

شکل (۱۹-۳) نتیجه با لامبدای ۰.۰۱

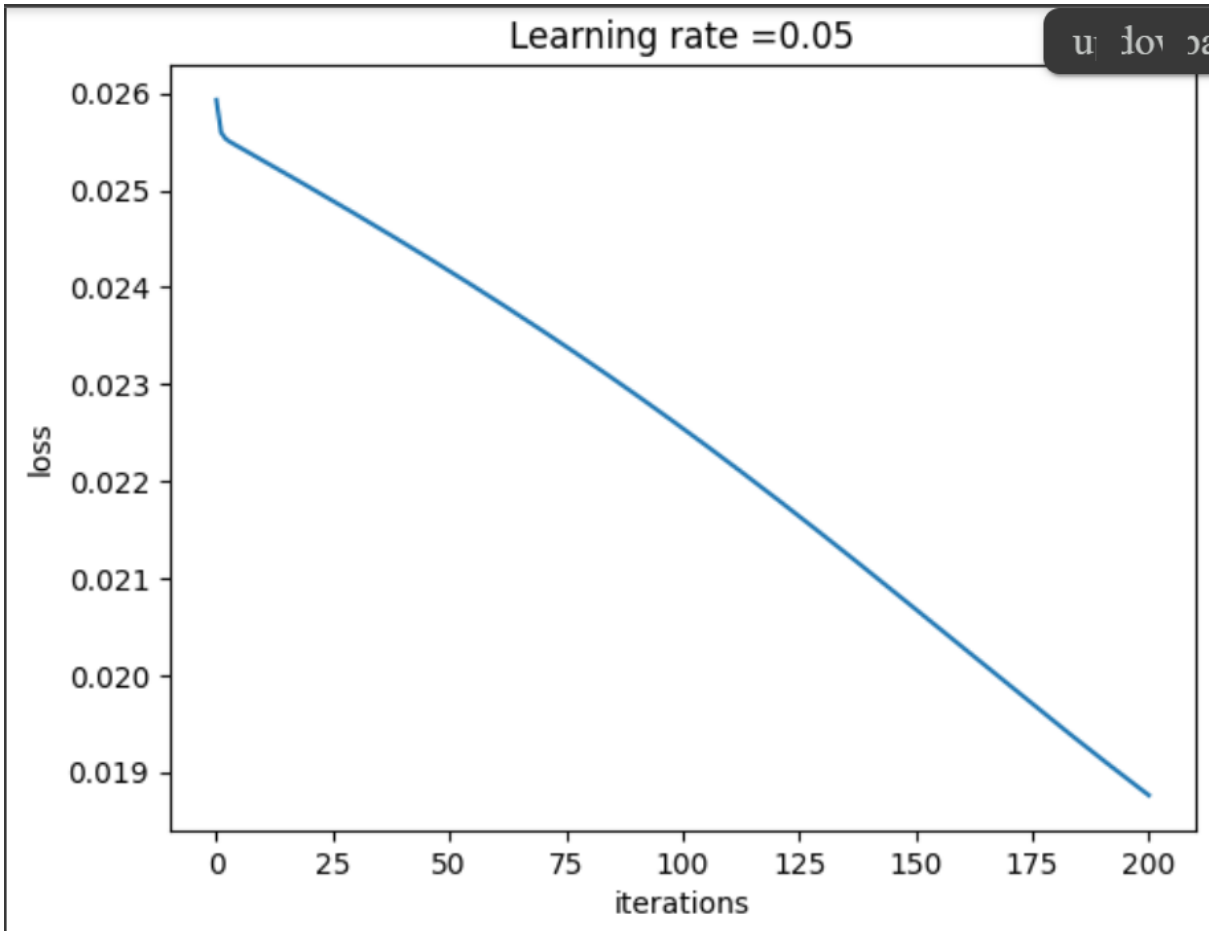
تغییری ایجاد نشد.

### ۱۱-۶-۳- رفع ارور

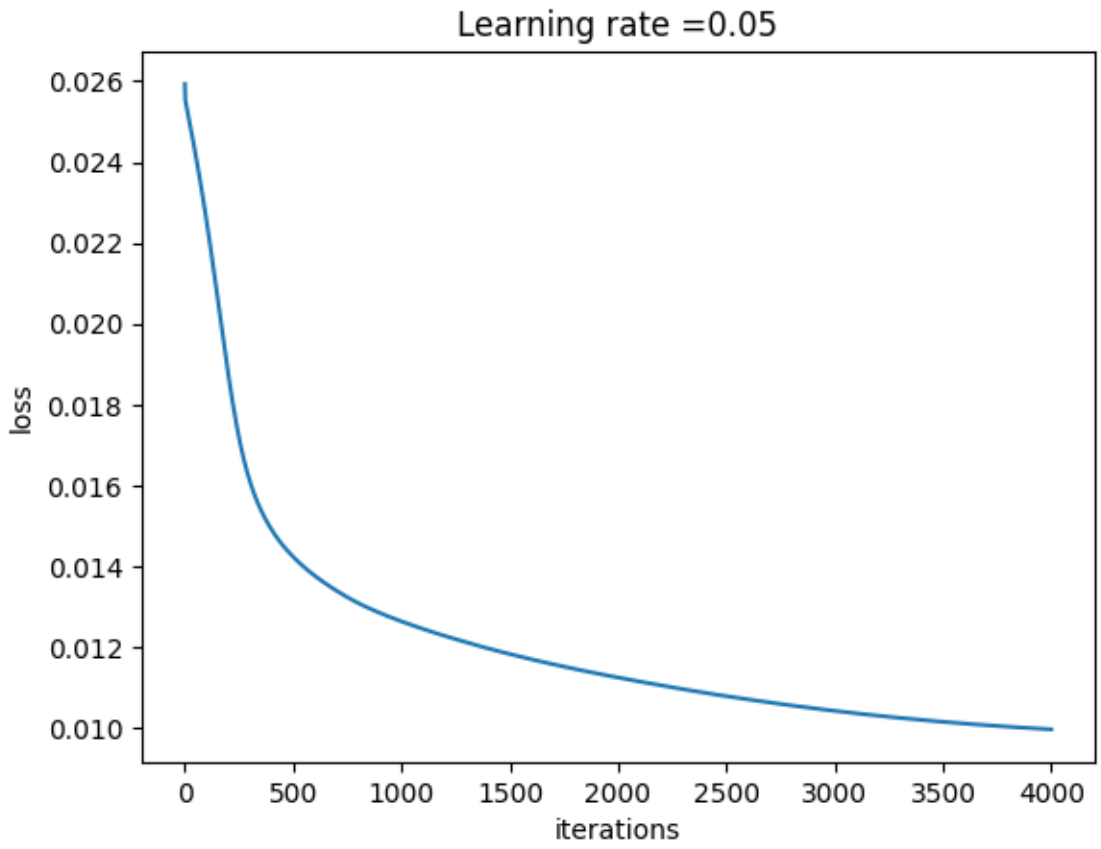
```
W1 = W1 - eta * grads['dW1']
b1 = b1 - eta * grads['db1']
W2 = W2 - eta * grads['dW2']
b2 = b2 - eta * grads['db2']
W3 = W3 - eta * grads['dW3']
b3 = b3 - eta * grads['db3']
W4 = W4 - eta * grads['dW4']
b4 = b4 - eta * grads['db4']
W5 = W5 - eta * grads['dW5']
b5 = b5 - eta * grads['db5']
```

