# بسم الله الرحمن الرحیم


محمد عرفان زارع زردینی

۹۸۱۴۱۱۴۳۲

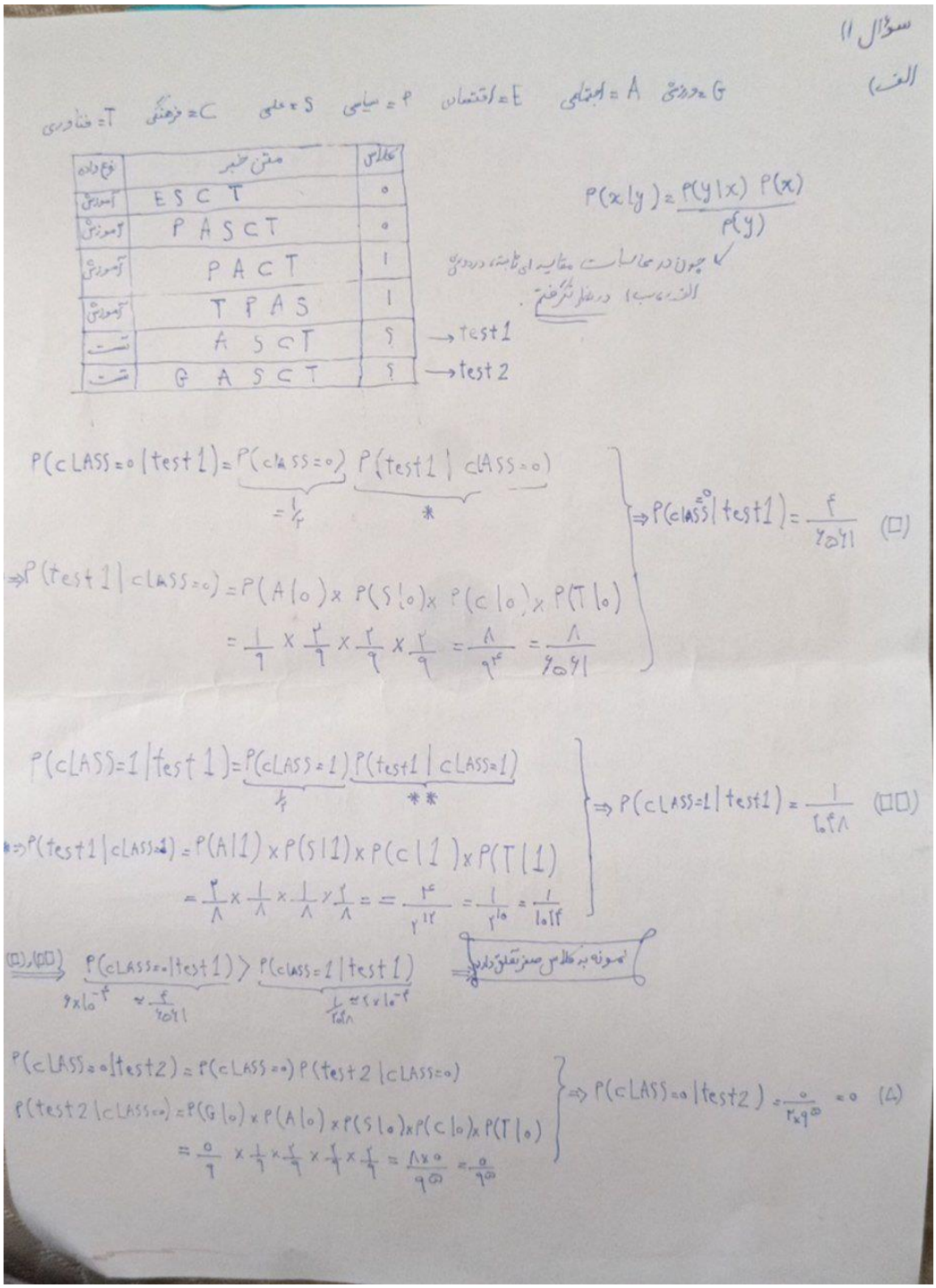
تمرین سری یک درس یادگیری عمیق

مدرس درس: استاد داوود آبادی

پاییز 1402

1) منبع:

https://towardsdatascience.com/laplace-smoothing-in-na%C3%AFve-bayes-algorithm-9c237a8bdece



$$P(class=1 \mid test2) = P(class=1) \times P(test2 \mid class=1)$$

$$P(test2 \mid class1) = P(G|2) \times P(A|1) \times P(S|1) \times P(c|1) \times P(T|1) = \frac{0}{8} \times \frac{1}{8} \times \frac{1}{8} \times \frac{1}{8} \times \frac{1}{8} = \frac{0}{2^{15}} = 0$$

$$\Rightarrow P(class=1 \mid test2) = 0 \quad (4\Delta)$$

$(\Delta) \Rightarrow (a\Delta) \Rightarrow 0=0 \Rightarrow$ به هیچ کدام از دو کلاس تعلق ندارد.

دلیل: به علت وجود داده جدید در فایل تست، دیگر با این مدل بررسی می‌کنیم، هیچ کدام از کلاس‌ها متعلق نخواهد بود.

