به نام خدا

گزارش مینی پروژه ۲

مبانی سیستم های هوشمند

دکترعلیاری

پاییز ۱۴۰۳

فرزاد مقدم

۴۰۰۰۹۴۵۳

لینک مخزن گیت هاب: https://github.com/Farzadmoghaddam/Intelligent-Systems

لینک گوگل کولب سوال ها:

https://colab.research.google.com/drive/1P4f0AsWxCCGJpLmg4_udiE4bCml5-g1b?usp=sharing

پرسش اول

۱.۱

استفاده از دو لایه انتهایی با فعال‌سازهای ReLU و سیگموید در یک مسئله طبقه‌بندی دوکلاسه می‌تواند مشکلات جدی ایجاد کند و ساختار مناسب شبکه را مختل کند.

فعال‌ساز ReLU معمولاً در لایه‌های مخفی استفاده می‌شود و هدف آن افزایش غیرخطی بودن شبکه است.

استفاده از ReLU می‌تواند مشکلاتی ایجاد کند:

۱. **اگر مقدار ورودی منفی باشد، خروجی صفر خواهد بود. در نتیجه، اطلاعات طبقه‌بندی ممکن است ناقص یا نادرست باشد.**
۲. **خروجی می‌تواند مقادیر بسیار بزرگ مثبت تولید کند که این با مقادیر احتمالی که بین ۰ و ۱ انتظار داریم (برای یک مسئله طبقه‌بندی) ناسازگار است.**
۳. **خروجی سیگموید با فرض اینکه ورودی آن می‌تواند مقادیر مثبت و منفی باشد طراحی شده است. اگر ورودی سیگموید تنها غیرمنفی باشد (به دلیل ReLU در لایه قبلی)، احتمال تولید شده توسط سیگموید ممکن است به طور غیرطبیعی به سمت ۰/۵ یا ۱ متمایل شود. این می‌تواند عملکرد مدل را تحت تأثیر قرار دهد.**
۴. **در مسائل طبقه‌بندی دوکلاسه، معمولاً از تابع زیان باینری کراس‌انتروپی (BCE) استفاده می‌شود. این تابع به ورودی سیگموید متکی است که مقادیر متقارن و با توزیع کامل از مثبت و منفی را می‌گیرد. اگر ورودی سیگموید توسط ReLU محدود به مقادیر غیرمنفی شده باشد، محاسبات احتمال و گرادیان دچار اشکال خواهند شد.**

فعال‌ساز سیگموید برای مسائل طبقه‌بندی دوکلاسه مناسب است. خروجی این فعال‌ساز را می‌توان به‌عنوان احتمال حضور در یکی از کلاس‌ها تفسیر کرد.

اگر لایه‌ای با فعال‌ساز ReLU مستقیماً به لایه‌ای با فعال‌ساز سیگموید متصل شود:

مقادیر خروجی لایه ReLU می‌توانند صفر یا مقادیر بزرگ مثبت باشند.

این ورودی‌ها می‌توانند در لایه سیگموید اثر منفی بگذارند.

مقادیر صفر که معمولاً به دلیل ReLU تولید شده‌اند در سیگموید مقدار 0.5 تولید می‌کنند. این مقدار ممکن است اشتباهاً به‌عنوان تصمیم "غیرقطعی" یا "نزدیک به مرز" تفسیر شود.

مقادیر بسیار بزرگ مثبت باعث می‌شوند که خروجی سیگموید به‌طور کامل به ۱ نزدیک شود، که می‌تواند منجر به مشکلات عددی و عدم همگرایی شبکه شود.

راهکار پیشنهادی

در مسائل طبقه‌بندی دوکلاسه، معمولاً تنها یک لایه خروجی با فعال‌ساز سیگموید کافی است.

لایه‌های مخفی می‌توانند از ReLU یا دیگر فعال‌سازهای مناسب برای یادگیری استفاده کنند.

۲.۱

Exponential Linear Unit (ELU)

$$\begin{array}{ll} \text{if } x > 0 & x \\ \text{if } x \leq 0 & \alpha(\exp(x) - 1) \end{array}$$

1. For $x > 0$:

$$f(x) = x \implies f'(x) = 1$$

2. For $x \leq 0$:

$$f(x) = \alpha(\exp(x) - 1) \implies f'(x) = \alpha \exp(x)$$

$$f'(x) = \begin{cases} 1 & \text{if } x > 0 \\ \alpha \exp(x) & \text{if } x \leq 0 \end{cases}$$

ELU نسبت به RELU دارای پایداری بیشتری است.زیرا در صورت منفی بودن ورودی مقداری نزدیک به صفر را میدهد که باعث بهبود همگرایی میشود.

باعث بهبود یادگیری در مراحل اولیه میشود.

در ReLU، اگر ورودی $x \leq 0$ باشد، گرادیان صفر است. این بدان معناست که نورون‌هایی که مقدار ورودی منفی دریافت می‌کنند، به طور کامل غیرفعال می‌شوند و در فرآیند یادگیری شرکت نمی‌کنند. این پدیده که به "مشکل مرگ نورون‌ها" (Dying ReLU Problem) معروف است، می‌تواند باعث شود بخش قابل توجهی از نورون‌ها در طول آموزش بی‌اثر شوند.

اما برای ELU، برای مقادیر $x \leq 0$، گرادیان ELU برابر αe^x است که هرگز صفر نمی‌شود (مگر اینکه $\alpha = 0$ باشد). این ویژگی تضمین می‌کند که نورون‌ها حتی در نواحی منفی همچنان گرادیان غیرصفر دارند و در فرآیند یادگیری مشارکت می‌کنند.

