Assignment NO.3 Solutions

Deep learning | winter 1400 | Dr.Mohammadi

Teacher Assistant:

Mohammad hossein khojaste

Student name : **Amin Fathi**

Student id : **400722102**

Problem 1

بیان مشکل

آنچه این مقاله قصد پرداخت به آن را دارد مشکلات مربوط به پاسخ دادن به سوالات با استفاده از شبکه های از پیش آموزش دیده و نمودار های دانش است . هر کدام از گراف های دانش و شبکه های از پیش آموزش یافته مشکلات خود را داشتند ، مثلا گراف های دانش دامنه پوشش محدودی در مسایل question answering داشتند و شبکه های از پیش آموزش دیده هم روی سوال هایی که نیاز به استدلال و استنتاج از گزاره ها بود عملکرد خوبی نداشتند . در این مقاله نگارندگان در پی ترکیب و ادغام این دو روش با هم هستند که البته با ۲ چالش هم رو به رو هستند ، چالش اول به دست اوردن اطلاعات مد نظر سوال است که الگورتیم در واقع باید بتواند اطلاعات مربوط به سوال را از گراف دانش بسیار بزرگی تشخیص دهد چالش دوم هم مربوط به درک کلمات و اصطلاحات مبهم در qa و kg بود .

بررسی نتایج

Problem 2

توابع فعال سازی توانایی یادگیری الگو های پیچیده غیر خطی را به شبکه می دهند ، همچنین خروجی توابع فعال سازی نسبت به ورودی آن ها معمولا در بازه محدود تری قرار دارند که این امر سبب آسانی محاسبات (خصوصا در شبکه های چند لایه) می شود . چنانچه از توابع فعال سازی استفاده نشود ، هر چه قدر هم که عمق شبکه را افزایش دهیم باز هم با یک شبکه خطی روبه رو هستیم :

$$W1*X1+b1 = W1(W2(W3 \ldots.(Wn(x)+bn)\ldots)+b1$$

درواقع در صورت عدم استفاده از توابع فعال سازی ، هر لایه از شبکه MLP در واقع یک تبدیل خطی ورودی ها (با استفاده از بایاس و وزن ها) خواهد بود که علی رغم سادگی complexity ، از توان یادگیری پایینی برخوردار خواهد بود و درواقع یک رگرسیون خطی صرفا خواهد بود .

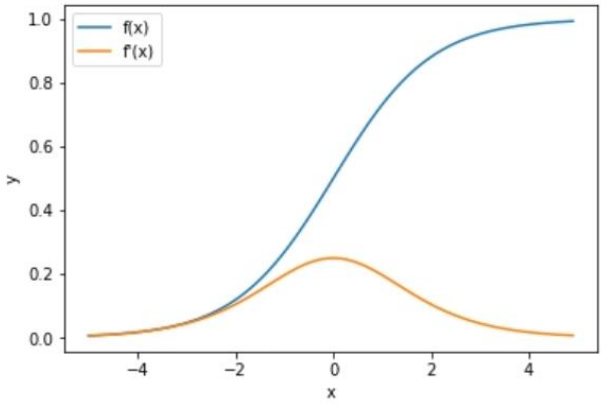
در میان توابع فعال سازی ، Relu از بقیه محبوب تر است ، چون با مشکل ناپدید شدن گرادیان که در سیگموید مواجهیم مواجه نیست و در برابر مشکل انفجار گرادیان هم چنانچه وزن های اولیه درست انتخاب شوند عملکرد خوبی دارد محاسبات و اجرای آن هم ساده تر است و از انجا که برای منفی ها مقدار صفر را بر میگرداند باعث می شود بعضی نورون ها غیر فعال باشند که این در شبکه عمیق بیسار میتواند مفید باشد . مشکل اشباع شدن Tanh و سیگوید را هم ندارد و تنها مشکل بزرگش شیب صفر در مقادیر منفی است که برای اصلاح آن leaky Relu معرفی شده است

ب)

