به نام خدا

دانشگاه تهران

دانشکده مهندسی

برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق

تمرین اول

نام و نام خانوادگی	بهراد موسایی شیرمحمد - محمد جواد رنجبر کلهرودی
شماره دانشجویی	۸۱۰۱۰۱۱۷۳-۸۱۰۱۰۱۲۷۸
تاریخ ارسال گزارش	۱۴۰۲.۰۸.۱۷

فهرست

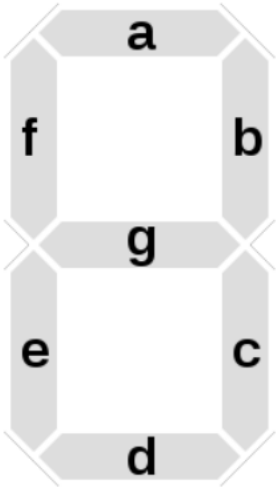
شکل‌ها

جدول‌ها

پاسخ ۱. شبکه عصبی Mcculloch-Pitts

۱-۱. نمایشگر segment-۷

طور که در شکل نمایشگر هفت قسمتی شود، مشاهده می‌شود. برای نشان دادن اعداد به صورت دیجیتالی استفاده می شود و دارای هفت LED باشد (دیود نورانی) که روشن یا خاموش بودن این LED ها اعداد .را به ما نشان دهد.



شکل ۱: نمایشگر هفت قسمتی

مثلاً برای نمایش عدد ۱ کافیست که فقط حرف **b** و **c** روشن باشند.

فرض کنید یک نمایشگر segment-۷ داریم که همواره یکی از اعداد ۹ ,۸ ,۷ ,۶ را نمایش می دهد. در ادامه قصد داریم که با داشتن وضعیت هر LED (روشن یا خاموش بودن) یک شبکه عصبی بسازیم عدد در حال نمایش را تشخیص دهد. نحوه نمایش این اعداد را در شکل زیر مشاهده می کنید.

شکل ۲: نحوه نمایش اعداد ۶و ۷ و ۸ و۹

پاسخ:

می‌توان به صورت دستی این وزن‌ها را محاسبه کرد، اما در اینجا ما وزن‌ها کد داده‌ایم. کد ما یک کلاس به نام McCulloch_Pitts_neuron ایجاد می‌کند، که یک نورون مدل McCulloch-Pitts را نمایش می‌دهد. این کد از کتابخانه NumPy برای محاسبات عددی استفاده می‌کند و یک ماتریس دو بعدی به نام X ایجاد می‌کند. این ماتریس شامل تمام ترکیب‌های ممکن از اعداد صحیح ۰ و ۱ با ابعاد معین (۱۲۸ سطر و ۷ ستون) است.

class McCulloch_Pitts_neuron: این کلاس یک نورون McCulloch-Pitts را نمایش می‌دهد و دارای دو ویژگی اصلی است:

weights: این ویژگی وزن‌های نورون را نشان می‌دهد. وزن‌ها مشخص می‌کنند که ورودی‌هایی با اهمیت متفاوت به نورون وارد می‌شوند

. threshold: این ویژگی آستانه نورون را نشان می‌دهد. آستانه تصمیم‌گیری است که نورون بر اساس محصول داخلی وزن‌ها و ورودی‌ها تصمیم می‌گیرد که خروجی ۰ یا ۱ باشد.

model: این متد به کلاس اضافه شده است و وظیفه‌اش این است که با ورودی x، خروجی نورون را محاسبه کند. در متد model، ابتدا محصول داخلی بین وزن‌ها و ورودی‌ها با استفاده از np.dot محاسبه می‌شود. سپس با مقایسه نتیجه با آستانه ، تصمیم‌گیری می کند: اگر محصول داخلی بیشتر یا مساوی آستانه باشد، خروجی نورون برابر با ۱ است. در غیر این صورت، خروجی برابر با ۰ خواهد بود.

کد اصلی ما پس از تعریف کلاس McCulloch_Pitts_neuron، یک ماتریس به نام X را ایجاد می‌کند. این ماتریس X شامل تمام ترکیب‌های ممکن از اعداد صحیح ۰ و ۱ با ابعاد معین (۱۲۸ سطر و ۷ ستون) است. این ماتریس احتمالاً برای تست و تحلیل عملکرد نورون McCulloch-Pitts استفاده می‌شود. برای تحلیل عملکرد نورون McCulloch-Pitts با این کد، شما می‌توانید یک شیء از کلاس McCulloch_Pitts_neuron ایجاد کنید و وزن‌ها و آستانه مورد نظر خود را به عنوان ورودی به این نورون ارائه دهید. سپس با استفاده از ماتریس X به عنوان ورودی‌ها، می‌توانید خروجی نورون را برای تمام ترکیب‌های ممکن از ورودی‌ها محاسبه کنید و نتایج را برای تحلیل به کار ببریم.

```
(128, 7) int64 [[0 0 0 0 0 0 0]
 [0 0 0 0 0 0 1]
 [0 0 0 0 0 1 0]
 [0 0 0 0 0 1 1]
 [0 0 0 0 1 0 0]
 [0 0 0 0 1 0 1]
 [0 0 0 0 1 1 0]
 [0 0 0 0 1 1 1]
 [0 0 0 1 0 0 0]
 [0 0 0 1 0 0 1]
 [0 0 0 1 0 1 0]
 [0 0 0 1 0 1 1]
```

شکل ۳: نتایج بخش اول پیاده سازی

کدی که ارائه داده‌ایم، از کلاس McCulloch_Pitts_neuron برای ایجاد چهار نورون مختلف با وزن‌ها و آستانه‌های متفاوت استفاده می‌کند. سپس با استفاده از این نورون‌ها، تمام ترکیب‌های ممکن از ورودی‌ها که در ماتریس X تعریف شده است، تست می‌شوند تا تشخیص داده شود که هر ترکیب به کدام رقم از اعداد ۶، ۷، ۸ و ۹ تعلق دارد یا آیا هیچکدام از این اعداد تشخیص داده نمی‌شود.

در اینجا توضیحی برای چگونگی عملکرد کد ارائه می‌شود:

ابتدا چهار نورون به نام n۶، n۷، n۸ و n۹ با وزن‌ها و آستانه‌های مختلف ایجاد می‌شوند. این نورون‌ها آماده‌اند تا وظیفه تشخیص اعداد ۶، ۷، ۸ و ۹ را انجام دهند.

سپس با استفاده از یک حلقه for، هر ترکیب ممکن از ورودی‌ها (که در ماتریس X قرار دارند) به ترتیب تست می‌شود. برای هر ترکیب ورودی، خروجی نورون‌های n۶، n۷، n۸ و n۹ محاسبه می‌شود. به ازای هر نورون، اگر محصول داخلی بین وزن‌ها و ورودی‌ها بیشتر یا مساوی آستانه آن نورون باشد، آن نورون خروجی ۱ می‌دهد و در غیر این صورت خروجی ۰ خواهد بود.

