نوید ابراهیمی

Q1)

x_1	x_2	T
1	1	-1
1	-1	1
-1	1	1
-1	-1	1

$y = w_1 x_1 + w_2 x_2 + b$ (1)

default values:

$\alpha = 0.1$

$b = 0.1$ ← b

$w_1 = 0.2$ ← x_1

$w_2 = 0.3$ ← x_2

step 1: line 1 → (1, 1) → 0

$y = 0.1 + 0.2 \times 1 + 0.3 \times 1 = 0.6$

$b_{new} = b_{old} + \eta(d - y) = 0.1 + 0.1(-1 - 0.6) = 0.1 - 0.16 = -0.06$

$w_{i\,new} = w_{i\,old} + \eta(d - y)x_i$ → $w_1 = 0.2 + 0.1(-1 - 0.6)1 = 0.2 - 0.16 = 0.04$

→ $w_2 = 0.3 + 0.1(-1 - 0.6)1 = 0.3 - 0.16 = 0.14$

step 2: line 2 → (1, 0) → 1

$y = -0.06 + 0.04 + 0.14(-1) = -0.16$

$b = -0.06 + 0.1(1 + 0.16) = -0.06 + 0.116 = 0.056$

$w_1 = 0.04 + 0.116 = 0.156$

$w_2 = 0.14 + 0.116(-1) = -0.076$

step 3: line 3 → (0, 1) → 1

$y = 0.056 + 0.156(-1) + (-0.076)(1) = -0.176$

$b = 0.056 + 0.1(1 + 0.176) = 0.1736$

$w_1 = 0.156 + 0.1176 \times (-1) = -0.0616$

$w_2 = -0.076 + 0.1176 = 0.0416$

Step 4: line4 → (-1, -1) → 1

$y = 0.1736 + 0.0616 - 0.0416 = 0.1936$

$b = 0.1736 + 0.1(1 - 0.1936) = 0.25424$

$w_1 = -0.0616 - 0.08064 = -0.14224$

$w_2 = 0.0416 - 0.08064 = -0.03904$

(Q2

الف)

تفاوت اصلی بین توابع فعالسازی خطی و غیرخطی در توانایی آنها برای مقابله با پیچیدگی و غیرخطی بودن در داده ها نهفته است.

عملکرد فعال سازی خطی:

- یک تابع فعال سازی خطی به شکل $f(x) = x$ است. یعنی خروجی متناسب با ورودی است.

- زمانی استفاده می شود که داده ها خطی هستند یا زمانی که می خواهیم یک مقدار پیوسته را پیش بینی کنیم.

- یک شبکه عصبی بدون تابع فعال سازی اساساً فقط یک مدل رگرسیون خطی است.

- با این حال، اگر یک تابع خطی را برای معرفی غیر خطی متمایز کنیم، نتیجه به ورودی x بستگی نخواهد داشت و تابع ثابت می شود.

- بنابراین، توابع فعال سازی خطی معمولاً فقط در لایه خروجی استفاده می شوند.

عملکرد فعال سازی غیرخطی:

- توابع فعال‌سازی غیرخطی به شبکه‌های عصبی اجازه می‌دهد تا از خطا یاد بگیرند و وزن‌ها و بایاس‌های نورون‌ها را در حین Backpropagation آپدیت کنند.

- آنها شبکه های عصبی را قادر می سازند تا پدیده های پیچیده و غیر خطی را مدل کنند.

- توابع فعال‌سازی غیرخطی مانند sigmoid، tanh، و ReLU می‌توانند داده‌هایی را مدیریت کنند که به‌طور خطی قابل تفکیک نیستند.

- به عنوان مثال، تابع سیگموئید بین ۰ و ۱ متغیر است و به ویژه برای مدل هایی که باید احتمال را به عنوان خروجی پیش بینی کنیم مفید است.

- توابع فعال‌سازی غیرخطی به مدل اجازه می‌دهد تا نقشه‌های پیچیده بین ورودی‌ها و خروجی‌های شبکه ایجاد کند.

به طور خلاصه، توابع فعال‌سازی غیرخطی برای شبکه‌های عصبی برای مدل‌سازی داده‌های پیچیده ضروری هستند، در حالی که توابع فعال‌سازی خطی به داده‌های ساده‌تر و قابل جداسازی خطی محدود می‌شوند.

