Q1)

پاسخ در pdf در فایل زیپ

Q2)

بدون نمره

Q3)

پاسخ در pdf در فایل zip

Q4)

الف)

چندین دلیل برای استفاده از توابع فعال‌سازی در شبکه‌های MLP وجود دارد:

غیرخطی بودن: توابع فعال‌سازی غیرخطی هستند که امکان مدل‌سازی رفتارهای غیرخطی در داده‌ها را فراهم می‌کنند. این قابلیت بسیار مهم است زیرا بسیاری از مسائل واقعی دارای الگوها و رفتارهای غیرخطی هستند که با استفاده از توابع خطی قابل تخمین نیستند.

نرمال‌سازی و وزن‌دهی: توابع فعال‌سازی می‌توانند ورودی‌ها را نرمال‌سازی کرده و وزن‌دهی کنند. به عنوان مثال، تابع سیگموید (Sigmoid)بازه ورودی را به بازه (0, 1) نرمال می‌کند و می‌تواند احتمالات را در مسائل دسته‌بندی مدل کند.

غیرخطی بودن گرادیان: توابع فعال‌سازی غیرخطی باعث می‌شوند گرادیان شبکه غیرصفر و غیرخطی باشد. این امر، بهینه‌سازی و آموزش شبکه را تسهیل می‌کند، زیرا بهینه‌سازهای گرادیان مبتنی بر کاهش گرادیان می‌توانند به طور موثر تابع هدف را در فضای پارامترها جستجو کنند.

تفاوت‌پذیری: برخلاف توابع خطی، توابع فعال‌سازی غیرخطی تفاوت‌پذیر هستند که به شبکه اجازه می‌دهد در صورت نیاز به تغییرات جزئی وزن‌ها و پارامترها پاسخ دهد.

با توجه به مزایای فوق، انتخاب توابع فعال‌سازی مناسب برای شبکه‌های MLP بسیار مهم است و بسته به مسئله مورد نظردارد.

پاسخ به کمک ChatGPT نوشته شده.

ب)

با توجه به تئوری Universal Approximation، از نظر تئوری از هر تابع غیرثابت، محدود، یکنواخت صعودی و پیوسته می‌توان استفاده کرد. اما در عمل میبینیم که هر تابعی نمی‌تواند مدل را به سرعت به پاسخ نزدیک کند و یا برخی مدل ها مثل ReLU هستند که خاصیت محدود بودن را ندارند ولی بسیار در مدل های MLP استفاده می‌شوند.

بنابراین در عمل تنها از توابعی میتوان استفاده کرد که علاوه بر غیرخطی بودن خاصیت‌های زیر را هم داشته باشند:

1- از نظر محاسباتی سبک باشند
2- مدل را دچار Vanishing gradient یا Exploding Gradient نکنند
3- یک نوا باشد
4- در برخی مواقع لازم است تابع مشتق پذیر باشد

Q5)

الف)

الگوریتم‌های شبکه‌های عصبی مصنوعی معمولاً از توابع فعال‌سازی خاصی برای افزایش قدرت مدلسازی و انتقال اطلاعات در لایه‌های مختلف استفاده می‌کنند. در ادامه، توابع فعال‌سازیSigmoid ، ReLU، tanhو softmax را توضیح می‌دهم و مزایا و معایب هرکدام را معرفی می‌کنم:

تابع:Sigmoid

تعریف: تابع Sigmoid به شکل f(x) = 1 / (1 + exp(-x)) تعریف می‌شود.

محدوده خروجی: مقدار خروجی تابع Sigmoid در بازه (0, 1) قرار دارد، که معمولاً به عنوان احتمال یک رویداد در مسائل دسته‌بندی باینری استفاده می‌شود.

مزایا: نرم‌افزاری و قابل مشتق‌پذیری، مناسب برای مسائل دسته‌بندی باینری در لایه خروجی، استفاده در شبکه‌های بازگشتی.

معایب: مشکل کاهش گرادیان (vanishing gradient) در شبکه‌های عمیق، خروجی محدود در بازه (0, 1) که می‌تواند باعث ایجاد مشکل در تعلیم شبکه شود. همچنین مقدار همیشه مثبت است که میتواند باعث حرکت زیگزاگی loss به سمت نقطه مینیمم شود.

تابع:ReLU (Rectified Linear Unit)

تعریف: تابع ReLU به شکل f(x) = max(0, x) تعریف می‌شود.

محدوده خروجی: مقدار خروجی تابع ReLU برای مقادیر مثبت برابر با خود عدد و برای مقادیر منفی برابر با صفر است.

مزایا: ساده و محاسباتی سریع، از مشکل کاهش گرادیان در شبکه‌های عمیق جلوگیری می‌کند، افزایش سرعت آموزش، عملکرد خوب در بزرگ‌نمایی داده‌ها.

معایب: خروجی برای مقادیر منفی برابر با صفر است که ممکن است منجر به از دست رفتن اطلاعات مربوط به گرادیان‌های منفی شود.

تابع:tanh (Hyperbolic Tangent)

تعریف: تابع tanh به شکل f(x) = (exp(x) - exp(-x)) / (exp(x) + exp(-x)) تعریف می‌شود.

محدوده خروجی: مقدار خروجی تابع tanh در بازه (-1, 1) قرار دارد.

مزایا: سیمتری حول صفر و متناظر با تابعSigmoid ، قابل استفاده در شبکه‌های بازگشتی، مشکل کاهش گرادیان در شبکه‌های عمیق را کاهش می‌دهد.

معایب: همچنان مشکل کاهش گرادیان در شبکه‌های عمیق وجود دارد.

تابع:softmax

تعریف: تابع softmax به شکل f(x_i) = exp(x_i) / sum(exp(x_j)) برای هر عنصر x_i در ورودی تعریف می‌شود.