سپس با استفاده از دستورات if-elif، تصمیم گرفته می‌شود که هر ترکیب ورودی به کدام رقم اعداد ۶، ۷، ۸ و ۹ تعلق دارد یا آیا هیچ کدام از این اعداد تشخیص داده نمی‌شوند. به عبارت دقیق‌تر، اگر o۱ برابر با ۱ باشد، ترکیب ورودی به عنوان عدد ۶ تشخیص داده می‌شود و همینطور برای o۲ به عنوان عدد ۷، o۳ به عنوان عدد ۸ و o۴ به عنوان عدد ۹. اگر هیچ کدام از این شروط برقرار نشود، به عنوان " No digit detected" نمایش داده می‌شود. این کد در واقعیت برای تست و تحلیل عملکرد چهار نورون مختلف در تشخیص اعداد از ۶ تا ۹ استفاده می‌شود و نتایج را به شما نمایش می‌دهد.

```
n6 = McCulloch_Pitts_neuron([1,-1,1,1,1,1,1], 6)
n7 = McCulloch_Pitts_neuron([1,1,1,-1,-1,-1,-1], 3)
n8 = McCulloch_Pitts_neuron([1,1,1,1,1,1,1], 7)
n9 = McCulloch_Pitts_neuron([1,1,1,1,-1,1,1], 6)
```

شکل ۴: وزن دهی های صورت گرفته

وزن‌دهی به این صورت در مدل نورون McCulloch-Pitts به منظور تعیین اهمیت و وزن مختلف ورودی‌ها در فرآیند تصمیم‌گیری نورون انجام می‌شود. وزن‌ها در این مدل نشان دهنده نیروهایی هستند که به ورودی‌ها اختصاص داده می‌شوند و تعیین می‌کنند که هر ورودی چقدر می‌تواند تأثیرگذار باشد.

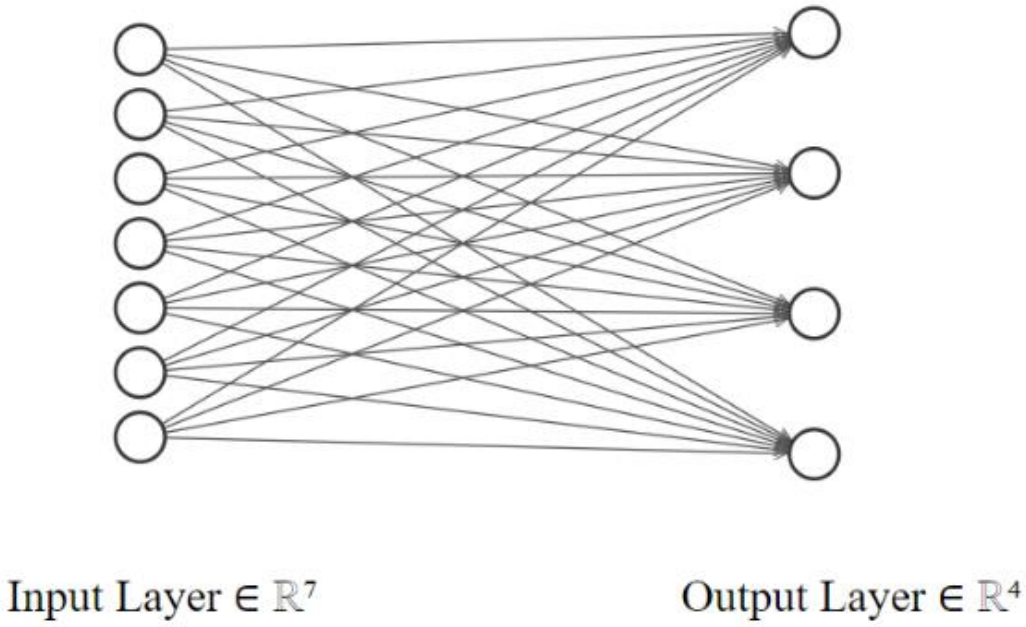
در کد ما، وزن‌های مختلف به ازای هر نورون به صورت دستی تعیین شده‌اند. این اعداد و وزن‌ها به طور عملی معمولاً به صورت تجربی یا با توجه به داده‌های آموزشی مشخص می‌شوند. عملیات آموزش معمولاً شامل تنظیم وزن‌ها به نحوی است که نورون بتواند الگوهای خاصی را تشخیص دهد. برای مثال، در کد شما، نورون n۶ با وزن‌های [۱, -۱, ۱, ۱, ۱, ۱, ۱] و آستانه ۶ تنظیم شده است. این وزن‌ها نشان می‌دهند که ورودی‌هایی که مقدار مثبت دارند (۱) بیشترین تأثیر را بر روی تصمیم‌گیری نورون دارند و ورودی‌هایی که مقدار منفی دارند (-۱) تأثیر معکوس دارند. آستانه ۶

۲-۱. شبکه عصبی یک لایه

در این قسمت یک شبکه عصبی یک لایه با ۴ نورون وجود دارد که خروجی هر نورون نشان دهنده مشاهده شدن یا نشدن یکی از اعداد است.این شبکه وضعیت هر LED را در ورودی دیافت می کند و به طوری که اگر LED روشن باشد ورودی مربوطه ۱ و اگر خاموش باشد ورودی مربوطه ۰ است.خروجی این شبکه به صورت زیر خواهد بود:

a	b	c	d	e	f	g	O_1	O_2	O_3	O_4
1	0	1	1	1	1	1	1	0	0	0
1	1	1	0	0	0	0	0	1	0	0
1	1	1	1	1	1	1	0	0	1	0
1	1	0	1	1	1	1	0	0	0	1
Any other combination							0	0	0	0

شکل ۵: عملکرد شبکه به ازای هر ورودی



شکل ۶: ساختار شبکه عصبی تک لایه

حال با در نظر گرفتن ωij به عنوان وزن شاخه ی بین ورودی i و نورون j و همچنین $\theta\ j$ به عنوان آستانه فعال سازی نورون j، پارامترهای این شبکه را طوری تعیین کنید که خروجی عملکرد مورد نظر را داشته باشد . (توجه داشته باشید که نورون ها یک پارامتر threshold نیز دارند که در نمودار مشخص نشده است.)

همان حداقل مقدار مورد نیاز برای فعال‌شدن نورون است. وزن‌ها و آستانه‌ها به صورت تجربی و با توجه به مسئله مورد بررسی تنظیم می‌شوند تا نورون به درستی الگوهای مورد نظر را تشخیص دهد. انتخاب درست وزن‌ها و آستانه‌ها از مهارت مهم در طراحی و آموزش مدل‌های عصبی مصنوعی است.