ب)

- بایاس رندم و وزن‌ها صفر:

اینکار باعث این میشود که چون وزن‌ها در ابتدا مقدار صفر داشتند، آموزش مدل به نسبت رندم بودن وزن‌ها کندتر صورت پذیرد. جدای از این مورد چون وزن‌ها در ابتدا مقدار یکسانی دارند، میزان تغییر آنها به یک اندازه و یک جهت انجام میگیرد که باعث میشود مدل ویژگی‌های مختلف را به خوبی یاد نگیرد.

- بایاس صفر و وزن‌ها رندم:

ممکنه تمام بایاس‌ها به یک سمت جهت‌گیری کنند در نتیجه مدل بایاس شود و مقدار خطایش بیشتر شود. به طور مثال مرز به درستی شیفت پیدا نمیکند و ممکنه باعث underfitting شود.

ج)

از نظر توانایی تعمیم، MLPها به طور کلی قدرتمندتر از شبکه های عصبی تک لایه مانند Perceptron و Adaline در نظر گرفته می شوند زیرا می توانند مرزهای تصمیم گیری پیچیده‌تری را یاد بگیرند. همچنین به دلیل غیرخطی و چندلایه‌ای بودن میتواند ویژگی‌های بیشتری را یاد بگیرد.

اگر توانایی تعمیم این مدل‌ها را مقایسه کنیم،Perceptron ضعیف‌ترین در بین آنهاست زیرا ساختار ساده‌تری دارد و فقط می‌تواند مرزهای تصمیم‌گیری خطی را یاد بگیرد. جدای از این مورد، perceptron نمیتواند بهترین خط برای جداسازی را پیدا کند و فقط یک خط پیدا میکند که ممکن است بهترین نباشد.

د)

مزیت:

استفاده از این فرمول این است که می تواند سریعتر از روش های دیگر همگرا شود، به خصوص در هنگام برخورد با مشکلات پیچیده.

معایب:

- برای محاسبه ماتریس Hessian به حافظه و قدرت محاسباتی زیادی نیاز دارد که می تواند در هنگام برخورد با مجموعه داده های بزرگ مشکل ساز باشد.
- می تواند به نویز در داده ها حساس باشد که می تواند منجر به overfit شود.

Q3)

dataset=circle

: Activation function=tanh

نمودار غیر خطی tanh به شبکه اجازه می دهد تا مرزهای دایره را که خطی نیستند، یاد بگیرد. در واقع این غیرخطی بودن به آن اجازه میدهد دایره را رسم کند.

: Activation function=sigmoid

نمودار غیر خطی sigmoid به شبکه اجازه می دهد تا مرزهای دایره را که خطی نیستند، یاد بگیرد. در واقع این غیرخطی بودن به آن اجازه میدهد دایره را رسم کند. در مقایسه با tanh، چون شیب کمتری (۰,۲۵ شیب tanh) نسبت به tanh دارد و چون تغییر وزن‌ها به دلیل شکل sigmoid یا همه مثبت میشوند یا همه منفی، به این دو دلیل همگرایی در sigmoid کندتر اتفاق میفتد.

: Activation function=ReLU

همانطور که در شکل مشخص است، دایره توسط خط‌هایی محصور شده‌اند. دلیل وجود خط برخلاف تو تابع قبلی، خطی بودن ReLU است که باعث میشود به صورت یک ۶ ضلعی Classification انجام شود. جدای از این مورد، به دلیل شیب بیشتری که ReLU دارد باعث میشود همگرایی زودتر اتفاق بیفتد.

: Activation function=linear

نمیتواند جداسازی را انجام دهد زیرا دیتاست به صورت دایره‌ای است و یک تابع خطی نمیتواند آنها را از هم با یک خط جدا کند. تفاوتش با ReLU این است که ReLU یک تکه خطی نیست.

Dataset=exclusive or

: Activation function=tanh

نمودار غیر خطی tanh به شبکه اجازه می دهد تا مرزهای ناحیه‌ها را که خطی نیستند، یاد بگیرد. در واقع این غیرخطی بودن به آن اجازه میدهد منحنی‌ها را رسم کند. چون دیتاست نسبت به circle در یک ناحیه متمرکز نیست، همگرایی دیرتر اتفاق میفتد و در epoch بالاتری به همگرایی میرسد.