ب)

طبق منبع مورد استفاده و برای محاسبه فرمول زیر داریم:

$$P(x \mid y) = \frac{\alpha + N_{yx}}{N_{y} + \alpha \times K}$$

$N_{yx}$: number of reviews with $y$ positive

$K$: number of dimensions (features) in data, $=1$

در مثال ما $N_y = 4$ (برای هر کلاس ۴ تا است) و $K = 4$ برای کلاس مثبت و $K = 5$ برای کلاس منفی. حال شروع به محاسبه می‌کنیم.

$$P(class=0 \mid test2) = P(class=0) \times P(test2 \mid class=0)$$

$$P(test2 \mid class=0) = P(G|0) \times P(A|0) \times P(S|0) \times P(c|0) \times P(T|0) = \frac{1}{4+4} \times \frac{3+1}{8} \times \frac{3}{8} \times \frac{3}{8} \times \frac{3}{8} = \frac{81}{2^{15}} = \frac{81}{32768}$$

$$P(class=1 \mid test2) = P(class=1) \, P(test2 \mid class=1)$$

$$P(test2 \mid class=1) = P(G|1) \times P(A|1) \times P(S|1) \times P(c|1) \times P(T|1) = \frac{1}{7} \times \frac{3}{7} \times \frac{2}{7} \times \frac{2}{7} \times \frac{3}{7} = \frac{36}{7^5} = \frac{36}{16807}$$

$\frac{18}{75} \approx 0.0010.7$

$0.0010.7 = \frac{36}{2^{15}}$ ⇒ احتمال جزو کلاس ۱ بالاتر است.

طبق توضیحات منبع، ما چند روش داریم که می توان به عدم در نظر گرفتن داده، (احتمال برابر با1) و یا احتمال صفر در نظر گرفتن برای حالت مدنظر( که سبب مشکل در محاسبه و خروجی می شود ) اشاره کرد. اما روش هموار ساز لاپلاس با فذمول موجود در برگه، وجود دارد که خروجی منطبق و ناصفر می دهد و سبب محاسبه و نتیجه گیری بهتر می شود.

2)

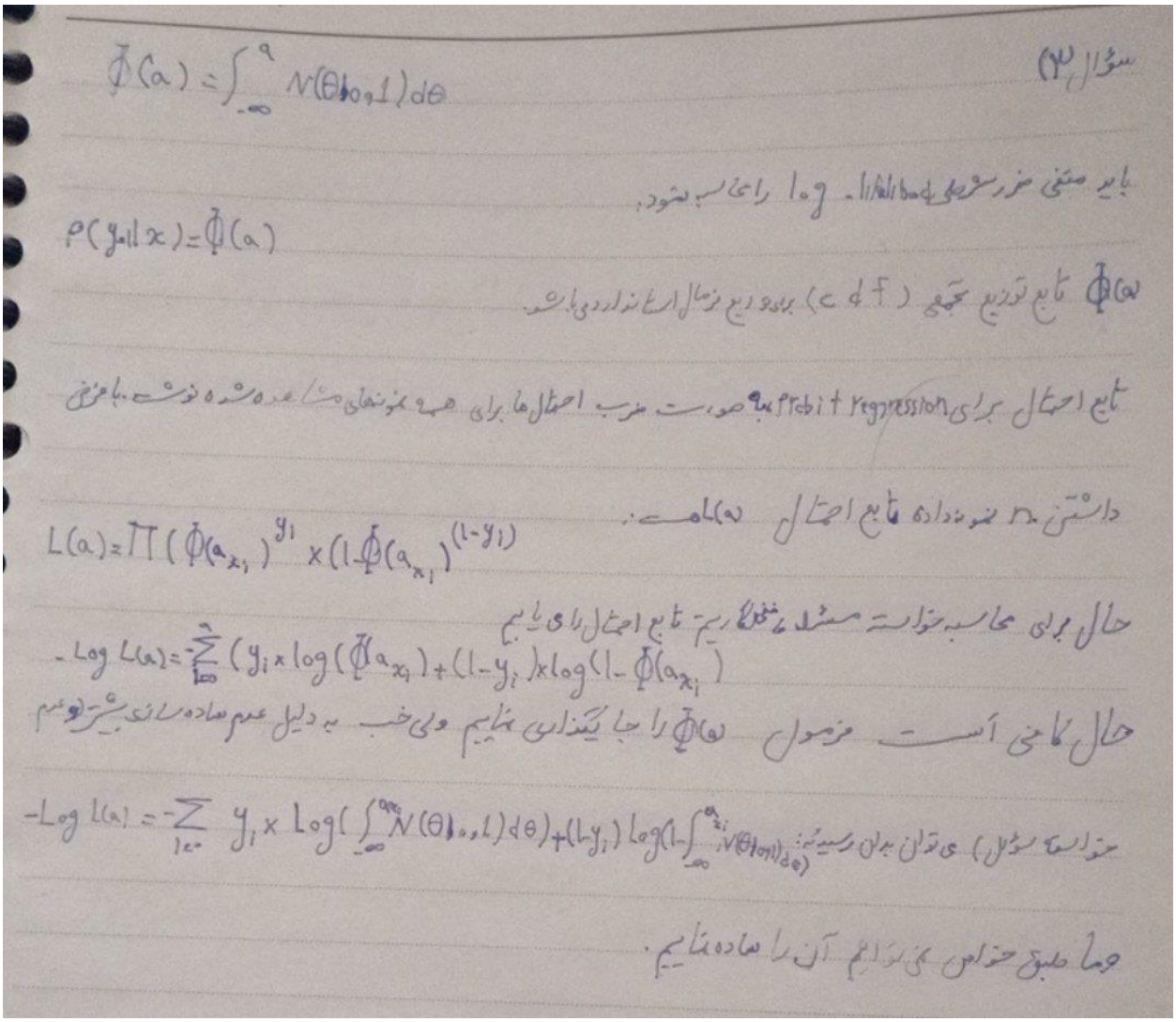
{کدها خوانده و ران شد.}

3)

منبع:

Chatgpt(prmpt: how can compute negative of the log-likelihood random loss with probit function and formulas below:

$\Phi(a)=\int N(\theta|0,1)\, d\theta$ (integral domain: $-\infty$ to a)




4)

منابع:

Chatgpt(prompt: explain some reason for using activation function in MLP networks?)