۳.۱

از نورون McCulloch-Pitts استفاده میکنیم.

از سه خط مرزی برای جداسازی ناحیه ها از هم استفاده میکنیم.

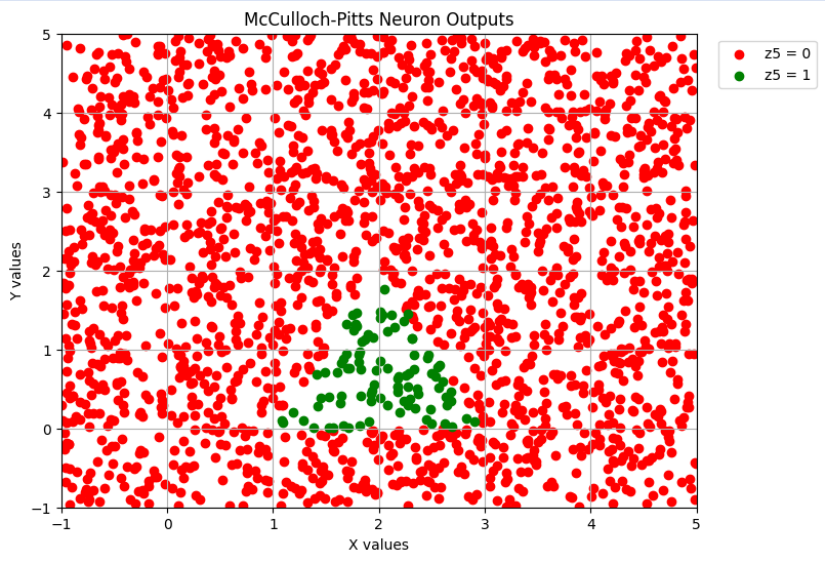
2x-y-2=0

-2x-y+6=0

0x+y+0=0

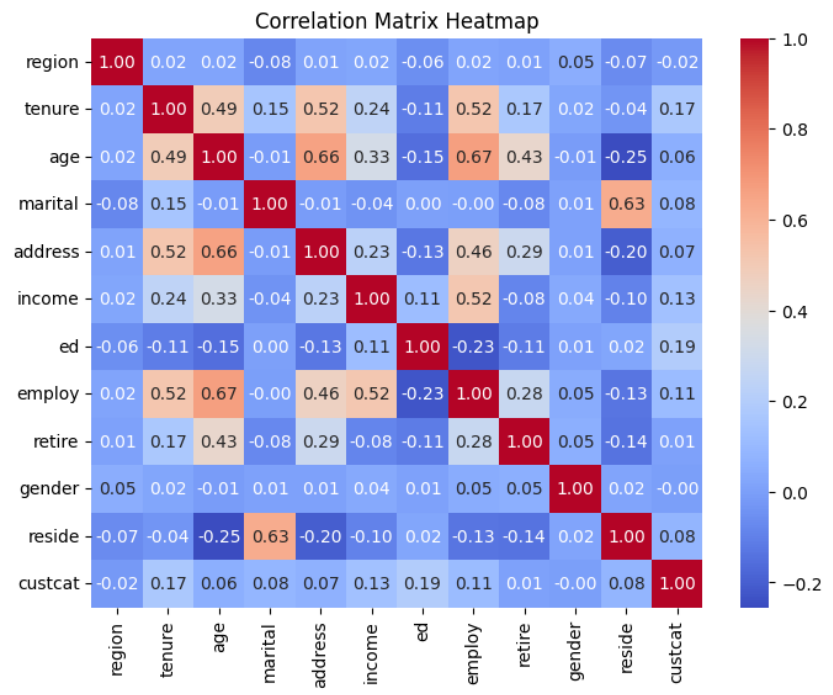
برای طراحی شبکه بر پایه نورون‌های McCulloch-Pitts که داخل مثلث را تشخیص دهد، می‌توان از سه شرط خطی (هرکدام یک نورون) استفاده کرد. این سه نورون با بررسی موقعیت نقطه نسبت به سه خط مثلث (و جهت آن‌ها) مشخص می‌کنند که نقطه داخل مثلث است یا خیر

برای حل این مسئله به ۳ نورون برای تعریف خطوط و یک نورون برای AND کردن خروجی ۳ نورون قبلی داریم. در نتیجه در کل به ۴ نورون نیاز داریم.