: Activation function=sigmoid

نمودار غیر خطی sigmoid به شبکه اجازه می دهد تا مرزهای منحنی‌ها را که خطی نیستند، یاد بگیرد. در واقع این غیرخطی بودن به آن اجازه میدهد منحنی‌ها را رسم کند. در مقایسه با tanh، چون شیب کمتری (۰,۲۵ شیب tanh) نسبت به tanh دارد و چون تغییر وزن‌ها به دلیل شکل sigmoid یا همه مثبت میشوند یا همه منفی، به این دو دلیل همگرایی در sigmoid کندتر اتفاق میفتد. چون دیتاست نسبت به circle در یک ناحیه متمرکز نیست، همگرایی دیرتر اتفاق میفتد و در epoch بالاتری به همگرایی میرسد.

: Activation function=ReLU

همانطور که در شکل مشخص است، منحنی توسط خط‌هایی محصور شده‌اند. دلیل وجود خط برخلاف تو تابع قبلی، خطی بودن ReLU است که باعث میشود با ۴ ضلع Classification انجام شود. جدای از این مورد، به دلیل شیب بیشتری که ReLU دارد باعث میشود همگرایی زودتر اتفاق بیفتد.

چون نسبت به circle تعداد ضلع‌های کمتری را برای classification نیاز دارد، سریع‌تر همگرا میشود.

: Activation function=linear

چون دو ناحیه با یک خط نمیتوان از هم جدایشان کرد، عمل classification را نمیتوان انجام داد.

Dataset= gaussian

: Activation function=tanh

چون دیتاست با یک خط میتواند classification را انجام دهد، به راحتی میتواند عمل جداسازی را انجام دهد.

: Activation function=sigmoid

چون دیتاست با یک خط میتواند classification را انجام دهد، به راحتی میتواند عمل جداسازی را انجام دهد. در مقایسه با tanh، چون شیب کمتری (۰,۲۵ شیب tanh) نسبت به tanh دارد و چون تغییر وزن‌ها به دلیل شکل sigmoid یا همه مثبت میشوند یا همه منفی، به این دو دلیل همگرایی در sigmoid کندتر اتفاق میفتد. چون دیتاست نسبت به circle در یک ناحیه متمرکز است، همگرایی زودتر اتفاق میفتد و در epoch کمتری به همگرایی میرسد.

: Activation function=ReLU

چون دیتاست با یک خط میتواند classification را انجام دهد، به راحتی میتواند عمل جداسازی را انجام دهد. جدای از این مورد، به دلیل شیب بیشتری که ReLU دارد باعث میشود همگرایی زودتر اتفاق بیفتد.

چون نسبت به circle تعداد ضلع‌های کمتری را برای classification نیاز دارد، سریع‌تر همگرا میشود.

: Activation function=linear

چون دو ناحیه را با یک خط میتوان از هم جدایشان کرد، عمل classification را میتوان انجام داد.

Dataset=spiral

: Activation function=tanh

چون دیتاست خیلی پیچیده است و یک hidden layer و تعداد نورون محدودی داریم، نمیتوان عمل classification را به خوبی انجام داد. تنها کاری که میتوان برای آن انجام داد overfitting است که هم تعداد hidden layerها و هم تعداد نورون‌ها را باید افزایش داد.

: Activation function=sigmoid

چون دیتاست خیلی پیچیده است و یک hidden layer و تعداد نورون محدودی داریم، نمیتوان عمل classification را به خوبی انجام داد. تنها کاری که میتوان برای آن انجام داد overfitting است که هم تعداد hidden layerها و هم تعداد نورون‌ها را باید افزایش داد.

: Activation function=ReLU

چون دیتاست خیلی پیچیده است و یک hidden layer و تعداد نورون محدودی داریم، نمیتوان عمل classification را به خوبی انجام داد. تنها کاری که میتوان برای آن انجام داد overfitting است که هم تعداد hidden layerها و هم تعداد نورون‌ها را باید افزایش داد.

: Activation function=linear

چون دیتاست خیلی پیچیده است و یک hidden layer و تعداد نورون محدودی داریم، نمیتوان عمل classification را به خوبی انجام داد. برای این مورد چون باید یک خط به هیچ وجه نمیتوان عمل classification را انجام داد، به هیچ صورتی نمیتوان از این تابع فعالساز نتیجه مطلوب گرفت.

