به نام خدا

گزارش تمرین شماره 5

هوش مصنوعی

علی عدالت

استاد دکتر صادقی

بهار 98

پرسش اول)

در ابتدا وزن ها را با صفر مقدار دهی کردیم و در ادامه به وزن دهی رندوم پرداختیم. با مقدار دهی وزن ها با صفر دقت شبکه به مقدار اندک 10 درصد رسید و با وزن دهی رندم به 24 درصد رسید. در وزن دهی به 0 دلیل این موضوع مشکل Vanishing gradient است. در این جا مقدار گرادیان به صفر میل می کند به همین دلیل هیچ تغییری در وزن ها ایجاد نمی شود و عملیات train متوقف می شود. دلیل صفر شدن گرادیان این است که برای محاسبه تغییرات وزن لایه ها از backpropagation استفاده می شود و تغییر وزن جلوترین لایه یک عدد کوچک بین 0 و 1 است. برای محاسبه یک وزن یک لایه لازم است این عدد کوچک چند بار در خود ضرب شود و در انتها تغییرات وزن یک عدد بسیار کوچک می شود. زمانی که وزن ها به صورت رندوم مقدار دهی می شوند یا مقدار وزن بسیار بالا است که در این صورت در تابع relu شیب تغییرات یک است که باعث کندی عملیات می شود یا اینکه به 0 بسیار نزدیک است که مانند وزن دهی به 0 است. همچنین ممکن است تغییر وزن بی نهایت شود که در این صورت اطراف مینیما می چرخیم و train جلو نمی رود و دقت پایین می آید. پس دقت در این حالت نیز کمتر خواهد بود ولی پیشرفت از حالت قبل بیشتر است. در این بخش در یافتیم که وزن دهی اولیه در دقت با توجه به راه حل SGD بسیار ماثر است. به همین دلیل کتابخانه از وزن دهی خاصی برای این کار با توجه به نوع لایه استفاده می کند. کد این بخش و نتایج در زیر آمده است.

```python
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv_layer = nn.Sequential(
            nn.Conv2d(in_channels=3, out_channels=32, kernel_size=3, padding=1),
            nn.BatchNorm2d(32),
            nn.ReLU(inplace=True),
            nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),
        )
        self.fc_layer = nn.Sequential(
            nn.Dropout(p=0.1),
            nn.Linear(8*8*128, 1024),
            nn.ReLU(inplace=True),
            nn.Linear(1024, 512),
            nn.ReLU(inplace=True),
            nn.Dropout(p=0.1),
            nn.Linear(512, 10)
        )

    def forward(self, x):
        x = self.conv_layer(x)
        x = x.view(x.size(0), -1)
        x = self.fc_layer(x)
        return x
```

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
print('without init')
trainNet()
findAccuracy()
net.apply(init_weights_zero)
print('init_weights_zero')
trainNet()
findAccuracy()
```

```
without init
Finished Training in  69175  ms
Accuracy of the network on the  10000  test images: 74 %
init_weights_zero
Finished Training in  69095  ms
Accuracy of the network on the  10000  test images: 10 %
```

```
[36] net = Net()
     def init_weights_random(m):
       if type(m) == nn.Conv2d:
         m.weight.data = torch.rand(m.weight.size(0),m.weight.size(1),m.weight.size(2),m.weight.size(3)) +\
         (torch.zeros(m.weight.size(0),m.weight.size(1),m.weight.size(2),m.weight.size(3)) * m.weight)
         m.bias.data = torch.rand(m.bias.size(0)) + (torch.zeros(m.bias.size(0)) * m.bias)
         return
       if type(m) == nn.Linear:
         m.weight.data = torch.rand(m.weight.size(0),m.weight.size(1)) +\
         (torch.zeros(m.weight.size(0),m.weight.size(1)) * m.weight)
         m.bias.data = torch.rand(m.bias.size(0)) + (torch.zeros(m.bias.size(0)) * m.bias)
         return

     net.apply(init_weights_random)
     net.to(device)
     criterion = nn.CrossEntropyLoss()
     optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
     print('init_weights_random')
     trainNet()
     findAccuracy()
```

