دانشگاه صنعتی خواجه نصیرالدین طوسی
دانشکدهٔ مهندسی برق - گروه مهندسی کنترل

یادگیری ماشین

پاسخ مینی پروژه دوم

نام و نام خانوادگی: سیدمحمد حسینی
شمارهٔ دانشجویی: ۹۹۰۱۳۹۹
تاریخ: مهرماه ۱۴۰۲

فهرست مطالب

فهرست تصاویر

فهرست جداول

فهرست برنامه‌ها

۱ سوال اول

۱.۱ بخش اول

سوال:
فرض کنید در یک مسألهٔ طبقه بندی دوکلاسه، دو لایهٔ انتهایی شبکهٔ شما فعال ساز ReLU و سیگموید است. چه اتفاقی می افتد؟

پاسخ:
توابع فعال‌ساز Relu و Sigmoid جزو پرکاربردترین اجزاء در یک مدل یادگیری ماشین هستند که به‌منظور اضافه‌کردن خاصیت غیرخطی به مدل مورداستفاده قرار می‌گیرند. به‌منظور تحلیل اثر این دو تابع فعال‌ساز بهتر است ابتدا خواص هرکدام را به‌صورت مجزا بررسی کنیم.

- تابع فعال‌سازی Sigmoid:
تابع سیگموید مقداری بین ۰ و ۱ خروجی می‌دهد که می‌تواند به‌عنوان احتمال تفسیر شود. این تابع فعال‌سازی اصولاً به‌عنوان فعال‌ساز در آخرین لایه مدل‌های طبقه‌بندی استفاده می‌شود؛ زیرا تفسیر احتمالی مستقیمی از خروجی ارائه می‌دهد. از دیگر خواص تابع سیگموید مشتق‌پذیری آن است که از لحاظ برنامه‌نویسی کار را بسیار ساده می‌کند؛ زیرا همان‌طور که در معادله ۱ مشاهده می‌کنید مشتق تابع سیگموید رابطه‌ای است متشکل از خود تابع سیگموید.

$$\frac{d}{dx}\sigma(x) = \sigma(x) \cdot (1 - \sigma(x)) \quad (۱)$$

- تابع فعال‌سازی ReLU:
تابع ReLU ساده‌ترین تابع فعال‌سازی غیرخطی است که مورداستفاده قرار می‌گیرد، این تابع به‌ازای مقادیر مثبت تابع همانی است و به‌ازای مقادیر منفی عدد ۰ را به‌عنوان خروجی در نظر می‌گیرد که این مسئله به‌سادگی محاسبات هنگام Back propagation بسیار کمک می‌کند. مشتق این تابع به‌ازای مقادیر مثبت برابر ۱ و به‌ازای مقادیر منفی برابر ۰ است.

حال باتوجه‌به توضیحات فوق می‌توانیم یک شبکه که دولایه انتهایی آن متشکل از یک تابع ReLU و Sigmoid است را بررسی کنیم. در این شبکه اطلاعات به‌دست‌آمده از backbone شبکه به لایه یکی مانده به آخر ارسال می‌شود که با انجام یک عملیات خطی این اطلاعات به محیط دیگری تصویر می‌شوند که می‌توانند مقادیر مثبت و منفی را با هر اندازه‌ای داشته باشند، این مسئله به مقادیر موجود در Wight و Bias نورون بستگی دارد. حال خروجی نرون‌ها به لایه فعال‌ساز ارسال می‌شوند که در نتیجه آن مقادیر مثبت تبدیل خطی نگه داشته می‌شوند و مقادیر منفی به عدد ۰ تصویر می‌شوند. باید توجه داشت که فعال‌ساز ReLU به تعبیری همانند یک ماژول Attention عمل می‌کند و در فرایند آموزش شبکه Wight و Bias نرون‌ها را به سمتی متمایل می‌کند که در صورت پیدانکردن Feature معنی‌دار از ورودی خود، یک عدد منفی تولید کند. ایراد فعال‌ساز ReLU این است که اگر لایه‌های پشت‌هم در شبکه، همگی مقدار مثبت تولید کنند، تمام تبدیل‌های خطی موجود در این لایه‌ها می‌توانند با یک تبدیل خطی شبیه‌سازی شوند که این مسئله باعث کاهش پیچیدگی مدل ما خواهد شد؛ بنابراین استفاده از روش‌های رگولاریزیشن و Batch Normalization در کنار توابع فعال‌سازی که به‌صورت پیوسته رفتار غیرخطی دارند موجب رفع این مشکل خواهد شد. در ادامه این مسئله خروجی لایه یکی مانده به آخر وارد لایه تصمیم‌گیری خواهد شد؛ ورودی این لایه تماماً اعداد مثبت است و تبدیل خطی برای اینکه بتواند کلاس مثبت را از منفی جدا کند باید به‌گونه‌ای تنظیم شود که بتواند ورودی‌های تماماً مثبت را به یک عدد مثبت یا منفی تبدیل کند؛ زیرا تابع سیگموید در نقطه ۰ مقدار ۰.۵ را دارد و اگر تبدیل خطی نتواند اعداد مثبت و منفی را از ورودی‌های تماماً مثبت استخراج کند، تابع سیگموید توانایی ایجاد خروجی مناسب را ندارد.