هنگام آموزش یک شبکه عصبی عمیق با یادگیری مبتنی بر گرادیان و backpropagation ، مشتقات جزئی را با عبور از شبکه از لایه نهایی به لایه اولیه پیدا می کنیم که این عمل با استفاده از قاعده مشتق زنجیره ای رخ می دهد . چنانچه مشتقات جزئی که در هر کدام از n لایه به دست می آیند و در یکدیگر ضرب می شوند ، مقدایر بالایی داشته باشنید ، مشتق خروجی شبکه نسبت به ورودی شبکه با شیب نمایی و تند افزایش پیدا خواهد کرد و این برای backpropagation مشکل ایجاد خواهد کرد که به این انفجار گرادیان گویند که با توجه به مقادیر بالای گرادیان ، در هر مرحله از بروزرسانی ، وزن ها و بایاس های لایه ها با تغییرات بزرگی روبه رو خواهند شد که باعث ناپایداری شبکه و عدم توانایی شبکه برای یادگیری می شود ، همچنین در نتیجه انفجار گرادیان ما با وزن های با مقادیر بسیار بزرگی (NaN) رو به رو هستیم که نمیتوان آن ها را به روز رسانی کرد ؛ همچنین اگر مقادیر مشتقات جزئی کوچک باشند در نهایت بنابه ضرب شدن این مقادیر کوچک در قائده زنجیری در همدیگر ، در نهایت با

یک مشتق خروجی نسبت به ورودی بسیار کم (در بدترین حالت در حد 0) در شبکه رو به رو هستیم که به این وضعیت ناپدید شدن گرادیان گویند که در این حالت وزن ها توانایی به روز شدن بر اساس ویژگی های ورودی را از دست خواهند داد که در نتیجه شبکه توانایی یادگیری خود را از دست خواهد داد ، وزن ها هم در بدترین حالت مقادیر معادل 0 خواهند داشت و وزن های لایه های نزدیک به لایه خروجی مقدار تغییرات بیشتری را نسبت به مقادیر وزن های لایه های نزدیک به لایه ورودی خواهند داشت .

تابع فعال سازی سیگموید مشکل vanishing gradient را بیشتر مواجه می شود ، همانطور که در شکل زیر مشاهده میشود ، مقدار مشتق در مقادیر بالا و پایین سیگموید بیسار بسیار پایین و در نهایت معادل صفر هستند که این باعث ناپدید شدن گرادیان در قائده زنجیری می شود .



Sigmoid gradient

برای مشکل انفجار گرادیان هم چنانچه Relu با مقادیر وزن بسیار بزرگ انتخاب شود می تواند منجر به انفجار گرادیان شود.

ج)

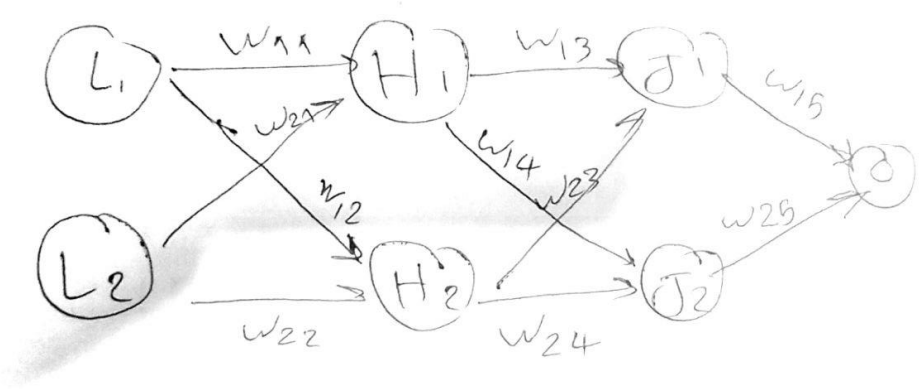
اضافه کردن لایه مخفی بیشتر سبب توانایی یافتن شبکه برای یادگیری الگو های پیچیده تر می شود ، البته که این کار تعداد پارامتر ها و سر بار زمانی زیادی دارد و complexity شبکه زیاد می شود که این اتفاق هم می تواند موجب کارایی بهتر شبکه در مسئله شود و هم میتواند منجر به over fit شدن شود .