الف)

از توابع فعالسازی در شبکه MLP به دلایل زیر استفاده می نماییم: (برای هر دلیل توضیح کوتاهی نوشته شده است)

1 .انتشار گرادیان: توابع فعال‌سازی در فرآیندbackward ، از انتشار گرادیان استفاده می‌کنند. در آموزش MLPبا استفاده از روش‌هایی همچونbackpropagation ، گرادیان تابع هزینه محاسبه و برای بروزرسانی وزن‌های شبکه استفاده می‌شود. توابع فعال‌سازی با مشتقات قابل تعریف همچون sigmoid, tanh, and ReLU، به گرادیان‌ها امکان عبور را در سرتاسر شبکه، بروزرسانی وزن‌ها و optimize کردن عملکرد شبکه را فراهم می نمایند.

2 .قدرت نمایش: توابع فعال‌سازی، ویژگی‌ها و توانایی‌های متفاوتی برای مدل‌سازی انواع مختلف داده ها دارند. به عنوان مثال، تابع sigmoid ، اغلب در وظایف طبقه‌بندی دودویی استفاده می‌شود، در حالی که تابع softmaxبرای مسائل طبقه‌بندی چنددسته‌ای مناسب است . یا توابع فعال‌سازی مانند ReLU به دلیل توانایی در مدیریت vanishing gradients و ترویج فعال‌سازی‌های پراکنده(sparse activations)، در یادگیری عمیق محبوب هستند .

3 .غیرخطی بودن :

توابع فعال‌سازی، تبدیلات غیرخطی را به خروجی هر نورون در شبکه اعمال می‌کنند. بدون وجود توابع فعال‌سازی غیرخطی، MLPتنها ترکیب خطی از ورودی‌های خود خواهد بود که سبب محدودیت ظرفیت نمایشی در آن می‌شود. توابع فعال‌سازی غیرخطی به MLP امکان می‌دهند روابط پیچیده و غیرخطی موجود در داده‌ها را یاد بگیرد و تقریب بزند.

4 .محدوده خروجی و نرمال‌سازی: توابع فعال‌سازی می‌توانند محدوده خروجی هر نورون را مشخص و محدود نمایند. به عنوان مثال، تابع sigmoid خروجی را به محدوده [1، 0] نگاشت می‌کند، در حالی که تابع تانژانت خروجی را به [1، 1-] نگاشت می‌کند. این نرمال‌سازی محدوده خروجی می‌تواند برای تضمین استحکام و جلوگیری از اشباع شدن نورون‌ها مفید باشد.

ویژگی‌های بیان شده سبب می شود که شبکه‌ها قابلیت یادگیری الگوهای پیچیده و دستیابی به عملکرد بهتر در تسک های مختلف یادگیری ماشین را دارا شوند.

ب)

تابع فعال ساز برای آن که امکان استفاده در یک شبکه عصبی را دارا باشد، باید نکات و ویژگی های زیر را رعایت نماید:

1. غیرخطی باشد. تا شبکه عصبی بتواند الگوهای پیچیده و غیرخطی را در داده ها یاد بگیرد. این در حالی است که یک تابع فعالساز خطی ، سبب می شود که شبکه اساسا همچون یک مدل خطی باشد و ظرفیت نمایش محدود شود.
2. تمایز پذیری داشته باشد. این ویژگی برای backpropagation هنگامی که الگوریتم مورد استفاده برای آموزش شبکه عصبی باشد،بسیار مهم وضروری است. تابع فعالساز باید یک مشتق تعریف شده داشته باشد تا امکان محاسبه برای گرادیان ها فراهم شود، که در جهت بروزرسانی وزن های شبکه در طول آموزش استفاده می شود.

توابع فعالساز مرسوم همچون sigmoid, tanh, and ReLU شامل این ویژگی ها می باشند و در نتیجه هنگامی هم که از آن ها استفاده می شود به خوبی کار می کنند. بدین سان آموزش و یادگیری کارآمد در شبکه عصبی امکان رخ دادن می یابد.

اما برخی توابع فعالساز غیرخطی نیز وجود دارند که شامل این ویژگی ها نمی شوند. همچون step function (Heaviside step function) . این تابع یک ورودی دارد و خروجی آن باینری و براساس بزرگتر یا مساوی بودن ورودی است. این تابع فعالساز دارای ناپیوستگی در صفر می باشد که سبب غیرقابل تمایز بودن آن می شود. بدین دلیل از این تابع در شبکه های عصبی به عنوان فعالساز استفاده نمی شود. همچنین به دلیل ناتوانی در محاسبه مشتقات برای backpropagation ، آموزش شبکه عصبی را دچار مشکل می نماید.

(5

منبع:

Chatgpt(prompt:compare relu,sigmoid,softmax and tanh . say advantages and disadvantages.)
chatgpt(for part c , when have bugs for run and give simple mlp model)
/https://pythonexamples.org/pillow-convert-image-to-grayscale

الف)

1. سیگموید(sigmoid):

فرمول:

1/(1+exp(-x))

فواید:

مقادیری میان 0و1 را بدست می آوزد و برای مسائل طبقه بندی باینری که خروجی احتمالات را نمایش می دهد، مناسب است.همچنین متمایز بودن و همچنین روش optimization براساس گرادیان همچون backpropagation می توان استفاده کرد.