شکل (۲۰-۳) اصلاح کد

متوجه شدم که مشکل از کد بود و وزن ها اصلاح نمی شدند که مشکل برطرف شد.




شکل (۲۱-۳) تابع تلفات




شکل (۲۲-۳) تابع تلفات با ۴۰۰۰ تعداد Iteration

مشاهده می شود که همه چیز به خوبی پیش رفته و با ۴۰۰۰ ایپاک به ۰.۰۱ تلفات رسیدیم.

```
#Let's see how well our model works on TEST data
df_test  = pd.read_csv('california_housing_test.csv')
scaler = MinMaxScaler()
scaler.fit(df)
test_scaled_data = scaler.transform(df_test)
test_scaled_df = pd.DataFrame(test_scaled_data,columns=df_test.columns) #make sure column names are reta

xtest = test_scaled_df.iloc[:,0:8]
ytest = test_scaled_df.iloc[:,8:9]
xtest  = xtest.values.T
ytest = ytest.values.reshape(1,-1)

ytest_pred = feed_forward(xtest,parameters_final)[0]
print(mean_squared_error(ytest,ytest_pred))
```

```
0.019718963150468425
```

شکل (۲۳-۳) تلفات روی دیتاست اصلی

# فصل ۴: پیاده سازی مدل روی دیتاست

# MNIST

## ۴-۱- آماده سازی دیتاست

```python
import kagglehub
import shutil
import os

# Download dataset (goes to default kagglehub cache)
path = kagglehub.dataset_download("oddrationale/mnist-in-csv")

# Your target folder
target_folder = "sample_data/mnist"

# Make sure the folder exists
os.makedirs(target_folder, exist_ok=True)

# Copy all files from the kagglehub download path to sample_data
for file in os.listdir(path):
    shutil.copy(os.path.join(path, file), target_folder)

print("Files copied to:", target_folder)
```

```python
import pandas as pd

mnistdf = pd.read_csv('/content/sample_data/mnist/mnist_train.csv')
df = mnistdf

scaler= MinMaxScaler()
scaler.fit(df)
scaled_values = scaler.transform(df)
scaled_df = pd.DataFrame(scaled_values, columns=df.columns)
```

شکل (۱-۴) دریافت و اسکیل دیتاست

## ۴-۲- طراحی تلفات با کراس آنتروپی

$$L = -\frac{1}{N}\left[\sum_{j=1}^{N}\left[t_j \log(p_j) + (1-t_j)\log(1-p_j)\right]\right]$$

for $N$ data points where $t_i$ is the truth value taking a value 0 or 1 and $p_i$ is the Softmax probability for the $i^{th}$ data point.

شکل (۴-۲) فرمول کراس آنتروپی

```python
import numpy as np

def loss_entropy(y_pred, yd):
    '''
    Inputs:
    - y_pred: numpy array of shape (num_classes, m), predicted probabilities
    - yd: numpy array of shape (num_classes, m), one-hot encoded true labels

    Output:
    - loss: scalar cross-entropy loss
    '''
    # START TODO #################
    epsilon = 1e-15
    y_pred = np.clip(y_pred, epsilon, 1 - epsilon)

    m = y_pred.shape[1]  # number of examples
    loss = -np.sum(yd * np.log(y_pred)) / m
    return loss
    # END TODO ###################
    raise NotImplementedError()
```

شکل (۴-۳) تابع تلفات کراس آنتروپی

## ۳-۴- آموزش مدل

پس از تغییر توابع برای تطبیق با مدل Mnist حالا وقت آن است که مدل را آموزش دهیم.

```
train_x = train_x.astype(np.float64)
xtest = xtest.astype(np.float64)
train_y = train_y.astype(np.float64)
ytest = ytest.astype(np.float64)

W = Model(X_train=train_x, y_train=train_y, X_test=xtest, y_test=ytest,
          layers_dims=[784, 16, 64, 64, 16, 10])
```

```
Epoch 0 | Loss: 23.9000 | Train Acc: 11.24% | Test Acc: 11.35%
Epoch 100 | Loss: 2.7588 | Train Acc: 11.24% | Test Acc: 11.35%
Epoch 200 | Loss: 2.4453 | Train Acc: 11.24% | Test Acc: 11.35%
Epoch 300 | Loss: 2.3515 | Train Acc: 11.24% | Test Acc: 11.35%
Epoch 400 | Loss: 2.3193 | Train Acc: 11.24% | Test Acc: 11.35%
Epoch 500 | Loss: 2.3077 | Train Acc: 11.24% | Test Acc: 11.35%
Epoch 600 | Loss: 2.3036 | Train Acc: 11.24% | Test Acc: 11.35%
Epoch 700 | Loss: 2.3020 | Train Acc: 11.24% | Test Acc: 11.35%
Epoch 800 | Loss: 2.3015 | Train Acc: 11.24% | Test Acc: 11.35%
Epoch 900 | Loss: 2.3013 | Train Acc: 11.24% | Test Acc: 11.35%
```

