تمرین چهارم یادگیری عمیق

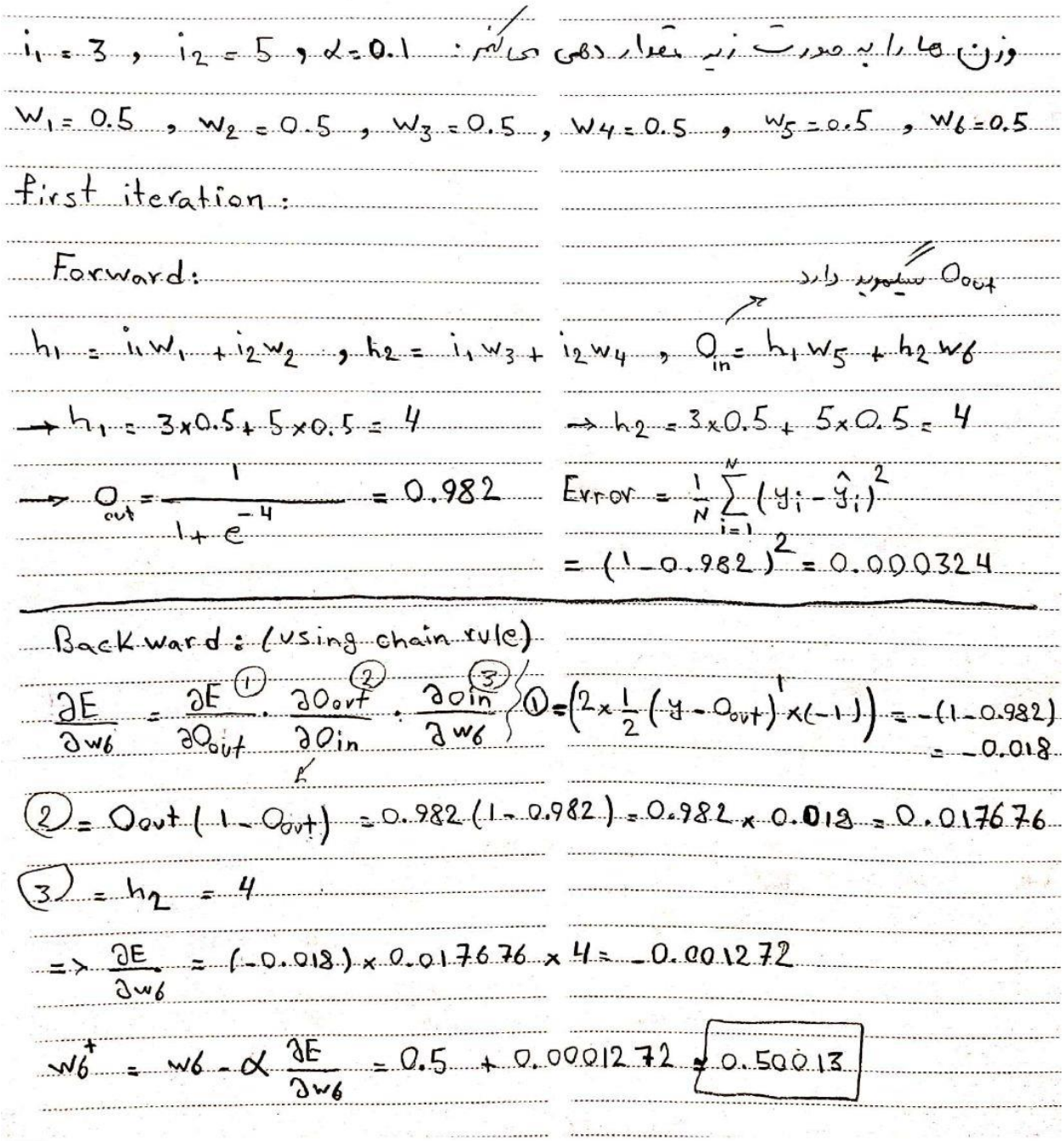
محمدعلی فراهت **۹۷۵۲۱۴۲۳**

سوال ۱)

