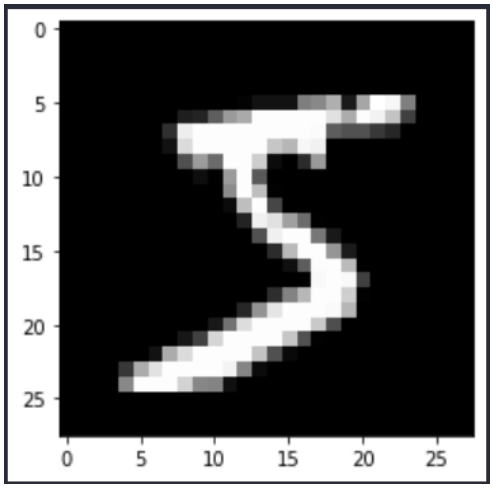
پروژه اول هوش محاسباتی

علی فدائی منش

۹۸۳۱۱۳۰

فاز اول:

در فاز اول داده ها را به صورت List در دو مجموعه train و test لود می کنیم. که داده و برچسب اش به صورت tuple در کنار هم موجود هستند. و برای اطمینان از درستی بارگیری داده ها اولین عکس را نمایش می دهیم به کمک کتابخانه matplotlib



فاز دوم (Feed Forward):

سپس در بخش config از کدم، تابع سیگموید رو تعریف کردم. و همینطور تابع محاسبه کننده cost.

```
# global config

sigmoid = lambda x: 1.0/(1.0 + np.exp(-x))

def calc_cost(train_set, A4,label_set):
    sum = 0
    for image in range(len(train_set[0])):
        X = A4[:,image]-label_set[image]

        sum += np.sum(X ** 2)
    return sum
```

در ابتدای فاز ۲، داده ها را از برچسب شان جدا کردم برای ۱۰۰ داده اول. سپس تعداد نورون های هر لایه، بردارهای B و ماتریس های W را تعریف کردم.

```
# define each layer neurons
Layer1Neurons = 784
Layer2Neurons = 16
Layer3Neurons = 16
Layer4Neurons = 10

# define each layer Biases
B2 = np.zeros((Layer2Neurons,1))
B3 = np.zeros((Layer3Neurons,1))
B4 = np.zeros((Layer4Neurons,1))

# define wights between each layer
W1 = np.random.normal(0,1,(Layer2Neurons, Layer1Neurons))
W2 = np.random.normal(0,1,(Layer3Neurons, Layer2Neurons))
W3 = np.random.normal(0,1,(Layer4Neurons, Layer3Neurons))
```