محدوده خروجی: مقادیر خروجی تابع softmax در بازه (0, 1) قرار دارند و مجموع آنها برابر با 1 است. این تابع معمولاً در لایه خروجی برای مسائل دسته‌بندی چند دسته‌ای استفاده می‌شود.

مزایا: تبدیل ورودی به یک توزیع احتمالی، تمرکز بر دسته‌ها و رقم‌های بزرگتر در خروجی.

معایب: حساس به مقادیر بزرگ و کوچک ورودی، ممکن است در مواردی که توزیع آشکاری وجود ندارد، نتایج نامطلوبی ایجاد کند.

هر کدام از این توابع کاربرد مخصوص خود را دارند. مثلا از تابع softmax برای خروجی دسته بندی چند کلاسه استفاده میشود یا از توابع sigmoid و tanh برای خروجی دسته بندی دو کلاسه استفاده میشود و از relu در لایه‌های میانی. لازم است با توجه به نوع مسئله و هزینه محاسباتی هر تابع، تابع مناسب را انتخاب کنیم.

ب)

این نوتبوک را در colab تکمیل کردم. گزارش کدهای نوشته شده به شرح زیر است:

برای پیاده سازی sigmoid از توابع numpy استفاده کردم تا این عملیات را به صورت vectorize شده انجام دهم:

```
# implement the sigmoid activation function
##### TO DO #####
def my_sigmoid(x):
  return 1 / (1 + np.exp(-1 * x))
```

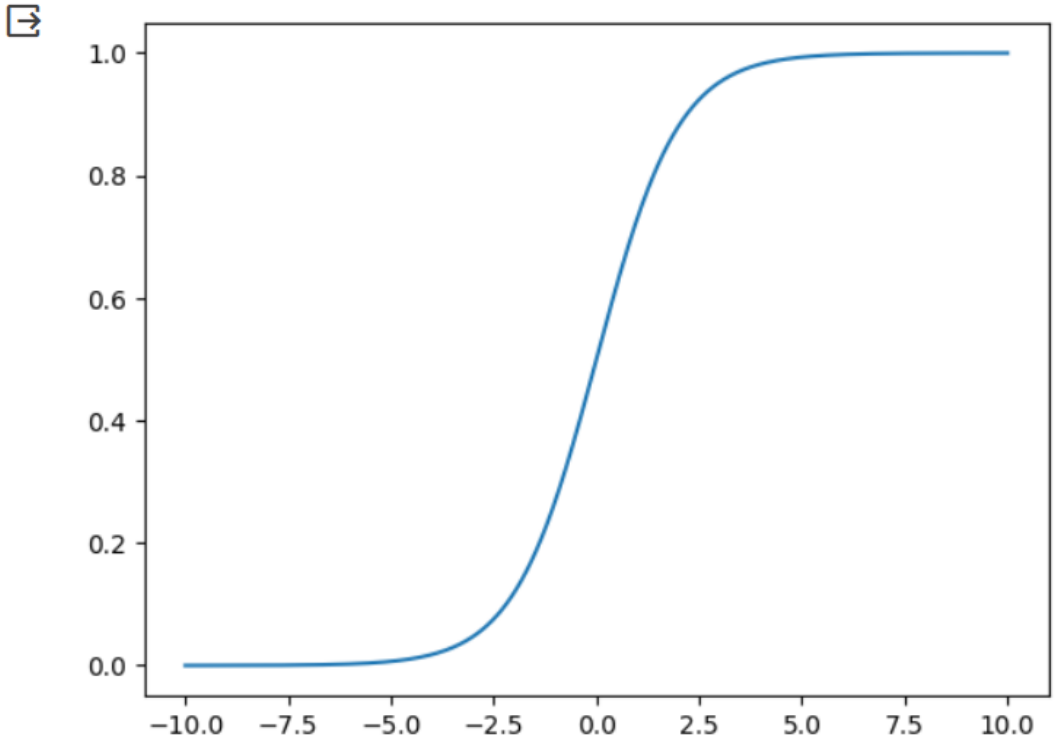
نتیجه طبق گزارش این بلوک کد، موفقیت آمیز بود:

```
# test your function against the sigmoid activation function from torch
result_sigmoid, output_sigmoid = test(test_data_torch, my_sigmoid , nn.Sigmoid())
print(result_sigmoid)
```

```
True
```

در نهایت نمودار تابع سیگوید را با اجرای بلوک بعدی رسم کردم که بازه آن به درستی بین 0 تا 1 میباشد:

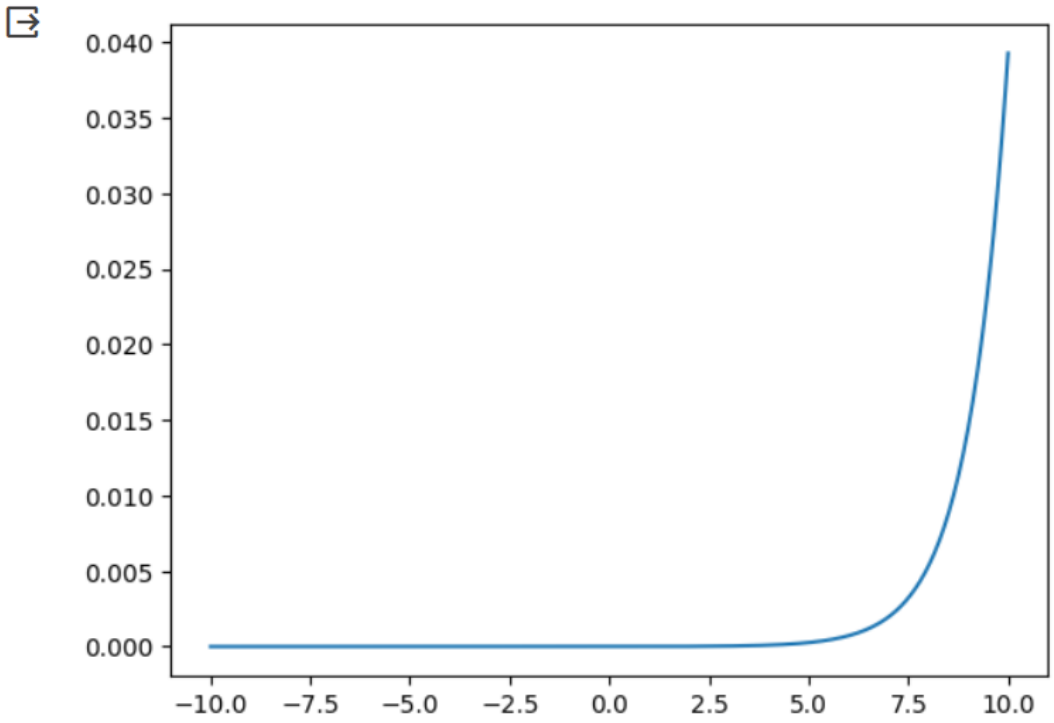
```
# plot the sigmoid activation function
plot_function(test_data_np, output_sigmoid)
```



