به نام خدا

دانشگاه صنعتی خواجه نصیرالدین طوسی

دانشکده برق

مبانی سیستم های هوشمند

گزارشکار مینی پروژه ۲

پارسا رشتیان
۴۰۰۱۰۸۲۳

استاد : آقای دکتر مهدی علیاری

دی ۱۴۰۳

پرسش ۱:

۱.

در یک مسألهٔ طبقه‌بندی دوکلاسه، انتخاب توابع فعال‌ساز در لایه‌های انتهایی بسیار مهم است، زیرا مستقیماً بر نحوه یادگیری مدل و عملکرد آن تأثیر می‌گذارد. حال، فرض کنید دو لایهٔ انتهایی شبکه به‌ترتیب از **ReLU** و **سیگموید** استفاده کنند. بررسی می‌کنیم چه اتفاقی می‌افتد:

رفتار لایه با فعال‌ساز ReLU:

ReLU (Rectified Linear Unit) معمولاً در لایه‌های مخفی استفاده می‌شود، زیرا به حل مشکل ناپدید شدن گرادیان کمک می‌کند. خروجی آن به‌صورت زیر است:

$$f(x) = \begin{cases} x, & \text{if } x > 0 \\ 0, & \text{if } x \leq 0 \end{cases}$$

این تابع معمولاً در لایهٔ انتهایی شبکه‌های طبقه‌بندی دوکلاسه استفاده می‌شود تا احتمال تعلق به یک کلاس را تولید کند.

رفتار لایه با فعال‌ساز سیگموید:

سیگموید یک تابع فعال‌ساز غیرخطی است که خروجی آن در بازهٔ (۰, ۱) قرار می‌گیرد:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

این تابع معمولاً در لایهٔ انتهایی شبکه‌های طبقه‌بندی دوکلاسه استفاده می‌شود تا احتمال تعلق به یک کلاس را تولیدکند.

ترکیب ReLU و سیگموید:

حال، اگر خروجی لایهٔ ReLU به لایهٔ سیگموید داده شود:

۱.**بزرگی خروجی ReLU**: ReLU مقادیر مثبت بزرگ تولید می‌کند. این مقادیر بزرگ ممکن است باعث اشباع سیگموید شوند.

وقتی مقدار ورودی سیگموید خیلی بزرگ باشد (مثلاً x >> 1)، خروجی سیگموید به ۱ نزدیک می‌شود.

وقتی مقدار ورودی سیگموید صفر یا کوچک باشد (مثلاً x <= 0)، خروجی سیگموید به ۰ نزدیک می‌شود.

در نتیجه، سیگموید فقط مقادیر بسیار کوچک یا بسیار نزدیک به ۱ تولید می‌کند و شبکه ممکن است نتواند تفاوت‌های ظریف بین کلاس‌ها را تشخیص دهد.

۲.**مشکل گرادیان**:

وقتی سیگموید در نواحی اشباع (ورودی‌های خیلی بزرگ یا خیلی کوچک) عمل می‌کند، گرادیان آن به صفر نزدیک می‌شود. این امر باعث **کندی یادگیری** یا حتی **توقف به‌روزرسانی وزن‌ها** می‌شود.

۳.**عدم تناسب نقش ReLU**:

از آنجایی که ReLU برای لایه‌های مخفی طراحی شده و برای یادگیری ویژگی‌های پیچیده مناسب است، استفاده از آن در لایهٔ قبل از سیگموید (در انتهای شبکه) **غیرمنطقی است**.
نتیجه‌گیری

مشکل اصلی: ترکیب ReLU و سیگموید در لایه‌های انتهایی شبکه می‌تواند منجر به اشباع سیگموید، کاهش گرادیان، و در نهایت ضعف در عملکرد شبکه شود.

پیشنهاد: برای مسألهٔ طبقه‌بندی دوکلاسه:

۱.اگر از **سیگموید** در لایهٔ انتهایی استفاده می‌کنید، نیازی به ReLU در لایهٔ قبل از آن نیست.

۲.معمولاً بهتر است از **فعال‌ساز خطی (بدون تغییر)** یا **تابع Softmax** استفاده کنید، به‌ویژه اگر از **Cross-Entropy Loss** برای آموزش استفاده می‌کنید.

۲.
در معادلهٔ (۱)، تابع **ELU (Exponential Linear Unit)** معرفی شده است که به‌عنوان یک جایگزین برای **ReLU** استفاده می‌شود. تابع ELU به‌صورت زیر تعریف شده است:

گرادیان تابع ELU

برای محاسبهٔ گرادیان این تابع:

۱. **زمانی که** x >= ۰:

$$\frac{d}{dx} ELU(x) = \frac{d}{dx} x = 1$$

۲. زمانی که x < ۰:

$$\frac{d}{dx} ELU(x) = \frac{d}{dx} [\alpha(e^x - 1)] = \alpha e^x$$

مزیت ELU نسبت به ReLU

ELU نسبت به ReLU حداقل یک مزیت مهم دارد:

۱.**مدیریت مقادیر منفی:**

در ReLU، مقادیر منفی به صفر نگاشت می‌شوند، که می‌تواند باعث **مردگی نورون‌ها** (**Dead Neurons**) شود. در ELU، مقادیر منفی با استفاده از عبارت زیر نگه‌داری می‌شوند:

$$\alpha(e^x - 1)$$

و مقدار خروجی به صفر نزدیک می‌شود، اما کاملاً صفر نمی‌شود.

این رفتار باعث می‌شود که شبکه بتواند اطلاعات بیشتری از ورودی‌های منفی را حفظ کند.

۲. **پیوستگی گرادیان:**

گرادیان ELU در نقطهٔ $x = 0$ **پیوسته** است، زیرا در هر دو حالت $x >= 0$ و $x < 0$، گرادیان به‌آرامی تغییر می‌کند. این ویژگی باعث می‌شود که آموزش شبکه روان‌تر انجام شود.

در ReLU، گرادیان در $x = 0$ ناپیوسته است، که ممکن است روی همگرایی مدل تأثیر منفی بگذارد.