افزون بر این به صورت تئوری می توان گفت شبکه عصبی با یک لایه مخفی با تعداد کافی نورون می تواند هر تابعی را به درستی تخمین بزند اما قدرت تعمیم بالایی نخواهد داشت ، مزیت چند لایه در توانایی یادگیری ویژگی ها در سطوح مختلف می باشد به طور مثال لایه اول یک سری ویژگی کلی در می آورد و به همین ترتیب لایه آخر ویژگی های جزئی تر و این باعث بالا رفتن قدرت تعمیم شبکه می شود ، همچنین بالا رفتن تعداد لایه ها باعث بالا رفتن هزینه محساباتی و سخت افزاری می شود .

منابع :

- Activation Functions | Fundamentals Of Deep Learning (analyticsvidhya.com)
- https://towardsdatascience.com/everything-you-need-to-know-about-activation-functions-in-deep-learning-models-84ba9f82c253
- https://medium.com/@snaily16/what-why-and-which-activation-functions-b2bf748c0441
- https://stats.stackexchange.com/questions/222883/why-are-neural-networks-becoming-deeper-but-not-wider
- Does adding more layers always result in more accuracy in convolutional neural networks? - Quora
- The vanishing gradient problem and ReLUs – a TensorFlow investigation – Adventures in Machine Learning
- **https://analyticsindiamag.com/can-relu-cause-exploding-gradients-if-applied-to-solve-vanishing-gradients/**
- **https://www.analyticsvidhya.com/blog/2020/01/fundamentals-deep-learning-activation-functions-when-to-use-them/**
- **صحبت های استاد در کلاس درس**

Problem3



H1 = i1 * w11 + i2 * w21 + b1 = -1 * 0.5 + 0 * -0.25 + 0.25 = -0.25

H2 = i1 * w12 + i2 * w22 + b2 = -1 * 0 + 0 * 1 + 0.25 = 0.25

Relu => H1 = 0 , H2 = 0.25

J1 = H1 * w13 + H2 * w23 + b3 = 0 * -0.1 + 0.25 * 0 + 0.25 = 0.25

J2 = H1 * w14 + H2 * w24 + b4 = 0 * 1 + 0.25 * 1 + 0.25 = 0.5

Relu => J1 = 0.25 , J2 = 0.5

o = J1 * w15 + J2 * w25 + b5 = 0.25*-0.5 + 0.5 * 1 + 0.5 = 0.875

MSE = (1 – 0.875) ^ 2 = 0.01562

Problem 4

بعد از دانلود دادگان و نمایش چند تا از آن ها به سراغ پیاده سازی مدل می رویم ، مدل ما دارای ۲ لایه FC مخفی می باشد که هر دو دارای ۶۰۰ نورون می باشد ، لایه خروجی هم به تعداد کلاس خروجی یعنی ۱۰ نورون دارد ، لایه ورودی نیز دارای ۷۸۴ نورون میباشد که در واقع اندازه تصویر ۲۸ * ۲۸ به صورت flatten می باشد و هر پیکسل به عنوان یک فیچر در این مدل به کار رفته شده است .

```
## Define the model
################ Your code ############
model = nn.Sequential(
            nn.Linear(784 ,600 ),
            nn.ReLU(),
            nn.Linear(600 , 600),
            nn.ReLU(),
            nn.Linear(600 , 10),
            nn.LogSoftmax()

)
#####################################
```

تابع ضرر و اپتیمایزر را مطابق با صورت سوال تعیین میکنیم ، نرخ یادگیری را هم برابر با 0.001 قرار می دهیم .

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

خلاصه مدل را در شکل زیر مشاهده میکنید ، اتابع فعال ساز لایه خروجی را سافت مکس قرار دادیم تا احتمالی برخورد شود .

```
Sequential(
  (0): Linear(in_features=784, out_features=600, bias=True)
  (1): ReLU()
  (2): Linear(in_features=600, out_features=600, bias=True)
  (3): ReLU()
  (4): Linear(in_features=600, out_features=10, bias=True)
  (5): LogSoftmax(dim=None)
)
```