شکل (۴-۴) خروجی آموزش اول Mnist

مقدار تلفات ما با مقدار دقت آموزش و تست مطابقت ندارد و این باگ باید برطرف شود چون در این وضعیت مدل ما یادگیری ندارد.

```
z = np.dot(W5, a) + b5
y_out = z
l.append(z)

return y_out, l
```

شکل (۴-۵) خروجی لایه آخر فید فوروارد

باید خروجی آخر را از soft_max رد شود.

```
Epoch 0 | Loss: 2.3026 | Train Acc: 11.23% | Test Acc: 11.23%
Epoch 100 | Loss: 2.3024 | Train Acc: 11.24% | Test Acc: 11.24%
Epoch 200 | Loss: 2.3023 | Train Acc: 11.24% | Test Acc: 11.24%
Epoch 300 | Loss: 2.3022 | Train Acc: 11.24% | Test Acc: 11.24%
Epoch 400 | Loss: 2.3021 | Train Acc: 11.24% | Test Acc: 11.24%
Epoch 500 | Loss: 2.3020 | Train Acc: 11.24% | Test Acc: 11.24%
```

شکل (۴-۶) خروجی جدید Mnist

مشاهده می شود که همچنان یک مشکلی وجود دارد و مدل دارد فقط پاسخ را حدس می زند و یادگیری در کار نیست.

```python
def Initialization(n_x, n_h1, n_h2, n_h3, n_h4, n_y):
    np.random.seed(1)
    parameters = {
        'W1': np.random.randn(n_h1, n_x) * np.sqrt(2. / n_x),
        'b1': np.zeros((n_h1, 1)),
        'W2': np.random.randn(n_h2, n_h1) * np.sqrt(2. / n_h1),
        'b2': np.zeros((n_h2, 1)),
        'W3': np.random.randn(n_h3, n_h2) * np.sqrt(2. / n_h2),
        'b3': np.zeros((n_h3, 1)),
        'W4': np.random.randn(n_h4, n_h3) * np.sqrt(2. / n_h3),
        'b4': np.zeros((n_h4, 1)),
        'W5': np.random.randn(n_y, n_h4) * np.sqrt(2. / n_h4),
        'b5': np.zeros((n_y, 1))
    }
    return parameters
```

شکل (۴-۷) اصلاح تابع Initialization

```python
W = Model(X_train=train_x, y_train=train_y, X_test=xtest, y_test=ytest,
          layers_dims=[784, 16, 64, 64, 16, 10],learning_rate = 0.5)
```

شکل (۴-۸) تغییر پارامتر نرخ یادگیری

```
W = Model(X_train=train_x, y_train=train_y, X_test=xtest, y_test=ytest,
          layers_dims=[784, 16, 64, 64, 16, 10],learning_rate = 0.5)
```

```
Epoch 0 | Loss: 2.3124 | Train Acc: 16.35% | Test Acc: 16.19%
Epoch 50 | Loss: 2.0864 | Train Acc: 31.95% | Test Acc: 32.31%
Epoch 100 | Loss: 1.4041 | Train Acc: 50.11% | Test Acc: 50.03%
Epoch 150 | Loss: 1.0945 | Train Acc: 55.34% | Test Acc: 54.95%
Epoch 200 | Loss: 0.9696 | Train Acc: 61.63% | Test Acc: 61.42%
Epoch 250 | Loss: 0.9011 | Train Acc: 66.18% | Test Acc: 65.85%
Epoch 300 | Loss: 0.8759 | Train Acc: 71.67% | Test Acc: 71.20%
Epoch 350 | Loss: 0.7292 | Train Acc: 75.80% | Test Acc: 75.18%
Epoch 400 | Loss: 0.9118 | Train Acc: 67.19% | Test Acc: 66.70%
Epoch 450 | Loss: 0.8716 | Train Acc: 68.88% | Test Acc: 68.48%
Epoch 500 | Loss: 0.8758 | Train Acc: 68.63% | Test Acc: 67.81%
Epoch 550 | Loss: 0.8658 | Train Acc: 68.32% | Test Acc: 67.55%
Epoch 600 | Loss: 0.8992 | Train Acc: 69.80% | Test Acc: 69.02%
Epoch 650 | Loss: 0.8428 | Train Acc: 71.93% | Test Acc: 71.02%
Epoch 700 | Loss: 0.7052 | Train Acc: 71.36% | Test Acc: 70.62%
Epoch 750 | Loss: 0.6628 | Train Acc: 72.31% | Test Acc: 71.43%
Epoch 800 | Loss: 0.6828 | Train Acc: 75.48% | Test Acc: 74.52%
Epoch 850 | Loss: 0.6924 | Train Acc: 76.62% | Test Acc: 75.36%
Epoch 900 | Loss: 0.6116 | Train Acc: 74.80% | Test Acc: 73.64%
Epoch 950 | Loss: 0.6832 | Train Acc: 77.59% | Test Acc: 76.40%
```