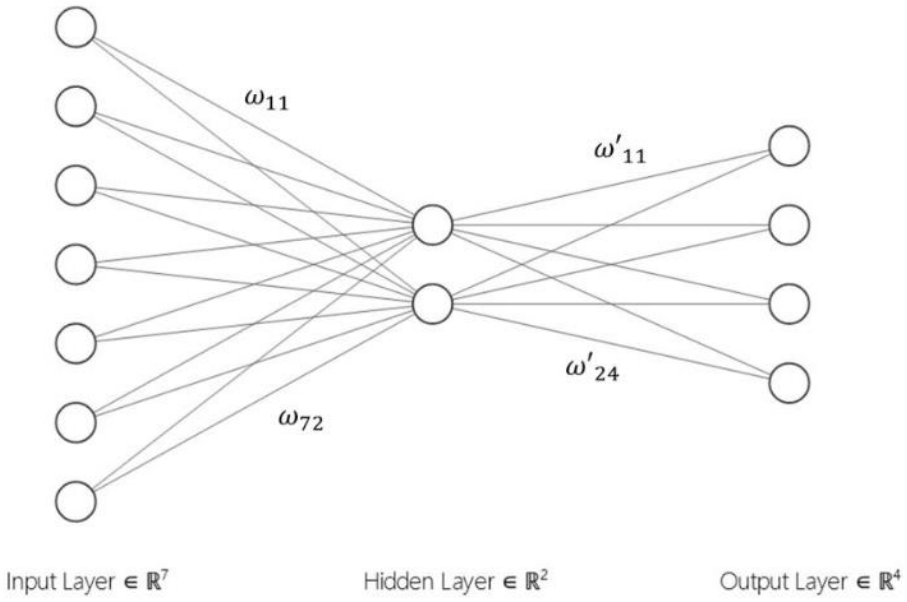
و در نهایت نتایج به صورت تصویر زیر گزارش می شود:

```
[1 1 0 0 1 1 0] - No digit detected
[1 1 0 0 1 1 1] - No digit detected
[1 1 0 1 0 0 0] - No digit detected
[1 1 0 1 0 0 1] - No digit detected
[1 1 0 1 0 1 0] - No digit detected
[1 1 0 1 0 1 1] - No digit detected
[1 1 0 1 1 0 0] - No digit detected
[1 1 0 1 1 0 1] - No digit detected
[1 1 0 1 1 1 0] - No digit detected
[1 1 0 1 1 1 1] - No digit detected
[1 1 1 0 0 0 0] - Digit 7
[1 1 1 0 0 0 1] - No digit detected
[1 1 1 0 0 1 0] - No digit detected
[1 1 1 0 0 1 1] - No digit detected
[1 1 1 0 1 0 0] - No digit detected
[1 1 1 0 1 0 1] - No digit detected
[1 1 1 0 1 1 0] - No digit detected
[1 1 1 0 1 1 1] - No digit detected
[1 1 1 1 0 0 0] - No digit detected
[1 1 1 1 0 0 1] - No digit detected
[1 1 1 1 0 1 0] - No digit detected
[1 1 1 1 0 1 1] - Digit 9
[1 1 1 1 1 0 0] - No digit detected
[1 1 1 1 1 0 1] - No digit detected
[1 1 1 1 1 1 0] - No digit detected
[1 1 1 1 1 1 1] - Digit 8
```

شکل ۷: نتایج

۳-۱. شبکه عصبی دو لایه

در این قسمت یک شبکه عصبی با یک لایه پنهان وجود دارد. لایه پنهان از دو نورون تشکیل شده که هرکدام وظیفه استخراج یک ویژگی یا الگو از ورودی را دارند. همچنین تضمین می شود که ورودی همواره یکی از اعداد ۶, ۷, ۸, ۹ می باشد و هیچ ترکیب دیگری در ورودی داده نمی شود. همانند بخش قبل وزن های بین لایه ورودی و لایه پنهان با ωij و وزن های بین لایه پنهان و لایه خروجی با $\omega' ij$ نشان داده می شوند. همچنین سازی آستانه فعال نورون های لایه پنهان با $\theta\ j$ و آستانه فعال سازی نورون های لایه خروجی با $\theta' j$ نشان داده می شودند.



شکل ۸: ساختار شبکه دو لایه

پاسخ:

کد ما از یک ترکیب از نورون‌های McCulloch-Pitts برای تشخیص اعداد و یافتن تشابه بین ورودی‌ها استفاده می‌کند. در اینجا توضیحی برای نحوه عملکرد کد ارائه می‌شود:

ابتدا، دو نورون به نام h۰ و h۱ با وزن‌ها و آستانه‌های مختلف ایجاد می‌شوند. این نورون‌ها برای تشخیص و تفکیک اعداد ۶ و ۹ مورد استفاده قرار می‌گیرند. ماتریس X۱ ایجاد می‌شود، که شامل چند ترکیب ورودی با طول ۷ است. این ترکیب‌ها به طور خاص برای تست و تحلیل مورد استفاده قرار می‌گیرند.

سپس چهار نورون جدید به نام‌های n۶۶، n۷۷، n۸۸ و n۹۹ با وزن‌ها و آستانه‌های مختلف ایجاد می‌شوند. این نورون‌ها برای تشخیص اعداد ۶، ۷، ۸ و ۹ مورد استفاده قرار می‌گیرند.

حالا با استفاده از دو نورون h۰ و h۱، تشخیص داده می‌شود که هر ورودی به کدامیک از اعداد ۶ و ۹ شبیه‌تر است. برای این منظور، ابتدا خروجی نورون‌های h۰ و h۱ برای هر ورودی محاسبه می‌شود و در X۲ ذخیره می‌شود.

سپس با استفاده از نورون‌های n۶۶، n۷۷، n۸۸ و n۹۹، تشخیص داده می‌شود که هر ورودی به کدام یک از اعداد ۶، ۷، ۸ و ۹ تعلق دارد. خروجی این نورون‌ها در o۱ تا o۴ ذخیره می‌شود.

در نهایت، با توجه به خروجی‌های نورون‌های h۰ و h۱، تصمیم گرفته می‌شود که هر ورودی به کدامیک از اعداد ۶ و ۹ شبیه‌تر است. سپس با توجه به خروجی‌های نورون‌های n۶۶ تا n۹۹، تشخیص داده می‌شود که ورودی به کدام یک از اعداد ۶، ۷، ۸ و ۹ تعلق دارد یا آیا هیچ کدام از این اعداد تشخیص داده نمی‌شود. کد شما در واقعیت برای تشخیص و تفکیک اعداد ۶ و ۹ از یکدیگر از این سیستم ترکیبی از نورون‌ها استفاده می‌کند و نتایج را به شما نمایش می‌دهد.

الف) وزن های این شبکه را به گونه ای تعیین کنید که خروجی مورد انتظار را به ازای هر ورودی ایجاد کند.

در این حالت، می‌توان از جدول کارنو به صورت معکوس برای تصمیم‌گیری استفاده کرد. به این صورت که اگر خروجی نورون h۱ برابر با ۱ شود و همچنین خروجی نورون h۲ برابر با ۱ باشد، این میانه مشابه عدد ۸ تلقی می‌شود. از آنجا که عدد ۸ همه سگمنت‌هایش فعال است، اگر خروجی‌های h۱ و h۲ همزمان ۱ باشند، نتیجه مشابه عدد ۸ خواهد بود. در اصطلاحات جدول کارنو، این معکوس معادل اعمال عملیات "و" (AND) بر روی خروجی‌های h۱ و h۲ است. اگر هر دو خروجی h۱ و h۲ برابر با ۱ باشند، این به معنای فعال بودن عدد ۸ با تمام سگمنت‌هاست.

```
n66 = McCulloch_Pitts_neuron([1,-1],1)
n77 = McCulloch_Pitts_neuron([-1,-1],0)
n88 = McCulloch_Pitts_neuron([1,1],2)
n99 = McCulloch_Pitts_neuron([-1,1],1)
```

شکل ۹:وزن های شبکه دو لایه از لایه مخفی به خروجی

```
h0 = McCulloch_Pitts_neuron([1,0,0,1,1,1,0], 4)
h1 = McCulloch_Pitts_neuron([1,1,1,0,0,0,1], 4)
```

شکل ۱۰: وزن های شبکه دو لایه از لایه ورودی به مخفی

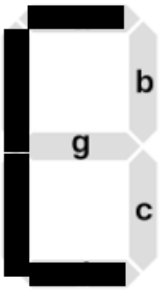
ب) توضیح دهید هر کدام از نورون های لایه پنهان چه الگویی را تشخیص می دهند.