مشکلات:

مشکل vanishing gradient را دارا می باشد که سبب می شود که گرادیان برای مقادیر ورودی شدید ،بسیار کوچک شود و سبب همگرایی کندتر در طول train شدن شود. همچنین مرکز آن غیر صفر است که سبب همگرایی نزولی گرادیان را کاهش دهد. همچنین دارای هزینه محاسباتی زیاد است(به دلیل نمایی بودن).

2 . Softmax

فرمول:

Exp(x)/sum(exp(x))

فواید:

برای مسائل طبقه بندی چندکلاسه استفاده می شود و توزیع احتمال برروی چندین کلاس را خروجی می دهد. این نکته سبب می شود که مجموع احتمال همه کلاس ها به 1 برسد.

مشکلات:

همچون سیگموید، دارای مشکل vanishing gradient می باشد. همچنین به داده های پرت حساس بوده و میتواند احتمال بالایی را به مقادیر ورودی کوچک دهد که سبب بی ثباتی در مدت آموزش شود.

3. ReLU

فرمول:

=max(0,x)

فواید:

از لحاظ محاسباتی در پیاده سازی کارآمد و آسان است. با دوری از اشباع شدن برای مقادیر ورودی مثبت، سبب می شود ،مشکل vanishing gradient را نداشته باشد. همچنین همگرایی را در مدت آموزش برای شبکه عصبی عمیق ، سرعت می بخشد.

مشکلات:

می تواند سبب مشکل dying ReLU شود که نورون ها به طور دائم، غیرفعال شده و در صورت خروج صفر برای همه ورودی ها، یادگیری متوقف می شود. این مشکل با استفاده از Leaky ReLU یا Parametric ReLU (PReLU) حل می شود.

## Tanh.4

فرمول:

=(exp(x)-exp(-x))/(exp(x)+exp(-x))

فواید:

صفر محور است که همگرایی سریعتری را در الگوریتم بهینه سازی براساس گرادیان را امکان می سازد.

همچنین مقادیر ورودی آن در بازه(-1و1) بوده و بدین سان برای داده های دارای مولفه مثبت و منفی مناسب است.

مشکلات:

مشکل vanishing gradient برای مقادیر ورودی خیلی بزرگ را دارد. محاسبات نمایی نیز دارد که سبب هزینه محاسباتی بالا می شود.

مقایسه:

سیگموید و tanh برای تسک های طبقه بندی باینری مناسب است، در حالی که softmax برای طبقه بندی چند کلاسه مناسب است.

relu در لایه های پنهان برای اکثر تسک ها استفاده می شود و به کاهش vanishing gradient کمک می کند. همچنین نسبت به سیگموید و tanh از نظر محاسباتی مناسب تر است.

سیگموید و tanh دارای خروجی smooth است ، درحالی که relu دارای خروجی piecewise-linear می باشد.

Relu می تواند سبب dying relu شود اما در استفاده از Leaky ReLU و PReLU این مشکل بهبود می یابد.

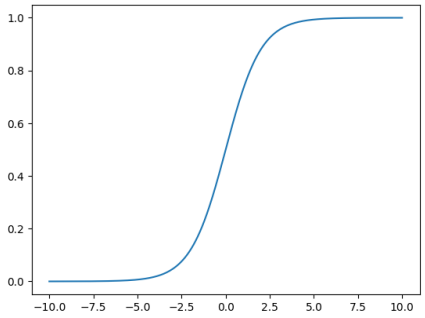
توابع softmax و سیگموید بیشتر به فذمت احتمالاتی قابل تفسیر اند درحالی که relu و tanh به صورت ذاتی احتمالاتی نیستند.

ب)

با توجه به اینکه قرار است پیاده سازی توابع فعالساز انجام شده و با دستورات آماده آن مقایسه شود، پس از کتابخانه های numpy برای نوشتن توابع استفاده می شود و فرمول ریاضیاتی آن پیاده سازی شده و در cell های بعدی آن ها به ترتیب مقایسه شده اند که یکسان هستند یا خیر و در نهایت روی نمودار نمایش داده شده(پلات شده) است. براساس فرمول توابع فعال ساز تابع های آن نوشته شده است که عکس های کد هر تابع و نمودار آن ها کشیده شده است .(فرمول های آن ها در سوالات قبلی اشاره شده است.)

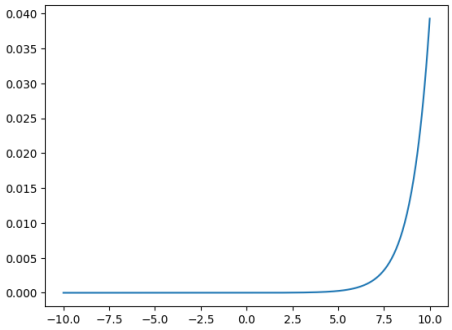
:Sigmoid

```python
# implement the sigmoid activation function
def sigmoid_func(x):
    return 1/(1+np.exp(-x))
```



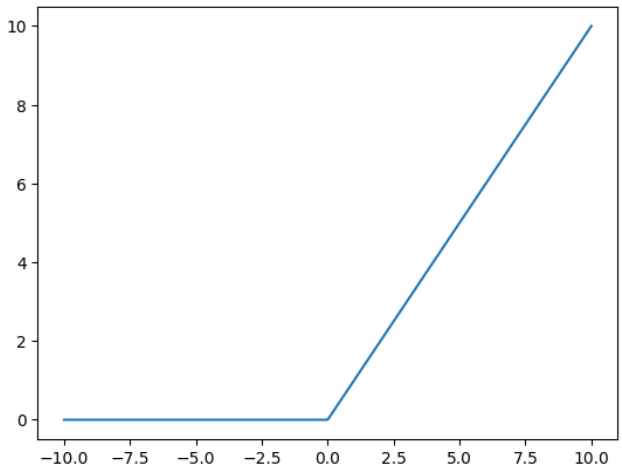

:softmax

```python
def softmax_func(x):
    e=np.exp(x)
    return e/e.sum()
```