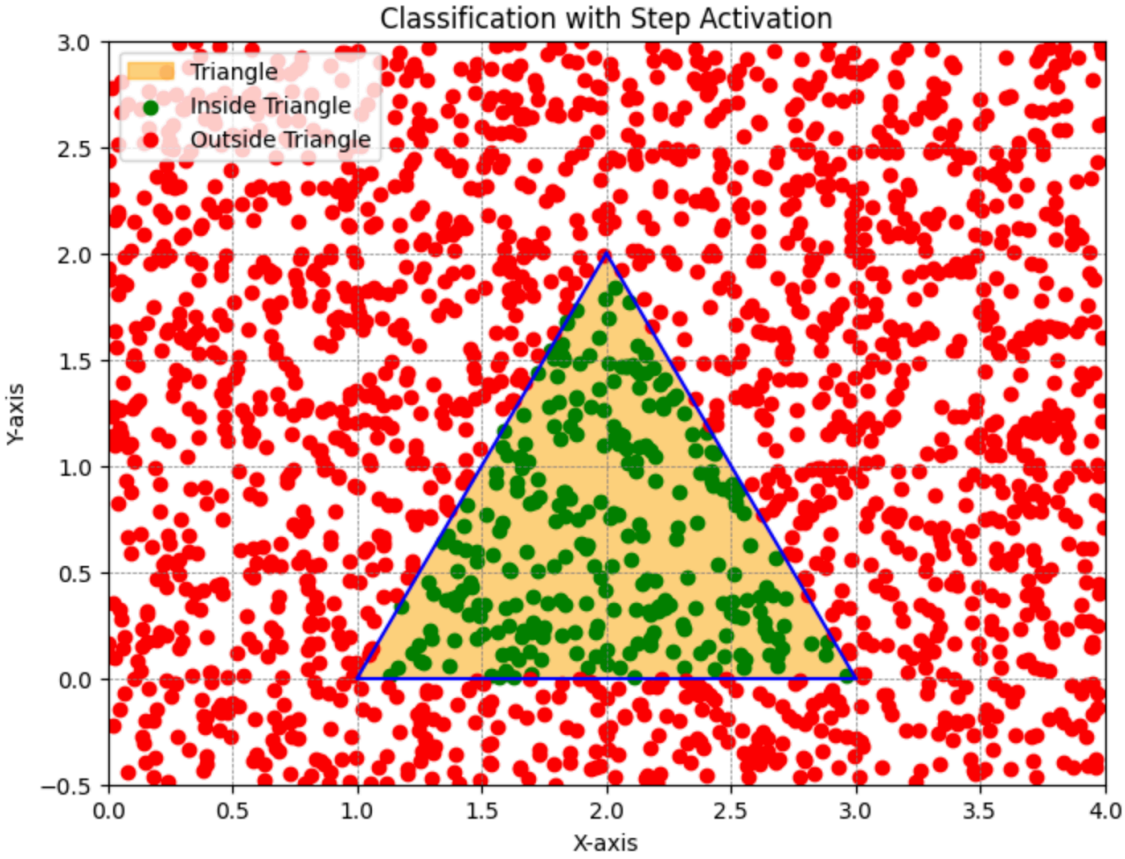
۳.**کمک به همگرایی سریع‌تر:**

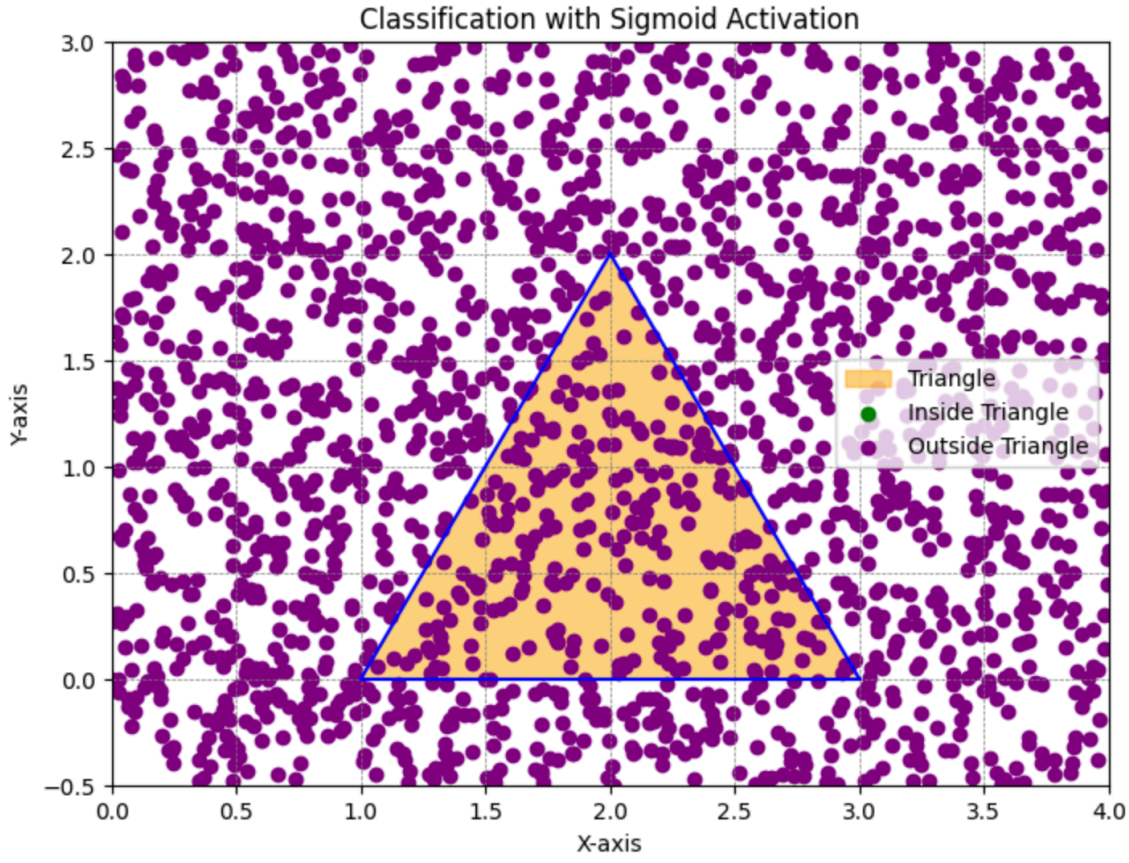
ELUخروجی‌های منفی تولید می‌کند، که میانگین خروجی لایه را به صفر نزدیک‌تر می‌کند. این خاصیت می‌تواند به کاهش پدیده **انباشت‌پذیری گرادیان‌ها** (**Vanishing Gradients**) کمک کند و سرعت یادگیری را افزایش دهد.