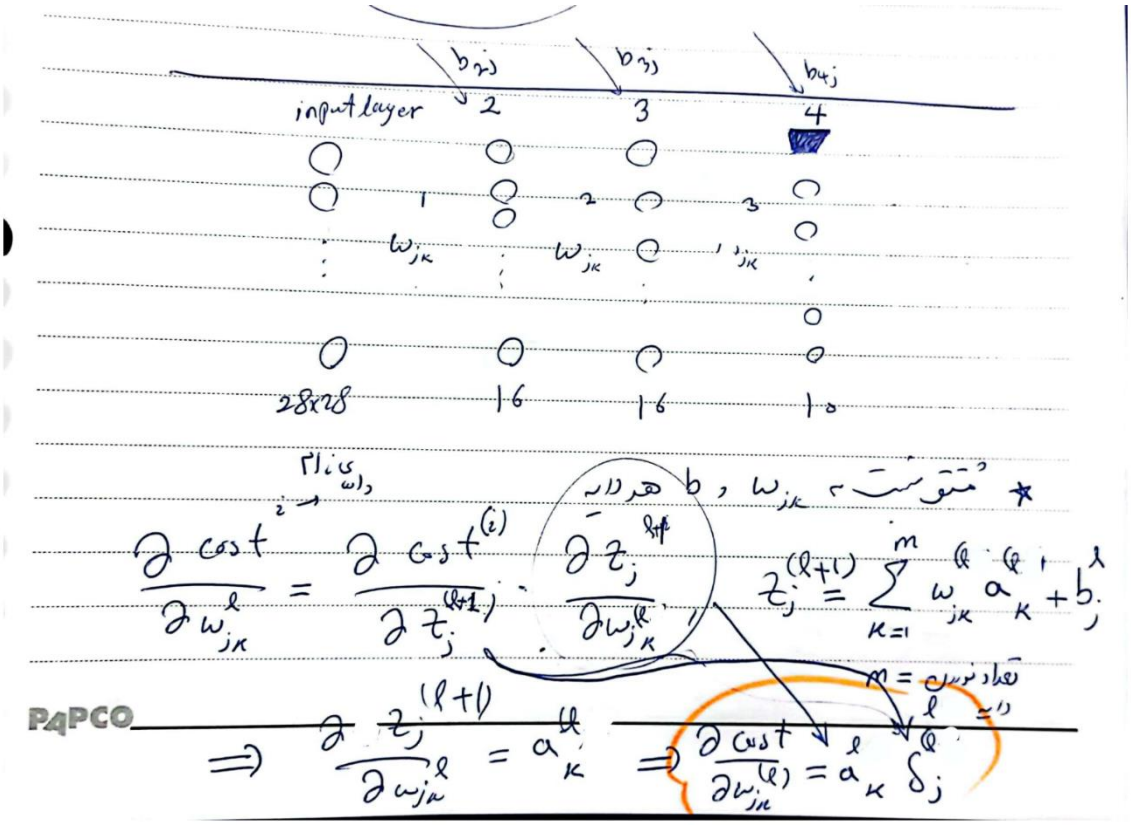
سپس activation خروجی هر لایه را تعریف و با ضرب برداری محاسبه کردم

```
A2 = sigmoid(W1 @ train_first_100 + B2)
A3 = sigmoid(W2 @ A2  + B3)
A4 = sigmoid(W3 @ A3 + B4)

correct_answers = 0
for i in range(100):
    detected_class = A4[:,i].argmax()
    if(detected_class == train_first_100_labels[:,i].argmax()):
        correct_answers += 1
print('accuracy: ' + str(correct_answers) + '%')
```

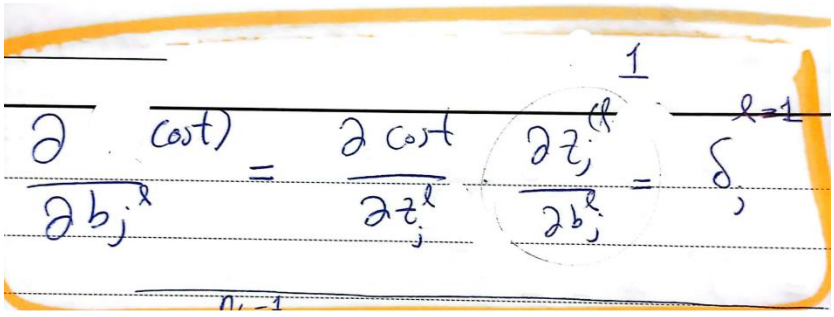
دقت به دست آمده ۱۰ درصد شد. (همانطور که انتظار می رفت)

فاز سوم(Back Propagation)

در این فاز باید مشتق تابع هزینه را نسبت به وزن های هر لایه و بایاس هایشان محاسبه کنیم. محاسبات را روی کاغذ نوشتم. می توانید مشاهده کنید:



عبارت دلتای j را برای لایه های مختلف اینطور تعریف کردم که مشتق هزینه نسبت به w_{ij} برای هر لایه معادل ضرب دلتای همان لایه در خروجی همان لایه سطر k ام اش شود. (بخش نارنجی) و همینطور ثابت کردم مشتق هزینه نسبت به bias هر لایه می شود دلتای لایه ی قبلی.



سپس دلتای لایه های اول تا سوم را محاسبه کردم:

Subject:
Date

$\star\ \delta_j^{(3)} = 2\,(a_j^{(4)} - y_j)\ \ a_j^{(4)}\,(1 - a_j^{(4)})$

در ابتدا مشتق ها را برای یک داده محاسبه می‌کنیم، سپس به روی همه‌ی داده‌ها جمع می‌کنیم.

$$cost = \sum_{i=1}^{10} (a_i - y_i)^2 \qquad a_i = \sigma(z_i^4)$$

$$\Rightarrow \delta_j^{(l)} = \frac{\partial cost}{\partial z_j^{(l+1)}} \xrightarrow{l=3} \frac{\partial cost}{\partial z_j^{(4)}} = 2 \sum_{i=1}^{10} (a_i - y_i) \times \sigma(a_i)(1-\sigma(a_i)) \times \frac{\partial z_i^{(4)}}{\partial z_j^{(4)}}$$