۲.۱ بخش دوم

سوال:

یک جایگزین برای ReLU تابع ELU می‌باشد . ضمن محاسبه گرادیان آن، حداقل یک مزیت آن را مطرح کنید

پاسخ:

ابتدا مشتق این تابع را محاسبه می‌کنیم:

$$ELU(x) = \begin{cases} x & \text{if} x > 0 \\ \alpha(e^x - 1) & \text{if} x \leq 0 \end{cases} \Rightarrow$$

$$\frac{d}{dx}ELU(x) = \begin{cases} 1 & \text{if} x > 0 \\ \alpha e^x & \text{if} x \leq 0 \end{cases} \quad (۲)$$

همانطور که در معادله ۲ مشخص است مشتق این تابع برای مقادیر مثبت با تابع ReLU تفاوتی ندارد اما برای مقادیر منفی برابر e^x می‌باشد. همانطور که در قسمت قبل مطرح شد تابع ReLU می‌تواند موجب این شود که در عمل تعداد لایه‌های شبکه کاهش پیدا کند اما با استفاده از تابع فعال‌ساز ELU و استفاده از تکنیک‌های مناسب، می‌توانیم به این مشکل غلبه کنیم. علاوه‌بر این تابع ELU در مشتق خود ناپیوستگی ندارد و بر خلاف ReLU مقدار آن به ازای مقادیر منفی به آرامی برابر $-\alpha$ می‌شود.[۱]

۳.۱ بخش سوم

در ابتدا با استفاده از توزیع یکنواخت تعداد ۲۰۰۰ داده ایجاد می‌کنیم و برچسب نقاطی که داخل مثلث مورد نظر هستند را به مقدار ۱ و برچسب بقیه نقاط را ۰ می‌کنیم. این عملات توسط کد زیر انجام گرفته است.

```
1
2  def point_in_triangle(point, v1, v2, v3):
3    """Check if point (px, py) is inside the triangle with vertices v1, v2, v3."""
4    # Unpack vertices
5    x1, y1 = v1
6    x2, y2 = v2
7    x3, y3 = v3
8    px, py = point
9
10   # Vectors
11   v0 = (x3 - x1, y3 - y1)
12   v1 = (x2 - x1, y2 - y1)
13   v2 = (px - x1, py - y1)
14
15   # Dot products
16   dot00 = np.dot(v0, v0)
17   dot01 = np.dot(v0, v1)
18   dot02 = np.dot(v0, v2)
19   dot11 = np.dot(v1, v1)
20   dot12 = np.dot(v1, v2)
```