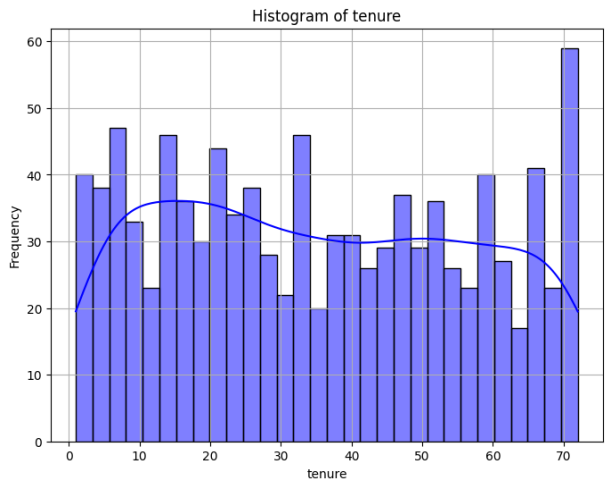


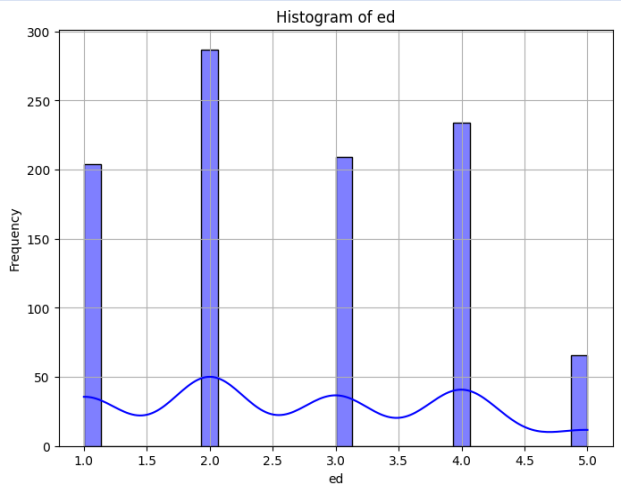
اگر بخواهیم از توابع فعالساز دیگر برای تصمیم گیری استفاده کنیم، به دلیل اینکه در لایه قبل از نورون Mcculloch-Pitts استفاده کرده‌ایم، تنها ۳ مقدار مختلف خواهیم داشت (۰.۷۳۱ و ۰.۸۸ و ۰.۹۵۲ برای سیگموید) که اگر آستانه سیگموید را روی ۰.۹ تنظیم کنیم، نقاطی که مقدار ۰.۹۵۲ را دارند (داخل مثلث هستند) را می‌توانیم تشخیص بدهیم. در نتیجه دقیقا خروجی قسمت قبل را خواهیم داشت

(۲.۲



همانطور که از هیت مپ پیداست، داده‌های موجود همبستگی خوبی با داده مورد نظر ما یعنی custcat ندارند به طوری که بیشترین همبستگی مربوط به داده ed است که برابر با ۰.۱۹ می‌باشد.



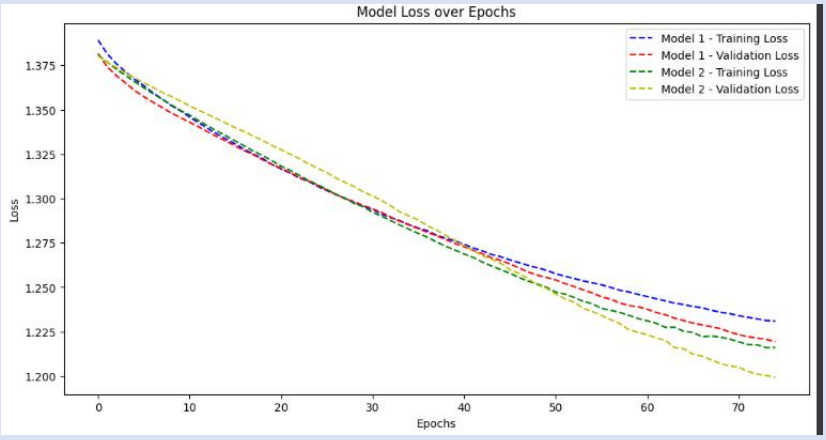


(۲.۴

SGD

```
# Mode                    ns one hidden lay
model_   = Sequen  ial([
    De  se(100, a  tivation='relu',
    De  se(num_cl  sses, activation=
])

# Compile Model 1
model_1.compile(optimizer=SGD(lear

# Train Model 1
history_1 = model_1.fit(X_train, y

# Model 2: Contains two hidden lay
model_   = Sequ  tial([
    De  se(50, a  tivation='relu', i
    De  se(50, a  tivation='relu'),
    De  se(num_cl  sses, activation=
])
```

```
Model 1 - Test Accuracy: 0.41, Test Loss: 1.27
Model 2 - Test Accuracy: 0.42, Test Loss: 1.27
```