شکل (۹-۴) خروجی یادگیری

مشـاهده می شـود که یادگیری انجام می شـود اما گاهی به نقطه ی مینیمم نزدیک می شـویم و ناگهان دور می شویم که نشان دهنده ی آن است که نرخ یادگیری بالا است.

```python
import os
import pickle

save_path = '/content/drive/MyDrive/mnist/'

files = [f for f in os.listdir(save_path) if f.endswith('.pkl')]

if not files:
    print("No model file found!")
else:
    latest_file = sorted(files)[-1]
    latest_path = os.path.join(save_path, latest_file)

    with open(latest_path, 'rb') as f:
        model = pickle.load(f)

    print(f"Model is now updated: {latest_file}")

# train

model = Model(X_train=train_x, y_train=train_y, X_test=xtest, y_test=ytest,
          layers_dims=[784, 32, 64, 64, 32, 10],learning_rate = 0.1,parameters=W,epochs=1000)

# Save to a file
save_path = '/content/drive/MyDrive/mnist/'
os.makedirs(save_path, exist_ok=True)

filename = f'mnist_model_{datetime.datetime.now().strftime("%Y-%m-%d_%H-%M-%S")}.pkl'

with open(os.path.join(save_path, filename), 'wb') as f:
    pickle.dump(W, f)
```

شکل (۱۰-۴) نوشتن کد برای ذخیره سازی و بارگزاری مجدد مدل برای ذخیره و از دست ندادن مدل ها

```python
model = Model(X_train=train_x, y_train=train_y, X_test=xtest, y_test=ytest,
          layers_dims=[784, 32, 64, 64, 32, 10],learning_rate = 0.1,parameters=W,epochs=1001)
```

شکل (۱۱-۴) آموزش مجدد با پارامتر های جدید

```
Model is now updated: mnist_model_2025-04-11_15-11-47.pkl
Epoch 0 | Loss: 0.7004 | Train Acc: 76.29% | Test Acc: 74.96%
Epoch 50 | Loss: 0.6434 | Train Acc: 77.33% | Test Acc: 75.87%
Epoch 100 | Loss: 0.6440 | Train Acc: 77.36% | Test Acc: 75.85%
Epoch 150 | Loss: 0.6444 | Train Acc: 77.36% | Test Acc: 75.82%
Epoch 200 | Loss: 0.6459 | Train Acc: 77.33% | Test Acc: 75.75%
Epoch 250 | Loss: 0.6474 | Train Acc: 77.28% | Test Acc: 75.73%
Epoch 300 | Loss: 0.6487 | Train Acc: 77.25% | Test Acc: 75.70%
Epoch 350 | Loss: 0.6498 | Train Acc: 77.25% | Test Acc: 75.67%
Epoch 400 | Loss: 0.6497 | Train Acc: 77.23% | Test Acc: 75.58%
Epoch 450 | Loss: 0.6504 | Train Acc: 77.22% | Test Acc: 75.54%
Epoch 500 | Loss: 0.6511 | Train Acc: 77.18% | Test Acc: 75.52%
Epoch 550 | Loss: 0.6518 | Train Acc: 77.16% | Test Acc: 75.45%
Epoch 600 | Loss: 0.6517 | Train Acc: 77.10% | Test Acc: 75.43%
Epoch 650 | Loss: 0.6524 | Train Acc: 77.07% | Test Acc: 75.32%
Epoch 700 | Loss: 0.6531 | Train Acc: 77.04% | Test Acc: 75.32%
Epoch 750 | Loss: 0.6541 | Train Acc: 77.03% | Test Acc: 75.26%
Epoch 800 | Loss: 0.6546 | Train Acc: 76.98% | Test Acc: 75.24%
Epoch 850 | Loss: 0.6552 | Train Acc: 76.92% | Test Acc: 75.25%
Epoch 900 | Loss: 0.6565 | Train Acc: 76.86% | Test Acc: 75.18%
Epoch 950 | Loss: 0.6570 | Train Acc: 76.80% | Test Acc: 75.15%
Epoch 1000 | Loss: 0.6586 | Train Acc: 76.81% | Test Acc: 75.11%
```

شکل (۱۲-۴) نتیجه آموزش

```
model = Model(X_train=train_x, y_train=train_y, X_test=xtest, y_test=ytest,
          layers_dims=[784, 64, 128, 128, 64, 10],learning_rate = 0.1,parameters=W,epochs=501)
```

شکل (۱۳-۴) اصلاح معماری شبکه

یک مشـــکلی که این تابع مدل ما دارد این اســـت که بهترین وزن ها را ذخیره نمی کند در عوض آخرین را ذخیره می کند که باید این را درست کنیم.

```
Model is now updated: mnist_model_2025-04-11_15-26-56.pkl
Epoch 0 | Loss: 0.6578 | Train Acc: 76.76% | Test Acc: 75.09%
Epoch 50 | Loss: 0.6587 | Train Acc: 76.74% | Test Acc: 75.08%
Epoch 100 | Loss: 0.6596 | Train Acc: 76.74% | Test Acc: 75.11%
Epoch 150 | Loss: 0.6606 | Train Acc: 76.74% | Test Acc: 75.01%
Epoch 200 | Loss: 0.6613 | Train Acc: 76.68% | Test Acc: 74.89%
Epoch 250 | Loss: 0.6628 | Train Acc: 76.66% | Test Acc: 74.84%
Epoch 300 | Loss: 0.6629 | Train Acc: 76.62% | Test Acc: 74.77%
Epoch 350 | Loss: 0.6627 | Train Acc: 76.54% | Test Acc: 74.78%
Epoch 400 | Loss: 0.6636 | Train Acc: 76.52% | Test Acc: 74.73%
Epoch 450 | Loss: 0.6642 | Train Acc: 76.48% | Test Acc: 74.77%
Epoch 500 | Loss: 0.6653 | Train Acc: 76.51% | Test Acc: 74.79%
```

شکل (۴-۱۴) خروجی آموزش دوباره مدل

می بینیم که باز هم این دقت افزایش نداشت که حتی کاهش هم داشت.