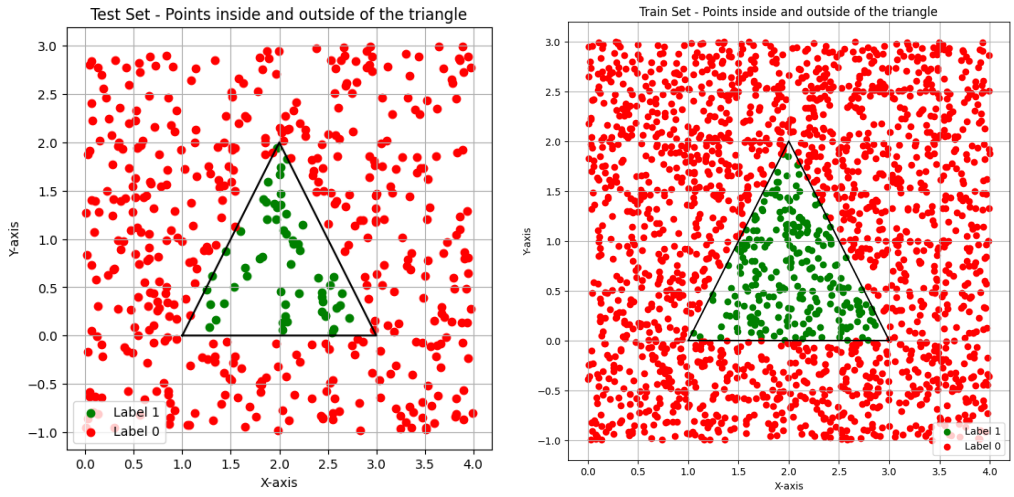
```
    # Barycentric coordinates
    invDenom = 1 / (dot00 * dot11 - dot01 * dot01)
    u = (dot11 * dot02 - dot01 * dot12) * invDenom
    v = (dot00 * dot12 - dot01 * dot02) * invDenom

    # Check if point is in triangle
    return (u >= 0) and (v >= 0) and (u + v < 1)

# Triangle vertices
v1 = (1, 0)
v2 = (2, 2)
v3 = (3, 0)

# Generate random points
np.random.seed(53)

x_coords = np.random.uniform(0, 4, 2000)
y_coords = np.random.uniform(-1, 3, 2000)
x_train = np.column_stack((x_coords, y_coords))

x_coords = np.random.uniform(0, 4, 500)
y_coords = np.random.uniform(-1, 3, 500)
x_test = np.column_stack((x_coords, y_coords))

# Label points based on whether they are inside the triangle
y_train = np.array([point_in_triangle(pt, v1, v2, v3) for pt in x_train]).astype(int)
y_test = np.array([point_in_triangle(pt, v1, v2, v3) for pt in x_test]).astype(int)
```

Code :۱ Example Python code

کد فوق به منظور ایجاد ۲۰۰۰ نقطه و بررسی اینکه هر نقطه درون مختصات مثلث قرار دارد یا خیر نوشته شده است. تابع `point_in_triangle` بررسی می‌کند که آیا یک نقطه داخل مثلثی با رئوس داده شده قرار دارد. مختصات رئوس مثلث و نقطه مورد نظر به تابع داده می‌شودو سپس نسبت مختصات نقطه به مثلث مشخص می‌شود. نقاط تصادفی برای مجموعه‌های آموزشی و تست تولید و با استفاده از تابع مذکور برچسب‌گذاری می‌شوند. همانطور که در شکل ۱ نمایش داده شده است، دو مجموعه داده به منظور آموزش و ارزیابی مدل تولید شده است.

در ادامه یک مدل MLP روی این مجموعه داده آموزش دیده است و نتایج آن به همراه Decision boundary آن نمایش داده شده است. مدل اول MLP با استفاده از توابع ReLU ایجاد شده است که جزییات آن به شرح زیر است:

```
MLP(
(layers): Sequential(
(0): Linear(in_features=2, out_features=8, bias=True)
```



(ب) (آ)

شکل ۱: مجموعه داده تولید شده

```
(1): ReLU()
(2): Linear(in_features=8, out_features=64, bias=True)
(3): ReLU()
(4): Linear(in_features=64, out_features=8, bias=True)
(5): ReLU()
(6): Linear(in_features=8, out_features=1, bias=True)
(7): Sigmoid()
)
)
```

در ادامه مدل فوق با config زیر به میزان ۱۵۰ Epoch آموزش داده شده است.

```
device = "cuda" if torch.cuda.is_available else "cpu"
model = MLP(input_size=2, hidden_size1=8, hidden_size2=64, hidden_size3=8, output_size=1).to(
device)
optimizer = optim.Adam(model.parameters(), lr=0.001)
criterion = nn.BCELoss()  # For binary classification
# DataLoader
train_loader = DataLoader(TensorDataset(x_train_tensor, y_train_tensor), batch_size=128,
shuffle=True)
test_loader = DataLoader(TensorDataset(x_test_tensor, y_test_tensor), batch_size=512, shuffle
=False)
```

Code :۲ Configuration

کد زیر به منظور اجرای حلقه آموزش نوشته شده است. در این کد، چندین متغیر برای نگه‌داری اطلاعات مانند تاریخچه‌ی خطاها و معیارهای متریکی مورد استفاده قرار گرفته است. سپس یک حلقه تکرار برای ایپاک‌های مختلف اجرا می‌شود. در هر ایپاک، داده‌های آموزشی بارگذاری شده و مدل در حالت آموزش قرار می‌گیرد. سپس خطا محاسبه می‌شود و بهینه‌سازی می‌شود. معیارهای متریکی مورد نظر نیز برای داده‌های آموزشی توسط توابعی که مستقلا پیاده‌سازی شده‌اند محاسبه می‌شوند. سپس مدل به حالت ارزیابی در آورده می‌شود و برای داده‌های تست خطا و معیارهای متریکی محاسبه می‌شود. در انتهای آموزش، این اطلاعات برای تحلیل و نمایش خروجی‌های آموزش به دست می‌آید.