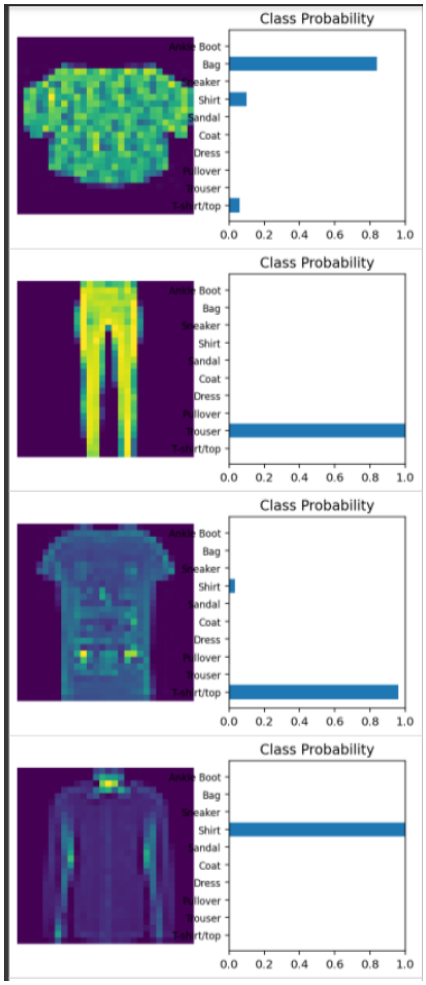
با 10 EPOCH کد را کامل می کنیم ، خطای ترین را مشاهده می کنید .

```python
## Train your model
epochs = 10

# Lists for knowing classwise accuracy

for e in range(epochs):
  running_loss = 0
  for images, labels in trainloader:

    images = images.view(images.shape[0],-1) #sqash the image in to 784*1 vector

    #reset the default gradients
    optimizer.zero_grad()

    # forward pass
    ################ Your code ############
    output = model(images)
    loss = criterion(output, labels)
    #######################################
    #backward pass calculate the gradients for loss
    loss.backward()
    #backward pass calculate the gradients for loss
    optimizer.step()

    running_loss = running_loss+loss.item()

  print(f"Training loss: {running_loss/len(trainloader)}")
```

```
/usr/local/lib/python3.7/dist-packages/torch/nn/modules/container.py:141: UserWarning: I
  input = module(input)
Training loss: 0.3017598070553752
Training loss: 0.21628437351697544
Training loss: 0.19223529064674366
Training loss: 0.1780912439420279
Training loss: 0.1718418008264607
Training loss: 0.16118162178368903
Training loss: 0.15441350926007671
Training loss: 0.14842582525196932
Training loss: 0.14571972785076734
Training loss: 0.1375994981943703
```

همان شبه کدی که برای ترین نوشتیم برای تست هم می نویسم ، و دیتای تست را به عنوان ورودی به مدل میدهیم و خروجی را با لیبل ها مقایسه کرده و نسبت تعداد درست پیش بینی شده به تعداد کل دادگان برابر ACCURACY می شود ، خطا را هم

همانند حالت قبل به دست می آوریم ، برای نمایش دادن تصاویر و احتمال هر یک از آن ها هم از تابع آماده ای که در لینک منابع ذکر شده است استفاده کردم.



برای OVERFIT شدن هم تعداد نورون های لایه ها را افزایش دادم و هم تعداد EPOCH ها را ، مدل جدید را در شکل زیر مشاهده می کنید .

```
## Define the model
################ Your code ############
model2 = nn.Sequential(
            nn.Linear(784 ,1500 ),
            nn.ReLU(),
            nn.Linear(1500 , 1500),
            nn.ReLU(),
            nn.Linear(1500 , 10),
            nn.LogSoftmax()

)
#######################################
```

```
## Train your model
epochs = 40
```

مقدار خطای نهایی بر روی داده ترین را مشاهده می کنید :

```
Training loss: 0.08436349696710087
Training loss: 0.08995782795485292
Training loss: 0.08848794223566943
Training loss: 0.0842612865212153
Training loss: 0.08104217079106166
Training loss: 0.07892891576663573
Training loss: 0.07649770510945914
Training loss: 0.08841666532247558
```

در نهایت همان پروسه قبلی را تکرار میکنیم و مشاهده می شود خطایی که بر روی داده تست به دست می آید حدود ۸ برابر خطای داده ترین است و این یعنی OVERFIT شدن

```
Loss: 0.6699567437171936, Accuracy: 89.2699966430664%
```

در ادامه ، تابع فعال سازی های مختلف و خطا های مختلف را تست کردیم که در نهایت بهترین عملکرد مربوط به تابع فعال سازی Relu و در لایه اخر Logsoftmax و با خطای ce می باشد .

س