$\star$ $\frac{\partial z_i^{(4)}}{\partial z_j^{(4)}}$ فقط در صورتی غیر صفر است که $i=j$ باشد

حال $\frac{\partial cost}{\partial b_j^4}$ ، $\frac{\partial cost}{\partial w_{jk}^4}$ را داریم طبق تقسیم.

$$\delta_j^{(3)} = 2 \sum_{i=1}^{10} (a_i^{(4)} - y_i)\ a_i^{(4)}(1 - a_i^{(4)}) \times \frac{\partial z_i^{(4)}}{\partial z_j^{(3)}}$$

$$z_i^{(4)} = \sum_{k=1}^{16} w_{ik}^{(3)} a_k^{(3)}, \qquad a_k^3 = \sigma(z_k^{(3)})$$

$\Rightarrow \frac{\partial z_i^4}{\partial z_j^3}$ تقاطعی $\Rightarrow \frac{\partial z_i^{(4)}}{\partial z_j^{(3)}} = w_{ij}^{(3)}\ a_j^{(3)}(1-a_j^{(3)})$

$j=k$ بستگی دارد که

$$\Rightarrow \delta_j^{(2)} = 2 \sum_{i=1}^{10} (a_i - y_i)\ a_i^{(4)}(1-a_i^{(4)}) \times w_{ij}^{(3)}\ a_j^{(3)}(1-a_j^{(3)})$$

$$\delta_j^{(1)} = \frac{\partial cost}{\partial z_j^{(2)}} = 2 \sum_{i=1}^{10} (a_i - y_i)\ a_i^{(4)}(1-a_i^{(4)}) \times \frac{\partial z_i^{(4)}}{\partial z_j^{(2)}}$$

Subject:
Date

$$= 2\sum_{i=1}^{10} (a_i - y_i)\, a_i^{(4)}(1 - a_i^{(4)})\, \partial\Big(\sum_{k=1}^{16} w_{ik}^{(3)} a_k^{(3)}\Big) / \partial z_j^{(2)}$$

$$a_k^{(3)} = \sigma\Big(\sum_{l=1}^{16} w_{kl}^{(2)} a_l^{(2)}\Big)$$

$$= \sum_{i=1}^{10} 2(a_i - y_i)\, a_i^{(4)}(1 - a_i^{(4)}) \sum_{k=1}^{16} w_{ik}^{(3)} a_k^{(3)}(1 - a_k^{(3)}) \times \frac{\partial\big(\sum_{l=1}^{16} w_{kl}^{(2)} a_l^{(2)}\big)}{\partial z_j^{(2)}},$$

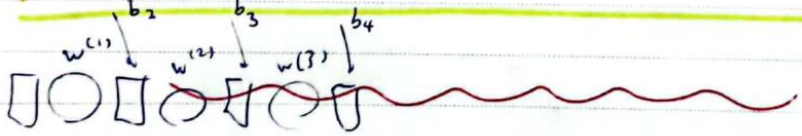
(این تاری نیست)

$$a_l^{(2)} = \sigma\Big(\sum_{p=1}^{28\times 28} w_{lp}^{(1)} a_p^{(1)}\Big)$$

$$\swarrow j = l$$

$$= w_{kj}^{(2)} a_j^{(2)}(1 - a_j^{(2)})$$

$$\Rightarrow \delta_j^{(1)} = \sum_{i=1}^{10}\sum_{k=1}^{16} 2(a_i^{(4)} - y_i)\, a_i^{(4)}(1 - a_i^{(4)})\, w_{ik}^{(3)} \times a_k^{(3)}(1 - a_k^{(3)}) \times w_{kj}^{(2)} a_j^{(2)}(1 - a_j^{(2)})$$



$$\Rightarrow \frac{\partial \text{Cost}}{\partial w_{jk}^{3}} = a_k^{l}\, \delta_j^{l} = a_k^{3}\, 2(a_j^{(4)} - y_j)\, a_j^{(4)}(1 - a_j^{(4)})$$

$$\frac{\partial \text{Cost}}{\partial b_j^{3}} = 2(a_j^{(4)} - y_j)\, a_j^{(4)}(1 - a_j^{(4)})$$