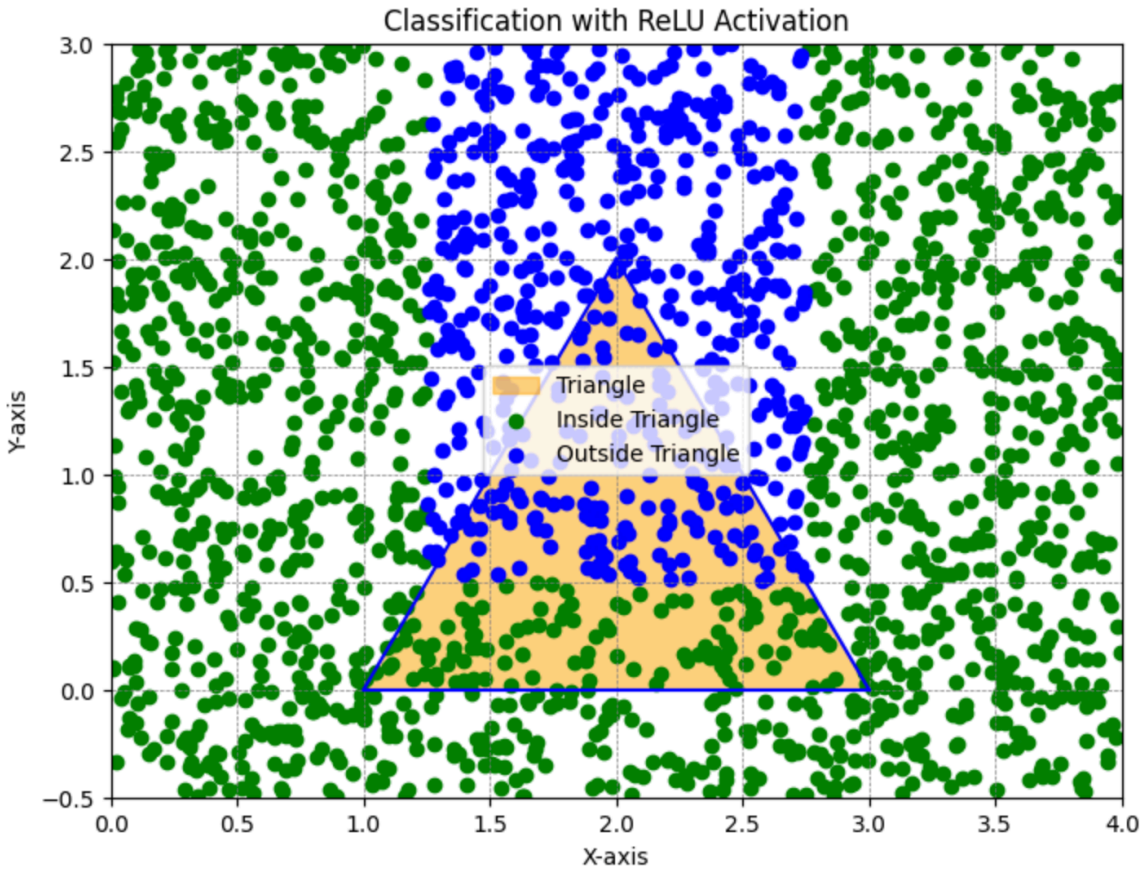
نتیجه‌گیری

ELUیک جایگزین مناسب برای ReLU است، به‌ویژه در مسائلی که مقادیر منفی ورودی اهمیت دارند یا مشکل مردگی نورون‌ها رخ می‌دهد. با این حال، مقدار alphaباید با دقت انتخاب شود تا عملکرد بهینه به‌دست آید.

۳.


Classification with Step Activation
Triangle
Inside Triangle
Outside Triangle
Y-axis
X-axis
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0


Classification with Sigmoid Activation
Triangle
Inside Triangle
Outside Triangle
Y-axis
X-axis
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0



همانطور که می‌بینیم در این مسئله که یک شبکه‌ی حاوی یک پرسپترون یا نورون ساده وظیفه‌ی کلاسیفیکیشن را بر عهده دارد با استفاده از step actvation به نتیجه‌ی مطلوب رسیدیم. به علت سادگی شبکه و نوع خروجی شبکه که باینری است (۰ و ۱) در صورت استفاده از activation function هایی همانند ReLU و سیگموید خروجی به هم می‌خورد. این به علت این است که سیگموید عددی بین ۰ و۱ را برای خروجی در نظر می‌گیرد و هیچوقت به مقدار دقیق ۰ و ۱ دست پیدا نمی‌کند. همچنین در دیگر تابع فعال‌ساز یعنی ReLU ما هیچوقت به صفر و یک مطلق نمی‌رسیم. این برای یک کلاسیفیکیشن باینری مناسب نیست و باعث پیش‌بینی اشتباه می‌شود که در نمودارهای بالا کاملا مشخص است.

پرسش ۲:

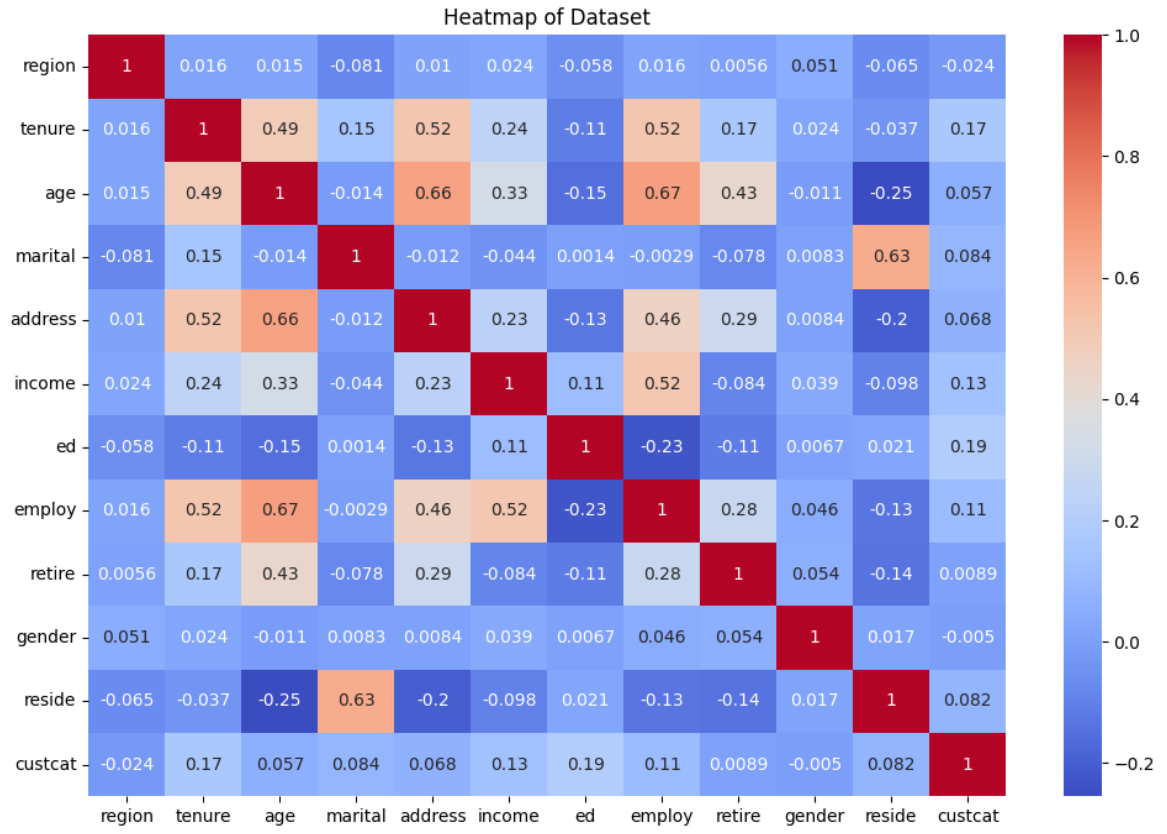
۱.