```
init_weights_random
Finished Training in  51279  ms
Accuracy of the network on the  10000  test images: 23 %
```

پرسش دوم)

تابع relu در عمق کم در برابر مشکلات گفته شده در بخش یک مقاوم تر است به همین دلیل در بخش رندم عملکرد بهتری دارد. ولی در وزن 0 دوباره گرادیان 0 می شود و مانند قبل عمل می کند. کد و نتایج این بخش در زیر آمده است.

```python
class Net2(nn.Module):
    def __init__(self):
        super(Net2, self).__init__()
        self.conv_layer = nn.Sequential(

            # Conv Layer block 1
            nn.Conv2d(in_channels=3, out_channels=256, kernel_size=3, padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),
        )


        self.fc_layer = nn.Sequential(
            nn.Dropout(p=0.1),
            nn.Linear(16*16*256, 10)
        )

    def forward(self, x):
        x = self.conv_layer(x)

        # flatten
        x = x.view(x.size(0), -1)

        # fc layer
        x = self.fc_layer(x)

        return x
```

```
without init
Finished Training in  65802  ms
Accuracy of the network on the  10000  test images: 63 %
init_weights_zero
Finished Training in  64433  ms
Accuracy of the network on the  10000  test images: 10 %
init_weights_random
Finished Training in  64881  ms
Accuracy of the network on the  10000  test images: 25 %
```

بخش سوم)

اعداد رنگ ها بین 0 و 255 هستند و برای جلوگیری از exploding/vanishing gradient problems بهتر است که این اعداد در بازه 0 و 1 قرار گیرند به همین دلیل از نرمالایز کردن استفاده می شود. که باعث بهتر شدن دقت می شود. هم چنین از اتمام عملیات زود تر از موعد جلوگیری می کند و کانورج شدن را سرعت می بخشد. در SGD نحوه به روز شدن وزن ها به صورت زیر است.

$$w_{t+1} = w_t - \gamma \nabla_w \ell(f_w(x), y)$$

با توجه به این موضوع در میابیم که نرمال کردن y که لیبل می باشد در تغییر وزن زمانی تاثیر می گذارد که در لیبل ها مقدار بسیار زیادی وجود داشته باشد که باعث سرریز شود. پس لازم نیست لیبل در نرمال کردن در نظر گرفته شود. کد این بخش و نتایج در زیر آمده است.

```
trainloader, testloader = fetchData(False, 32)
print('whitout normalization')
net = Net()
net.to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
trainNet()
findAccuracy()
trainloader, testloader = fetchData(True, 32)
print('normalize')
net = Net()
net.to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
trainNet()
findAccuracy()
```

```
Files already downloaded and verified
Files already downloaded and verified
whitout normalization
Finished Training in  50779  ms
Accuracy of the network on the  10000  test images: 73 %
Files already downloaded and verified
Files already downloaded and verified
normalize
Finished Training in  61236  ms
Accuracy of the network on the  10000  test images: 73 %
```

بخش چهارم)

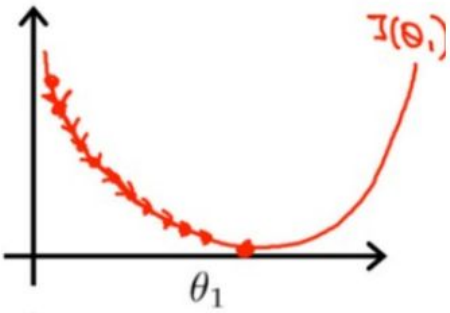
ریت یادگیری پارامتری است که نشان می دهد وزن ها چقدر با توجه به گرادیان loss تغییر کنند. مقدار کم این پارامتر نشان دهنده سرعت پایین حرکت در جهت شیب است. فرمول تغییر وزن ها به صورت زیر است.