خیر، نمیتوانیم بگوییم که این الگوریتم همواره بهتر است و باید همیشه از Adam استفاده کنیم. در بعضی مسائل حتی استفاده از Adam عملکرد بدتری برای ما ایجاد میکند. طبق تحقیقاتی که در سال ۲۰۱۷ در دانشگاه برکلی انجام شده، ثابت شده با اینکه Adam عملکرد بهتری در عملیات training دارد و با سرعت بالاتری همگرا میشود و دقت بالاتری را فراهم میکند، اما از لحاظ generalization ، از SGD عملکرد کمتری دارد. با اینکه SGD با سرعت کمتری همگرا میشود و اکثرا دقت پایین‌تری دارد، اما اگر بخواهیم برای دیتای بزرگتری از آن استفاده کنیم که نمیتوانیم همه دیتا را آموزش دهیم، یا دیتاست sparse باشد، این بهینه‌ساز عملکرد بهتری خواهد داشت. Adam سریعتر از بقیه overfit میشود و باید حواسمان به این باشد و با انتخاب یک loss function بهتر از این اتفاق تا جای ممکن جلوگیری کنیم. چون SGD به تنهایی خیلی کند است و عملا قابل استفاده نیست، بهتر است از روش های دیگر در کنار آن استفاده کنیم، مثلا SGD + Momentum میتواند انتخاب بهتری در این مسائل باشد.

سوال۲)

بعد از انجام محاسبات و آپدیت کردن وزن‌ها ، آن‌ها را تحلیل میکنیم. (منبع)



* در آپدیت کردن وزن w6 ، در جمله اول اشتباهاً ½ ضرب شده که بعدا فهمیدم و باید محاسبات را کلا از اول مینوشتم که متاسفانه وقت‌گیر بود و فقط اینجا ذکر میکنم.

$$w_5^+ = w_5 - \alpha \frac{\partial E}{\partial w_5}$$

$$\frac{\partial E}{\partial w_5} = \frac{\partial E}{\partial o_{out}} \cdot \frac{\partial o_{out}}{\partial o_{in}} \cdot \frac{\partial o_{in}}{\partial w_5}$$

① ② ③

قبلاً محاسبه شده‌اند ← ①, ②

③ $= h_1 = 4$

$$\Rightarrow \frac{\partial E}{\partial w_5} = -0.018 \times 0.017676 \times 4 = -0.001272$$

$$\Rightarrow \boxed{w_5^+ \simeq 0.50013}$$

$$w_4^+ = w_4 - \alpha \frac{\partial E}{\partial w_4}$$

$$\frac{\partial E}{\partial w_4} = \frac{\partial E}{\partial h_2} \cdot \frac{\partial h_2}{\partial w_4}$$
① ②

① $= \frac{\partial E}{\partial h_2} = \frac{\partial E}{\partial o_{in}} \cdot \frac{\partial o_{in}}{\partial h_2} = (-0.018)(0.017676)(0.5) = 0.00016$

← قبلاً حساب شده ↳ w_6

② $= i_2 = 5$ $\Rightarrow w_4^+ = 0.5 - 0.1(-0.0008) \simeq \boxed{0.50008}$

$$w_3^+ = w_3 - \alpha \frac{\partial E}{\partial w_3}$$

$$\frac{\partial E}{\partial w_3} = \frac{\partial E}{\partial h_2} \cdot \frac{\partial h_2}{\partial w_3}$$ ← محاسبات قبل

$$\frac{\partial E}{\partial w_3} = -0.00016 \times 3 = -0.00048 \Rightarrow \boxed{w_3^+ \simeq 0.500048}$$

$$w_2^+ = w_2 - \alpha \frac{\partial E}{\partial w_2}$$

$$\frac{\partial E}{\partial w_2} = \frac{\partial E}{\partial h_1} \cdot \frac{\partial h_1}{\partial w_2}$$

$$\frac{\partial E}{\partial w_2} = \frac{\partial E}{\partial o_{in}} \cdot \frac{\partial o_{in}}{\partial h_1} \cdot \frac{\partial h_1}{\partial w_2} = -0.0003182 \times 0.5 \times 5 = -0.0008$$

← قبلاً ↳ w_5 ↳ i_2

$$\Rightarrow \boxed{w_2^+ \simeq 0.50008}$$

$$w_1^+ = w_1 - \alpha \frac{\partial E}{\partial w_1}$$

$$\frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial h_1} \cdot \frac{\partial h_1}{\partial w_1}$$

$$\frac{\partial E}{\partial w_1} = -0.00048 \qquad \boxed{w_1^+ \simeq 0.500048}$$

Second iteration

$w_1 = w_3 = 0.500048$, $w_2 = w_4 = 0.50008$

$w_5 = w_6 = 0.50013$

Forwad :

$h_1 = 4.000544$, $h_2 = 4.000544$

$O_{in} \simeq 4.0016$, $O_{out} \simeq 0.98204$

$E = (1 - 0.98204)^2 = 0.000322$ ← از قبل محاسبه شده است

Backward :

* چون محاسبات یکبار پیوسته شده و این بار فقط اعداد تغییر می‌کنند فقط حواب‌های آخر را می‌نویسم.