برای پیاده سازی تابع softmax هم از np استفاده کردم و نتیجه آن صحیح بود:

```
[ ]  # implement the softmax activation function
     ##### TO DO #####
     def my_softmax(x):
       return np.exp(x) / np.exp(x).sum()
```

نمودار رسم شده برای softmax:

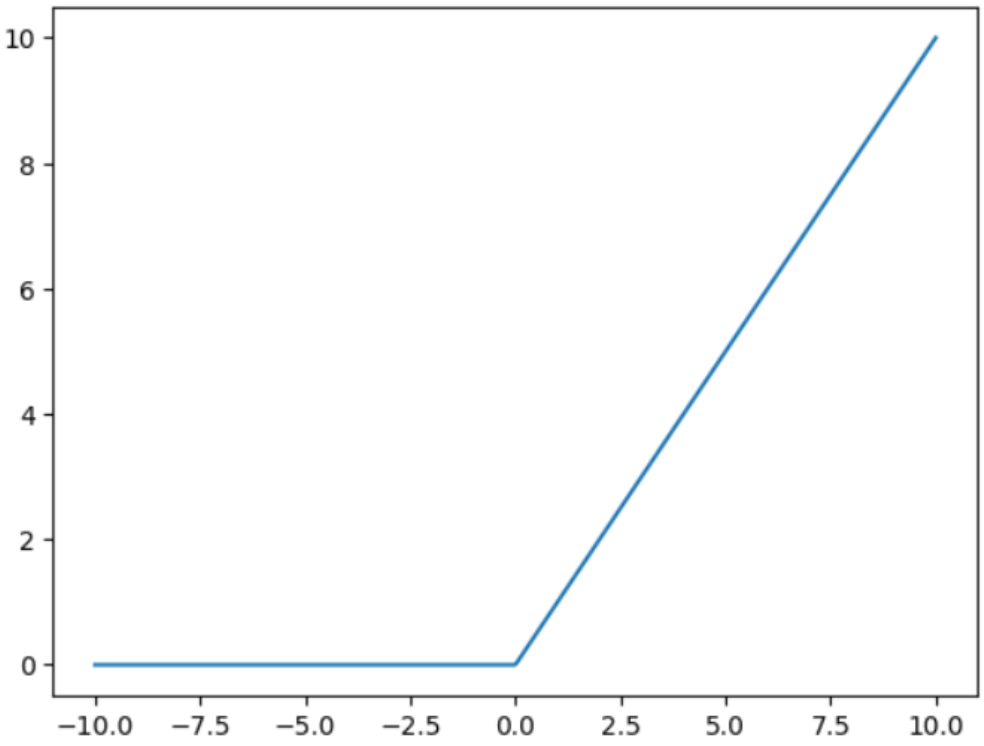


میبینیم که مانند مقادیر احتمال، ورودی های کوچک تر خروجی نزدیک به صفر و بزرگترین ورودی ها خروجی‌های نزدیک به 1 دارند. همچنین به صورت تقریبی می‌توان مشاهده کرد که مساحت زیر نمودار یعنی جمع همه احتمالات، برابر 1 است که درست میباشد.

برای تابع ReLU از np.maximum استفاده کردم:

```
]  # implement the ReLU activation function
   ##### TO DO #####
   def my_relu(x):
     return np.maximum(0, x)
```

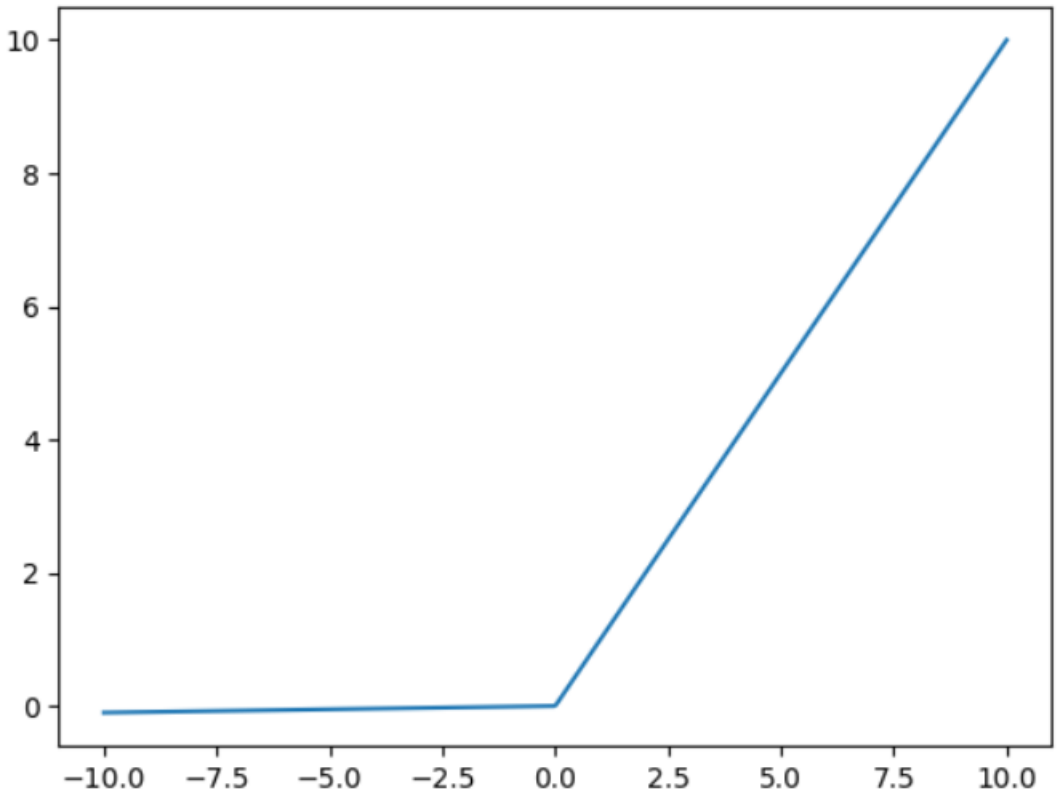
نتیجه رسم نمودار:



برای پیاده سازی Leaky ReLU هم تنها در قسمت منفی، به جای صفراز alpha * x استفاده کردم.

```
# implement the Leaky ReLU activation function
##### TO DO #####
def my_leakyrelu(x, alpha=0.01):
  return np.maximum(alpha * x, x)
```

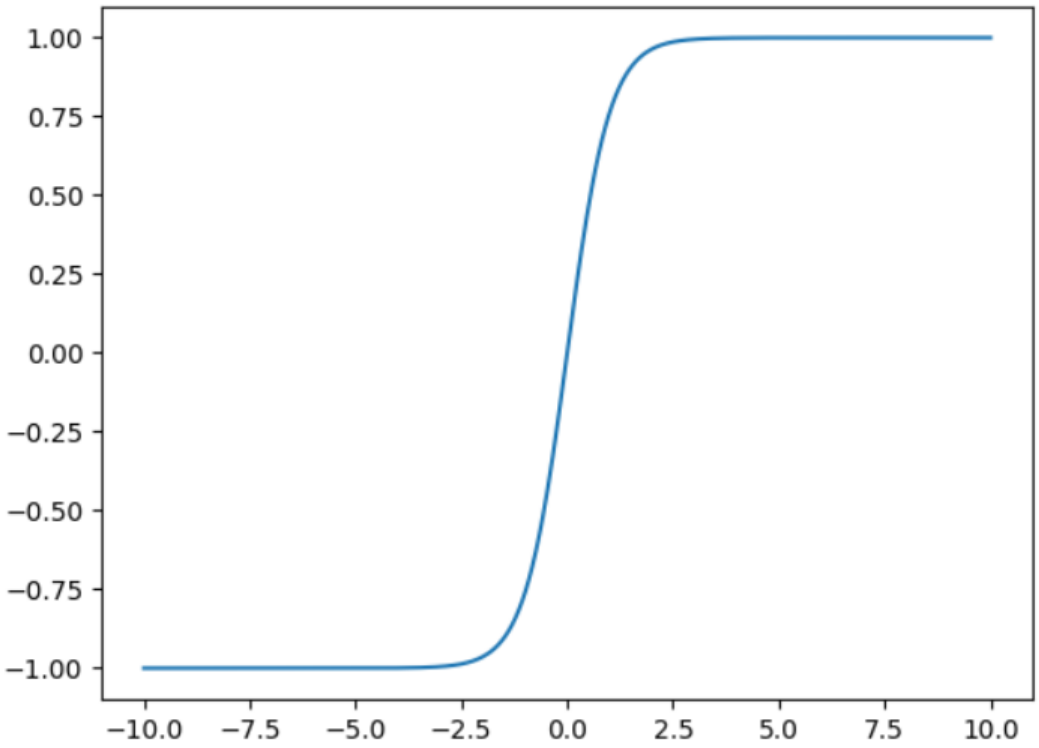
مقدار alpha در تابع nn.LeakyReLU، 0.01 است. بنابراین من هم از مقدار 0.01 به صورت پیشفرض استفاده کردم. نمودار Leaky ReLU به عت کوچک بودن alpha، شبیه ReLU شده. اگر ضریب alpha را بیشتر کنیم، راحت تر می‌توان تفاوت این دو تابع را در نمودار متوجه شد:



تابع tanh را هم مانند توابع دیگر به کمک np.exp پیاده سازی کردم:

```
[ ]  # implement the tanh activation function
     ##### TO DO #####
     def my_tanh(x):
       return (np.exp(x) - np.exp(-1 * x)) / (np.exp(-1 * x) + np.exp(x))
```

نمودار آن به این صورت شد که میبینیم که شکلی شبیه sigmoid دارد با این تفاوت که بازه آن، از -1 تا 1 است که از زیگزاگی شدن حرکت loss به سمت نقطه minimum جلوگیری میکند:



ج)

- تعداد لایه‌ها: برای تشخیص این سه ورودی از همدیگر، نیاز به مدل پیچیده‌ای نخواهیم داشت. من از سه لایه برای ساخت مدل استفاده کردم. لایه اول، وروی است. لایه سوم خروجی، و لایه میانی برای کمی قدرت و انعطاف پذیری بیشتر دادن به مدل میباشد.
- تعداد نورون‌های هر لایه: لایه اول لایه ورودی است و تصاویر ورودی 8*8 هستند. بنابراین، در لایه ورودی از 64 نورون استفاده کردم. ورودی‌های ما به 3 دسته تقسیم می‌شوند. بنابراین در لایه خروجی از 3 نورون استفاده کردم تا هر نورون احتمال مربوط به هر کلاس را پیشبینی کند. در لایه میانی هم به صورت تقریبی و تجربی از 10 نورون استفاده کردم.
- تابع فعالسازی هر لایه: برای لایه دوم از تابع فعالسازی ReLU استفاده کردم. تابع ReLU در اکثر مواقع انتخاب مناسبی است و در این مورد هم این تابع را تست کردم و نتیجه خوبی داشت. برای لایه بعدی هم از تابع softmax استفاده کردم زیرا یک دسته بندی چند کلاسه را انجام میدهیم و در خروجی نیاز به تابعی مانند softmax داریم که خروجی های هر نورون تبدیل به احتمال کند.
- تابع ضرر: با توجه به این که مسئله ما، دسته بندی چند کلاسه هست، از تابع CrossEntropy استفاده کردم.