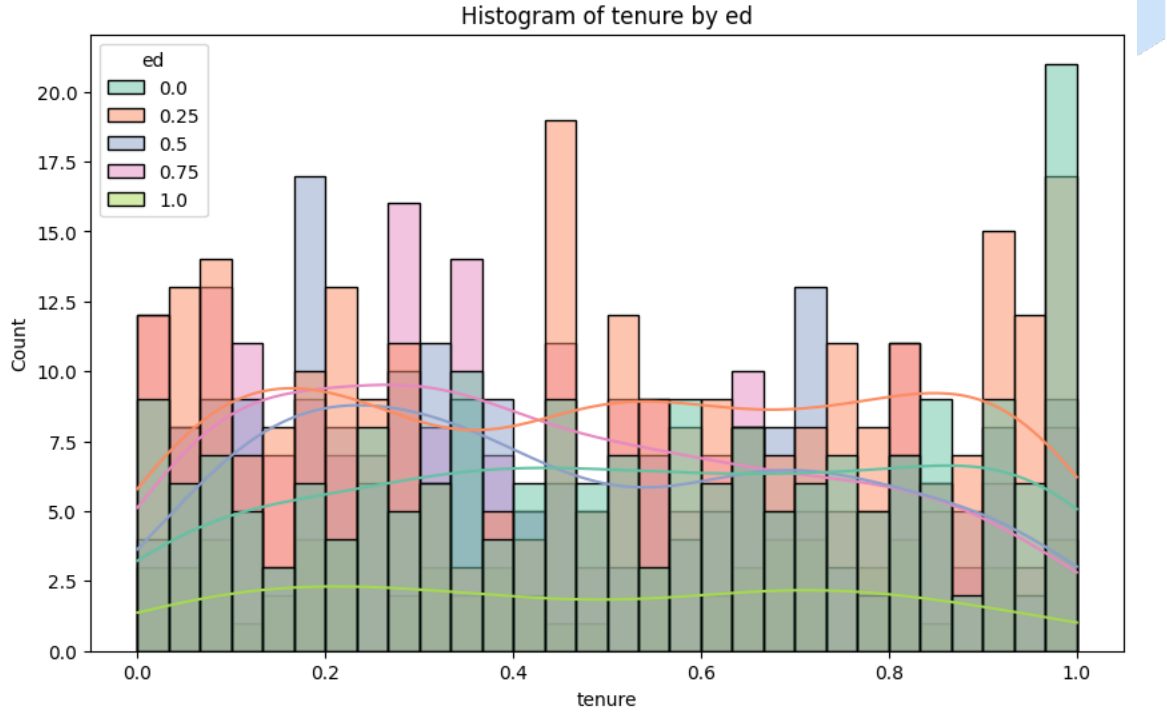
```
# Load the dataset
data = pd.read_csv('/content/drive/My Drive//teleCust1000t.csv')

# Inspect the first few rows
print(data.head())
```

```
   region  tenure  age  marital  address  income  ed  employ  retire
gender  \
0       2      13   44        1        9    64.0   4       5     0.0
0
1       3      11   33        1        7   136.0   5       5     0.0
0
2       3      68   52        1       24   116.0   1      29     0.0
1
3       2      33   33        0       12    33.0   2       0     0.0
1
4       2      23   30        1        9    30.0   1       2     0.0
0

    reside  custcat
0        2        1
1        6        4
2        2        3
3        1        1
```

همانطور که مشاهده می‌شود ویژگی‌های مختلفی که در این داده وجود دارد را استخراج کردیم.

.۲


Heatmap of Dataset
region 1 0.016 0.015 -0.081 0.01 0.024 -0.058 0.016 0.0056 0.051 -0.065 -0.024
tenure 0.016 1 0.49 0.15 0.52 0.24 -0.11 0.52 0.17 0.024 -0.037 0.17
age 0.015 0.49 1 -0.014 0.66 0.33 -0.15 0.67 0.43 -0.011 -0.25 0.057
marital -0.081 0.15 -0.014 1 -0.012 -0.044 0.0014 -0.0029 -0.078 0.0083 0.63 0.084
address 0.01 0.52 0.66 -0.012 1 0.23 -0.13 0.46 0.29 0.0084 -0.2 0.068
income 0.024 0.24 0.33 -0.044 0.23 1 0.11 0.52 -0.084 0.039 -0.098 0.13
ed -0.058 -0.11 -0.15 0.0014 -0.13 0.11 1 -0.23 -0.11 0.0067 0.021 0.19
employ 0.016 0.52 0.67 -0.0029 0.46 0.52 -0.23 1 0.28 0.046 -0.13 0.11
retire 0.0056 0.17 0.43 -0.078 0.29 -0.084 -0.11 0.28 1 0.054 -0.14 0.0089
gender 0.051 0.024 -0.011 0.0083 0.0084 0.039 0.0067 0.046 0.054 1 0.017 -0.005
reside -0.065 -0.037 -0.25 0.63 -0.2 -0.098 0.021 -0.13 -0.14 0.017 1 0.082
custcat -0.024 0.17 0.057 0.084 0.068 0.13 0.19 0.11 0.0089 -0.005 0.082 1
region tenure age marital address income ed employ retire gender reside custcat
1.0 0.8 0.6 0.4 0.2 0.0 −0.2


Histogram of tenure by ed
ed
0.0
0.25
0.5
0.75
1.0
Count
20.0 17.5 15.0 12.5 10.0 7.5 5.0 2.5 0.0
0.0 0.2 0.4 0.6 0.8 1.0
tenure

۳. نشان‌دهنده‌ی اینکه نرمالیزیشن با موفقیت انجام شده:

```
Min value in scaled train data: 0.0
Max value in scaled train data: 1.0
Mean value in scaled train data: 0.3192609017240444
Standard deviation of scaled train data: 0.3562674854627224
```

۴. طبق خواسته‌ی سوال مدل اول را یک mlp با یک لایه‌ی پنهان و مدل دوم را یک mlp با دولایه‌ی پنهان قرار می‌دهیم.

```
Accuracy for model1 with 10 neurons: 0.3667
Accuracy for model2 with 10 neurons: 0.3133


Accuracy for model1 with 30 neurons: 0.4000
Accuracy for model2 with 30 neurons: 0.3600


Accuracy for model1 with 70 neurons: 0.4000
Accuracy for model2 with 70 neurons: 0.3733
```

هیچکدام از نتایج آورده شده نتیجه‌ی مطلوبی نیست. درصد صحت بسیار پایین است.