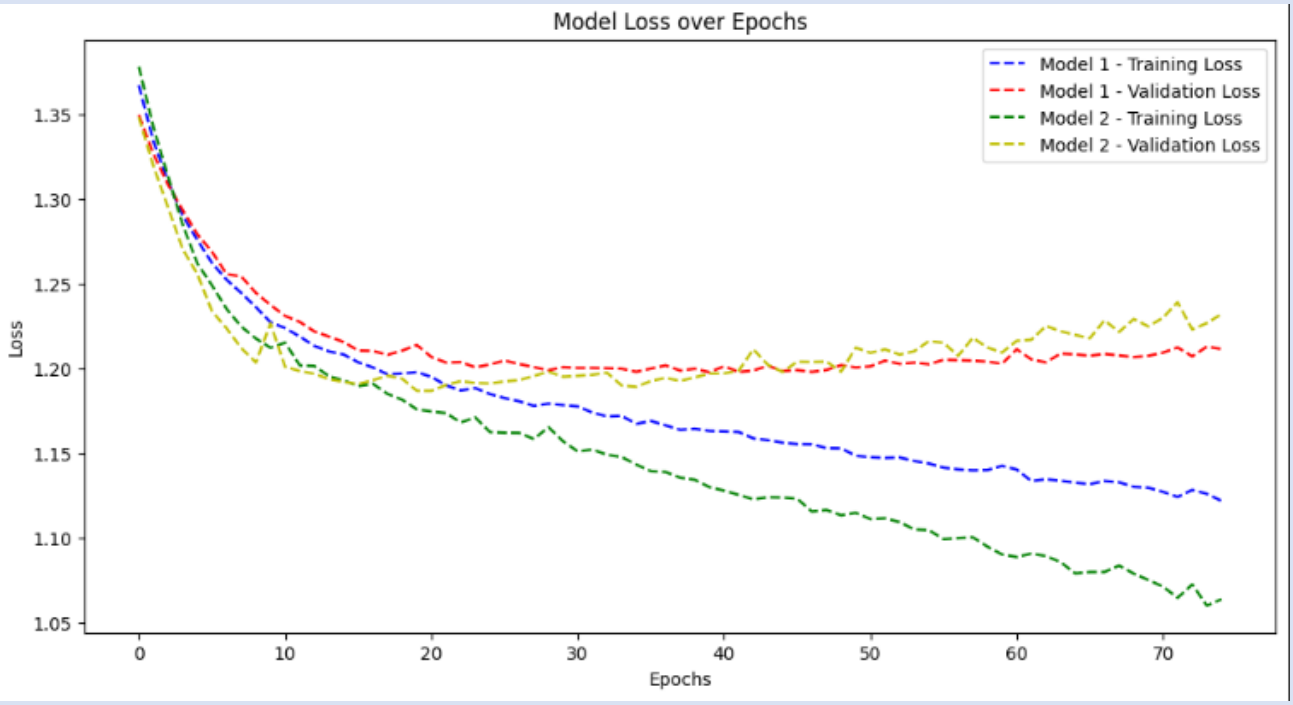


در حالت کلی می‌توان گفت که مدل‌ها عملکرد قابل قبولی داشتند.

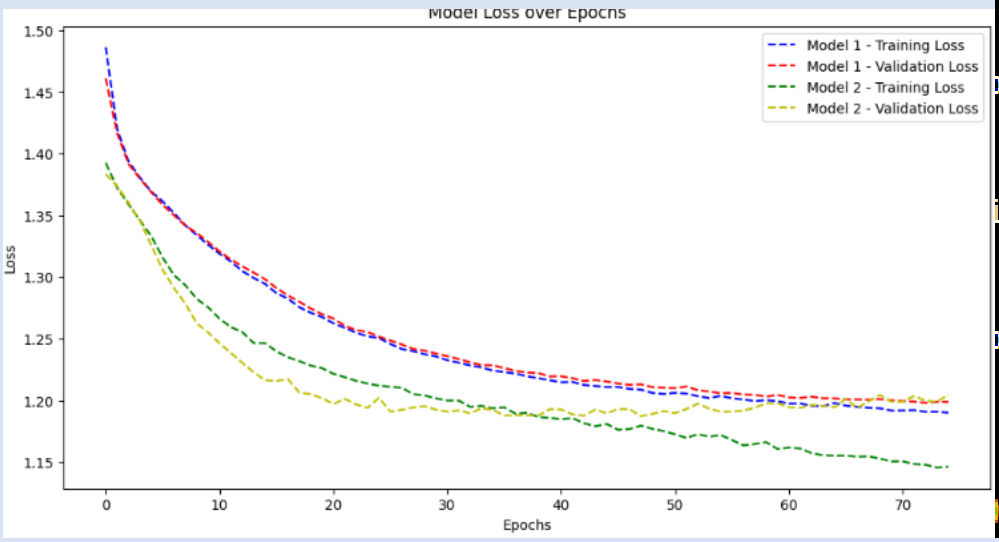
ADAM

```
model_1 = Sequential([
    Dense(100, activation
    Dense(num_classes, ac
])

# Compile Model 1
model_1.compile(optimizer

# Train Model 1
history_1 = model_1.fit(X

# Model 2: Contains two h
model_2 = Sequential([
    Dense(50, activation=
    Dense(50, activation=
    Dense(num_classes, ac
])
```

```
Model 1 - Test Accuracy: 0.41, Test Loss: 1.31
Model 2 - Test Accuracy: 0.41, Test Loss: 1.35
```



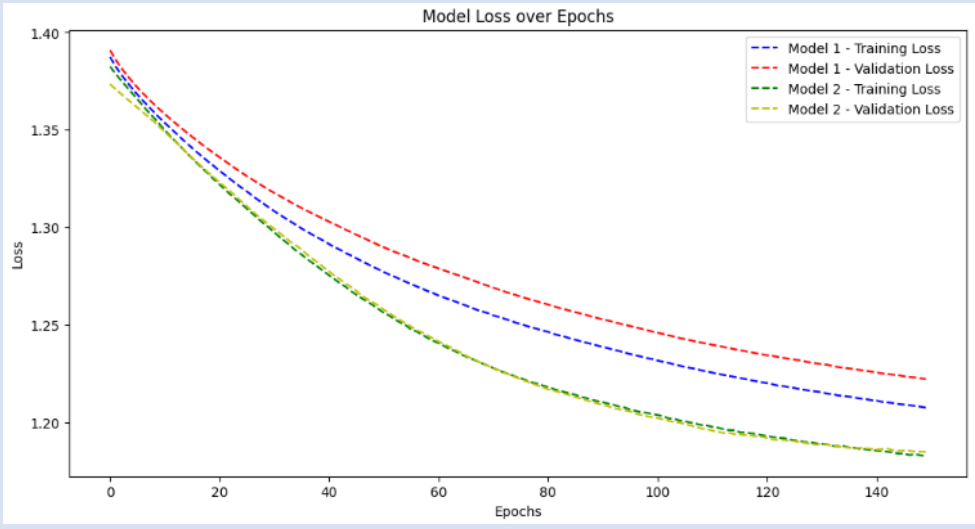
مشاهده می‌شود که val loss برای هر دو مدل بعد از طی کردن روند نزولی شروع به افزایش می‌کند. این به خاطر پیچیدگی پیش از حد مدل است در نتیجه برای بهبود نتایج باید تعداد نورون‌ها را کاهش دهیم.