در اینجا، ما تمرکز خود را روی لایه مخفی و نورون اول قرار می‌دهیم تا اعدادی که شبیه به ۶ هستند، تفکیک شوند. برای این کار، ما به سه ویژگی توجه می‌کنیم. سه ویژگی ممکن هستند: شبیه‌به-۶ بودن، شبیه‌به-۸ بودن و شبیه‌به-۹ بودن.

حالا وزن‌هایی را برای این ویژگی‌ها تنظیم می‌کنیم. برای ویژگی "شبیه به ۶ بودن"، وزن‌های مربوط به موقعیت‌هایی که شبیه‌به-۶ هستند را برابر با ۱ قرار می‌دهیم و وزن‌های مربوط به موقعیت‌هایی که نشانه‌گذاری نشده‌اند (به عبارت دقیق‌تر، مشکی نشده‌اند) را برابر با ۰ قرار می‌دهیم. این به معنای این است که نورون تنها به ورودی‌هایی با ویژگی "شبیه به ۶ بودن" عکس‌العمل می‌دهد و نورون به ورودی‌هایی با ویژگی‌های دیگر توجهی ندارد.

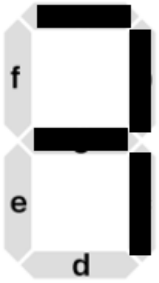
نتیجه این تنظیمات این است که نورون به تشخیص و تفکیک اعدادی که شبیه به ۶ هستند می‌پردازد و به عبارت دیگر، اگر ویژگی "شبیه به ۶ بودن" برای ورودی فعال باشد، نورون به عنوان معرف عدد ۶ عمل می‌کند. از آنجا که عدد ۸ نیز شبیه به ۶ است و تنها در یک جزء متفاوت است، این تنظیمات به ما امکان می‌دهند تا هم عدد ۶ و هم عدد ۸ را تشخیص دهیم.

اما برای عدد ۹، چون از ترشهلد یکی کمتر است، ویژگی "شبیه به ۸ بودن" فعال است و نورون h۱ را فعال نمی‌کند. بنابراین، اگر ویژگی "شبیه به ۶ بودن" فعال باشد و "شبیه به ۸ بودن" غیرفعال باشد، نتیجه نورون h۱ برابر با ۰ خواهد بود و عدد ۹ به عنوان عدد شبیه‌به-۹ تشخیص داده خواهد شد.



شکل ۱۱: الگو برای نورون اول لایه مخفی

برای نورون h۲، تمرکز خود را روی اعدادی که به عدد ۹ شبیه‌تر هستند (به عبارت دقیق‌تر، شکل ۴) قرار می‌دهیم و وزن‌ها را بر اساس این تمرکز تنظیم می‌کنیم. به این تنظیمات و وزن‌دهی‌ها نتیجه می‌دهد که نورون h۲ بتواند هم عدد ۸ و هم عدد ۹ را به عنوان خروجی تولید کند.



شکل ۱۲: الگو برای نورون دوم لایه مخفی

به علت فرض اینکه ورودی‌ها تنها ۴ حالت ممکن در قسمت دوم هستند، اگر نورون h۲ هیچ کدام از عددها را تشخیص ندهد، خروجی o۲ به ما عدد ۷ را می‌دهد.

حال با توجه به آن چه که در بالا در مورد کد و تنضیماتی که اعمال کردیم بیان کردیم داریم:

```
output of hidden neurons are:   [1, 0]
[1 0 1 1 1 1 1] - similar to 6
[1 0 1 1 1 1 1] - Digit 6
output of hidden neurons are:   [0, 0]
[1 1 1 0 0 0 0] - No digit detected
[1 1 1 0 0 0 0] - Digit 7
output of hidden neurons are:   [1, 1]
[1 1 1 1 1 1 1] - 8
[1 1 1 1 1 1 1] - Digit 8
output of hidden neurons are:   [0, 1]
[1 1 1 1 0 1 1] - similar to 9
[1 1 1 1 0 1 1] - Digit 9
```

شکل ۱۳: نتایج شبکه دولایه

ج) تعداد پارامترهای شبکه یک لایه و دو لایه را محاسبه و مقایسه کنید.

برای یک شبکه با یک لایه مخفی، باید تعدادی وزن به تعداد ورودی‌ها (در اینجا ۷۴ یعنی ۲۸ وزن) تعیین کنیم، همچنین ۴ عدد ترشهلد را نیز تعیین می‌کنیم. اما برای یک شبکه با دو لایه مخفی، ابتدا باید یک لایه اولیه با تعداد وزن‌هایی برابر با تعداد ورودی‌ها (در اینجا ۷۲ یعنی ۱۴ وزن) و ۲ ترشهلد اولیه داشته باشیم. سپس باید یک لایه دومی با تعداد وزن‌هایی برابر با تعداد خروجی‌ها (در اینجا ۲*۴ یعنی ۸ وزن) و ۴ ترشهلد بعدی داشته باشیم.

در مجموع، برای یک لایه تنها ۳۲ پارامتر مورد نیاز است، اما برای دو لایه مخفی ۲۸ پارامتر نیاز است. این نشان می‌دهد که شبکه با یک لایه مخفی می‌تواند بهتر عمل کند و در عین کارایی بهتر، تعداد پارامترهای کمتری دارد.

پاسخ ۲ – آموزش شبکه های Adaline و Madaline

در این پرسش به بررسی دو شبکه‌ی عصبی Adaline و Madaline پرداخته می شود.

۲-۱. Adaline

در این بخش یک شبکه‌ی عصبی Adaline آموزش داده خواهد شد که در مجموعه داده iris(که از ۳ نوع عنبیه مختلف Setosa، Versicolour و Virginica تشکیل شده)، نوع عنبیه Setosa را از سایرها تشخیص دهد.

الف) ابتدا نمودار پراکندگی داده‌ها را در دو بعد رسم کنید (برای سادگی از دو ویژگی اول یعنی Sepal-width و Sepal-length Adaline استفاده شود)، سپس شبکه را آموزش می‌دهیم.