P4PCO

سپس در قسمت پیاده سازی یک تابع استخراج batch تعریف کردم که شماره اندیس batch و خود دیتا ست و سایز بچ را میگیرد و در خروجی داده ها را از برچسب شان جدا می کند و تحویل می دهد.

```
# Step 3
def extract_batch(dataset, index, batch_size):
    X, y = [], []
    min_ = min(batch_size * (index + 1), len(dataset))
    tmp = dataset[index * batch_size: min_]
    for img, label in tmp:
        X.append(img)
        y.append(label)
    return np.array(X).reshape(len(tmp),784).transpose(), np.array(y),
len(tmp)
```

و البته حالتی که تعداد داده ها به batch_size بخش پذیر نباشد را نیز handle کردم.

بعد از تعریف کردن وزن های رندوم نرمال و بایاس های صفر، تابع محاسبه دلتای هر لایه را تعریف کردم:

```
def delta_1(j,image,A4,label_set, A3, A2):
    term = 0
    for i in range(Layer4Neurons):
        for k in range(Layer3Neurons):
            term += 2 * (A4[:,image][i]  - label_set[image,i]) * A4[:,image][i] * (1-
A4[:,image][i]) * W3[i,k] * A3[:,image][k] * (1-A3[:,image][k]) * W2[k,j] * A2[:,
image][j] * (1-A2[:, image][j])
    return term

def delta_2(j,image,A4,label_set, A3):
    term = 0
    for i in range(Layer4Neurons):
        term += 2 * (A4[:,image][i]  - label_set[image,i]) * A4[:,image][i] * (1-
A4[:,image][i]) * W3[i,j] * A3[:,image][j] * (1 - A3[:,image][j])
    return term

def delta_3(j,image, A4,label_set):
    return 2 * (A4[:,image][j]  - label_set[image,j]) * A4[:,image][j] * (1-
A4[:,image][j])
```

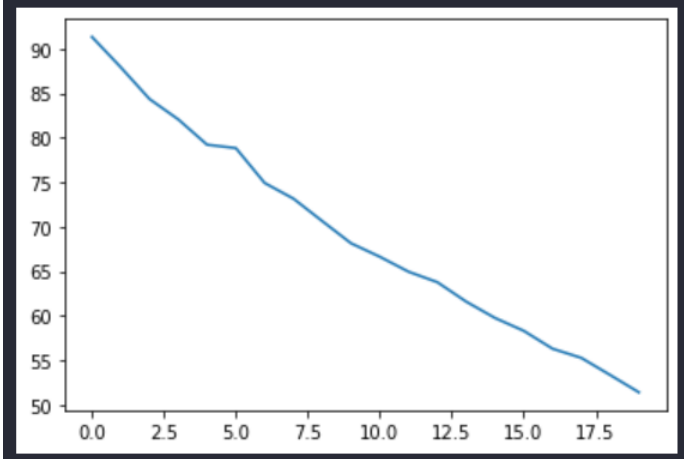
سپس الگوریتم کلی epoch با mini-batch رو پیاده سازی کردم که در کد مشاهده می کنید. در ابتدای هر epoch، هزینه کل رو محاسبه و در لیستی ذخیره می کنم. سپس داده ها را بر می زنم.

```
  train_, label_, l = extract_batch(phase_3_train_set,0,100)
    A2 = sigmoid(W1 @ train_  + B2)
    A3 = sigmoid(W2 @ A2 + B3)
    A4 = sigmoid(W3 @ A3 + B4)
    print('epoch is:', epoch)
    cost = calc_cost(train_,A4,label_)
    epoch_costs.append(cost)
    print("temp cost is: ", cost)
    np.random.shuffle(phase_3_train_set)
```

سپس به محاسبه مشتق ها و دلتا های هر لایه می پردازم. و در هر iteration، W و B ها را تغییر می دهم.
سپس بعد از فرایند train، دقت train را محاسبه می کنم. با استفاده از متود argmax

```
correct_answers = 0
for i in range(100):
    detected_class = A4[:,i].argmax()
    if(detected_class == labels_[i].argmax()):
        correct_answers += 1
print('accuracy: ' + str(correct_answers) + '%')
```