د)

مدل شرح داده شده را به کمک pytorch، ایجاد کردم. برای نوشتن این کد از این ریپازیتوری کمک گرفتم.

ابتدا کتابخانه های لازم را وارد کردم:

```
import os
import torch
from torch import nn
# from torchvision.datasets import CIFAR10
from torch.utils.data import DataLoader
from torchvision import transforms
import numpy as np
```

سپس مدل MLP را طبق ساختار و لایه‌های شرح داده شده در قسمت قبل ایجاد کردم:

```
class MLP(nn.Module):
  '''
    Multilayer Perceptron.
  '''
  def __init__(self):
    super().__init__()
    self.layers = nn.Sequential(
      nn.Flatten(),
      nn.Linear(8 * 8, 10),
      nn.ReLU(),
      nn.Linear(10, 3),
      nn.Softmax()
    )


  def forward(self, x):
    '''Forward pass'''
    return self.layers(x)
```

سپس تصاویر مشخص شده در صورت سوال را به شکل کد درآوردم تا قابل پردازش باشد. کلاس تصاویر را به ترتیب از چپ به راست، 0 و 1 و 2 قرار دادم:

```
7] x1 = np.array([[0, 0, 0, 0, 0, 0, 0, 0,],
                  [0, 0, 0, 0, 0, 0, 0, 0,],
                  [0, 0, 0, 0, 0, 1, 1, 0,],
                  [0, 0, 1, 0, 1, 0, 0, 1,],
                  [1, 0, 1, 1, 1, 1, 1, 1,],
                  [1, 0, 1, 0, 0, 0, 0, 0,],
                  [1, 1, 1, 0, 0, 0, 0, 0,],
                  [0, 0, 0, 0, 0, 0, 0, 0,]], dtype=np.float32)
  y1 = 0

  x2 = np.array([[0, 0, 0, 0, 0, 1, 0, 1,],
                 [0, 0, 0, 0, 1, 0, 1, 0,],
                 [0, 0, 0, 0, 0, 1, 0, 0,],
                 [1, 0, 0, 0, 1, 0, 0, 0,],
                 [1, 1, 1, 1, 1, 0, 0, 0,],
                 [0, 0, 0, 0, 0, 0, 0, 0,],
                 [0, 0, 0, 0, 0, 0, 0, 0,],
                 [0, 0, 0, 0, 0, 0, 0, 0,]], dtype=np.float32)
  y2 = 1

  x3 = np.array([[0, 0, 0, 0, 0, 0, 0, 0,],
                 [0, 0, 0, 0, 0, 0, 0, 0,],
                 [0, 0, 0, 0, 0, 0, 0, 0,],
                 [0, 0, 1, 0, 0, 0, 0, 1,],
                 [0, 0, 1, 1, 1, 1, 1, 1,],
                 [0, 0, 0, 0, 0, 0, 0, 0,],
                 [0, 0, 0, 0, 1, 0, 0, 0,],
                 [0, 0, 0, 0, 0, 0, 0, 0,]], dtype=np.float32)
  y3 = 2
```

سپس دیتاست را با این نمونه ها ایجاد کردم و از یک dataloader برای enumerate کردن داده ها استفاده کردم:

```
dataset = [(x1, y1), (x2, y2), (x3, y3)]
trainloader = torch.utils.data.DataLoader(dataset, batch_size=10, shuffle=True, num_workers=1)
```

سپس مدل از کلاس طراحی شده، یک آبجکت ساختم تا عملیات train را روی آن انجام دهم:

```
torch.manual_seed(27)
mlp = MLP()
```

همچنین تابع loss و روش optimization و تعداد epoch ها را تعیین کردم. چون تعداد نمونه ها بسیار کم بود، تعداد epoch ها را بزرگ گذاشتم تا learning به خوبی انجام شود:

```
loss_function = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(mlp.parameters(), lr=1e-4)
epochs = 1000
```

در نهایت در یک حلقه، فرایند آموزش را طبق manual های pytorch انجام دادم:

```
for epoch in range(epochs):
  current_loss = 0.0
  for i, data in enumerate(trainloader, 0):
    inputs, targets = data
    # Zero the gradients
    optimizer.zero_grad()

    # Perform forward pass
    outputs = mlp(inputs)

    # Compute loss
    loss = loss_function(outputs, targets)

    # Perform backward pass
    loss.backward()

    # Perform optimization
    optimizer.step()

    # Print statistics
    current_loss += loss.item()
  if epoch % 100 == 0:
    print(f"loss after epoch {epoch} = {current_loss}")
    current_loss = 0.0
```

و آخر هر epoch، loss را پرینت کردم تا پیشرفت مدل قابل مشاهده باشد. نتایج به این صورت بود:

```
    current_loss = 0.0
```

```
loss after epoch 0 = 1.1124707460403442
loss after epoch 100 = 1.0937315225601196
loss after epoch 200 = 1.0711406469345093
loss after epoch 300 = 1.0450184345245361
loss after epoch 400 = 1.0183435678482056
loss after epoch 500 = 0.99102383852005
loss after epoch 600 = 0.9621233344078064
loss after epoch 700 = 0.9320527911186218
loss after epoch 800 = 0.9014608263969421
loss after epoch 900 = 0.870947539806366
```

میبینیم که مقدار loss به مرور کاهش پیدا می‌کند و یادگیری مدل به درستی انجام می‌شود.