```
new_weight = existing_weight — learning_rate * gradient
```

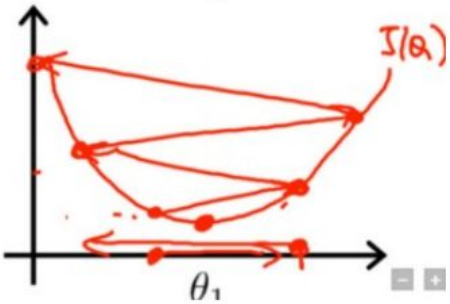
که با توجه به این فرمول حالات زیر برای نحوه حرکت در هنگام تمرین ممکن است. با توجه به این حالات به این نکته می رسیم که که اگر این ریت خیلی زیاد باشد یا اطراف یک مینیما حرکت می کنیم یا ممکن است از مسیر دور شویم. اگر مقدار آن کم باشد سرعت حرکت ما کم خواهد بود و در یک epoch کم دقت ما از حالتی که یک ریت کمی بیشتر داشته باشیم کمتر خواهد بود. در این جا در ابتدا ریت ما زیاد است و دقت کم است چون train به خوبی انجام نمی شود و با کاهش ریت سرعت حرکت در جهت تغیرات کمتر می شود و train بهتر انجام می شود و دقت بالا تر است. کد و نتایج در ادامه آمده است.

$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$$

If α is too small, gradient descent can be slow.



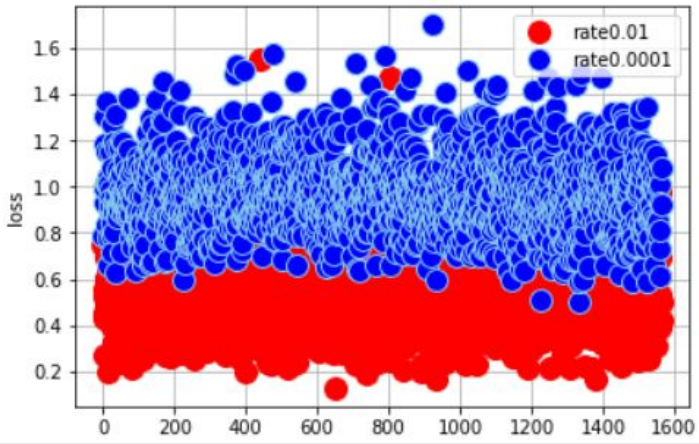
If α is too large, gradient descent can overshoot the minimum. It may fail to converge, or even diverge.




```
[44] import matplotlib.pyplot as plt
     import pandas as pd

     trainloader, testloader = fetchData(True, 32)
     print('lr : ', 0.01)
     net = Net()
     net.to(device)
     criterion = nn.CrossEntropyLoss()
     optimizer = optim.SGD(net.parameters(), lr=0.01, momentum=0.9)
     lossList = trainNet()
     findAccuracy()
     trainloader, testloader = fetchData(True, 32)
     print('lr : ', 0.0001)
     net = Net()
     net.to(device)
     criterion = nn.CrossEntropyLoss()
     optimizer = optim.SGD(net.parameters(), lr=0.0001, momentum=0.9)
     lossList2 = trainNet()
     findAccuracy()
     xlist = lossList[(len(lossList)-(int(len(lossList)/5))):]
     df=pd.DataFrame({'x': range(1,len(xlist)+1),\
     'rate0.01': lossList[(len(lossList)-(int(len(lossList)/5))):],\
     'rate0.0001': lossList2[(len(lossList2)-(int(len(lossList2)/5))):]})
     plt.plot( 'x', 'rate0.01', data=df, marker='o', markerfacecolor='red', markersize=12, color='red', linewidth=0)
     plt.plot( 'x', 'rate0.0001', data=df, marker='o', markerfacecolor='blue', markersize=12, color='skyblue', linewidth=0)
     plt.xlabel('batch')
     plt.ylabel('loss')
     plt.grid()
     plt.legend()
```