یک نمودار پراکندگی برای داده‌های iris را با استفاده از کتابخانه‌های `matplotlib` و `sklearn` ایجاد می‌کند. این کد مراحل زیر را انجام می‌دهد:

۱. در مرحله اول، داده‌های مربوط به مجموعه داده گل‌های آیریس با استفاده از تابع `datasets.load_iris()` از ماژول `sklearn.datasets` بارگیری می‌شود. سپس داده‌ها (بردار ویژگی‌ها) و برچسب‌های کلاس‌ها (تارگت) از متغیر `iris` استخراج می‌شوند.

```
# Step 1: Load the Iris dataset
iris = datasets.load_iris()
data = iris.data
target = iris.target
```

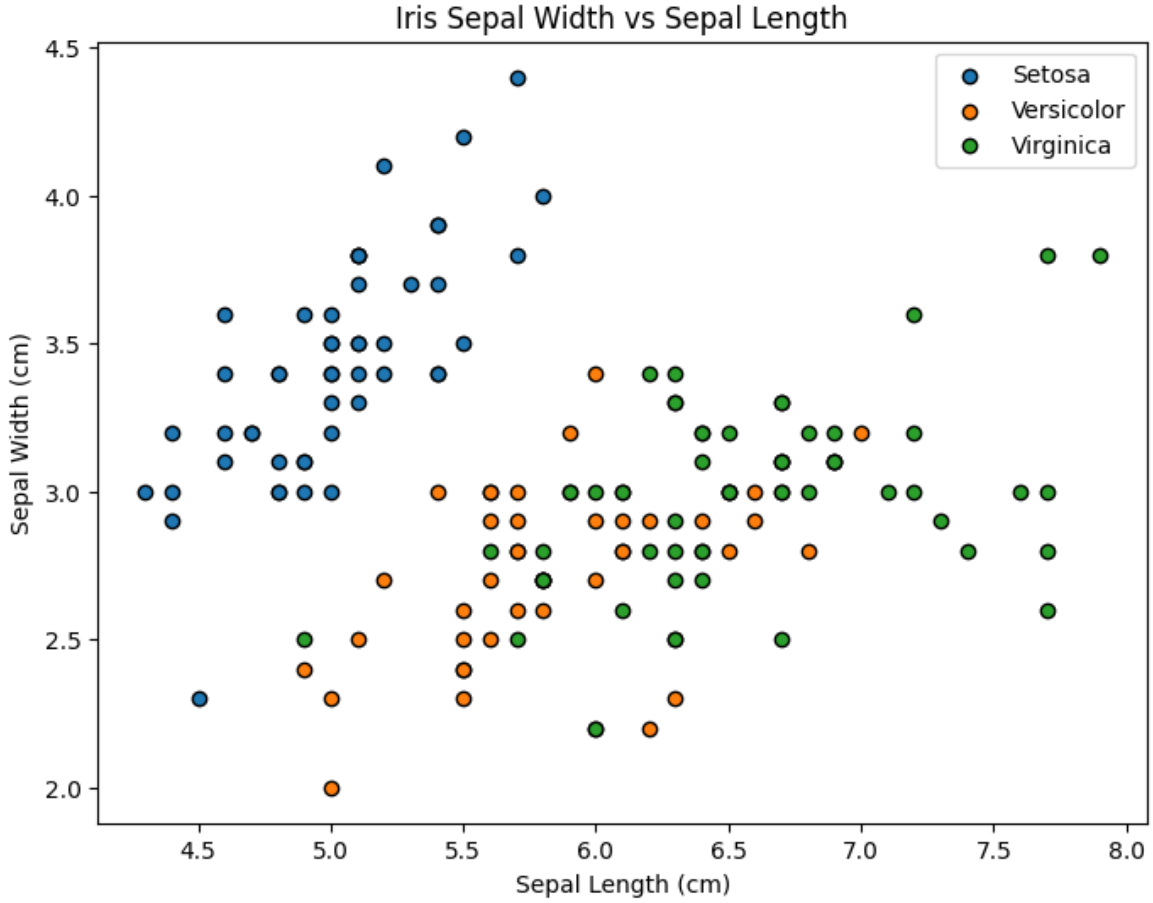
شکل ۱۴ مجموعه‌داده **Iris**

۲. حال دو ویژگی مورد نظر ما در این سوال را انتخاب می‌کنیم. در اینجا، از طول سپال (feature ۰) و عرض سپال (feature ۱) به عنوان متغیرهای مستقل برای محور x و y نمودار استفاده می‌شود.

```
sepal_width = data[:, 1]
sepal_length = data[:, 0]
```

شکل ۱۵ انتخاب ویژگی‌ها

۳. در این بخش یک نمودار پراکندگی، برای داده‌های Iris رسم می‌کنیم. از تابع `plt.scatter()` برای ایجاد نمودار پراکندگی استفاده می‌شود. همچنین، با استفاده از `cmap=plt.cm.Set۱`، یک نقشه رنگ برای نمودار تعیین می‌شود و مرزهای نقاط با رنگ سیاه (`edgecolor='k'`) نمایش داده می‌شود.



شکل ۱۶: نمودار پراکندگی در دو بعد

سپس مشخصات نمودار از جمله محور x و y، عنوان نمودار و لجند تنظیم می‌شوند. در نهایت، نمودار ایجاد شده با تابع `plt.show()` نمایش داده می‌شود. این نمودار نشان می‌دهد رابطه بین طول و عرض سپال گل‌های آیریس برای هر یک از کلاس‌ها و از لجند می‌توانید تشخیص دهید کدام نقطه به کدام کلاس تعلق دارد.

```python
# Define class labels and colors for the legend
class_labels = ['Setosa', 'Versicolor', 'Virginica']
colors = ['#1f77b4', '#ff7f0e', '#2ca02c']  # Custom colors

# Step 3: Create a scatter plot with a legend
plt.figure(figsize=(8, 6))
for i in range(3):
    plt.scatter(sepal_length[target == i], sepal_width[target == i], c=colors[i], label=class_labels[i], cmap=plt.cm.Set1, edgecolor='k')

plt.xlabel('Sepal Length (cm)')
plt.ylabel('Sepal Width (cm)')
plt.title('Iris Sepal Width vs Sepal Length')
plt.legend()
plt.show()
```

شکل ۱۷ پلات کردن مجموعه‌داده **iris**

کد ما یک کلاس به نام "Adaline" را پیاده‌سازی می‌کند. این کلاس برای اجرای یک الگوریتم یادگیری ماشین به نام Adaline (Adaptive Linear Neuron) به کار می‌رود. این الگوریتم یک نوع از شبکه‌های عصبی مصنوعی به حساب می‌آید که برای مسائل دسته‌بندی باینری مورد استفاده قرار می‌گیرد.

این کد Adaline را پیاده‌سازی کرده و قادر به آموزش مدل و انجام پیش‌بینی‌های مربوط به داده‌های ورودی است.

```python
from IPython.core import history
class Adaline():
  def __init__(self,input_size,activation_function):
    self.weights = np.random.rand(input_size)  # Initialize weights randomly
    self.biases = 0
    self.lr=None
    self.history={"train_acc":[],"train_loss":[],"val_acc":[],"val_loss":[]}
    self.activation_function=activation_function
  def fit(self, X, Y, epochs, lr, error_func,val_split=0.1):

    X_train, X_val, Y_train, Y_val = train_test_split(X, Y, test_size=val_split)
    self.lr = lr
    for epoch in range(epochs):
        outputs = self.forward(X_train)
        self.backward(X_train, Y_train, outputs)

        accuracy = self.evaluate(X_train, Y_train)
        error = error_func(Y_train, outputs)
        self.history["train_loss"].append(np.mean(error))
        self.history["train_acc"].append(accuracy)

        outputs = self.forward(X_val)
        accuracy = self.evaluate(X_val, Y_val)
        error = error_func(Y_val, outputs)
        self.history["val_loss"].append(np.mean(error))
        self.history["val_acc"].append(accuracy)
```