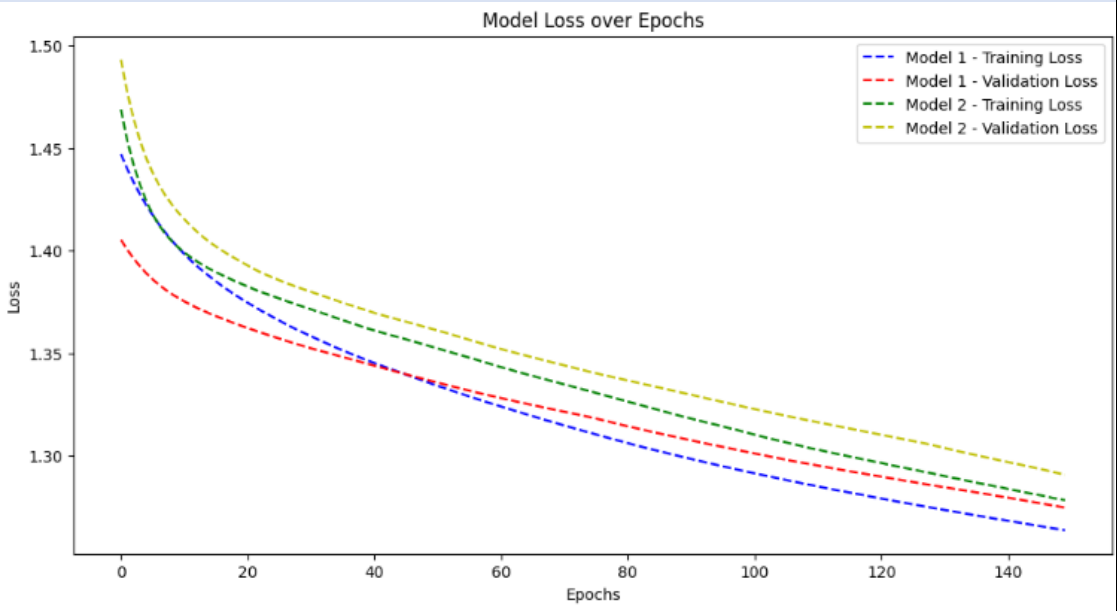

```
odel_1 = Sequential([
    Dense(20, activatio
    Dense(num_classes,
])

 Compile Model 1
odel_1.compile(optimiz

 Train Model 1
istory_1 = model_1.fit

 Model 2: Contains two
odel_2 = Sequential([
    Dense(20, activatio
    Dense(20, activatio
    Dense(num_classes,
])
```

مشاهده می‌شود که مشکل Overfit شدن مدل بهبود یافته است و هر دو مدل تا epoch های بیشتری روند نزولی خود را ادامه می‌دهند.

ADOPT



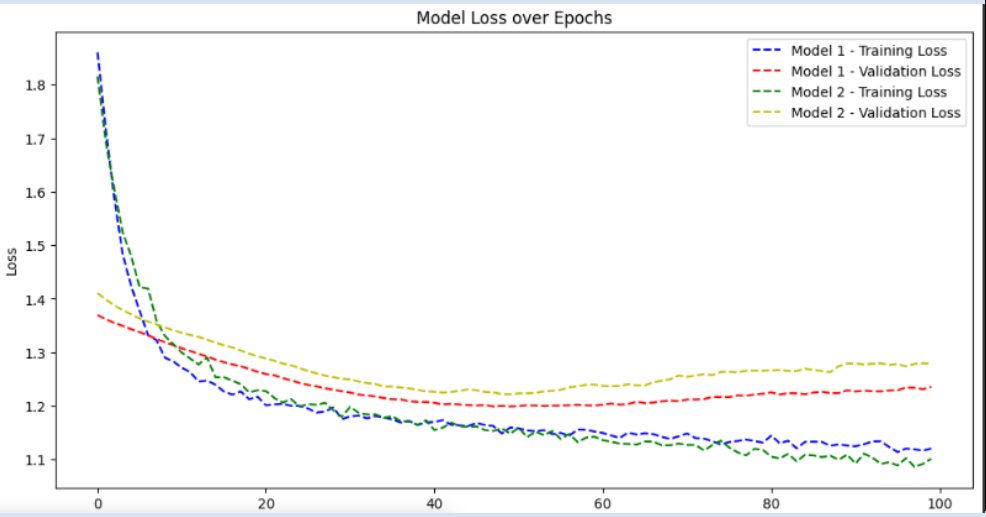

```
    # Model 1: Contai
    model_1 = Sequent
        Dense(100, ac
        Dense(num_cla
    ])

    # Compile Model 1
    model_1.compile(o

    # Train Model 1
    history_1 = model

    # Model 2: Contai
    model_2 = Sequent
        Dense(50, act
        Dense(50, act
        Dense(num_cla
    ])
```




```
# Model 1: Contains
model_1 = Sequential
    Dense(30, activa
    Dense(num_classe
])

# Compile Model 1
model_1.compile(opti

# Train Model 1
history_1 = model_1.

# Model 2: Contains
model_2 = Sequential
    Dense(20, activa
    Dense(10, activa
    Dense(num_classe
])

# Compile Model 2
```

به نظر می‌رسد که تعداد epoch کمی برای آموزش دادن مدل استفاده شده است. همچنین learning rate را هم ۱۰ برابر افزایش می‌دهیم تا اصلاحات بزرگتری روی مدل صورت گیرد.