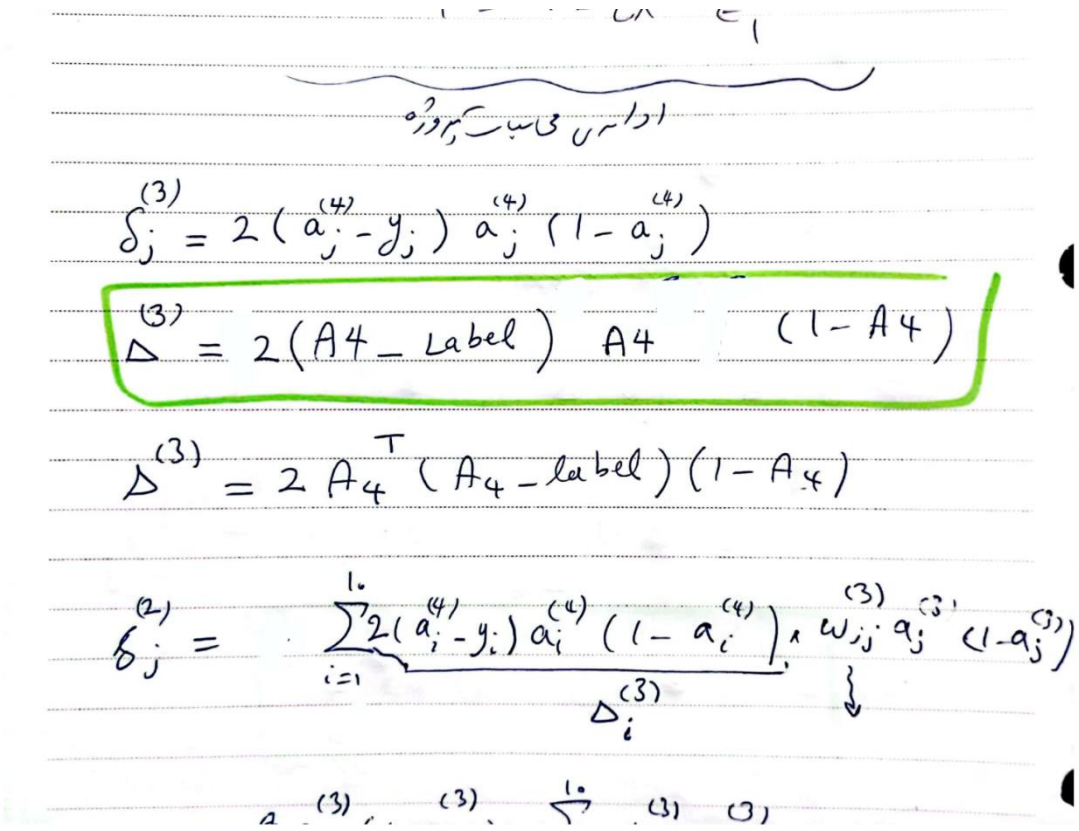
```
accuracy: 69%

[<matplotlib.lines.Line2D at 0x2089689a850>]
```



سپس تابع هزینه را به ازای ایپاک ها رسم می کنیم و در نهایت به دقت **۶۹** درصد رسیدم. طی **۱۱۰** ثانیه.

فاز چهارم (Vectorization):

در این فاز مشتقات رو برداری و ماتریسی می کنیم تا از for اجتناب کنیم و سرعت بالاتر رود.
برای این کار نیاز به محاسبات ریاضی و تبدیل سیگما به ماتریس هست که در زیر مشاهده می کنید:



ubject:
Date

Layer 4 × Layer 3 ، Layer 4 × 1 شکل

Layer 4

$$\sum_{i=1}^{n_4} W^{(3)T}_{ji}\ \Delta^{(3)}_{i1} = \left(W^{(3)T}\Delta^{(3)}\right)_j$$

$$a^{(3)}_j\left(1-a^{(3)}_j\right)$$

$$\downarrow$$

$$\left(A3\,(1-A3)\right)_{3j}$$

$$\Rightarrow \delta^{(2)}_j = \left(W^{(3)T}\Delta^{(3)}\right)_j \times \left(A_3(1-A_3)\right)_j$$

$$\delta^{(2)}_j = \left(A_3(1-A_3)\times\left(W^{(3)T}\Delta^{(3)}\right)\right)_j$$