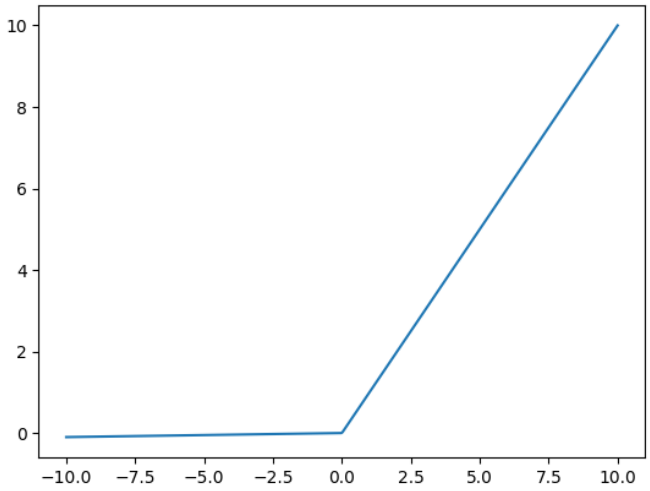

:ReLU

```python
# implement the ReLU activation function
def relu_func(x):
    return np.maximum(0,x)
```



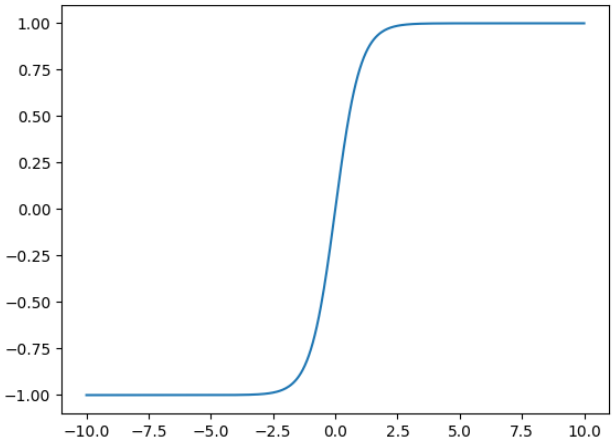

:Leaky ReLU

```python
# it has a constant parameter .we consider it 0.01
def leaky_relu(x):
    return np.maximum(x*0.01,x)
```




:Tanh

```python
# implement the tanh activation function
def tanh_func(x):
    return (np.exp(x)-np.exp(-x))/(np.exp(x)+np.exp(-x))
```




ج)

برای پیاده سازی مدلی براساس mlp ، من ساختار زیر را پیش بردم. مدل شامل سه لایه کلی ، (یک لایه ورودی، 1 لایه پنهان و یک لایه خروجی است که به علت نیاز به مدلی اولیه و ساده برای آموززش پایه ای روی

تعداد کمی داده از این تعداد لایه استفاده شد و از پیچیده کردن بیش از حد مدل و استفاده نابجا از لایه های اضافی جلوگیری شد.) که ابعاد لایه ورودی آن را براساس ابعاد تصویر در نظر گرفتم و با توجه به اینکه تصویر را با فرمت سیاه سفید مدنظر داشتم، فرمت ورودی 1*h*w است. سایز لایه مخفی را هم براساس تجربه و نرم کلی مناسب برای مدل با این ساختار 256 در نظر گرفتم. این معماری ،معماری پایه مدل براساس mlp است که توانایی و ظرفیت مناسب برای نمایش الگو های پیچیده را دارا می باشد. تعداد نورون ها براساس سایز ورودی برابر است با 1*512*461 (دلیل آن براساس ابغاد تصویر ورودی است و برای تحلیل بهتر عکس و هر پیکسل تصویر می باشد. میشد از تعداد لایه بیشتر استغاده کردولی سبب پیچیدگی بیی مورد و اضافه محاسبات می شد.) و برای سایز لایه مخفی 256 می باشد. انتخاب سایز این لایه دلخواه بوده و به صورت دستی مشخص شده . البته افزایش نورون های لایه پنهان می تواند سبب افزایش ظرفیت مدل برای یادگیری الگوهای پیچیده تر در داده ها باشد. اما لایه خروجی و تعداد نورن آن براساس تعداد کلاس ها مشخص می شود که اینجا سه هست که هرکدان احتمال یک کلاس را نشان می دهند. پیرامون تابع فعالساز یا توجه به معمول بودن ReLu به علت سادگی و توانایی در مدیریت مشکلات گرادیان مانند ناپدید شدن گرادیان، مناسب است. همچنین سبب غیرخطی شدن در شبکه می شود و بدین سان به مدل در آموزش روابط پیچیده میان ویژگی های ورودی و کلاس مدنظر همیاری می رساند. پیرامون تابع ضرر ، باتوجه به تعداد کلاس های موجود و اینکه نیاز به طبقه بندی چند کلاسه بود از CrossEntropy استفاده نمودم. همچنین در این تابع ضرر بین احتمالات کلاس پیش بینی شده و واقعی با softmax خود را اپدیت می کند و تابع ضرر را محاسبه می کند. همچنین بدین سان مدل را به سمت آنکه احتمالات بالاتری را برای کلاس صحیح تولید نماید ، و جریمه مناسب را برای انحراف از لیبل مناسب در نظر گیرد.همچنین مناسب ترین بهینه ساز برای این مدل adam می باشد که در بهینه سازی در مدل کمک می نماید.