سعی می کنیم با نرخ یادگیری بیشتر هم تلاش کنیم هرچند که این مدل را در نظر نمی گیریم.

```
    losses = []
    best_parameters = parameters

    for i in range(epochs):
        A, caches = feed_forward(X_train, parameters)
        loss = loss_entropy(A, y_train)
        grads = Backpropagation1(X_train, y_train, caches, A, parameters, lambda_reg)
        parameters = Update(parameters, grads, learning_rate)

        if(loss <= min(losses)):
          #update best model if it's the bes
          best_parameters = parameters

        # add to losses
        losses.append(loss)
```

شکل (۴-۱۵) تغییر تابع مدل برای ذخیره بهترین مدل با کمترین تلفات

```
model = Model(X_train=train_x, y_train=train_y, X_test=xtest, y_test=ytest,
          layers_dims=[784, 64, 128, 128, 64, 10],learning_rate = 0.8,parameters=W,epochs=501)
```

شکل (۴-۱۶) تغییر پارامتر های مدل

```
Model is now updated: mnist_model_2025-04-11_15-26-56.pkl
Epoch 0 | Loss: 0.6697 | Train Acc: 76.78% | Test Acc: 75.18%
Epoch 50 | Loss: 7.9591 | Train Acc: 38.87% | Test Acc: 38.64%
Epoch 100 | Loss: 11.2525 | Train Acc: 31.08% | Test Acc: 30.62%
Epoch 150 | Loss: 6.7834 | Train Acc: 48.81% | Test Acc: 48.75%
Epoch 200 | Loss: 6.4163 | Train Acc: 50.32% | Test Acc: 50.03%
Epoch 250 | Loss: 6.2594 | Train Acc: 50.59% | Test Acc: 50.21%
Epoch 300 | Loss: 6.1863 | Train Acc: 50.54% | Test Acc: 50.17%
Epoch 350 | Loss: 6.1345 | Train Acc: 50.44% | Test Acc: 50.31%
Epoch 400 | Loss: 6.1082 | Train Acc: 50.23% | Test Acc: 50.01%
Epoch 450 | Loss: 6.2515 | Train Acc: 51.03% | Test Acc: 50.93%
Epoch 500 | Loss: 6.3467 | Train Acc: 50.79% | Test Acc: 50.75%
```

شکل (۱۷-۴) خروجی آموزش مدل

مشاهده می کنید که این خروجی هم زیاد جالب نیست بنابراین از همان معماری که بیشترین دقت را به ما داد استفاده می کنیم.

سعی ما باید این باشد که با استفاده از تغییر دیگر هایپر پارامتر ها دقت را بهبود دهیم.

```
model = Model(X_train=train_x, y_train=train_y, X_test=xtest, y_test=ytest,
          layers_dims=[784, 32, 64, 64, 32, 10],learning_rate = 0.05,parameters=W,epochs=500)
```

شکل (۱۸-۴) کاهش نرخ یادگیری

به نادرستی مدل را w داده بودیم که پس از هر کدام از یادگیری ها دوباره روی وزن های اولین یادگیری آموزش انجام می شد که این درست شد.

الآن مدل ما بهترین وزن ها را بر می گرداند و همان ها بروزرسانی می شوند تا به سمت بهینه ترین برویم.

```
model = Model(X_train=train_x, y_train=train_y, X_test=xtest, y_test=ytest,
          layers_dims=[784, 32, 64, 64, 32, 10],learning_rate = 0.5,parameters=model,epochs=500)
```

شکل (۱۹-۴) تغییر مدل

```
Model is now updated: mnist_model_2025-04-11_15-11-47.pkl
Epoch 0 | Loss: 0.7004 | Train Acc: 78.54% | Test Acc: 76.98%
Epoch 50 | Loss: 0.6482 | Train Acc: 77.36% | Test Acc: 75.81%
Epoch 100 | Loss: 0.6463 | Train Acc: 77.10% | Test Acc: 75.66%
Epoch 150 | Loss: 0.7043 | Train Acc: 78.36% | Test Acc: 76.75%
Epoch 200 | Loss: 0.6723 | Train Acc: 77.61% | Test Acc: 76.05%
Epoch 250 | Loss: 0.6506 | Train Acc: 75.41% | Test Acc: 73.93%
Epoch 300 | Loss: 0.7256 | Train Acc: 77.34% | Test Acc: 75.63%
Epoch 350 | Loss: 0.6654 | Train Acc: 75.45% | Test Acc: 73.80%
Epoch 400 | Loss: 0.6958 | Train Acc: 74.88% | Test Acc: 73.42%
Epoch 450 | Loss: 0.6854 | Train Acc: 74.95% | Test Acc: 73.45%
Epoch 500 | Loss: 0.7409 | Train Acc: 76.76% | Test Acc: 75.28%
Best accuracy in epoch:1 & loss:0.6153474888325049
```

شکل (۲۰-۴) نتیجه خروجی

```
model = Model(X_train=train_x, y_train=train_y, X_test=xtest, y_test=ytest,
        layers_dims=[784, 32, 64, 64, 32, 10],learning_rate = 0.05,parameters=model,epochs=500)
```

شکل (۲۱-۴) کاهش نرخ یادگیری

```
with open(os.path.join(save_path, filename), 'wb') as f:
    pickle.dump(model, f)
```