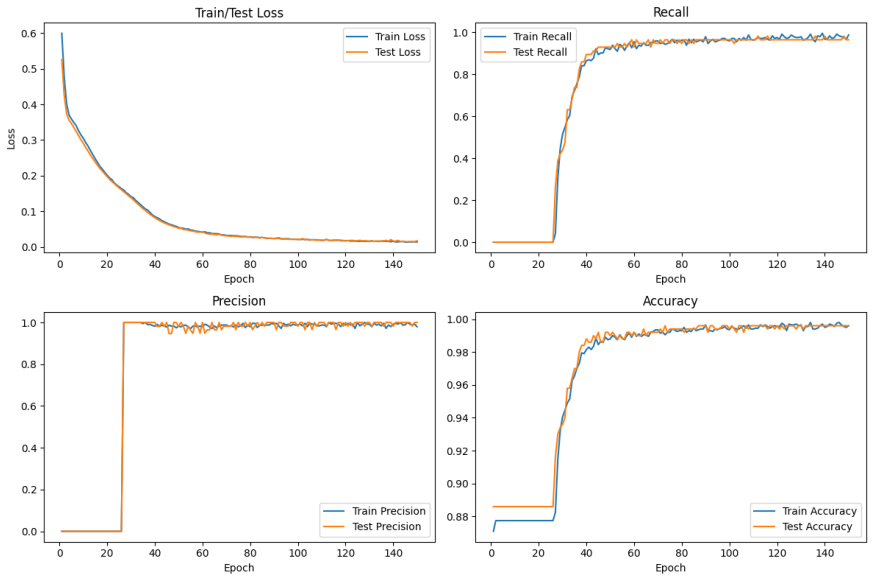
```
num_epochs = 150
train_loss_hist = []
test_loss_hist = []
train_metrics = []
test_metrics = []

for epoch in range(num_epochs):
    loop = tqdm(train_loader)
    model.train()
    train_loss = 0.0
    train_TP, train_FP, train_TN, train_FN = 0, 0, 0, 0

    print("train")
    for inputs, labels in loop:
        inputs = inputs.to(device)
        labels = labels.to(device)

        outputs = model(inputs)
        loss = criterion(outputs, labels)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        train_loss += loss.item()

        TP, FP, TN, FN = calculate_metrics(outputs, labels)
        train_TP += TP
        train_FP += FP
        train_TN += TN
        train_FN += FN

        loop.set_postfix(
            epoch=epoch+1,
            total_loss=train_loss / len(train_loader),
        )
```

```
    train_metrics.append((train_TP, train_FP, train_TN, train_FN))
    train_loss_hist.append(train_loss / len(train_loader))

    model.eval()
    torch.cuda.empty_cache()
    test_loss = 0.0
    test_TP, test_FP, test_TN, test_FN = 0, 0, 0, 0
    print("Test:")

    with torch.no_grad():
        loop = tqdm(test_loader)
        for inputs, labels in loop:
            inputs = inputs.to(device)
            labels = labels.to(device)

            outputs = model(inputs)
            loss = criterion(outputs, labels)

            test_loss += loss.item()

            TP, FP, TN, FN = calculate_metrics(outputs, labels)
            test_TP += TP
            test_FP += FP
            test_TN += TN
            test_FN += FN

            loop.set_postfix(
                loss=loss.item(),
                total_loss=test_loss / len(test_loader),
            )
    test_metrics.append((test_TP, test_FP, test_TN, test_FN))
    test_loss_hist.append(test_loss / len(test_loader))