Batch Normalization

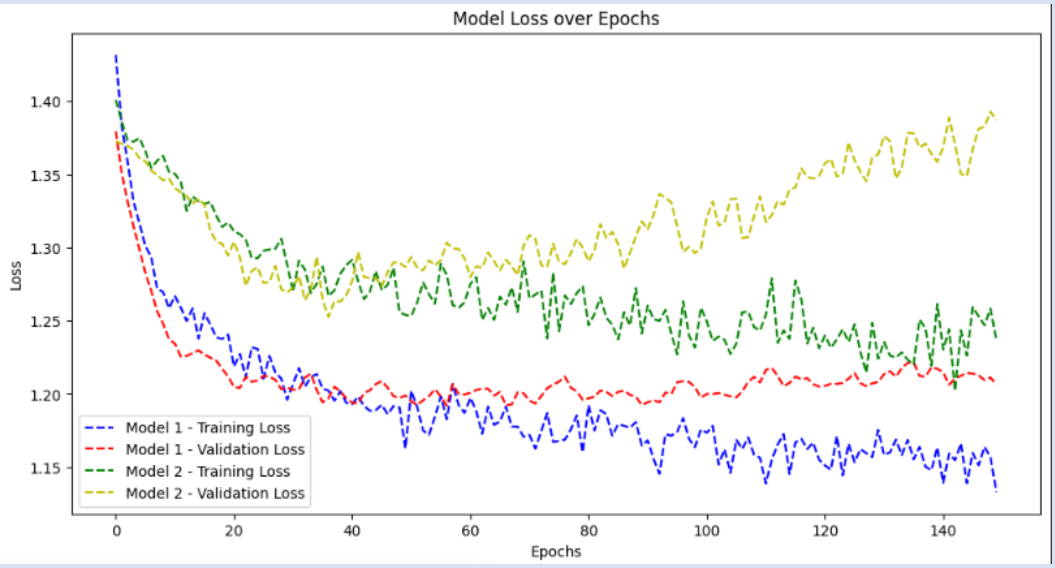



```
    Dense(30, act
    BatchNormaliz
    Dense(num_cla
])

# Compile Model 1
model_1.compile(o

# Train Model 1
history_1 = model

# Model 2: Contai
model_2 = Sequent
    Dense(20, act
    BatchNormaliz
    Dense(8, acti
```

```
Model 1 - Test Accuracy: 0.38, Test Loss: 1.46
Model 2 - Test Accuracy: 0.44, Test Loss: 1.35
```

همانطور که از نتایج پیداست اضافه کردن لایه نرمال سازی برای مدل ۲، موجب بهبود عملکرد آن شده ولی برای مدل ۱ برعکس نتیجه داده است.

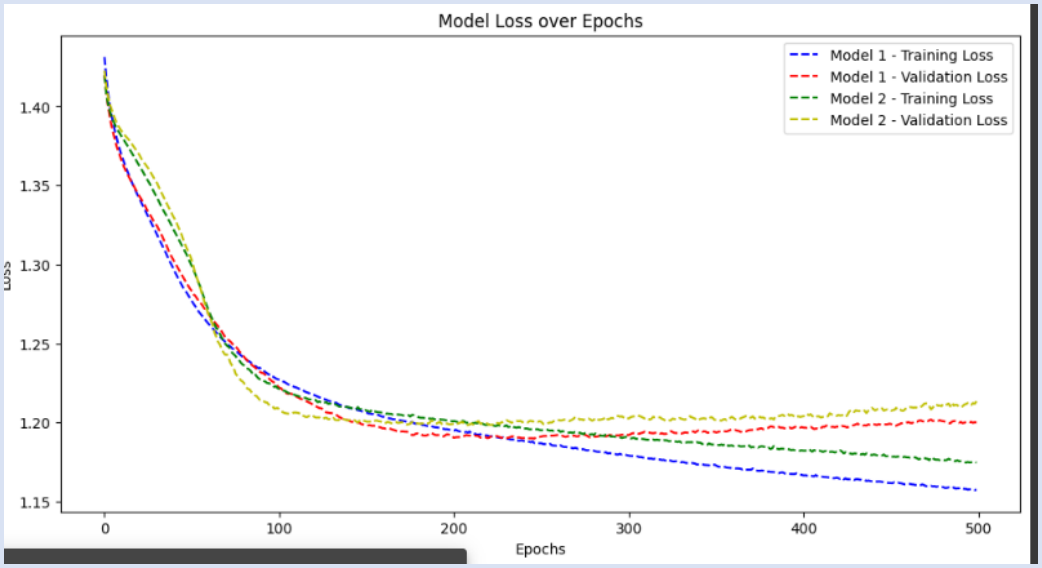
Dropout




```
model_1 = Sequentia
    Dense(20, activ
    Dropout(0.3),
    Dense(num_class
])

# Compile Model 1
model_1.compile(opt

# Train Model 1
history_1 = model_1

# Model 2: Contains
model_2 = Sequentia
    Dense(15, activ
    Dropout(0.3),
    Dense(5, activa
    Dropout(0.3),
    Dense(num_class
])
```

استفاده از روش Dropout موجب بهبود عملکرد مدل‌ها تا تعداد مشخصی از epoch می‌شود و اگر از Early Stop استفاده شود، می‌توان آموزش مدل را در بهترین حالت خود (epoch ۶۰ برای مدل ۲) متوقف کنیم.

L2-Regularization




```
# Model 1: Contains one
model_1 = Sequential([
    Dense(20, activatio
    Dense(num_classes,
])

# Compile Model 1
model_1.compile(optimi

# Train Model 1
history_1 = model_1.fi

# Model 2: Contains two
model_2 = Sequential([
    Dense(15, activatio
    Dense(5, activation
    Dense(num_classes,
])

# Compile Model 2
model_2.compile(optimi
```

```
Model 1 - Test Accuracy: 0.44, Test Loss: 1.33
Model 2 - Test Accuracy: 0.42, Test Loss: 1.32
```

مشاهده می‌شود که عملکرد مدل ۲ بسیار پایدار بوده ولی نمودار loss مدل ۱ بعد از تعداد مشخصی epoch، روند افزایشی را در پیش می‌گیرد.