$w_6^+ = 0.50013 - 0.1(-0.01796 \times 0.01764 \times 4.000544) \simeq 0.50026$

w_5^+ ← $h_1 = h_2$ چون ← $w_5^+ = w_6^+ \simeq 0.50026$

$w_4^+ = 0.50008 - 0.1(-0.1796 \times 0.01764 \times 0.50013 \times 5) \simeq 0.50016$

$w_3^+ = 0.500048 - 0.1(-0.1796 \times 0.01764 \times 0.50013 \times 3) \simeq 0.500128$

w_2^+ ← $h_1 = h_2$ چون ← $w_2^+ = w_4^+ \simeq 0.50016$

w_1^+ ← // ← $w_1^+ = w_3^+ \simeq 0.500128$

می‌بینیم که با پیدا کردن وزن‌های جدید، loss کاهش داشت ولی برای آموزش کامل این شبکه ما به تعداد iteration بیشتری نیاز داریم و با ۲ مرحله انجام این کار وزن‌ها قابلیت تغییر زیادی در آینده خواهند داشت و همگرا نشده‌اند. اما چون وزن های انتخاب شده من به صورت شانسی جواب را درست میدادند (خروجی نزدیک ۱ بود) ، دقت ما ۱۰۰٪ بود! در کل این یعنی ما برای آموزش یک شبکه نیاز به تعداد خیلی خیلی بیشتری iteration داریم که این محاسبات را به شدت زیاد میکند و برای همین آموزش شبکه‌های عصبی زمان بر است.

سوال ۳)

در این سوال یک شبکه CNN را روی دیتاست CIFAR10 آموزش می‌دهیم.
هدف این سوال کار کردن با PyTorch است.
در این سوال از دو منبع اصلی استفاده شده.(۱ و ۲)
در زیر به ترتیب مراحل انجام کار و عملکرد هر قسمت از کد را می‌بینیم:
- اول تمامی کتابخانه‌های مورد نیاز و فانکشن‌های استفاده شده را import می‌کنیم.

```
import torch
import numpy as np
from torchvision import datasets
import torchvision.transforms as transforms
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
```

- سپس برای افزایش سرعت محاسبات ، از CUDA استفاده می‌کنیم (در صورت دسترسی به GPU)

```
# check if CUDA is available
train_on_gpu = torch.cuda.is_available()

if not train_on_gpu:
    print('CUDA is not available.  Training on CPU ...')
else:
    print('CUDA is available!  Training on GPU ...')
```

- سپس با استفاده از Data augmentation ، ورودی **آموزش** را به بهترین شکل در می‌آوریم، یکی از مزایای این کار این است که معمولا باعث افزایش generalization در مدل می‌شود.

```
transform_train = transforms.Compose([transforms.Resize((32,32)),  #resises the image so it can be perfect for our model.
                                      transforms.RandomHorizontalFlip(), # FLips the image w.r.t horizontal axis
                                      transforms.RandomRotation(10),     #Rotates the image to a specified angel
                                      transforms.RandomAffine(0, shear=10, scale=(0.8,1.2)), #Performs actions like zooms, change shear angles
                                      transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2), # Set the color params
                                      transforms.ToTensor(), # comvert the image to tensor so that it can work with torch
                                      transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)) #Normalize all the images
                                ])
```