(Q4

تحلیل‌های روی نمودار:

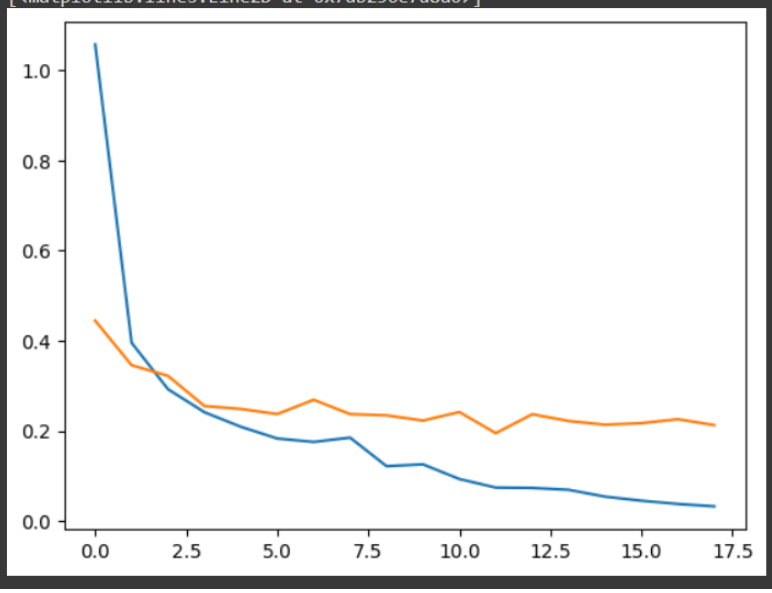
- مدل به دقت بالایی در هر دو مجموعه داده train و validation دست یافته که به ترتیب در دوره آخرین دوره به ۹۹,۶٪ و ۹۴٪ رسید. یعنی مدل توانسته است الگوهای موجود در داده ها را یاد بگیرد و به خوبی به داده های جدید تعمیم دهد.

- این مدل همچنین در هر دو مجموعه داده train و validation به loss کم دست یافت و به ترتیب در دوره گذشته به ۰,۰۳۲۶ و ۰,۲۱۲۹ رسید. این بدان معناست که پیش‌بینی‌های مدل به labelهای واقعی نزدیک بود.

- فاصله loss training data از loss validation data زیاد نیست، پس مدل overfit نشده. همچنین هردو loss کاهشی‌ست، پس مدل به خوبی یاد گرفته است.

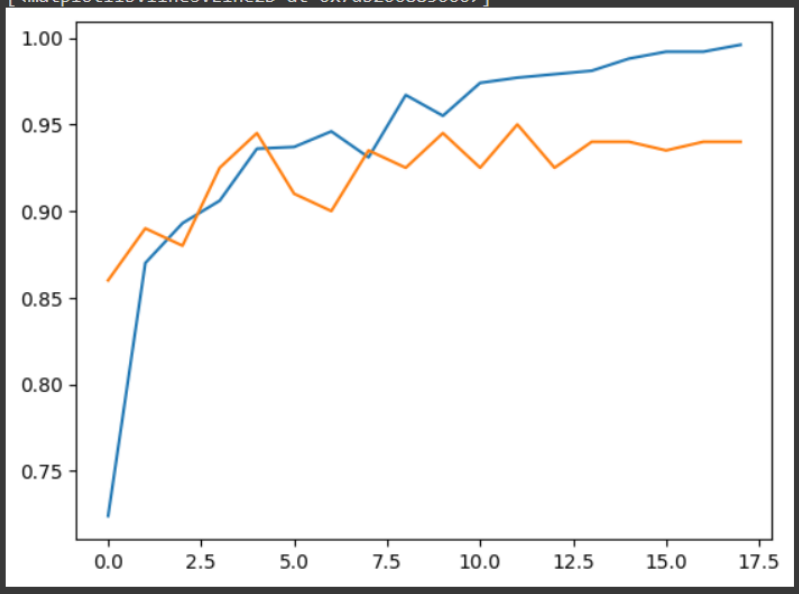
```
[25] # plot Loss
     plt.plot(MLP_model.history["loss"])
     plt.plot(MLP_model.history["val_loss"])
```

```
[<matplotlib.lines.Line2D at 0x7db256e7d8a0>]
```




```
# plot accuracy
plt.plot(MLP_model.history["accuracy"])
plt.plot(MLP_model.history["val_accuracy"])
```

```
[<matplotlib.lines.Line2D at 0x7db266885660>]
```