شکل (۲۲-۴) اصلاح ذخیره سازی

در ذخیره سازی هم اشتباه کرده بودیم که اصلاح شد.

```
Model is now updated: mnist_model_2025-04-11_15-11-47.pkl
Epoch 0 | Loss: 0.7004 | Train Acc: 75.96% | Test Acc: 74.59%
Epoch 50 | Loss: 0.6431 | Train Acc: 77.33% | Test Acc: 75.93%
Epoch 100 | Loss: 0.6436 | Train Acc: 77.33% | Test Acc: 75.85%
Epoch 150 | Loss: 0.6441 | Train Acc: 77.33% | Test Acc: 75.86%
Epoch 200 | Loss: 0.6442 | Train Acc: 77.35% | Test Acc: 75.84%
Epoch 250 | Loss: 0.6439 | Train Acc: 77.35% | Test Acc: 75.86%
Epoch 300 | Loss: 0.6446 | Train Acc: 77.35% | Test Acc: 75.80%
Epoch 350 | Loss: 0.6454 | Train Acc: 77.33% | Test Acc: 75.75%
Epoch 400 | Loss: 0.6461 | Train Acc: 77.32% | Test Acc: 75.74%
Epoch 450 | Loss: 0.6467 | Train Acc: 77.30% | Test Acc: 75.73%
Epoch 500 | Loss: 0.6476 | Train Acc: 77.29% | Test Acc: 75.72%
Best accuracy in epoch:26 & loss:0.6423698195975674
```

شکل (۲۳-۴) خروجی آموزش

مشـاهده می شـود که این تقریبا بهینه ترین حالت ما هسـت که یا در مینیمم محلی هسـتیم یا بهینه ترین هست.

یا باید دوباره با مقادیر اولیه ی رندوم شـــروع کنیم یا اینکه با نرخ یادگیری کم ســـعی کنیم همین دقت را کمی افزایش دهیم.

```
print(f"Epoch {i} | Loss: {loss:.4f} | Train Acc: {acc_train:.2f}% | Test Acc: {acc_test:.2f}%")
if(len(acc_tests)!=0 and acc_test >= max(acc_tests)):
  #update best model if it's the bes
  best_parameters = parameters
  best_epoch = i
  print(f"Best Epoch is updated to {best_epoch}")
```

شکل (۲۴-۴) اصلاح تابع مدل

تابع را اشتباه حساب کرده بودیم که با مینیمم کار میکرد و بهترین آپدیت نمی شد.

دوباره شروع به آموزش مدل با مقادیر ابتدایی می کنیم.

```
model = Model(X_train=train_x, y_train=train_y, X_test=xtest, y_test=ytest,
          layers_dims=[784, 32, 64, 64, 32, 10],learning_rate = 0.5,parameters=-1,epochs=200,lambda_reg=0.1)
```

شکل (۲۵-۴) اصلاح پارامتر های مدل

```
Epoch 177 | Loss: 0.6580 | Train Acc: 75.64% | Test Acc: 75.89% | Best_epoch:176
Epoch 178 | Loss: 0.6530 | Train Acc: 75.60% | Test Acc: 75.97% | Best_epoch:177
Epoch 179 | Loss: 0.6521 | Train Acc: 75.81% | Test Acc: 76.03% | Best_epoch:178
Epoch 180 | Loss: 0.6476 | Train Acc: 75.73% | Test Acc: 76.07% | Best_epoch:179
Epoch 181 | Loss: 0.6474 | Train Acc: 75.96% | Test Acc: 76.14% | Best_epoch:180
Epoch 182 | Loss: 0.6432 | Train Acc: 75.86% | Test Acc: 76.13% | Best_epoch:181
Epoch 183 | Loss: 0.6436 | Train Acc: 76.08% | Test Acc: 76.23% | Best_epoch:181
Epoch 184 | Loss: 0.6394 | Train Acc: 75.94% | Test Acc: 76.28% | Best_epoch:183
Epoch 185 | Loss: 0.6403 | Train Acc: 76.18% | Test Acc: 76.35% | Best_epoch:184
Epoch 186 | Loss: 0.6361 | Train Acc: 76.04% | Test Acc: 76.34% | Best_epoch:185
Epoch 187 | Loss: 0.6378 | Train Acc: 76.30% | Test Acc: 76.40% | Best_epoch:185
Epoch 188 | Loss: 0.6336 | Train Acc: 76.10% | Test Acc: 76.38% | Best_epoch:187
Epoch 189 | Loss: 0.6362 | Train Acc: 76.36% | Test Acc: 76.48% | Best_epoch:187
Epoch 190 | Loss: 0.6321 | Train Acc: 76.15% | Test Acc: 76.39% | Best_epoch:189
Epoch 191 | Loss: 0.6354 | Train Acc: 76.48% | Test Acc: 76.56% | Best_epoch:189
Epoch 192 | Loss: 0.6315 | Train Acc: 76.12% | Test Acc: 76.47% | Best_epoch:191
Epoch 193 | Loss: 0.6350 | Train Acc: 76.52% | Test Acc: 76.66% | Best_epoch:191
Epoch 194 | Loss: 0.6320 | Train Acc: 76.07% | Test Acc: 76.47% | Best_epoch:193
Epoch 195 | Loss: 0.6348 | Train Acc: 76.56% | Test Acc: 76.62% | Best_epoch:193
Epoch 196 | Loss: 0.6325 | Train Acc: 76.01% | Test Acc: 76.43% | Best_epoch:193
Epoch 197 | Loss: 0.6329 | Train Acc: 76.61% | Test Acc: 76.61% | Best_epoch:193
Epoch 198 | Loss: 0.6295 | Train Acc: 76.09% | Test Acc: 76.47% | Best_epoch:193
Epoch 199 | Loss: 0.6310 | Train Acc: 76.65% | Test Acc: 76.64% | Best_epoch:193
Epoch 200 | Loss: 0.6268 | Train Acc: 76.15% | Test Acc: 76.48% | Best_epoch:193
Best accuracy in epoch:193 | loss:0.6349660021997134 | Train Acc:76.51666666666667 | Test Acc:9.01
```