- سپس برای داده تست هم ورودی را مانند صورت سوال درست می‌کنیم، ولی تغییرات زیادی مثل داده آموزش به آن نمی‌دهیم. بقیه جزئیات loading را مثل داک سوال انجام می‌دهیم.
- حالا به سراغ پیاده سازی شبکه CNN مورد نظر می‌رویم و لایه‌های آن را مانند شکل‌های خواسته شده و با مقادیر دلخواه (چندین بار تست کردم و این مقادیر بهترین نتیجه را می‌داد، خیلی از جزئیات اینکه چرا اینطوری هستن رو نمی‌دونم) پر کردم و در forwarding ، خروجی لایه convolutional را به ورودی لایه fully connected دادم.

```
self.conv_layer = nn.Sequential(
    nn.Conv2d(3, 6, 3, padding = 1),
    nn.BatchNorm2d(6),
    nn.ReLU(),
    nn.Conv2d(6, 10, 3, padding = 1),
    nn.ReLU(),
    nn.MaxPool2d(2),#

    nn.Conv2d(10, 20, 3, padding = 1),
    nn.BatchNorm2d(20),
    nn.ReLU(),
    nn.Conv2d(20, 40, 3, padding = 2),
    nn.ReLU(),
    nn.MaxPool2d(2),
    nn.Dropout(0.1),#

    nn.Conv2d(40, 80, 3, padding = 2),
    nn.BatchNorm2d(80),
    nn.ReLU(),
    nn.Conv2d(80, 160, 3, padding = 2),
    nn.ReLU(),
    nn.MaxPool2d(2))
```

```
self.fc_layer = nn.Sequential(
    nn.Dropout(0.1),
    nn.Flatten(),
    nn.Linear(160*6*6, 1024),
    nn.ReLU(),
    nn.Linear(1024, 128),
    nn.ReLU(),
    nn.Dropout(0.1),
    nn.Linear(128, 10))
```

```
def forward(self, x):
  x = self.conv_layer(x)
  x = self.fc_layer(x)
  return x
```

- سپس مدل را ساختم و CUDA را برای آن فعال کردم.
- حالا Loss function و Optimizer را انتخاب میکنم.

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr = 0.001)
```

- حالا نوبت آموزش شبکه است، با epoch = 2 ، این کار را انجام میدهیم(چون شبکه بزرگ است، زمان زیادی می‌گیرد) . برای اینکار روی dataloader با یک for حرکت می‌کنیم و هربار input ها را به شبکه می‌دهیم و بعد از محاسبه loss ، با optimizer مدل را آپدیت می‌کنیم. در هر epoch ، من مقدار loss را هم در خروجی چاپ کردم که بتوانیم آن‌ها را باهم مقایسه کنیم. (کد در صفحه بعد)

```
for epoch in range(1, len(n_epochs)+1):

    train_loss = 0.0
    model.train()

    for data, target in trainloader:
        if train_on_gpu:
            data, target = data.cuda(), target.cuda()

        optimizer.zero_grad()
        output = model(data)
        loss = criterion(output, target)

        loss.backward()
        optimizer.step()

        train_loss += loss.item()*data.size(0)

    # print training/validation statistics
    print('Epoch: {} \tTraining Loss: {:.6f} '.format(
        epoch, train_loss))
```

```
Epoch: 1        Training Loss: 97168.819641
Epoch: 2        Training Loss: 84011.326695
```

- در آخر نوبت به تست می‌رسد. در این قسمت مقادیر prediction و تعداد کل تست‌ها را برای هرکلاس جداگانه در دیکشنری ذخیره کردم تا بتوانم دقت آن‌ها را جدا جدا نشان دهم.
بعد از اینکه ورودی تست را به مدل می‌دهیم ، جواب پیش‌بینی شده آن را با جواب اصلی تست مقایسه می‌کنیم و جواب‌های درست را ذخیره می‌کنیم. در آخر هم دقت کل و دقت هر کلاس را در خروجی نشان می‌دهیم.

```
for data, target in testloader:
    if train_on_gpu:
        data, target = data.cuda(), target.cuda()

    output = model(data)
    pred = output.argmax(1)

    for out, prediction in zip(target, pred):
        class_total[classes[out]] += 1
        if (out == prediction):
            total_correct += 1
            class_correct[classes[out]] += 1
```

```
Total accuracy of tests: 44.66%
Accuracy of each class:

Accuracy of plane: 52.900000000000006%
Accuracy of car: 72.0%
Accuracy of bird: 44.6%
Accuracy of cat: 19.3%
Accuracy of deer: 24.6%
Accuracy of dog: 31.1%
Accuracy of frog: 51.1%
Accuracy of horse: 56.2%
Accuracy of ship: 63.800000000000004%
Accuracy of truck: 31.0%
```