با این حال مشاهده می‌شود تعداد نورون‌ها در نتیجه تاثیر دارد. اما باید مراقب این باشیم که تعداد نورون‌ها از حدی بیشتر نشود که پیچیدگی بسیار زیاد شود و مدل overfit شود. اما در این سه حالت مشاهده می‌کنیم که ۷۰ نورون نتیجه‌ی مناسب‌تری برای ما به ارمغان می‌آورد.

```
Evaluating Model 1 on test set with BatchNormalization and without
Dropout...
Model 1 Accuracy: 0.4000

Evaluating Model 2 on test set with BatchNormalization and without
Dropout...
Model 2 Accuracy: 0.3467

Evaluating Model 1 on test set without BatchNormalization and without
Dropout...
Model 1 Accuracy: 0.4000

Evaluating Model 2 on test set without BatchNormalization and without
Dropout...
Model 2 Accuracy: 0.3067

Evaluating Model 1 on test set with BatchNormalization and with
Dropout...
Model 1 Accuracy: 0.3733
```

```
Evaluating Model 2 on test set with BatchNormalization and with
Dropout...
Model 2 Accuracy: 0.3867
Evaluating Model 1 on test set without BatchNormalization and with
Dropout...
Model 1 Accuracy: 0.3400

Evaluating Model 2 on test set without BatchNormalization and with
Dropout...
Model 2 Accuracy: 0.3067
```

همینطور که در این ۴ حالت می‌بینیم اولین حالت یعنی مدل با BatchNormalization و بدون دراپ اوت نتیجه‌ی نسبتا بهتری نسبت به حالت‌های دیگر داشته است. درنتیجه استفاده از دراپ‌اوت اینجا فایده‌ای ندارد و به مدلمان کمکی نمی‌کند.

در نهایت با استفاده از L۲-Regularization با نرخ ۰.۰۰۰۱ به نتایج زیر دست پیدا می‌کنیم:

```
Evaluating Model 1 on test set with BatchNormalization, without
Dropout, L2=0.0001, rmsprop optimizer...
Model 1 Accuracy: 0.3733

Evaluating Model 2 on test set with BatchNormalization, without
Dropout, L2=0.0001, rmsprop optimizer...
Model 2 Accuracy: 0.4200
```

مدل دوم بهترین دقت را بین مدل‌هایی که تا به حال تست کردیم به ما داد.

۵.

```
Model: rmsprop optimizer, Dropout Rate: 0.0, BatchNorm: with
BatchNormalization
Sample 1:
  - Actual class: 2
  - Predicted class: 3
---------------------------------------
Sample 2:
  - Actual class: 0
  - Predicted class: 2
---------------------------------------
Sample 3:
  - Actual class: 2
  - Predicted class: 3
---------------------------------------
Sample 4:
  - Actual class: 0
  - Predicted class: 3
---------------------------------------
```

```
Sample 5:
  - Actual class: 3
  - Predicted class: 2
----------------------------------------
Sample 6:
  - Actual class: 2
  - Predicted class: 3
----------------------------------------
Sample 7:
  - Actual class: 1
  - Predicted class: 2
----------------------------------------
Sample 8:
  - Actual class: 2
  - Predicted class: 3
----------------------------------------
Sample 9:
  - Actual class: 3
  - Predicted class: 1
----------------------------------------
Sample 10:
  - Actual class: 3
  - Predicted class: 3
----------------------------------------
  Model Accuracy on Test Data: 0.3267
```

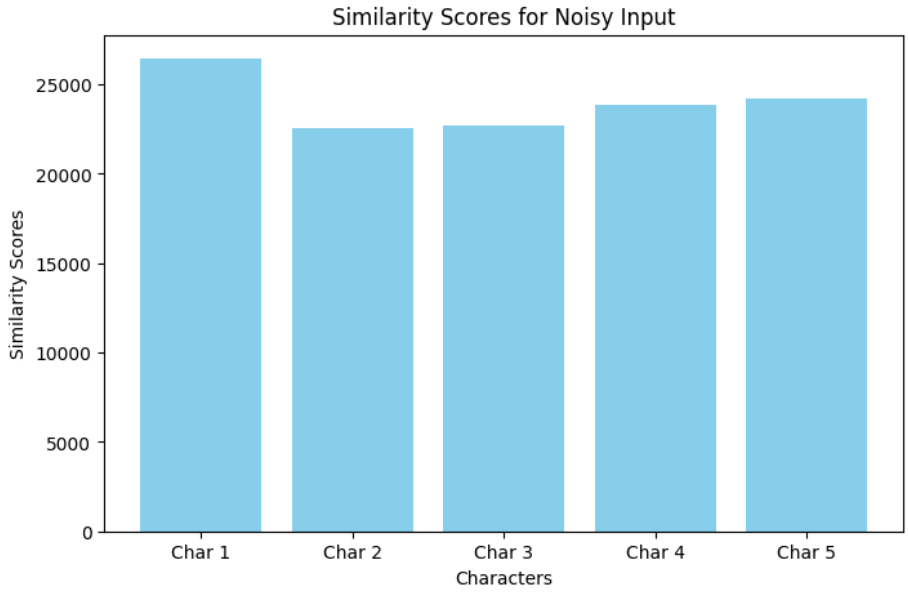
۶.

مدل دوم با ۲ لایه پنهان و تعداد نورون بالا به همراه L۲-Regularization بهترین عملکرد را نشان داد. اضافه کردن لایه Batch Normalization باعث بهبود نتایج شد زیرا با نرمال‌سازی خروجی‌ها، روند همگرایی را سرعت بخشید و حساسیت مدل به مقداردهی اولیه را کاهش داد. همچنین این تکنیک باعث کاهش overfitting شد. استفاده از L۲-Regularization نیز مدل را از بزرگ شدن بیش از حد وزن‌ها جلوگیری کرده و به تعادل و سادگی آن کمک کرد. ترکیب این دو تکنیک موجب کاهش overfitting و افزایش پایداری مدل شده است. در مقابل، استفاده از DropOut تأثیر منفی بر عملکرد مدل داشت.

پرسش ۳:

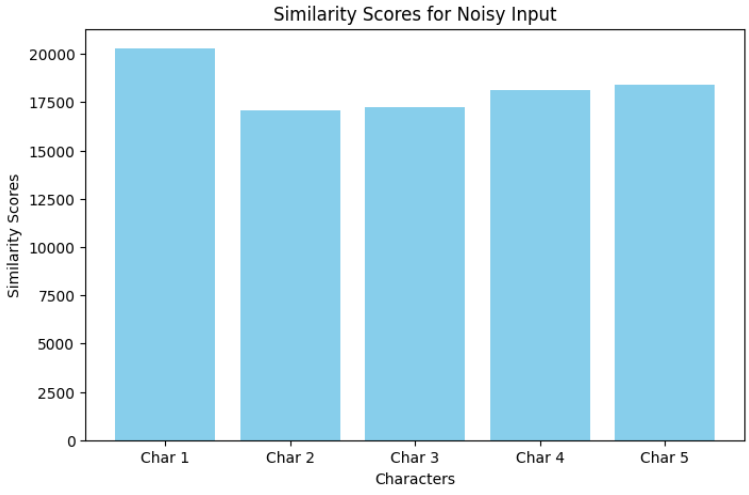
```
noise_factor = 10
```

```
Noisy image 1 recognized as character 1
Noisy image 2 recognized as character 2
Noisy image 3 recognized as character 3
Noisy image 4 recognized as character 4
Noisy image 5 recognized as character 5
```