```
Files already downloaded and verified
Files already downloaded and verified
lr :  0.01
Finished Training in  61208  ms
Accuracy of the network on the  10000  test images: 74 %
Files already downloaded and verified
Files already downloaded and verified
lr :  0.0001
Finished Training in  61459  ms
Accuracy of the network on the  10000  test images: 65 %
<matplotlib.legend.Legend at 0x7f6b0044cef0>
```



بخش پنجم)

افزایش batch size باعث کند شدن یادگیری می شود ولی واریانس یادگیری را کاهش می دهد و در نهایت ما به مدل های استیبل تری می رسیم. دیگر عیب batch size بزرگ استفاده بیشتر مموری برای هر به روز رسانی است.

با افزایش batch size باید learning rate را افزایش داد. با افزایش batch size دقت محاسبه گرادیان افزایش می یابد و تعداد گام های کلی بررسی داده ها کاهش می یابد پس باید تاثیر گرادیان و تغییرات وزن را افزایش داد و باید learning rate را افزایش داد. کد و نتایج در ادامه آمده است.

```
[ ]  trainloader, testloader = fetchData(True, 32)
     print('batch size : ', 32, 'lr: ', 0.001)
     net = Net()
     net.to(device)
     criterion = nn.CrossEntropyLoss()
     optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
     trainNet()
     findAccuracy()
     print('batch size : ', 32, 'lr: ', 0.01)
     net = Net()
     net.to(device)
     criterion = nn.CrossEntropyLoss()
     optimizer = optim.SGD(net.parameters(), lr=0.01, momentum=0.9)
     trainNet()
     findAccuracy()
     print('batch size : ', 32, 'lr: ', 0.0001)
     net = Net()
     net.to(device)
     criterion = nn.CrossEntropyLoss()
     optimizer = optim.SGD(net.parameters(), lr=0.0001, momentum=0.9)
     trainNet()
     findAccuracy()

     trainloader, testloader = fetchData(True, 64)
     print('batch size : ', 64, 'lr: ', 0.001)
     net = Net()
     net.to(device)
     criterion = nn.CrossEntropyLoss()
     optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
     trainNet()
     findAccuracy()
     print('batch size : ', 64, 'lr: ', 0.01)
     net = Net()
     net.to(device)
```

```
     criterion = nn.CrossEntropyLoss()
     optimizer = optim.SGD(net.parameters(), lr=0.01, momentum=0.9)
     trainNet()
     findAccuracy()
     print('batch size : ', 64, 'lr: ', 0.0001)
     net = Net()
     net.to(device)
     criterion = nn.CrossEntropyLoss()
     optimizer = optim.SGD(net.parameters(), lr=0.0001, momentum=0.9)
     trainNet()
     findAccuracy()

     trainloader, testloader = fetchData(True, 256)
     print('batch size : ', 256, 'lr: ', 0.001)
     net = Net()
     net.to(device)
     criterion = nn.CrossEntropyLoss()
     optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
     trainNet()
     findAccuracy()
     print('batch size : ', 256, 'lr: ', 0.01)
     net = Net()
     net.to(device)
     criterion = nn.CrossEntropyLoss()
     optimizer = optim.SGD(net.parameters(), lr=0.01, momentum=0.9)
     trainNet()
     findAccuracy()
     print('batch size : ', 256, 'lr: ', 0.0001)
     net = Net()
     net.to(device)
     criterion = nn.CrossEntropyLoss()
     optimizer = optim.SGD(net.parameters(), lr=0.0001, momentum=0.9)
     trainNet()
     findAccuracy()
```