```

آموزش حلقه ۳: Code

۱.۳.۱ نتایج MLP با تابع فعال‌ساز ReLU

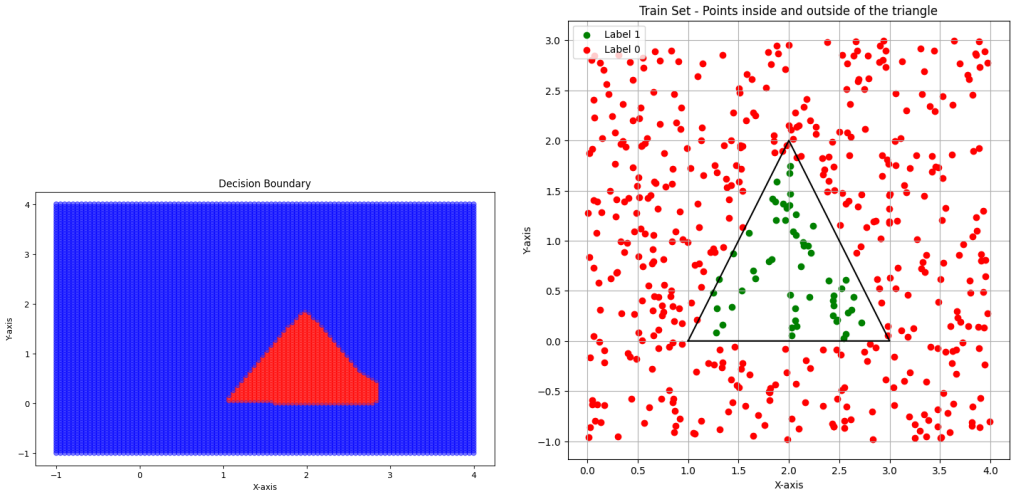
در شکل ۲ نتایج مربوط به این آموزش نشان داده شده است. همانطور که مشخص است فرآیند آموزش برای مدل MLP به درستی انجام شده و مدل روی دیتاست آموزش over fit نشده است این درحالی است که میزان متریک‌ها برای هر دو دیتاست به ۱ بسیار نزدیک شده است.



شکل ۲: نمودارهای متریک و تابع هزینه حین آموزش

در ۳(آ) نتیجه عملکرد مدل روی دیتای ارزیابی قابل ملاحضه است و همانطور که مشخص است تنها ۲ نقطه در راس بالایی مثلث اشتباه طبقه بندی شده‌اند این درحالی است که هیچ کلاسی که متعلق به داخل دایره بوده، به اشتباه به عنوان یک کلاس در خارج از دایره تشخیص داده نشده است، به عبارت دیگر False Negative برابر صفر است و Precision برابر ۱ است.

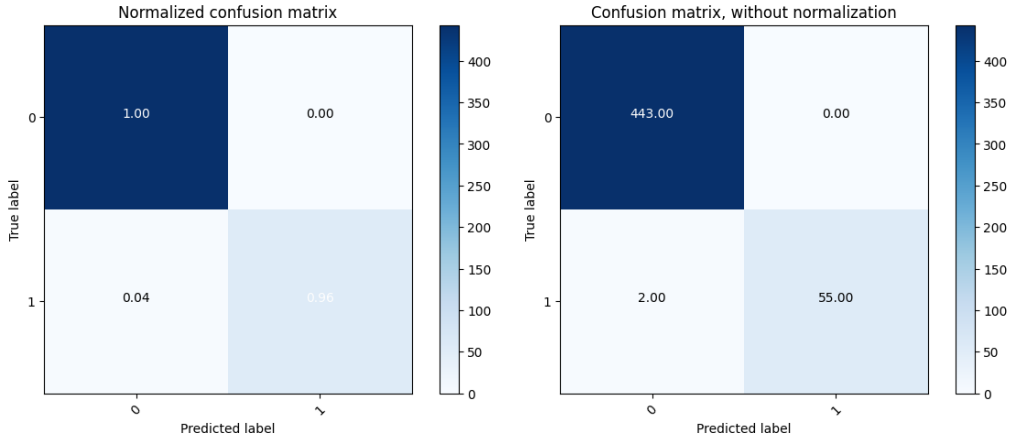
در انتها در ۳(ب) مرز تصمیم مدل مشخص است که با توجه با این نمودار مرز حوالی راس پایین سمت راست مثلث از دقت کافی برخوردار نیست.



(ب) (آ)

شکل ۳: نتایج مدل MLP و مرز تصمیم مربوط به مدل

در شکل ۴ ماتریس درهم‌ریختگی به صورت نرمال و غیرنرمال نشان داده شده است که نشان می‌دهد مدل تنها دو نقطه را که مطعلق به داخل مثل بوده است به اشتباه به عنوان کلاس خارج از مثلث تشخیص داده است.