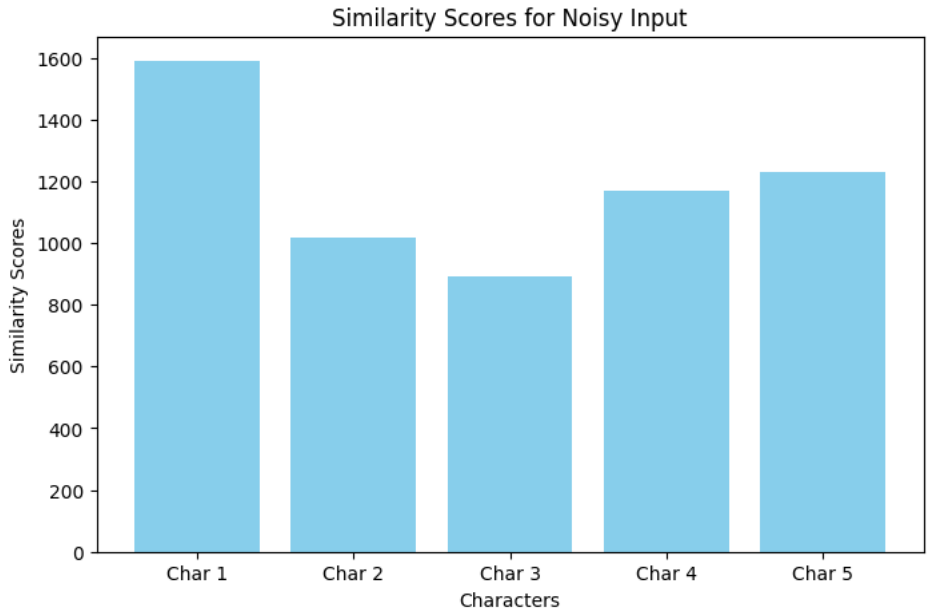



```
noise_factor = 100
```

```
Noisy image 1 recognized as character 1
Noisy image 2 recognized as character 2
Noisy image 3 recognized as character 3
Noisy image 4 recognized as character 4
Noisy image 5 recognized as character 5
```

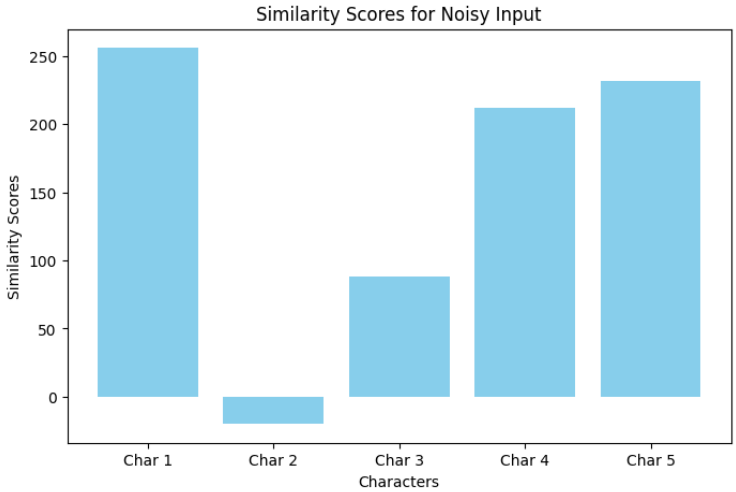


```
noise_factor = 1000

Noisy image 1 recognized as character 1
Noisy image 2 recognized as character 2
Noisy image 3 recognized as character 3
Noisy image 4 recognized as character 4
Noisy image 5 recognized as character 5
```