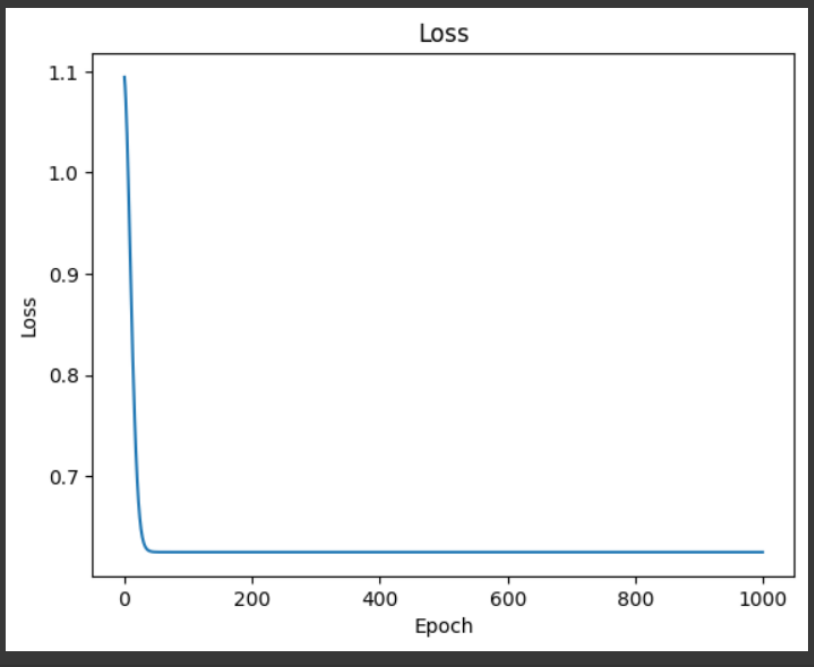


(Q5

```
mlp = MLP([weights_of_hiddenLayer, weights_of_outputLayer], [bias_of_hiddenLayer, bias_of_outputLayer])
losses = mlp.train(x_train, y_train, epochs=1000, learning_rate=0.06)
```

```
Epoch 0 loss: 1.09
Epoch 100 loss: 0.63
Epoch 200 loss: 0.63
Epoch 300 loss: 0.63
Epoch 400 loss: 0.63
Epoch 500 loss: 0.63
Epoch 600 loss: 0.63
Epoch 700 loss: 0.62
Epoch 800 loss: 0.62
Epoch 900 loss: 0.62
```

```
plt.plot(losses)
plt.title('Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.show()
```



(Q6

دلیل انتخاب دو hidden layer و تعداد نورون‌های هر لایه براساس تجربه و میزان دقت و loss بوده است. ولی به طور کلی، مدل با نورون بیشتر توانایی یادگیری توابع پیچیده‌تری را دارد، ولی باید حواسمان به overfit شدن تابع هم باشد.

برای جلوگیری از overfit، باید مدل را تا حد امکان ساده انتخاب کنیم که 1 hidden layer را برای همین انتخاب کردم. برای تعداد نورون هم با ۶۴ زیاد خوب عمل نکرد و با ۲۵۶ هم نزدیک به overfit شدن بود. پس ۱۲۸ نورون را انتخاب کردم.