د)

```
from PIL import Image
import matplotlib.pyplot as plt
import torch
import torch.optim as optim
import torch.nn as nn
from torch.utils.data import Dataset as ds
from torchvision import transforms as tf
```

کتابخانه های مورد نیاز برای پیاده سازی کد را اضافه می نماییم. از کتابخانه pil برای کار برروی عکس های داده و از کتابخانه matplot برای نمایش داده استفاده می شود.

```
class MLP(nn.Module):
    def __init__(self, inp_size, h_size, num_classes): # input size,hidden size, nu
        super(MLP, self).__init__()
        self.layer1 = nn.Linear(inp_size, h_size)
        self.relu1 = nn.ReLU()
        self.layer2 = nn.Linear(h_size, h_size)
        self.relu2 = nn.ReLU()
        self.layer3 = nn.Linear(h_size, num_classes)

    def forward(self, x):
        outp = x.view(x.size(0), -1)
        outp = self.layer1(outp)
        outp = self.relu1(outp)
        outp = self.layer2(outp)
        outp = self.relu2(outp)
        outp = self.layer3(outp)
        return outp
```

همانطور که در قسمت قبل هم گفتیم برای مدل مدنظر از سه لایه با سایز های و فرمت بیان شده در قسمت قبل استفاده می نماییم و از تابع فعالساز relu کمک میگیریم. همچنین می دانیم که باید در forward خروجی هر لایه را به ورودی لایه بعد بدهیم.

```
class CustomDataset(ds):
    def __init__(self, img_names, labels, transform=None):
        self.img_names = img_names
        self.labels = labels
        self.transform = transform

    def __len__(self):
        return len(self.img_names)

    def __getitem__(self, idx):
        img = self.img_names[idx]
        label = self.labels[idx]
        image = Image.open(img).convert('L')
        if self.transform:
            image = self.transform(image)
        return image, label
```

برای این قسمت از chatgpt کمک گرفته شده. و باتوجه به داده های خارج از محدوده داده های مشهور، از این کلاس نوشته شده برای خواند تصویر و تبدیل داده های به سیاه و سفید استفاده شده است.

```
w = 512
h = 461
inp_size = 1 * h * w
h_size = 256
num_classes = 3
model = MLP(inp_size, h_size, num_classes)
```

در اینجا براساس طول و عرض تصویرمان سایز ورودی را مشخص میکنیم همچنین داده های  دیگری شامل تعداد کلاس ها و سایز لایه چنهان را مشخص می نماییم و در نهایت مدل را تعریف میکنیم.

```
transformer = tf.Compose([tf.Resize((w, h)),tf.ToTensor(),])
dataset = CustomDataset(img_names, labels, transform=transformer)
optimizer = optim.Adam(model.parameters(), lr=0.005)
losses = nn.CrossEntropyLoss()
```

در این قسمت براساس ماژول transforms در torchvision، چندین تبدیل را همزمان می سازد. همچین دیتاسیت مد نظر را تعریف میکنیم و لیبل های مد نظر را نسبت میدهیم. و بهینه ساز و تابع ضرر مدنظرمان را تعریف مینماییم. نرخ یادگیری استفاده شده در بهینه ساز براساس آزمون وخطا بدست آمده.

```
epoch_num = 20
for epoch in range(epoch_num):
    all_loss = 0
    for image, target in dataset:
      optimizer.zero_grad()
      image = image.unsqueeze(0)
      outp = model(image)
      target = torch.tensor([target])
      loss = losses(outp, target)
      all_loss += loss.item()
      loss.backward()
      optimizer.step()

    print(f'Epoch {epoch + 1}, Loss: {all_loss}')
```

حال تعداد epoch مد نظرمان را مشخص میکنیم. من براساس ران کردن کد ب هاین نتیجه رسیدن که مدل به تعداد زیاد epoch برای آموزش نیاز ندارد و در تعداد نسبتا کمی آموزش میبینند و خطای آن کاهش میابد.

```
for i in range(len(img_names)):
    img = img_names[i]
    label = labels[i]
    image = Image.open(img).convert('L')
    image = transformer(image)

    outp = model(image.unsqueeze(0))
    pred_index = torch.argmax(outp, dim=1).item()
    pred_label = labels[pred_index]

    plt.imshow(image.view(w, h).numpy(), cmap='gray')
    plt.title(f"original Label: {label}\npredict Label: {pred_label}")
    plt.show()
```

در نهایت هم  تصویر و لیبل آنرا پردازش کرده و تصویر را سیاه سفید مینماییم و به سایز مدنظرمان( در صورت عدم همخوانی ) میبریم و تغییرات مورد نیاز را اعمال می نماییم. در نهایت ایندکس ولیبل پیش بینی را ذخیره نموده و با لیبل اصلی در کنار هم برای مقایسه چاپ می نماییم.

(6

منابع:


1. https://ai.stackexchange.com/questions/9828/what-happens-when-i-mix-activation-functions#:~:text=Combining%20activation%20functions%20to%20form,a%20single%2C%20effective%20network%20design

2. https://datascience.stackexchange.com/questions/72559/different-activation-function-in-same-layer-of-a-neural-network#:~:text=You%20may%20try%20using%20two,they're%20zero%20or%20positive.