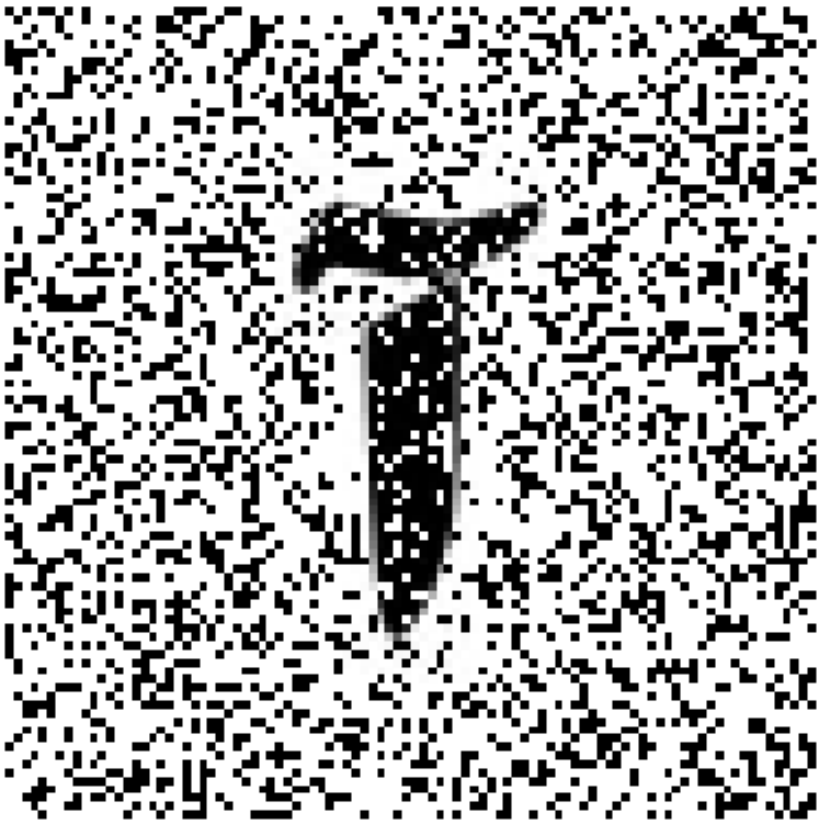


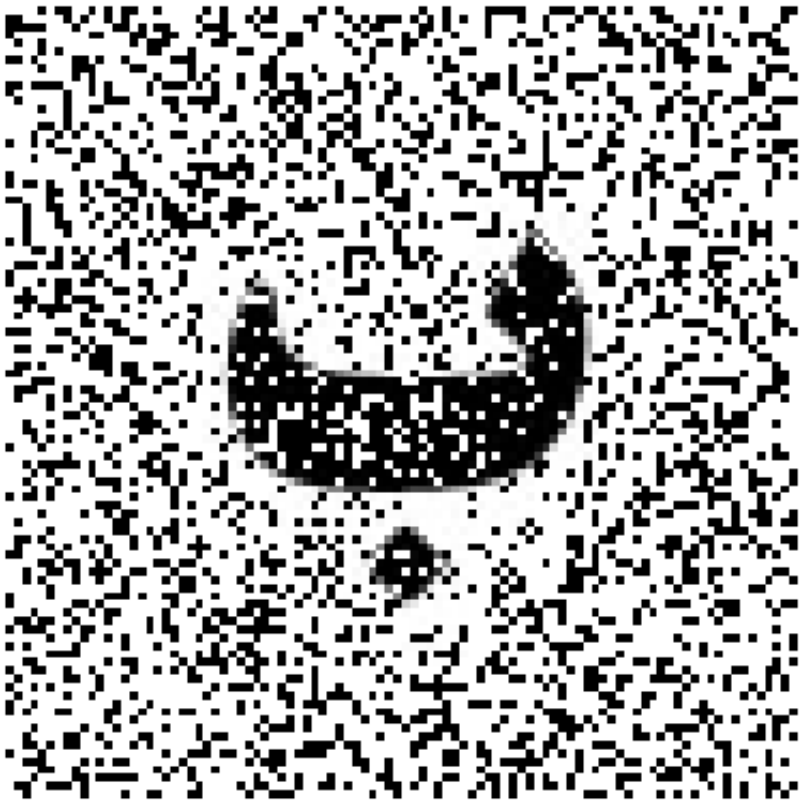

```
noise_factor = 5000

Noisy image 1 recognized as character 1
Noisy image 2 recognized as character 2
Noisy image 3 recognized as character 5
Noisy image 4 recognized as character 4
Noisy image 5 recognized as character 1
```

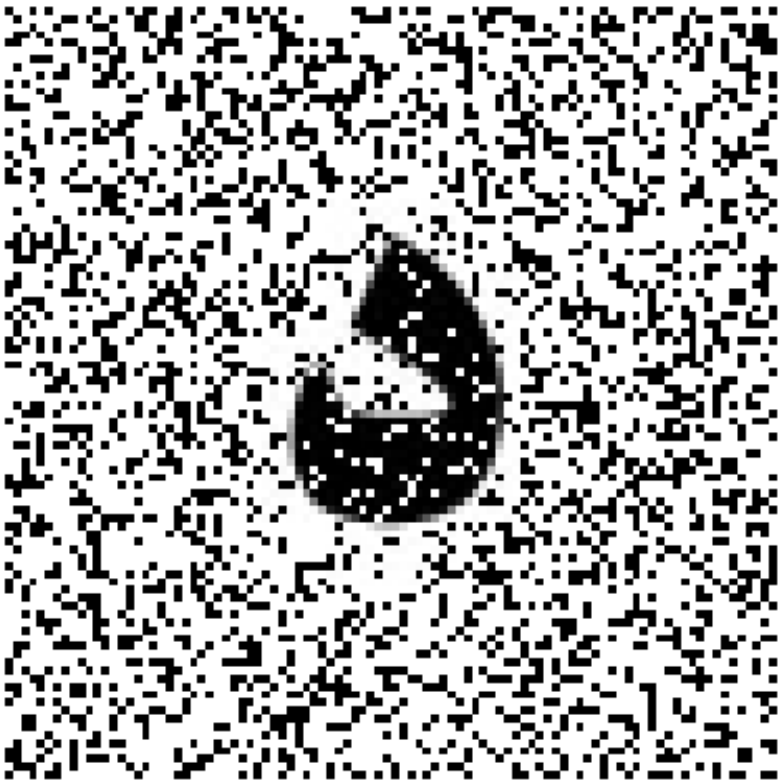
همانطور که مشاهده می‌شود. در نویز فاکتور ۵۰۰۰ دو کرکتر اشتباه تشخیص داده شدند و مدل قادر به تشخیص درست نبود.



عکس های تولید شده توسط تابع تولید عکس با missing point در زیر با نرخ ۰.۳ آورده شده است:







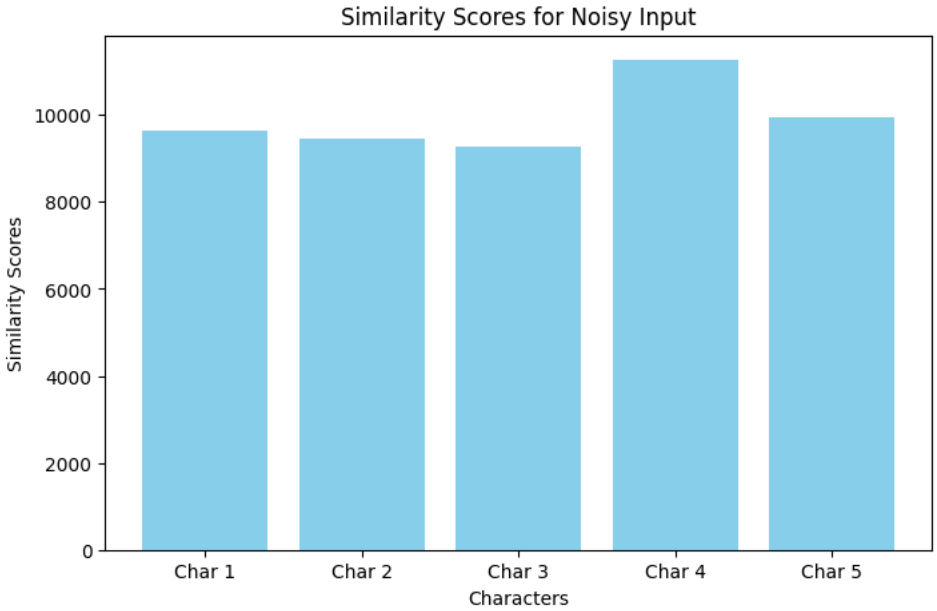
این عکس ها به کمک تابع زیر به دست آمدند.