۲.۵)

مدل۱

```
Random Samples: Actual vs Predicted
Sample 1: Actual: 2, Predicted: 2
Sample 2: Actual: 1, Predicted: 1
Sample 3: Actual: 0, Predicted: 3
Sample 4: Actual: 0, Predicted: 0
Sample 5: Actual: 3, Predicted: 2
Sample 6: Actual: 2, Predicted: 2
Sample 7: Actual: 3, Predicted: 0
Sample 8: Actual: 2, Predicted: 2
Sample 9: Actual: 0, Predicted: 2
Sample 10: Actual: 3, Predicted: 2
```

مدل۲

```
Random Samples: Actual vs Predicted
Sample 1: Actual: 2, Predicted: 2
Sample 2: Actual: 1, Predicted: 3
Sample 3: Actual: 1, Predicted: 1
Sample 4: Actual: 3, Predicted: 3
Sample 5: Actual: 0, Predicted: 0
Sample 6: Actual: 3, Predicted: 1
Sample 7: Actual: 0, Predicted: 1
Sample 8: Actual: 3, Predicted: 1
Sample 9: Actual: 0, Predicted: 1
Sample 10: Actual: 3, Predicted: 1
```

(۳.۱

تابع اول ابتدا طول و عرض عکس دریافت شده را استخراج می‌کند و سپس یک فاکتور (factor) یا ترشولد (threshold) برای تعیین مرز سفید یا سیاه بودن پیکسل مورد نظر تعریف می‌کند. سپس مقادیر RGB تک تک پیکسل‌ها را استخراج کرده و total intensity را محاسبه می‌کند. سپس با توجه به فاکتور از قبل تعیین شده، سفید یا سیاه بودن پیکسل را تعیین کرده و آن را به عکس خالی binary_representation اضافه می‌کند.

البته برای این کار می‌توان به شکل زیر هم عمل کرد:

```
# Convert to binary (thresholding)
binary_img = np.array(img) > 75  # Threshold at 75
return binary_img.astype(np.int8)  # Convert to 0s and 1s
```

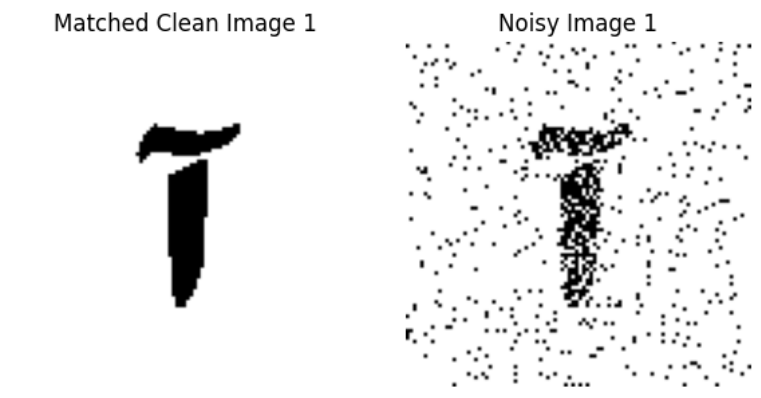
در اینجا binary_img ماتریسی به همان ابعاد عکس دریافت شده می‌باشد با این تفاوت که اینجا عکس به gray scale تبدیل شده و مقادیر RGB را ندارد. در نتیجه پیکسل‌های با مقدار بیشتر از ۷۵ به True و کمتر از آن به False تبدیل می‌شوند. که با astype(np.int8) به اعداد ۱ و ۰ تبدیل خواهند شد.

در تابع دوم برای ایجاد نویز روی عکس می‌توان با تنظیم noise_factor مقدار نویز ایجاد شده را تنظیم کرد. تولید نویز هم به این صورت است که یک مقدار تصادفی از بازه noise_factor- تا noise_factor را انتخاب کرده و مقادیر RGB پیکسل مورد نظر را با این عدد جمع می‌کنیم.