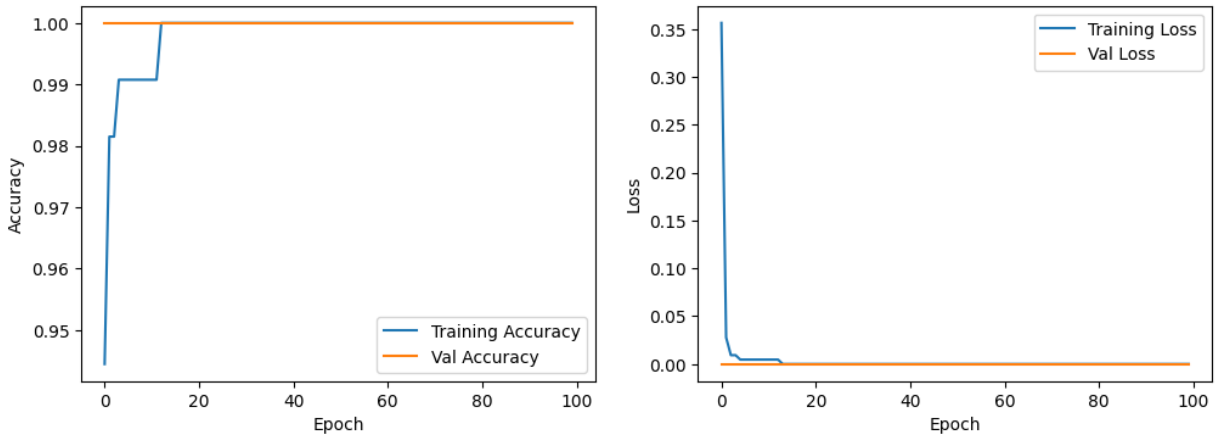
شکل ۱۸ پیاده سازی **Aalaine**

کلاس Adalaine را پیاده‌سازی کرده‌ایم که فانکشن‌هایی همچون predict برای پیش‌بینی و fit برای آموزش دارد. حال ۱۰۰ دوره با lr=۰.۰۱ این مدل را روی داده‌ها آموزش می‌دهیم، همچنین ۰.۱ داده‌ها را به عنوان داده‌ی validation جدا می‌کنیم:

```
# Create an instance of the Adaline class
input_size = X.shape[1]
adaline = Adaline(input_size=input_size, activation_function=step_activation)

# Train the Adaline model
epochs = 100
learning_rate = 0.01
history=adaline.fit(X_train, Y_train, epochs, learning_rate, error_func=mean_squared_error)

# Evaluate the model
train_accuracy = adaline.evaluate(X_train, Y_train)
test_accuracy = adaline.evaluate(X_test, Y_test)

print("Training Accuracy:", train_accuracy)
print("Testing Accuracy:", test_accuracy)
```

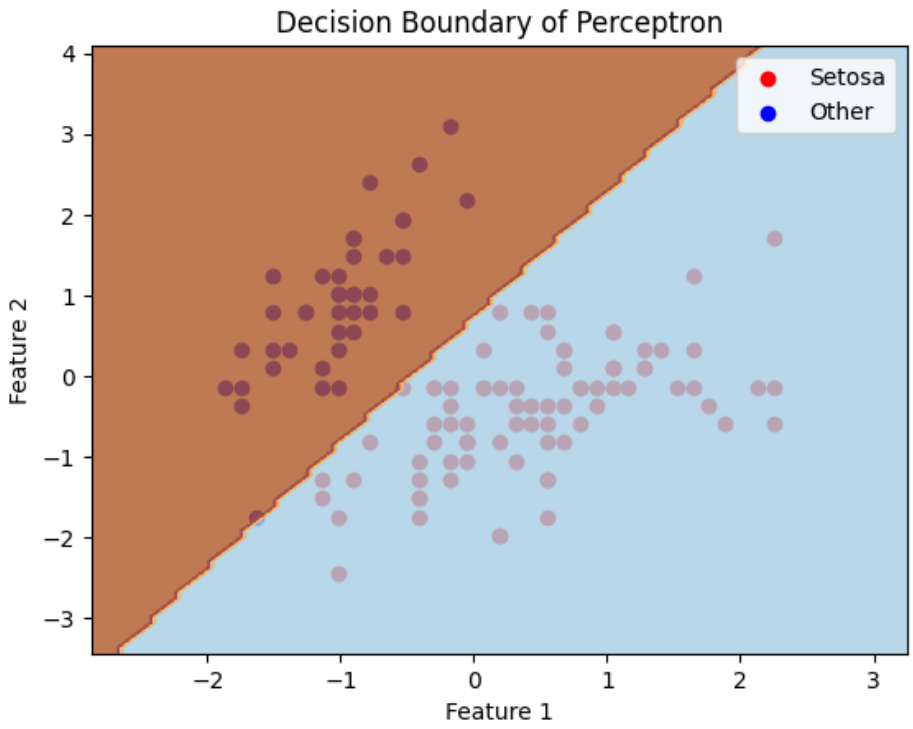
```
Training Accuracy: 1.0
Testing Accuracy: 1.0
```

شکل ۱۹ کد مربوط به آموزش **Adalaine** برای **Setosa**



شکل ۲۰: نتایج به ازای نرخ یادگیری و **epoch**

حال می‌توان خطوط جدا ساز مربوط به این کلاس‌ها را رسم کرد، و همچنین ناحیه تصمیم‌گیری را نشان داده‌ایم. از آنجا که کلاس Setosa نسبت به سایر کلاس‌ها به صورت خطی قابل جداسازی بود، شبکه Adaline آموزش داده شده توسط ما قابلیت جداسازی این داده‌ها را با دقت ۱۰۰ درصد داشت.



شکل ۲۱: **Decision boundry**

ب) حال این کار را برای عنبیه versicolor انجام می‌دهیم(یعنی که مجموعه داده را به دو بخش versicolor و non- versicolor تقسیم کرده و آموزش بر روی این داده‌ها انجام شود) سپس دلیل خوب یا بد جدا شدن داده ها را نسبت به بخش الف توضیح می‌دهیم.