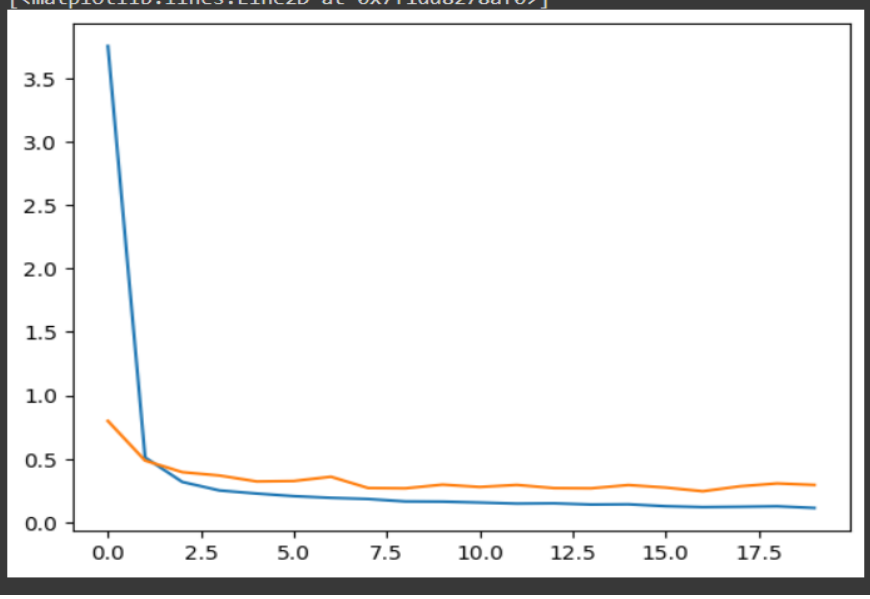
```
MLP_model = model_mnist.fit(x_train, y_train, batch_size=64, epochs=20, validation_split=0.2)

Epoch 1/20
750/750 [==============================] - 5s 6ms/step - loss: 3.7516 - accuracy: 0.8571 - val_loss: 0.7996 - val_accuracy: 0.8787
Epoch 2/20
750/750 [==============================] - 3s 4ms/step - loss: 0.5124 - accuracy: 0.8985 - val_loss: 0.4874 - val_accuracy: 0.9064
Epoch 3/20
750/750 [==============================] - 4s 5ms/step - loss: 0.3193 - accuracy: 0.9255 - val_loss: 0.3964 - val_accuracy: 0.9241
Epoch 4/20
750/750 [==============================] - 5s 6ms/step - loss: 0.2523 - accuracy: 0.9375 - val_loss: 0.3698 - val_accuracy: 0.9212
Epoch 5/20
750/750 [==============================] - 3s 5ms/step - loss: 0.2280 - accuracy: 0.9422 - val_loss: 0.3233 - val_accuracy: 0.9306
Epoch 6/20
750/750 [==============================] - 4s 5ms/step - loss: 0.2074 - accuracy: 0.9460 - val_loss: 0.3268 - val_accuracy: 0.9338
Epoch 7/20
750/750 [==============================] - 4s 6ms/step - loss: 0.1940 - accuracy: 0.9505 - val_loss: 0.3602 - val_accuracy: 0.9337
Epoch 8/20
750/750 [==============================] - 4s 5ms/step - loss: 0.1849 - accuracy: 0.9522 - val_loss: 0.2713 - val_accuracy: 0.9449
Epoch 9/20
750/750 [==============================] - 3s 4ms/step - loss: 0.1658 - accuracy: 0.9580 - val_loss: 0.2689 - val_accuracy: 0.9492
Epoch 10/20
750/750 [==============================] - 4s 5ms/step - loss: 0.1641 - accuracy: 0.9590 - val_loss: 0.2980 - val_accuracy: 0.9433
Epoch 11/20
750/750 [==============================] - 4s 5ms/step - loss: 0.1571 - accuracy: 0.9594 - val_loss: 0.2797 - val_accuracy: 0.9441
Epoch 12/20
750/750 [==============================] - 4s 5ms/step - loss: 0.1490 - accuracy: 0.9620 - val_loss: 0.2957 - val_accuracy: 0.9449
Epoch 13/20
750/750 [==============================] - 4s 5ms/step - loss: 0.1510 - accuracy: 0.9640 - val_loss: 0.2706 - val_accuracy: 0.9524
Epoch 14/20
750/750 [==============================] - 4s 6ms/step - loss: 0.1416 - accuracy: 0.9656 - val_loss: 0.2691 - val_accuracy: 0.9473
Epoch 15/20
750/750 [==============================] - 4s 5ms/step - loss: 0.1434 - accuracy: 0.9642 - val_loss: 0.2949 - val_accuracy: 0.9446
Epoch 16/20
750/750 [==============================] - 4s 5ms/step - loss: 0.1283 - accuracy: 0.9676 - val_loss: 0.2750 - val_accuracy: 0.9513
Epoch 17/20
750/750 [==============================] - 4s 6ms/step - loss: 0.1212 - accuracy: 0.9705 - val_loss: 0.2458 - val_accuracy: 0.9518
Epoch 18/20
750/750 [==============================] - 4s 5ms/step - loss: 0.1238 - accuracy: 0.9694 - val_loss: 0.2850 - val_accuracy: 0.9513
Epoch 19/20
750/750 [==============================] - 3s 4ms/step - loss: 0.1275 - accuracy: 0.9701 - val_loss: 0.3077 - val_accuracy: 0.9498
Epoch 20/20
750/750 [==============================] - 4s 5ms/step - loss: 0.1144 - accuracy: 0.9727 - val_loss: 0.2958 - val_accuracy: 0.9525
```

Plot Loss

```
plt.plot(MLP_model.history["loss"])
plt.plot(MLP_model.history["val_loss"])
```

```
[<matplotlib.lines.Line2D at 0x7f1dd8278af0>]
```



Plot Accuracy

```
plt.plot(MLP_model.history["accuracy"])
plt.plot(MLP_model.history["val_accuracy"])
```

```
[<matplotlib.lines.Line2D at 0x7f1de32413f0>]
```