سپس پیشبینی مدل را انجام دادم:

```python
for i, data in enumerate(trainloader, 0):
    inputs, targets = data
    output = mlp(inputs)
    for i in range(len(output)):
      print(f"label = {targets[i]} \t output = {output[i]}")
```

```
label = 2        output = tensor([0.2508, 0.1114, 0.6378], grad_fn=<SelectBackward0>)
label = 1        output = tensor([0.3333, 0.5749, 0.0918], grad_fn=<SelectBackward0>)
label = 0        output = tensor([0.6270, 0.1984, 0.1747], grad_fn=<SelectBackward0>)
```

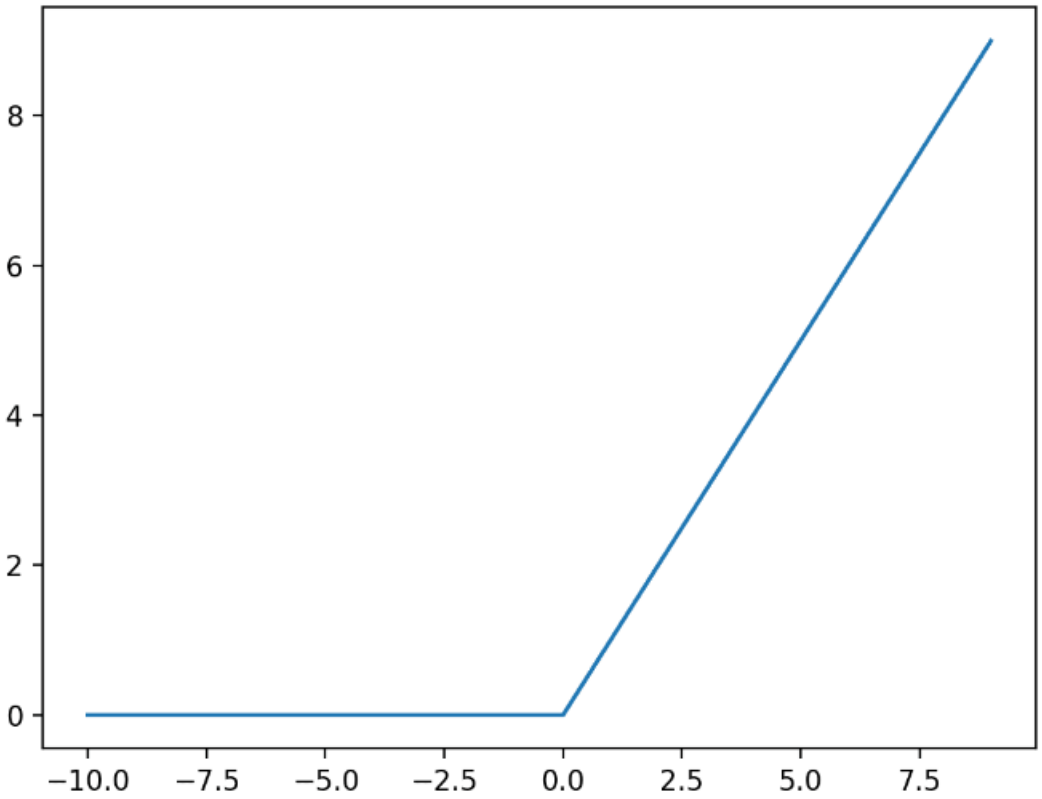
خروجی مدل را به جای شماره کلاس به صورت احتمالات هر کلاس چاپ کردم تا جزعیات بیشتری از مدل متوجه شویم.

میبینیم که دقت مدل 100٪ شده زیرا برای هر سه نمونه، احتمالی که برای کلاس صحیح پیشبینی کرده از سایر احتمالات بیشتر است. اما میبینیم که همچنان مدل جای یادگیری بیشتر و کمتر کردن loss دارد زیرا اختلاف احتمال کلاس صحیح را با سایر کلاس ها میتوان خیلی بیشتر کرد. همچنین این فقط 3 نمونه است و مدل روی همین 3 نمونه هم train شده و نمیتوان مدل را روی این 3 نمونه تست کرد. برای تست بهتر مدل باید نمونه های دیگری در اختیار داشته باشیم که مدل روی آنها train نشده.