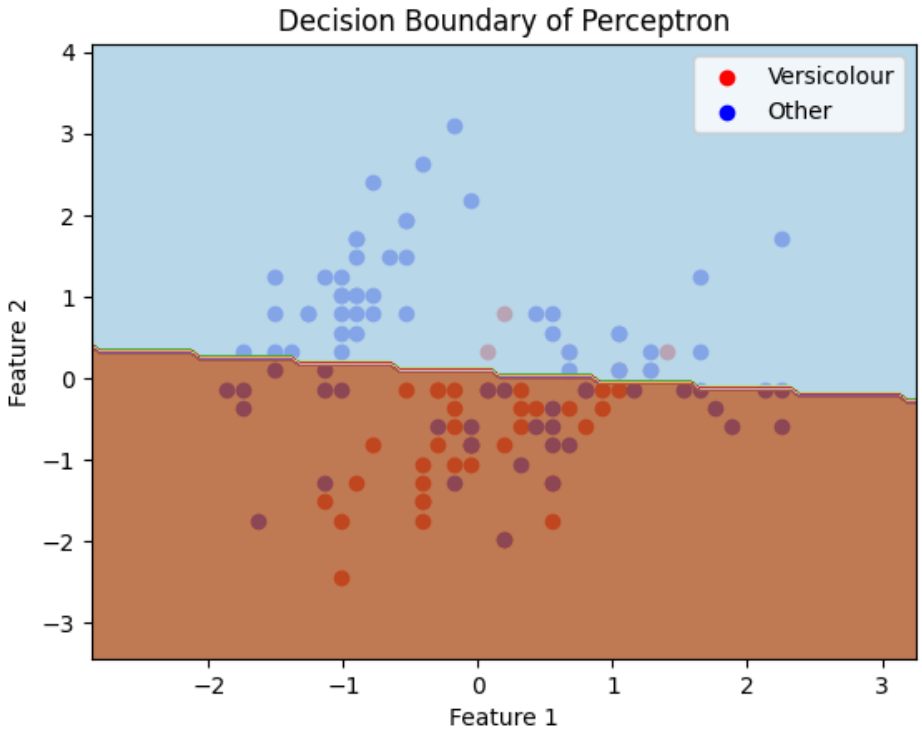
حال مراحل قبل را برای این داده‌ها تکرار می‌کنیم، ابتدا داده‌های آموزش و تست را جدا کرده، و مدل Adaline را با استفاده از این داده‌های جدید آموزش می‌دهیم:

```python
# Train the Adaline model
epochs = 100
learning_rate = 0.01
history=adaline.fit(X_train, Y_train, epochs, learning_rate, error_func=mean_squared_error)

# Evaluate the model
train_accuracy = adaline.evaluate(X_train, Y_train)
test_accuracy = adaline.evaluate(X_test, Y_test)

print("Training Accuracy:", train_accuracy)
print("Testing Accuracy:", test_accuracy)
```

```
Training Accuracy: 0.675
Testing Accuracy: 0.6
```

شکل ۲۲ مدل **Adaline** برای **Versicolor**

مشخص است که دقت برای این داده‌ها به شدت کاهش پیدا کرده است، دلیل این اتفاق در شکل زیر قابل مشاهده است.



شکل ۲۳: نتایج مربوط به **versicolor**

در واقعیت، دلیلی که مدل Adaline در حالت دوم (که داده‌های "Versicolour" از داده‌های دیگر تفکیک‌پذیر نیستند) نتوانسته به خوبی عملکرد کند این است که مدل Adaline توانایی تفکیک داده‌های غیرخطی را ندارد و فقط برای داده‌هایی کاربرد دارد که به صورت خطی تفکیک‌پذیر باشند. در حالت اول، داده‌های "Setosa" از داده‌های دیگر به صورت خطی تفکیک‌پذیر هستند، بنابراین مدل Adaline به خوبی

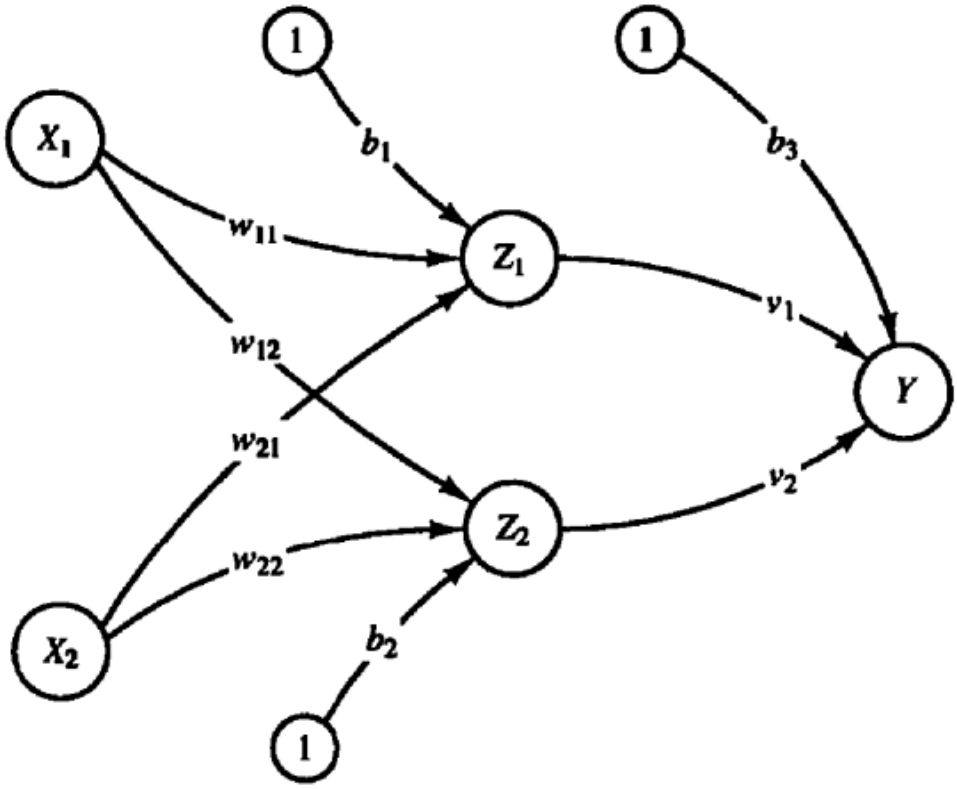
کار می‌کند و می‌تواند مرز تصمیم‌گیری خوبی برای آنها ایجاد کند. اما در حالت دوم، چون داده‌های "Versicolour" از داده‌های دیگر به صورت خطی تفکیک‌پذیر نیستند (یعنی مرز تصمیم‌گیری غیرخطی است)، مدل Adaline نمی‌تواند مرز تصمیم‌گیری مناسبی ایجاد کند. به عبارت دیگر، Adaline به توانایی مدل کردن مسائل دسته‌بندی غیرخطی ندارد و برای داده‌هایی که توسط مرزهای تصمیم‌گیری غیرخطی تفکیک می‌شوند، عملکرد ناپایداری دارد. این امر نشان می‌دهد که مدل‌های دیگری که توانایی مدل‌سازی دسته‌بندی غیرخطی دارند (مانند شبکه عصبی چند لایه یا SVM با هسته غیرخطی) برای مسائلی که Adaline ناتوان در آن است، مناسب‌تر هستند.

۲-۲. Madaline

در این بخش به پیاده سازی شبکه Madaline بر روی یک مجموعه‌داده ماه شکل می‌پردازیم.

الف) ابتدا یکی از الگوریتم های MRI و MRII را که کتاب مرجع موجود است پیاده سازی کرده وتوضیح دهید.

الگوریتم MRI یک روش ساده برای تشخیص الگو در شبکه‌های مدالین است و می‌تواند برای مسائل تشخیص الگو و تصمیم‌گیری دودویی مورد استفاده قرار گیرد. این الگوریتم به تطبیق الگوها با الگوهای مرجع و به‌روزرسانی وزن‌ها بر اساس بازخورد از خروجی شبکه تکیه می‌کند.



شکل ۲۴: یک شبکه MadaLine با دو نورون AdaLine مخفی

نحوه کار این شبکه به این صورت بوده که در ابتدا تابع فعالسازی (Activation Function) به صورت زیر تعریف می‌شود:

$$f(x)=\begin{cases}1 & \text{if } x\geq 0\\ -1 & \text{if } x<0\end{cases}$$

و سپس حاصلضرب وزن‌های ورودی این نورون‌های Z۱ و Z۲ و Y در ورودی بعلاوه بایاس‌های مربوطه تحت این تابع فعالساز اثر داده می‌شود. دقت داشته باشید که وزن و بایاس لایه مخفی مقادیر تصادفی و کوچک بوده و وزن و بایاس لایه خروجی مقادیر فیکس هستند.