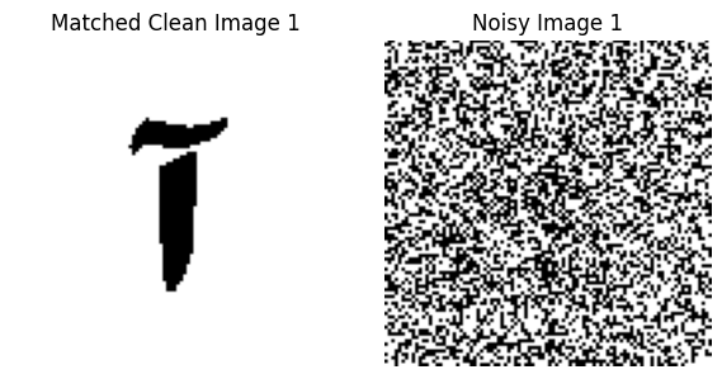
۱ (۳.۲

---

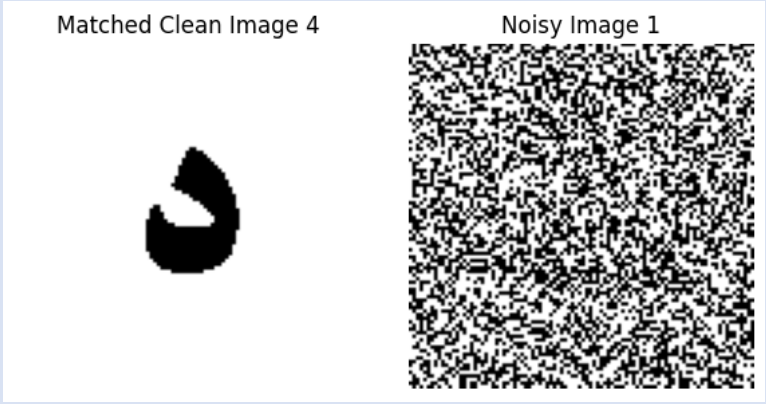
noise_factor = 200



noise_factor = 1000



noise_factor = 2000

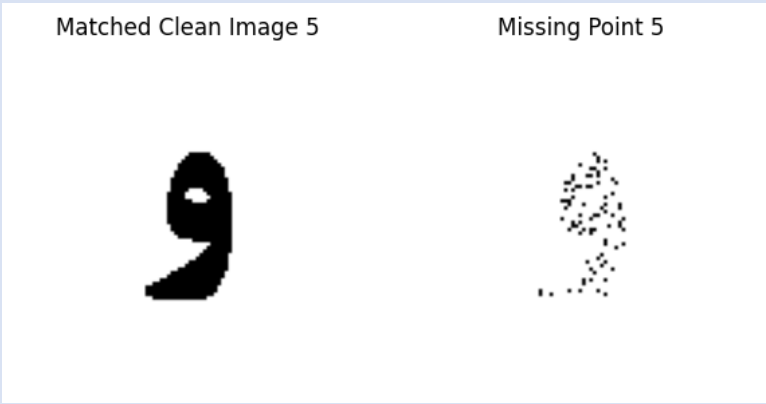


با این مقدار نویز بعضی اوقات جواب اشتباه از شبکه می‌گیریم که در شکل بالا یک مثال از آن را می‌بینیم.

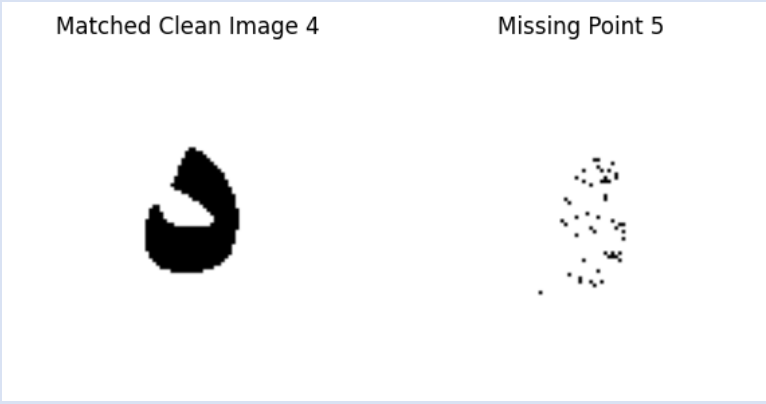
۲)

---

noise_factor = 500

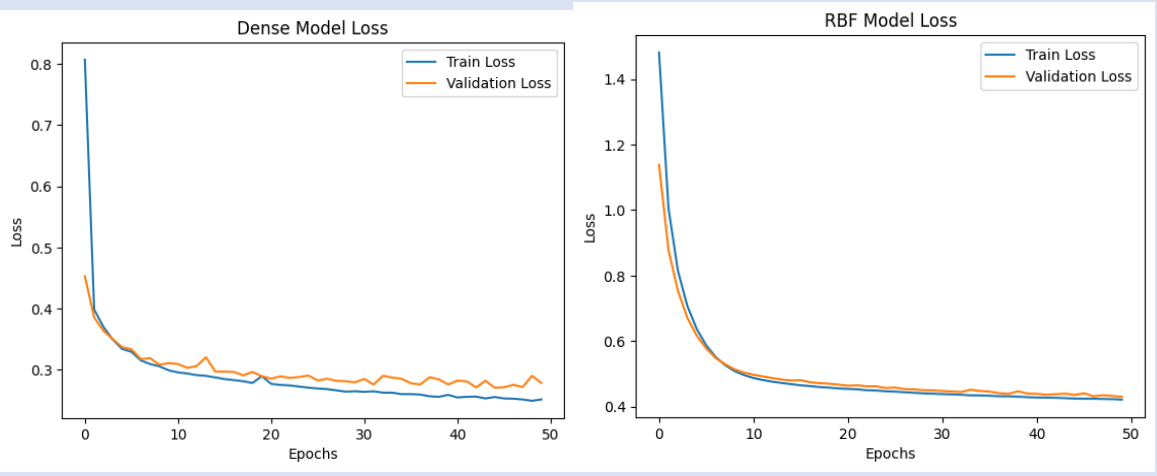


noise_factor = 1000



برای بهبود عملکرد شبکه در عکس‌های Missing Point، می‌توان چند نمونه عکس Missing Point برای Train کردن مدل در نظر بگیریم.

۳ ۴)



مشاهده می‌شود که loss مدل Dense کمتر از مدل RBF است و نشان دهنده‌ی عملکرد بهتر این مدل در این مسئله است.

مدل RBF زمانی عملکرد خوبی خواهد داشت که داده‌ها پراکندگی خوشه‌ای داشته باشند ولی در این مسئله، داده‌ها روابط پیچیده و غیر خطی با هدف دارند که مدل Dense گزینه مناسب‌تری برای حل این مسئله خواهد بود.