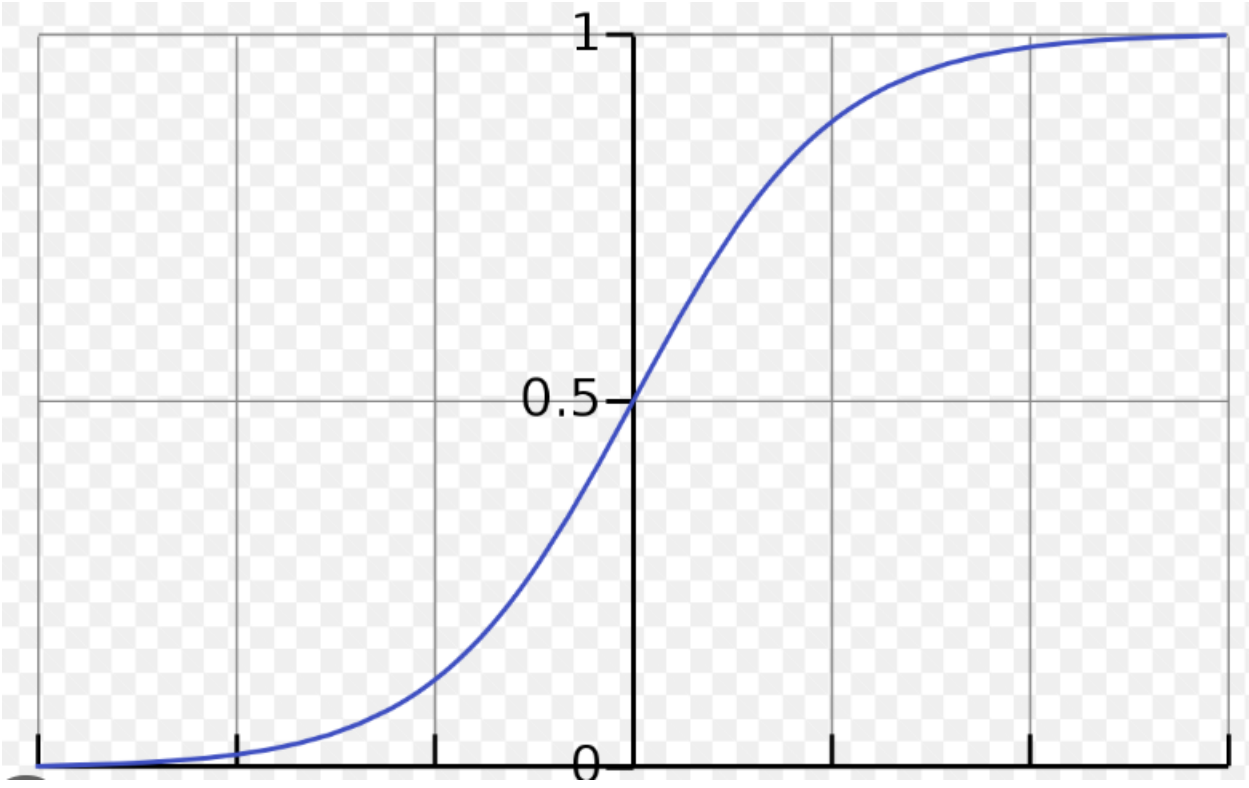
Q6)

مشکل استفاده از ReLU قبل از sigmoid این است که مقدار z، میتواند منفی یا مثبت باشد اما با اعمال ReLU، مقادیر قبل از صفر از بین میروند. این تغییر اطلاعات، احتمال را از توزیع نرمال خارج میکند و باعث میشود که مدل در یادگیری دچار مشکل شود.

اما مشکل مهم ترمقدار آستانه برابر با 0.5 است. تابع ReLU تمام مقادیر کم تر از صفر را تبدیل به 0 میکند:



سپس این مقادیر وارد تابع sigmoid میشوند. نمودار تابع sigmoid به این صورت است:



میبینیم که تابع sigmoid به ازای مقادیر مثبت بزرگ تر از 0.5 است. وقتی داده از از تابع ReLU عبور میکنند همگی بزرگتر یا مساوی صفر میشوند و وقتی این مقادیر وارد sigmoid میشوند، همواره مقدار خروجی sigmoid، از حد آستانه بالا تر است و این به این معناست که این مدل هیچگاه کلاس 0 را پیشبینی نمیکند.

Q7)

الف)

یادگیری ماشین (Machine Learning) و یادگیری عمیق (Deep Learning) هر دو مفاهیم مرتبط با هوش مصنوعی هستند، اما تفاوت‌هایی در روش‌ها و مفهوم اصلی آن‌ها وجود دارد.

یادگیری ماشین به مجموعه‌ای از الگوریتم‌ها و تکنیک‌ها اشاره دارد که به کامپیوتر امکان می‌دهد از داده‌ها یاد بگیرد و بر اساس آن‌ها پیش‌بینی‌ها و تصمیماتی را انجام دهد. در یادگیری ماشین، معمولاً از رویکردهای آماری و احتمالاتی برای مدل‌سازی داده‌ها و استنتاج استفاده می‌شود. الگوریتم‌های یادگیری ماشین، بر اساس استخراج ویژگی‌های معنادار از داده‌ها، مدلی را برای پیش‌بینی و تصمیم‌گیری ایجاد می‌کنند.

به عنوان مقابل، یادگیری عمیق به یک زیرمجموعه از یادگیری ماشین اشاره دارد که بر اساس شبکه‌های عصبی مصنوعی با تعداد بسیار زیادی لایه عمل می‌کند. شبکه‌های عصبی عمیق قادر به خودآموزش و استخراج ویژگی‌های همانند انسان هستند و برای بسیاری از وظایف پیچیده تر و داده‌های بزرگ مناسب هستند. با استفاده از لایه‌های عمیق، شبکه‌های عصبی عمیق قادر به یادگیری نمایش‌های سلسله مراتبی از داده‌ها هستند، که این نمایش‌ها می‌توانند ویژگی‌ها و الگوهای پیچیده‌تر را مدل کنند.

بنابراین، مهمترین تفاوت بین یادگیری ماشین و یادگیری عمیق در استفاده از شبکه‌های عصبی عمیق است. در حالی که یادگیری ماشین ممکن است از الگوریتم‌های مختلفی مانند درخت تصمیم، ماشین بردار پشتیبان و غیره استفاده کند، یادگیری عمیق به طور اصلی بر روی شبکه‌های عصبی عمیق تمرکز دارد که قدرت بالاتری در مدل‌سازی و استخراج ویژگی‌ها دارند.

جواب به کمک ChatGPT

ب)

پاسخ به این سوال به عوامل مختلفی بستگی دارد، از جمله نوع داده‌های آموزش، معماری شبکه و هدف طبقه‌بندی.

به طور کلی، لایه‌های عمیق‌تر شبکه‌های یادگیری عمیق معمولاً اطلاعات بیشتری در مورد داده‌ها دارند. این به این دلیل است که لایه‌های عمیق‌تر به ویژگی‌های پیچیده‌تری از داده‌ها دسترسی دارند که در لایه‌های کم‌عمق‌تر قابل شناسایی نیستند.