(ب) (آ)

شکل ۴: Matrix Confusion

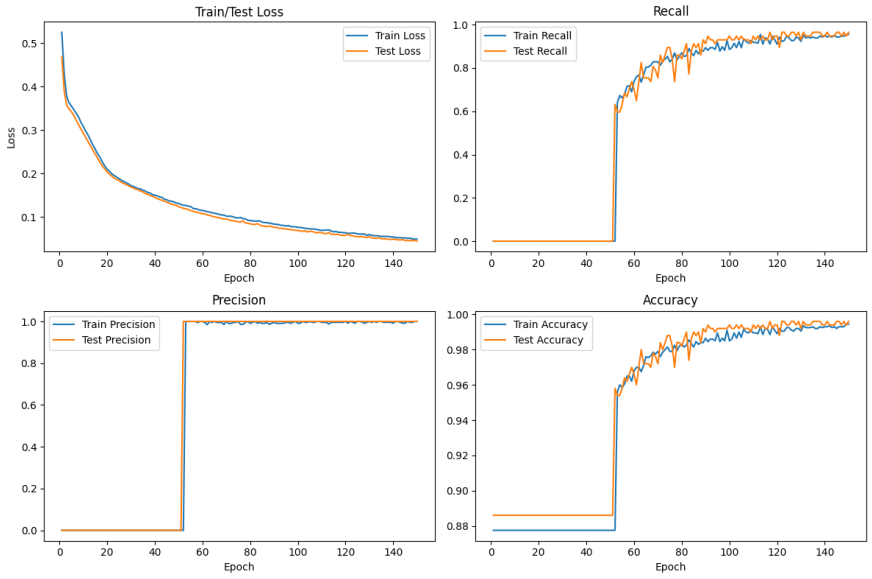
۲.۳.۱ نتایج MLP با تابع فعال‌ساز ELU

این مدل با استفاده از تابع ELU ساخته شده است و جزییات آن به شرح زیر می‌باشد:

```
MLP(
  (layers): Sequential(
```

```
    (0): Linear(in_features=2, out_features=8, bias=True)
    (1): ELU(alpha=1.0)
    (2): Linear(in_features=8, out_features=64, bias=True)
    (3): ELU(alpha=1.0)
    (4): Linear(in_features=64, out_features=8, bias=True)
    (5): ELU(alpha=1.0)
    (6): Linear(in_features=8, out_features=1, bias=True)
    (7): Sigmoid()
  )
)
```

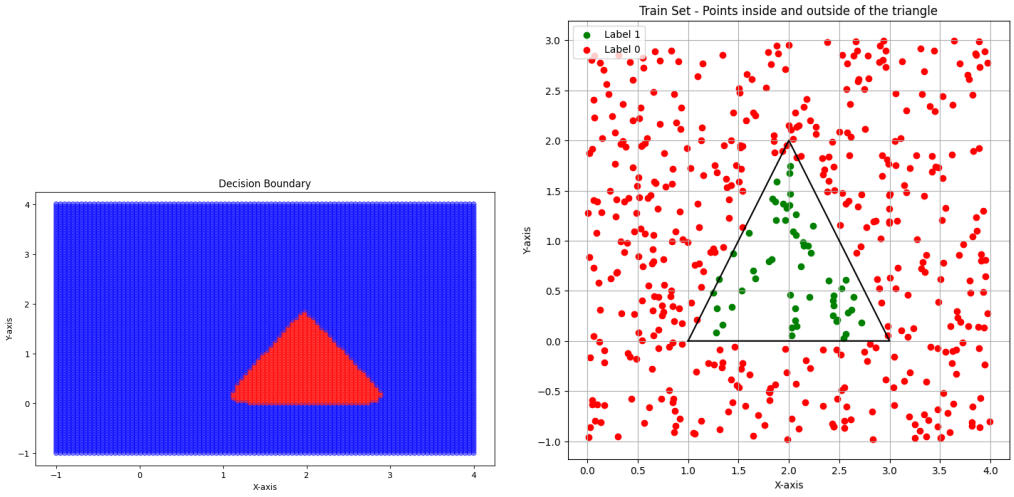
در شکل ۵ نتایج مربوط به این آموزش نشان داده شده است. همانطور که مشخص است فرآیند آموزش برای مدل MLP با تابع فعال‌ساز ELUبه درستی انجام شده و مدل روی دیتاست آموزش over fit نشده است این درحالی است که شیب نمودار هزینه همچنان کاهشی می‌باشد.در ادامه میزان متریک‌ها برای هر دو دیتاست به ۱ بسیار نزدیک شده است.