3.chatgpt(prompts ,a: if we use two activation function relu and sigomid on it, for nueral network for binary classification , whats problem? ,b: if we use two activation function for nueral network , whats problem? )


چالش های استفاده از دو تابع فعال ساز گفته شده، شامل موارد زیر می باشد:

1.Vanishing Gradients :

تابع فعالساز ReLU ممکن است سبب رخداد این مشکل برای ورودی منفی می شود و گرادیان ها بسیار کوچک می شود ، که میتواند یادگیری و بروزرسانی وزن ها در هنگام backpropagation را دچار مشکل کند و سبب همگرایی کندتر شود. همچنین میتواند DYING RELU هم رخ دهد که به شبکه اسیب می رساند.

2. اشباع سیگموید:

برای ورودی های مثبت و منفی بسیار بزرگ اشباع می شود و سبب خروجی های نزدیک 0 یا 1 ایجاد می شود. این اشباع سبب حساسیت شبکه نسبت به تغییرات ورودی کمتر و روند یادگیری، کند شود.

3.مرز تصمیم:

نکته ای که وجود دارد این است که استانه 0.5 که در اینجا مورد استفاده است ممکن است همیشه برای طبقه بندی دوکلاس بهینه نباشد. آن وابسته به پیچیدگی داده ها و توانایی شبکه ما برای یادگیری است. مرز و آستانه تصمیم ممکن است که با آستانه واقعی کلاس هماهنگ نباشد و سبب طبقه بندی اشتباه شود.

4.داده های نامتعادل:

اگر مجموعه داده ای که در اختیار داریم ، توزیع نامتعادلی داشته باشد و از یک کلاس تعداد بیشتری موجود باشد ، استانه 0.5 ممکن است مناسب نباشد و منجر به پیش بینی نادرست نسبت به طبقه با داده بیشتر شود که سبب عملکرد ضعیف در طبقه دیگر( داده کمتر) شود.

5. تنظیم Hyperparameter :

استفاده از چندین توابع فعالساز سبب ایجاد هایپرپارامتر اضافی برای شبکه می شود .ما نیاز به بهترین تعادل و پارامتر ها برای شبکه داریم و باید تعادلی میلن دو تابع فعالساز برقرار شود و همچین پیرامون تعداد واحد هایی که هر تابع فعالساز عمل می نماید.

برای برطرف کردن این چالش ها از روش هایی همچون تنظیم آستانه ، استفاده از توابع فعالساز همسو با نیاز،استفاده از معماری های پیشرفته تر همچون کانولوشنال یا RNN که بسته به ماهیت داده و مشکل طبقه بندی مان است.

7)

منابع:

a) https://levity.ai/blog/difference-machine-learning-deep-learning

a') https://www.zendesk.com/blog/machine-learning-and-deep-learning/

a") https://flatironschool.com/blog/deep-learning-vs-machine-learning/#:~:text=Machine%20learning%20is%20about%20computers,needs%20less%20ongoing%20human%20intervention.

الف)

توضیحات اولیه: یادگیری عمیق زیر مجموعه ای از یادگیری ماشین هست که بخشی از هوش مصنوعی را شامل می شود. یادگیری ماشین پیرامون فرآیند هایی است که کامپیوتر بتواند با دخالت کمتر انسانی عمل و فکر نماید، درحالی که در یادگیری عمیق تفکر و ساختار ذهنی مشابه مغز انسان داشته باشد.همچنین یادگیری ماشین نیازمند قدرت محاسباتی کمتر است.

مهمترین تفاوت های آن شامل مهندسی فیچرها، داده های مورد نیاز برای آموزش، پیچیدگی مدل، کارایی و دیتا های غیرساختارمند، و همانطور که گفتم، منابع محاسباتی می باشد. یادگیری ماشین از مدل های سنتی با پیچیدگی محدود استفاده می کند،نیازمند مهندسی فیچر ها به شکل دستی است، همچنین فرمت سنتی آن نیازمند داده های برچسب خورده کمتری برای رسیدن به حالت optimal خود می باشد، همچنین همانطور که گفتیم قدرت محاسباتی کمتری را می خواهد.

این درحالی است که یادگیری عمیق ، شبکه عصبی با چند لایه برای آموزش الگوهای پیچیده به شکل خودکار استفاده می نماید، فیچرهارا به شکل خودکار می آموزد، نیاز به داده های برچسب خورده زیادی برای آموزش دارد، و همچین قدرت محاسباتی زیادی نیازمند است وگاهی باید از سخت افزار های قوی همچون gpu استفاده شود.به علاوه اینکه می تواند داده های غیر ساختارمند همچون تصویر،صدا و متن را بررسی نماید.