شکل (۲۶-۴) نتیجه

باز به همان نتیجه رسیدیم.

```
Epoch 111 | Loss: 0.6142 | Train Acc: 76.79% | Test Acc: 77.11% | Best_epoch:110
Epoch 112 | Loss: 0.6142 | Train Acc: 76.79% | Test Acc: 77.11% | Best_epoch:111
Epoch 113 | Loss: 0.6142 | Train Acc: 76.79% | Test Acc: 77.11% | Best_epoch:112
Epoch 114 | Loss: 0.6141 | Train Acc: 76.79% | Test Acc: 77.11% | Best_epoch:113
Epoch 115 | Loss: 0.6141 | Train Acc: 76.79% | Test Acc: 77.11% | Best_epoch:114
Epoch 116 | Loss: 0.6141 | Train Acc: 76.79% | Test Acc: 77.11% | Best_epoch:115
Epoch 117 | Loss: 0.6141 | Train Acc: 76.79% | Test Acc: 77.11% | Best_epoch:116
Epoch 118 | Loss: 0.6140 | Train Acc: 76.79% | Test Acc: 77.11% | Best_epoch:117
Epoch 119 | Loss: 0.6140 | Train Acc: 76.79% | Test Acc: 77.11% | Best_epoch:118
Epoch 120 | Loss: 0.6140 | Train Acc: 76.80% | Test Acc: 77.11% | Best_epoch:119
Epoch 121 | Loss: 0.6140 | Train Acc: 76.80% | Test Acc: 77.12% | Best_epoch:120
Epoch 122 | Loss: 0.6139 | Train Acc: 76.80% | Test Acc: 77.12% | Best_epoch:121
Epoch 123 | Loss: 0.6139 | Train Acc: 76.80% | Test Acc: 77.12% | Best_epoch:122
Epoch 124 | Loss: 0.6139 | Train Acc: 76.80% | Test Acc: 77.12% | Best_epoch:123
Epoch 125 | Loss: 0.6139 | Train Acc: 76.80% | Test Acc: 77.12% | Best_epoch:124
Epoch 126 | Loss: 0.6138 | Train Acc: 76.80% | Test Acc: 77.12% | Best_epoch:125
Epoch 127 | Loss: 0.6138 | Train Acc: 76.80% | Test Acc: 77.12% | Best_epoch:126
Epoch 128 | Loss: 0.6138 | Train Acc: 76.80% | Test Acc: 77.12% | Best_epoch:127
Epoch 129 | Loss: 0.6138 | Train Acc: 76.80% | Test Acc: 77.12% | Best_epoch:128
Epoch 130 | Loss: 0.6137 | Train Acc: 76.80% | Test Acc: 77.12% | Best_epoch:129
Epoch 131 | Loss: 0.6137 | Train Acc: 76.80% | Test Acc: 77.12% | Best_epoch:130
Epoch 132 | Loss: 0.6137 | Train Acc: 76.80% | Test Acc: 77.12% | Best_epoch:131
Epoch 133 | Loss: 0.6137 | Train Acc: 76.80% | Test Acc: 77.12% | Best_epoch:132
Epoch 134 | Loss: 0.6136 | Train Acc: 76.81% | Test Acc: 77.12% | Best_epoch:133
Epoch 135 | Loss: 0.6136 | Train Acc: 76.81% | Test Acc: 77.12% | Best_epoch:134
Epoch 136 | Loss: 0.6136 | Train Acc: 76.81% | Test Acc: 77.12% | Best_epoch:135
Epoch 137 | Loss: 0.6136 | Train Acc: 76.81% | Test Acc: 77.13% | Best_epoch:136
Epoch 138 | Loss: 0.6135 | Train Acc: 76.82% | Test Acc: 77.13% | Best_epoch:137
Epoch 139 | Loss: 0.6135 | Train Acc: 76.82% | Test Acc: 77.13% | Best_epoch:138
```

شکل (۴-۲۷) گیر کردن در نقطه کمینه محلی

## ۴-۴- افزایش تعداد لایه های شبکه

برای آنکه مدل پیچیده تری داشــته باشــیم باید تعداد لایه هارا افزایش دهیم همچنین تعداد نورون ها را هم افزایش دهیم که نیازمند تغییر توابع پایه ای مدل ما است.

```
# train
model = Model9(train_x, train_y, X_test=xtest, y_test=ytest,
          layers_dims=[784, 32, 64,128,256,512,512,256,128, 64, 32, 10],learning_rate = 0.5,parameters=-1,epochs=200,lambda_reg=0)
```