ضرب تناظر ← نه ماتریسی!

$$\Rightarrow \Delta^{(2)} = A_3(1-A_3)\,W_3^T\,\Delta^{(3)}$$

$$\delta^{(2)}_j = \left[A_2(1-A_2)\right]_j \sum_{k=1}^{n_3} A3_k(1-A3_k)\,W^{(2)}_{kj}$$

$$\sum_{i=1}^{n_4} \Delta^{(3)}_i W^{(3)}_{ik}$$

$$= \left[A_2(1-A_2)\right]_j \sum_{k=1}^{n_3} A^{(3)}_k\left(1-A^{(3)}_k\right)W^{(2)}_{kj} \sum_{i=1}^{n_4} W^{(3)}_{ik}\Delta^{(3)}_i$$

$\Delta^{(2)}_k$ می‌شود

$$= \left[A_2(1-A_2)\right]_j \sum_{k=1}^{n_3} W^{(2)T}_{jk}\,\Delta^{(2)}_{k1}$$

$$= \left[A_2(1-A_2)\right]_j \left[W^{(2)T}\Delta^{(2)}\right]_j$$

PqPCO

$$\Rightarrow \Delta^{(2)}_{j1} = \delta^{(2)}_{j1} = A_2(1-A_2)\,W^{(2)T}\Delta^{(2)}$$

Date

$$\rightarrow \Delta W^{(l)} = \Delta^{(l)} \cdot A^{T(l)}$$

$$L^{+} \times 1 \qquad 1 \times L$$

حالا اگر ستونهای داده را ~ Δ و A را بصورت کنار هم N داده

$$\Delta W^{(l)} = \Delta^{(l)} \; A^{T(l)}$$

$$L^{+} \times N \qquad N \times L$$

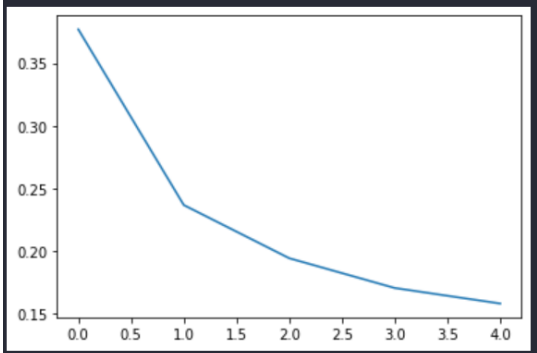
به صورت خودکار $\Delta W^{(l)}$ برای داده‌های تمام روی هم جمع می‌شود

و ΔW جمع ΔW خواهد شد.

$$\Delta B^{(l)} = \Delta^{(l-1)} \cdot \text{sum}(\text{axis} = 1)$$

$$\text{cost} = \left((A4 - \text{label})^{T} (A4 - \text{label})\right) \cdot \text{trace}()$$

```
train accuracy: 0.9000166666666667
test accuracy: 0.8987

[<matplotlib.lines.Line2D at 0x2089692cac0>]
```



همانطور که مشاهده می کنید مشتقات برداری با این روش بسیار فشرده و ساده به دست می آیند.

دقت روی داده های **train**، **۹۰ درصد شد. و روی test ۸۹ درصد و طی زمان ۸.۹ ثانیه**

فاز ششم (دقت داده های تست)

در این فاز دقت مدل را با داده هایی که برای test جدا کرده بودیم بدست می آوریم:

```
# evaluate accuracy on train set
phase_5_test_set = test_set
test_, labels_, size = extract_batch(phase_5_test_set, 0,
len(phase_5_test_set))
A2 = sigmoid(W1 @ test_ + B2)
A3 = sigmoid(W2 @ A2 + B3)
A4 = sigmoid(W3 @ A3  + B4)

#test accuracy
correct_answers = 0
for i in range(len(phase_5_test_set)):
    detected_class = A4[:,i].argmax()
    if(detected_class == labels_[:,i].argmax()):
        correct_answers += 1
print('test accuracy: ' + str(correct_answers/len(phase_5_test_set)))
```

۹۰.۴۵ درصد

با تشکر از توجه شما