شکل ۵: نمودارهای متریک و تابع هزینه حین آموزش

در ۶(آ) نتیجه عملکرد مدل روی دیتای ارزیابی قابل ملاحضه است و همانطور که مشخص است تنها ۲ نقطه در راس بالایی مثلث اشتباه طبقه بندی شده‌اند این درحالی است که هیچ کلاسی که متعلق به داخل دایره بوده، به اشتباه به عنوان یک کلاس در خارج از دایره تشخیص داده نشده است، به عبارت دیگر False Negative برابر صفر است و Precision برابر ۱ است.

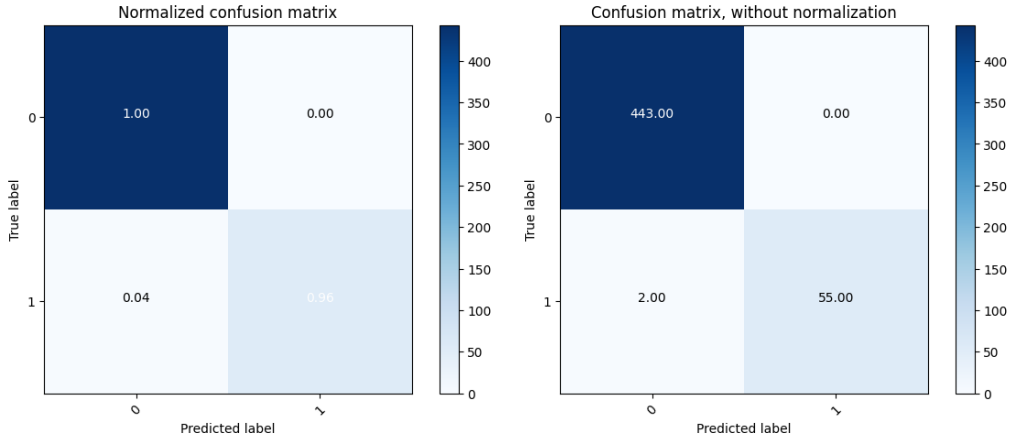
در ۶(ب) مرز تصمیم مدل مشخص است که با توجه به این نمودار و مقایسه آن با ۳(ب)، مشخص است که نتایج بدست آمده از این مدل توانایی تعمیم‌پذیری بیشتری دارند و هر سه راس این مثلث به یک شکل هستند.



(ب) (آ)

شکل ۶: نتایج مدل MLP و مرز تصمیم مربوط به مدل

در شکل ۷ماتریس درهم‌ریختگی به صورت نرمال و غیرنرمال نشان داده شده است که نشان می‌دهد مدل تنها دو نقطه را که مطعلق به داخل مثل بوده است به اشتباه به عنوان کلاس خارج از مثلث تشخیص داده است.



(ب) (آ)

شکل ۷: Matrix Confusion

۳.۳.۱ نتایج مدل بر مبنای mcculloch-pitts نرون

مدل‌های مبتنی بر نرون‌های mcculloch-pitts به ما این امکان را می‌دهند که با استفاده از دانش انسانی، بهترین پاسخ را برای مسئله خود بدست بیاوریم. راه حلی که می‌توانیم نقاط داخل مثلث را شناسایی کنیم حاصل ترکیب AND ۳ گزاره متفاوت است؛ هر یک از این گذاره‌ها نشان‌دهنده این است که نقطه مد نظر ما نسبت به خط گذرنده از هر ضلع مثلث چه وضعیتی دارد. بدین منظور باید سه معادله

خط بدست آوریم و با قرار دادن نقاط اطراف خط مشخص کنیم که تغییر علامت چه زمانی رخ می‌دهد و با چه ترکیب منطقی از این تغییر علامت‌ها می‌توانیم مثلث را پیدا کنیم.

در شکل ۵ نتایج مربوط به این آموزش نشان داده شده است. همانطور که مشخص است فرآیند آموزش برای مدل MLP با تابع فعال‌ساز ELUبه درستی انجام شده و مدل روی دیتاست آموزش over fit نشده است این درحالی است که شیب نمودار هزینه همچنان کاهشی می‌باشد.در ادامه میزان متریک‌ها برای هر دو دیتاست به ۱ بسیار نزدیک شده است. روش محاسبه هر یک از این معادلات خط با قرار دادن یک جفت از راس‌های مثلث در معادله زیر می‌باشد:

$$y = w(x - x_1) + y_1, \quad w = \frac{y_1 - y_2}{x_1 - x_2} \tag{۳}$$

با استفاده از رابطه فوق سه جفت مقدار برای هر معادله خط بدست آمد که این مقادیر عبارتند از $(-2, 1)$، $(2, 1)$ و $(0, 1)$ که سه ضلع مثلث را تشکیل می‌دهند. در ادامه یک نرون لازم است تا عملیات منطقی AND را انجام دهد. با جایگذاری نقاط مختلف از صفحه در معادلات خط بدست آمده و مقایسه آنها رابطه منطقی زیر بدست می‌آید:

$$l'_1 + l'_2 + l_3$$

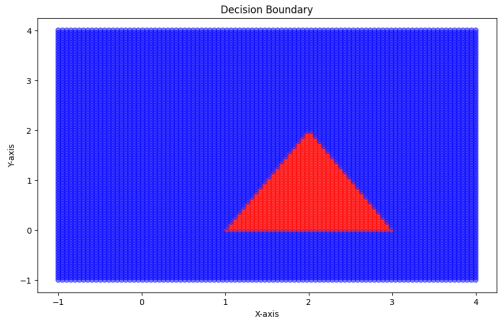
مدل مرتبط با روابط فوق به شکل زیر در پایتون قابل پیاده‌سازی می‌باشد:

```python
#define muculloch pitts
class McCulloch_Pitts_neuron():

  def __init__(self , weights ,bias, threshold):
    self.weights = np.array(weights).reshape(-1, 1)     #define weights
    self.threshold = threshold    #define threshold
    self.bias = np.array(bias)

  def model(self , x):
    #define model with threshold
    return (x.T @ self.weights + self.bias >= self.threshold).astype(int)

def model(x):
    neur1 = McCulloch_Pitts_neuron([-2, 1],2, 0)
    neur2 = McCulloch_Pitts_neuron([2, 1], -6, 0)
    neur3 = McCulloch_Pitts_neuron([0, 1], 0, 0)
    neur4 = McCulloch_Pitts_neuron([1, 1, 1], 0, 3)

    z1 = neur1.model(np.array(x))
    z2 = neur2.model(np.array(x))
    z3 = neur3.model(np.array(x))
    z4 = np.squeeze(np.array([1-z1, 1-z2, z3]), axis=-1)
    z4 = neur4.model(z4)

    # 3 bit output
    # return str(z1) + str(z2)
    return z4
```

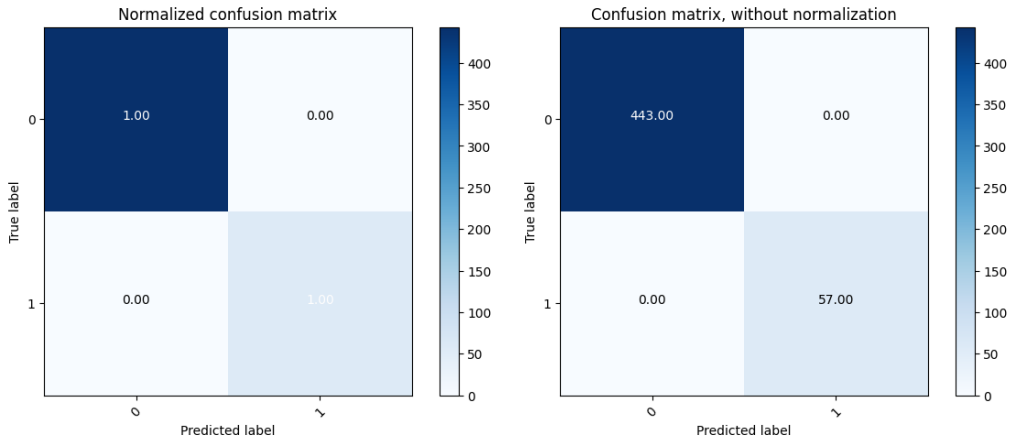
```
۲۸
۲۹
```

آموزش حلقه ۴: Code

همانطور که در شکل ؟؟ مشخص است یک مثلث دقیق بدست آمده است و انتظار می‌رود که نتایج ماتریس درهم‌ریختگی آن بهترین وضعیت ممکن باشد که این مسئله در شکل ۹ قابل مشاهده است.



شکل ۸: mcculloch-pitts و مرز تصمیم مربوط به مدل



(ب) (آ)

شکل ۹: Matrix Confusion

مراجع

[۱] Avinash. Yuan. Learning Machine: Cheatsheet Activation، Functions. n.d. Accessed: May ۱۸، ۲۰۲۴.