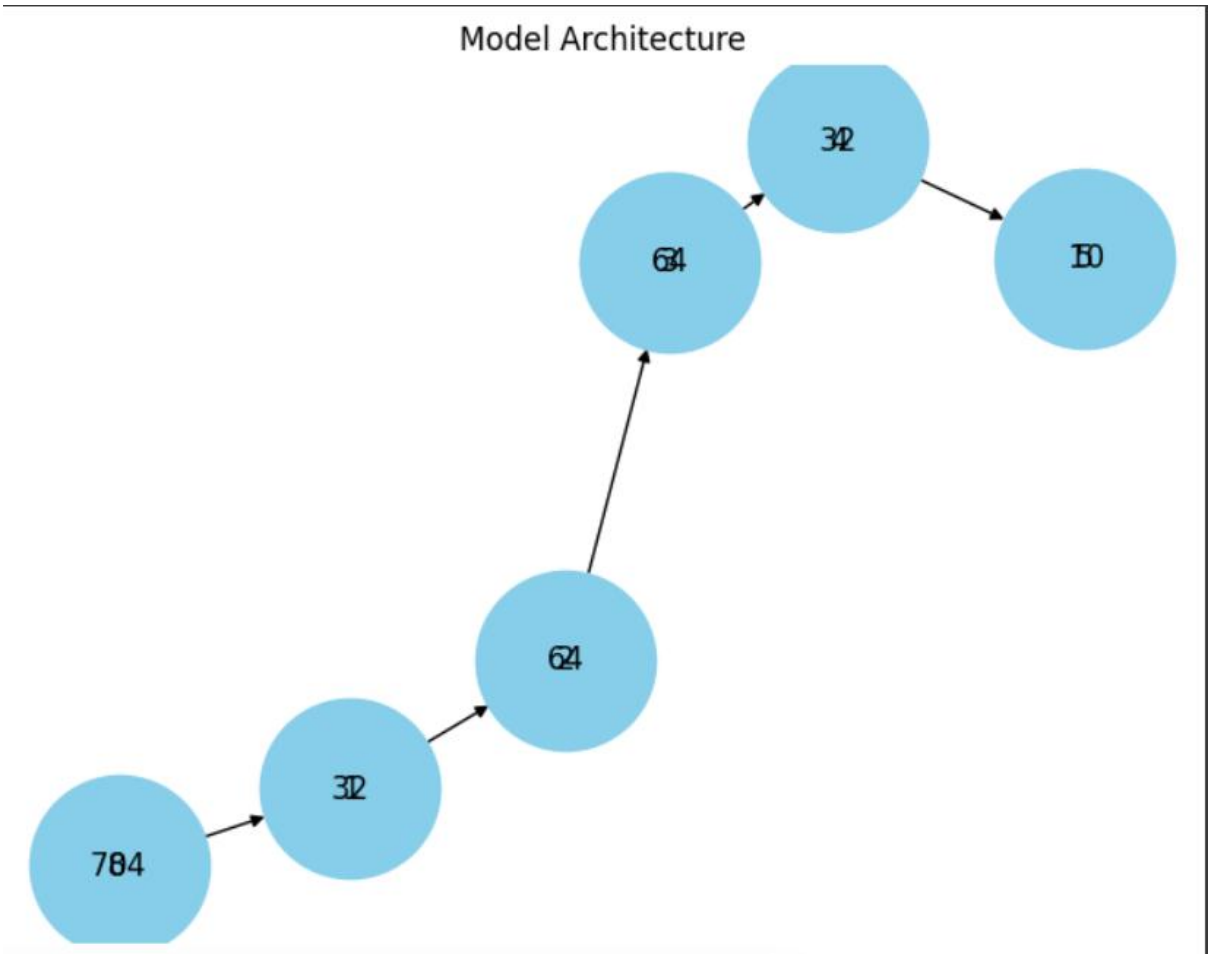
شکل (۴-۲۸) لایه های شبکه

```
Epoch 0 | Loss: 29.9997 | Train Acc: 0.00% | Test Acc: 0.00% | Best_epoch:0
Epoch 1 | Loss: 10.3645 | Train Acc: 0.00% | Test Acc: 0.00% | Best_epoch:0
Epoch 2 | Loss: 24.1743 | Train Acc: 0.00% | Test Acc: 0.00% | Best_epoch:1
Epoch 3 | Loss: nan | Train Acc: 0.00% | Test Acc: 0.00% | Best_epoch:2
Epoch 4 | Loss: nan | Train Acc: 9.87% | Test Acc: 9.80% | Best_epoch:3
Epoch 5 | Loss: nan | Train Acc: 9.87% | Test Acc: 9.80% | Best_epoch:4
Epoch 6 | Loss: nan | Train Acc: 9.87% | Test Acc: 9.80% | Best_epoch:5
Epoch 7 | Loss: nan | Train Acc: 9.87% | Test Acc: 9.80% | Best_epoch:6
Epoch 8 | Loss: nan | Train Acc: 9.87% | Test Acc: 9.80% | Best_epoch:7
Epoch 9 | Loss: nan | Train Acc: 9.87% | Test Acc: 9.80% | Best_epoch:8
Epoch 10 | Loss: nan | Train Acc: 9.87% | Test Acc: 9.80% | Best_epoch:9
Epoch 11 | Loss: nan | Train Acc: 9.87% | Test Acc: 9.80% | Best_epoch:10
Epoch 12 | Loss: nan | Train Acc: 9.87% | Test Acc: 9.80% | Best_epoch:11
Epoch 13 | Loss: nan | Train Acc: 9.87% | Test Acc: 9.80% | Best_epoch:12
Epoch 14 | Loss: nan | Train Acc: 9.87% | Test Acc: 9.80% | Best_epoch:13
Epoch 15 | Loss: nan | Train Acc: 9.87% | Test Acc: 9.80% | Best_epoch:14
Epoch 16 | Loss: nan | Train Acc: 9.87% | Test Acc: 9.80% | Best_epoch:15
Epoch 17 | Loss: nan | Train Acc: 9.87% | Test Acc: 9.80% | Best_epoch:16
Epoch 18 | Loss: nan | Train Acc: 9.87% | Test Acc: 9.80% | Best_epoch:17
Epoch 19 | Loss: nan | Train Acc: 9.87% | Test Acc: 9.80% | Best_epoch:18
Epoch 20 | Loss: nan | Train Acc: 9.87% | Test Acc: 9.80% | Best_epoch:19
Epoch 21 | Loss: nan | Train Acc: 9.87% | Test Acc: 9.80% | Best_epoch:20
Epoch 22 | Loss: nan | Train Acc: 9.87% | Test Acc: 9.80% | Best_epoch:21
Epoch 23 | Loss: nan | Train Acc: 9.87% | Test Acc: 9.80% | Best_epoch:22
Epoch 24 | Loss: nan | Train Acc: 9.87% | Test Acc: 9.80% | Best_epoch:23
Epoch 25 | Loss: nan | Train Acc: 9.87% | Test Acc: 9.80% | Best_epoch:24
Epoch 26 | Loss: nan | Train Acc: 9.87% | Test Acc: 9.80% | Best_epoch:25
Epoch 27 | Loss: nan | Train Acc: 9.87% | Test Acc: 9.80% | Best_epoch:26
```

شکل (۲۹-۴) خروجی مدل با ۱۰ لایه پنهان

مشاهده می شود که این خروجی درست نیست و متاسفانه برخلاف تلاش زیاد نتوانستم مشکل آن را برطرف کنم.

## ۵-۴- چاپ معماری مدل




شکل (۳۰-۴) معماری مدل