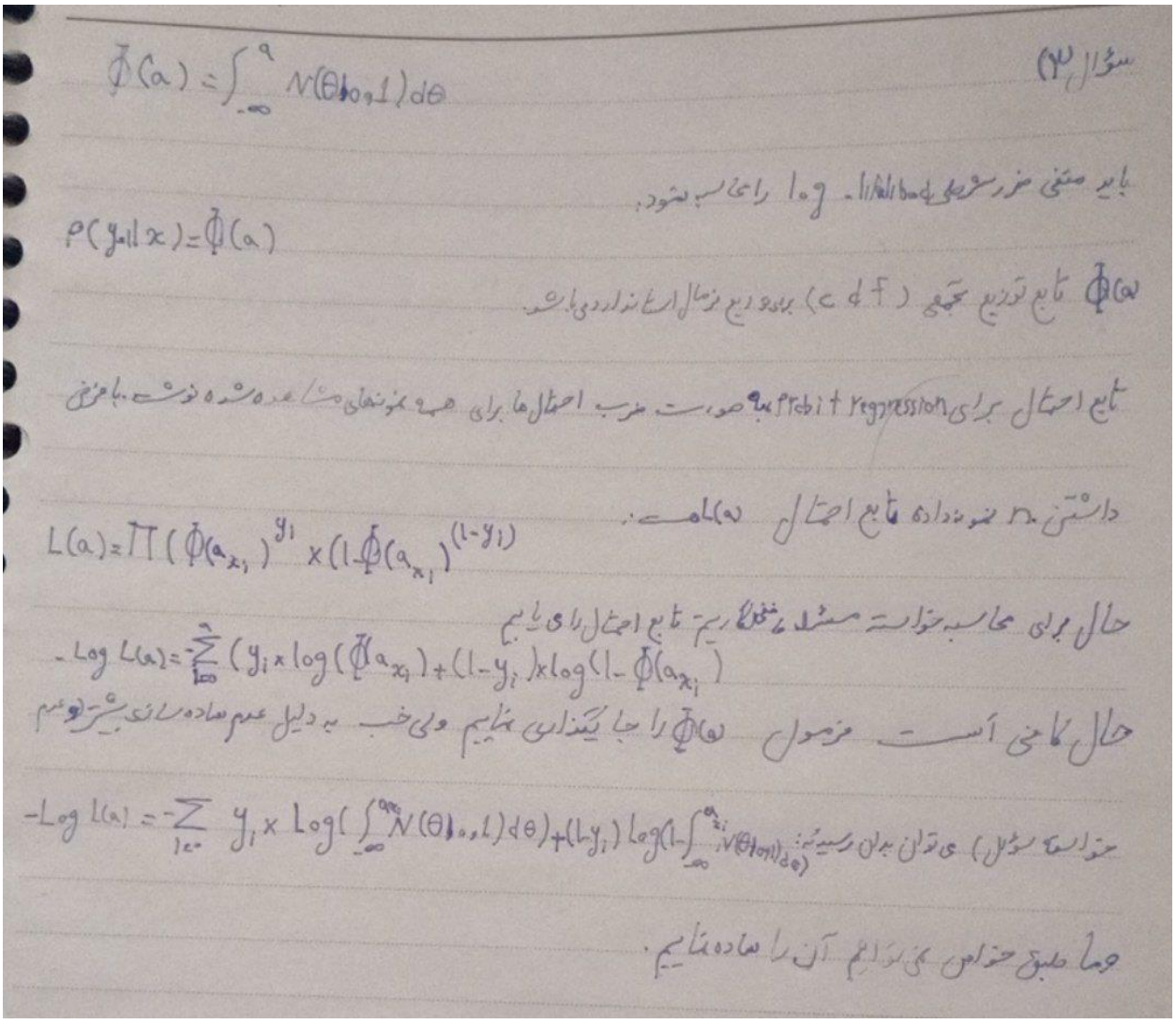
ب)

نکته ای که در این سوال هست این است که بسته تسک می تواند حالت مختلف رخ دهد(جلوتر توضیح می دهم) ولی استنباط کلی که می توان نمود این است که با توجه به برخی پارامتر، در لایه های آخر باشد، بهتر می باشد. پس لایه 11ام بهتر است.(دلیل آن هم در توضیحات پایین می گویم) ولی نکته کلی که وجود دارد، انتخاب معماری وطراحی خاص شبکه ، میتواند بر مناسب بودن لایه های مختلف اثرگذار باشد

لایه های عمیق تر ویژگی های پیچیده تر و معنایی بالاتری دارند، که این سبب می شود لایه یازدهم مناسب تر است. از طرفی لایه هفتم، ویژگی های سطح پایین تر و به ورودی نزدیکتر را نمایان می کند درحالی که لایه های یازدهم برروی ویژگی های عمیق تر و high level تری کار کرده و نمایش می دهد.بدین سان لایه های 11 ویژگی های مرتبط تری را برای طبقه بندی دریافت کند.

از سویی اگر مشکل Vanishing Gradients وجود داشه باشد، لایه های پایین تر به دلیل تاثیر کمتری که از آن گرفته اند، مناسب تر است. پس در این حالت خاص لایه 7ام بهتره.

.



ج)

معماری و انتخاب طراحی در بهینه سازی عملکرد شبکه نقش دارد.معیارهای آن پیچیدگی تابع ،داده های موجود، محدودیت های محاسباتی و ویژگی های generalization است .

همچنین شبکه با عرض بیشتر، بار و هزینه محاسباتی بالایی دارد و اگر داده مان ناکافی باشد، ممکن است سبب overfitting شود .

شبکه عمیق تر ، برای یادگیری الگوهای پیچیده و دستیابی به generalization بهتر کارایی بیشتری دارد. همچنین شبکه عمیق تر، کارایی محاسباتی بهتری دارد و میتواند قدرت بالایی را با پارامترهای کمتر بدست آورند.

در نتیجه می توان اشاره کرد که **شبکه عمیق تر** مناسب تر است.

د)

افزودن لایه به شبکه عصبی میتواند مزایا و معایب گوناگونی داشته باشد که شامل:

مزایا:

1.افزایش generalization برای داده های دیده نشده.

2.یادگیری های سلسله مراتبی برای انتزاع بهتر

3. انتقال توانایی یادگیری برای سود بردن در مدل پیش آموزش دیده

4.ظرفیت مدل برای اسخراج و گرفتن الگو های پیچیده تر را فراهم می نماید.

معایب:

1.چالش در معماری و tune کردن هایپر پارامترها

2.نیاز به داده های بیشتر و دیتاست بزرگ تر

3.افزایش پیچیدگی محاسبات و منابع مورد نیاز

4.افزایش پتانسیل رخداد vanishing gradients یا exploding gradients

5.افزایش رخدادoverfitting بدون regularization مناسب