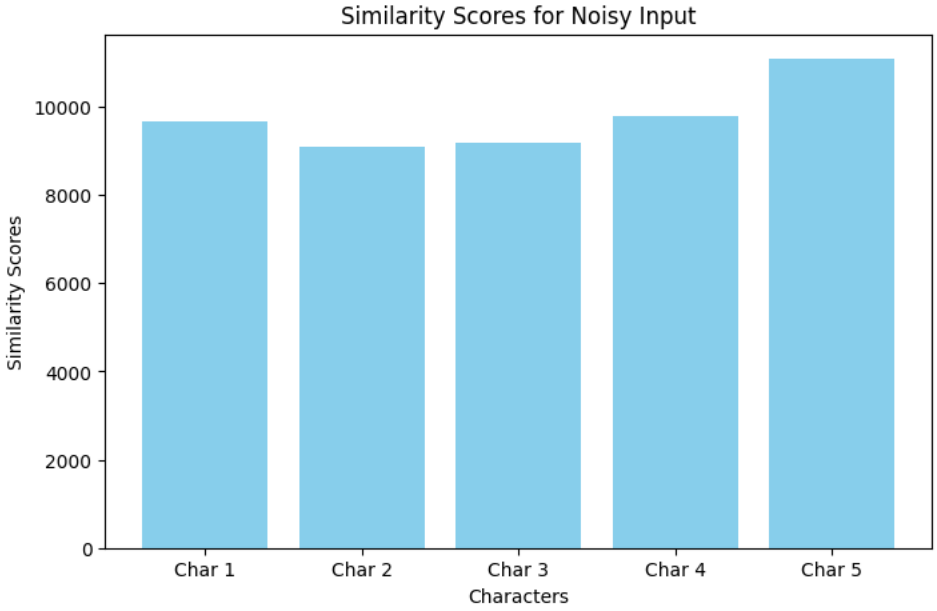
```
def generateMissingCharacterPoints(input_path, output_path,
missing_point_probability=0.05):
    """
    Adds missing points to the image where white pixels are turned
black and black pixels are turned white.

    Args:
        input_path (str): The file path to the input image.
        output_path (str): The file path to save the image with
missing points.
        missing_point_probability (float): Probability of a pixel
being turned into black or white.
    """
    # Open the input image.
    image = Image.open(input_path)

    # Determine the image's width and height in pixels.
    width, height = image.size

    # Load pixel values for the image.
    pix = image.load()
```

```
    # Loop through all pixels in the image.
    for i in range(width):
        for j in range(height):
            # Get the current pixel's RGB values
            r, g, b = pix[i, j]

            # Check if the pixel is white or black
            if r == 255 and g == 255 and b == 255:
                # Simulate missing data by turning white pixels to
black with a certain probability
                if random.random() < missing_point_probability:
                    pix[i, j] = (0, 0, 0)  # Set the pixel to black
(representing missing data)
            elif r == 0 and g == 0 and b == 0:
                # Simulate missing data by turning black pixels to
white with a certain probability
                if random.random() < missing_point_probability:
                    pix[i, j] = (255, 255, 255)  # Set the pixel to
white (representing missing data)

    # Save the image with missing points as a new file.
    image.save(output_path)

    return image
```

```
missing_point_probability=0.3 =>

Noisy image 1 recognized as character 1
Noisy image 2 recognized as character 2
Noisy image 3 recognized as character 3
Noisy image 4 recognized as character 4
Noisy image 5 recognized as character 5
```

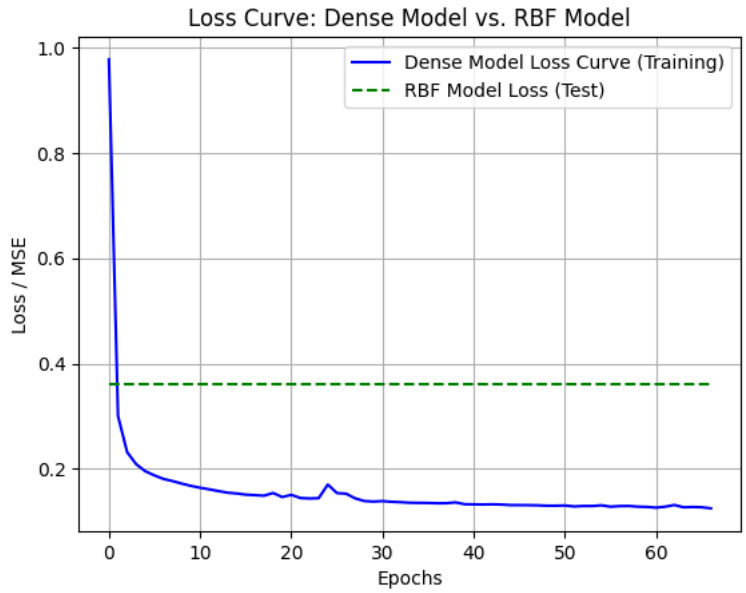
با تغییر دادن و زیاد کردن متغیر missing_point_probability به عددی حدود ۰.۴۵ می‌رسیم. این حداکثر مقداریست که مدل کنونی می‌تواند پیش بینی کند. راه حل برای حل این مشکل استفاده از یک کاهش‌دهنده‌ی نویز قبل از ورود عکس‌ها به شبکه‌ی عصبی است.

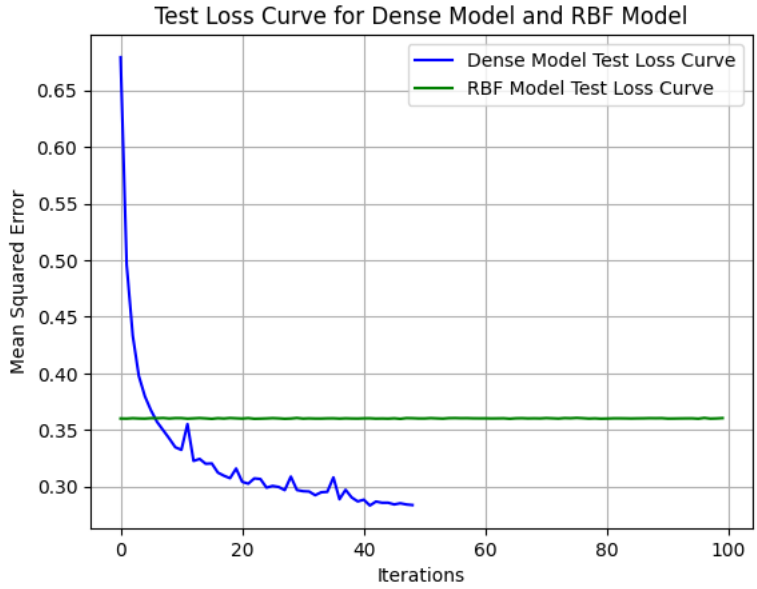




پرسش ۴:

```
Mean Squared Error of Dense Neural Network: 0.27099387608468906
Mean Squared Error of RBF Model: 0.3600210317659691
```





Rbf عملکرد بهتری به نسبت مدل dense داشته است. چرا که همانند شبکه‌ی dense نیاز به iterationهای بالا برای رسیدن به پاسخ با خطای کم‌تر ندارد. در مقاطعی مشاهده می‌شود که خطای شبکه‌ی dense کمتر می‌شود. همینطور mseکمتری نسبت به rbf دارد. اما مدل rbf بدون iterationبالا عملکرد پایداری داشته و می‌توان گفت هزینه‌ی کمتری برای گرفتن نتیجه‌ی بهتر روی دست می‌گذارد. پس می‌توان به این نتیجه رسید برای این نمونه سوال خاص مدل rbf بهتر عمل کرده است.

به پایان آمد این دفتر حکایت همچنان باقی
به صد دفتر نشاید گفت حسب‌الحال مشتاقی

با تشکر از توجهتان.