بنابراین، اگر هدف طبقه‌بندی یک کار دشوار باشد که نیاز به شناسایی ویژگی‌های پیچیده‌ای از داده‌ها دارد، لایه‌های عمیق‌تر معمولاً عملکرد بهتری خواهند داشت.

با این حال، لایه‌های عمیق‌تر همچنین می‌توانند منجر به overfitting شوند .زمانی اتفاق می‌افتد که شبکه به قدری به داده‌های آموزش خود تنظیم می‌شود که دیگر نمی‌تواند داده‌های جدید را به درستی طبقه‌بندی کند. در این صورت استفاده از لایه های قبل تر میتواند نتیجه بهتری را به همراه داشته باشد.

در مورد فرضی که مطرح کردید، اگر داده‌های آموزش پیچیده باشند و هدف طبقه‌بندی یک کار دشوار باشد، لایه ۱۱ ام احتمالاً عملکرد بهتری خواهد داشت. البته، این فقط یک فرض است و برای تعیین اینکه کدام لایه عملکرد بهتری دارد، باید آزمایش‌های بیشتری انجام شود.

پاسخ به کمک Bard

ج)

پاسخ به این سوال به عوامل مختلفی بستگی دارد، از جمله نوع تابعی که باید تقریب شود، حجم داده‌های آموزشی و منابع محاسباتی موجود.

به طور کلی، شبکه‌های عمیق‌تر معمولاً در تقریب توابع پیچیده‌تر کارآمدتر هستند. این به این دلیل است که شبکه‌های عمیق‌تر می‌توانند ویژگی‌های پیچیده‌تری از داده‌ها را شناسایی کنند که در شبکه‌های کم‌عمق‌تر قابل شناسایی نیستند.

با این حال، شبکه‌های عمیق‌تر همچنین می‌توانند منجر به overfitting شوند .زمانی اتفاق می‌افتد که شبکه به قدری به داده‌های آموزش خود تنظیم می‌شود که دیگر نمی‌تواند داده‌های جدید را به درستی تقریب کند. همچنین ممکن است مشکلات vanishing gradient و exploading gradient در شبکه‌های عمیق به وجود بیاید.

شبکه‌های عریض‌تر معمولاً در تقریب توابع ساده‌تر کارآمدتر هستند. این به این دلیل است که شبکه‌های عریض‌تر می‌توانند به طور مستقیم ویژگی‌های داده‌ها را یاد بگیرند، بدون اینکه نیاز به شناسایی ویژگی‌های پیچیده‌تر باشد.

پاسخ به کمک Bard

د)

افزودن لایه‌های بیشتر به یک شبکه عصبی عمیق مزایا و معایب خاص خود را دارد. در ادامه به برخی از مزایا و معایب افزودن لایه‌ها به شبکه عصبی عمیق اشاره می‌کنیم:

مزایا:

قدرت یادگیری بیشتر: افزودن لایه‌های بیشتر به شبکه عصبی می‌تواند قدرت یادگیری آن را افزایش دهد. هر لایه جدید به شبکه امکان یادگیری ویژگی‌های جدید را می‌دهد و درک بهتری از ساختار داده‌ها و الگوهای پیچیده‌تر را فراهم می‌کند.

توانایی نمایش سلسله مراتبی ویژگی‌ها: با افزودن لایه‌های بیشتر، شبکه قادر به یادگیری ویژگی‌های سلسله مراتبی از داده‌ها می‌شود. لایه‌های ابتدایی معمولاً ویژگی‌های ساده‌تری را استخراج می‌کنند و لایه‌های بالاتر به تدریج ویژگی‌های پیچیده‌تر و سطح بالاتر را نمایش می‌دهند.

انتقال یادگیری: شبکه‌های عصبی عمیق با لایه‌های بیشتر، قادر به انتقال یادگیری بهتری هستند. با استفاده از مدل‌های پیش‌آموزش دیده شده و انتقال وزن‌ها، می‌توان لایه‌های ابتدایی شبکه را با ویژگی‌های مفید از مسائل مشابه پیش‌آموزش داده و سپس لایه‌های بالاتر را برای ویژگی‌ها و الگوهای خاص مربوط به مسئله فعلی آموزش داد.

معایب:

پیچیدگی محاسباتی: افزودن لایه‌های بیشتر به شبکه می‌تواند پیچیدگی محاسباتی را افزایش دهد. این ممکن است منجر به زمان آموزش بیشتر و نیاز به منابع محاسباتی قوی‌تر شود.

overfitting: شبکه‌های عصبی عمیق با تعداد زیادی لایه، در مواردی ممکن است به overfitting دچار شوند. این به معنای یادگیری و تطبیق بیش از حد به داده‌های آموزشی است، که ممکن است باعث کاهش عملکرد شبکه در داده‌های جدید شود.

پارامترهای بیشتر: با افزودن لایه‌هایبیشتر، تعداد پارامترهای شبکه نیز افزایش می‌یابد. این به معنای نیاز به بیشترین حجم داده آموزشی و منابع محاسباتی برای آموزش و بهینه‌سازی شبکه است.

مشکل کاهش گرادیان: در شبکه‌های عمیق، ممکن است با مواجهه با مشکل کاهش گرادیان (vanishing/exploding gradients)روبرو شوید. این مشکل ممکن است باعث کاهش سرعت یادگیری و ناپایداری آموزش شبکه شود.

به طور کلی، افزودن لایه‌های بیشتر به شبکه عصبی عمیق می‌تواند بهبودهای قابل توجهی در یادگیری و نمایش داده‌ها به همراه داشته باشد. با این حال، در انتخاب تعداد لایه‌ها باید مواردی مانند مسئله مورد نظر، حجم داده، پیچیدگی مسئله و منابع محاسباتی را در نظر گرفت و به طور دقیق تعیین کرد که چند لایه برای مسئله مورد نظر بهینه است.

پاسخ به کمک ChatGPT