```
Files already downloaded and verified
Files already downloaded and verified
batch size :  32 lr:  0.001
Finished Training in  61695  ms
Accuracy of the network on the  10000  test images: 73 %
batch size :  32 lr:  0.01
Finished Training in  61315  ms
Accuracy of the network on the  10000  test images: 75 %
batch size :  32 lr:  0.0001
Finished Training in  61629  ms
Accuracy of the network on the  10000  test images: 65 %
Files already downloaded and verified
Files already downloaded and verified
batch size :  64 lr:  0.001
Finished Training in  71352  ms
Accuracy of the network on the  10000  test images: 70 %
batch size :  64 lr:  0.01
Finished Training in  71190  ms
Accuracy of the network on the  10000  test images: 75 %
batch size :  64 lr:  0.0001
Finished Training in  71407  ms
Accuracy of the network on the  10000  test images: 60 %
Files already downloaded and verified
Files already downloaded and verified
batch size :  256 lr:  0.001
Finished Training in  73793  ms
Accuracy of the network on the  10000  test images: 68 %
batch size :  256 lr:  0.01
Finished Training in  73416  ms
Accuracy of the network on the  10000  test images: 73 %
batch size :  256 lr:  0.0001
Finished Training in  73495  ms
Accuracy of the network on the  10000  test images: 49 %
```

پرسش ششم)

کد و نتایج این بخش در زیر آمده است.

```
trainloader, testloader = fetchData(True, 32)
print('relu')
net = Net()
net.to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.01, momentum=0.9)
trainNet()
findAccuracy()
print('tanh')
net = NetTanh()
net.to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.01, momentum=0.9)
trainNet()
findAccuracy()
print('leaky relu')
net = NetLeakyReLU()
net.to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.01, momentum=0.9)
trainNet()
findAccuracy()
print('softplus')
net = NetSoftplus()
net.to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.01, momentum=0.9)
trainNet()
findAccuracy()
```

```
Files already downloaded and verified
Files already downloaded and verified
relu
Finished Training in  61658  ms
Accuracy of the network on the  10000  test images: 73.69%
tanh
Finished Training in  61866  ms
Accuracy of the network on the  10000  test images: 70.15%
leaky relu
Finished Training in  61439  ms
Accuracy of the network on the  10000  test images: 75.05%
softplus
Finished Training in  61864  ms
Accuracy of the network on the  10000  test images: 65.31%
```

با توجه به نتایج دیده می شود که زمان train برای relu و leaky relu از بقیه کمتر است که دلیل آن حجم محاسبات کمتر این تابع ها است. دقت relu , leaky relu از بقیه بیشتر است که دلیل آن حل مشکل vanishing gradient در این دو تابع است. دقت leaky relu از relu بیشتر است که دلیل آن حل مشکل صفر شدن گرادیان و dying ReLU است که باعث می شود نرون های بیشتری در train دخیل باشند و وزنشان تغییر کند و مدل بهتری تولید شود.

بخش هفتم)

دیده می شود که ممنتم باعث می شود که دقت افزایش یابد و همچنین زمان train را افزایش می دهد. ممنتم تغییر جهت گرادیان را نرم می کند و باعث کند شدن حرکت در جهت گرادیان می شود و گام های حرکت کوچک تر می شود و دقت بالا تر می رود و کانورج بهتر اتفاق می افتد. همچنین این موضوع باعث افزایش زمان یادگیری می شود. کد و نتایج در زیر آمده است.

```
trainloader, testloader = fetchData(True, 32)
print('leaky relu')
net = NetLeakyReLU()
net.to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.01, momentum=0.9)
trainNet()
findAccuracy()
trainloader, testloader = fetchData(True, 32)
print('leaky relu without momentum')
net = NetLeakyReLU()
net.to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.01)
trainNet()
findAccuracy()
```

```
Files already downloaded and verified
Files already downloaded and verified
leaky relu
Finished Training in  61096  ms
Accuracy of the network on the  10000  test images: 75.12%
Files already downloaded and verified
Files already downloaded and verified
leaky relu without momentum
Finished Training in  60797  ms
Accuracy of the network on the  10000  test images: 69.50%
```