در نهایت مراحل الگوریتم MadaLine را به صورت زیر پیگیری می‌نماییم:

الگوریتم **MRI** شبکه مدالین را برای تشخیص الگوها استفاده می‌کند. این الگوریتم به صورت زیر عمل می‌کند:

۱. مقداردهی اولیه وزن‌ها و بایاس‌ها: وزن‌ها و بایاس‌های شبکه با مقادیر تصادفی مقداردهی می‌شوند. این وزن‌ها و بایاس‌ها نقش مهمی در تصمیم‌گیری شبکه ایفا می‌کنند.

۲. برای هر مجموعه آموزشی ورودی (**s**) و خروجی مرجوعی (**t**)، مراحل زیر انجام می‌شوند:

۳. ست کردن ورودی شبکه: ورودی (**s**) به شبکه داده می‌شود تا شبکه از آن الگوی ورودی را دریافت کند.

۴. محاسبه حاصل جمع ورودی لایه مخفی: در این مرحله، حاصل جمع ورودی‌های لایه مخفی محاسبه می‌شود. این حاصل جمع شامل ترکیب خطی از ورودی‌ها با وزن‌ها و بایاس‌های لایه مخفی است. این محاسبات به صورت زیر انجام می‌شود:

حاصل جمع لایه مخفی = (**Σ** ورودی * وزن) + بایاس

۵. تابع فعال‌سازی: حاصل جمع لایه مخفی به تابع فعال‌سازی اعمال می‌شود تا سیگنال خروجی محاسبه شود. تابع فعال‌سازی ممکن است تابع تطابق معمولاً مثل تابع سیگموئید باشد.

۶. مقایسه با الگوی مرجع: سیگنال خروجی با الگوی مرجع (**t**) مقایسه می‌شود. اگر تطابق وجود داشته باشد، شبکه تصمیم مثبت می‌گیرد، در غیر این صورت، تصمیم منفی می‌گیرد.

۷. به‌روزرسانی وزن‌ها و بایاس‌ها: اگر شبکه تصمیم منفی بگیرد (یعنی تشخیص اشتباهی داشته باشد)، وزن‌ها و بایاس‌ها به‌روزرسانی می‌شوند تا به تدریج به تطابق بیشتری با الگوها برسند. این به‌روزرسانی معمولاً توسط قوانین یادگیری انجام می‌شود.

این مراحل به تکرار برای مجموعه داده‌های آموزشی انجام می‌شوند تا شبکه به تصمیم درستی برسد یا به حد مشخصی تکرار شوند. این الگوریتم به تدریج تطابق بیشتری با الگوها را یاد می‌گیرد و وزن‌ها را بهبود

می‌بخشد.

$$\begin{cases} z\_in_1 = b_1 + x_1 w_{11} + x_2 w_{21} \\ z\_in_2 = b_2 + x_1 w_{12} + x_2 w_{22} \end{cases}$$

۸. خروجی‌های لایه مخفی را مشخص می‌کنیم (f تابع فعالساز بود):

$$\begin{cases} z_1 = f(z\_in_1) \\ z_2 = f(z\_in_2) \end{cases}$$

۹.خروجی شبکه را به دست می‌آوریم:

$$\begin{cases} y\_in = b_3 + z_1 v_1 + z_2 v_2 \\ y = f(y\_in) \end{cases}$$

۱۰.در این مرحله نیز در صورتی که t=y باشد به روز رسانی وزن‌ها و بایاس را انجام نمی‌دهیم و در غیر اینصورت مطابق دستورالعمل کتاب به آپدیت وزن‌ها می‌پردازیم.(به منظور پرهیز از اطناب نوشتار از تفصیل این دستورالعمل‌ها اجتناب ورزیده و در ضمن تشریح کدهای نوشته شده به بیان آنها می‌پردازیم.)

در قسمت بعد مراحل برای رسیدن به پاسخ به صورت زیر به دست می آید:

۱. مقداردهی اولیه وزن‌ها و نرخ یادگیری:

در ابتدا، مقدارهای اولیه برای وزن‌ها (W)، وزن‌های خروجی (V)، و عرض تفسیر (b) تعیین می‌شوند. این مقدارها اغلب به صورت تصادفی مقداردهی می‌شوند.

۲. محاسبه مقدار ورودی (z_in):

مقدار ورودی (z_in) با محاسبه ترکیب خطی از وزن‌ها (W) و ورودی‌ها (X) به علاوه عرض تفسیر (b) به شکل زیر محاسبه می‌شود:

$$z\_in = W * X + b$$

۳. تابع فعال‌ساز:

سپس، مقدار ورودی (z_in) به تابع فعال‌ساز اعمال می‌شود. در اینجا اگر مقدار z از صفر بزرگتر یا مساوی باشد، تابع فعال‌ساز مقدار ۱ را برمی‌گرداند و در غیر این صورت مقدار -۱ را برمی‌گرداند:

activation(z_in) = ۱، اگر z_in >= ۰

activation(z_in) = -۱، اگر z_in < ۰

۴. محاسبه خروجی (z_out):

مقدار خروجی (z_out) به عنوان ورودی تابع فعال‌ساز (activation) با وزن‌های خروجی (V) و عرض تفسیر (b) محاسبه می‌شود:

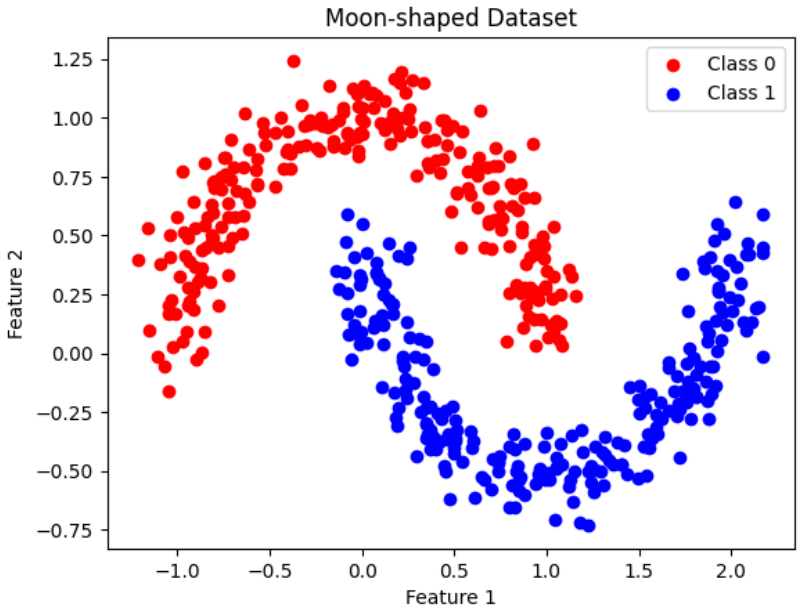
z_out = V * activation(z_in) + b

۵. به‌روزرسانی وزن‌ها:

در نهایت، اگر برچسب اصلی با برچسب پیش‌بینی شده یکسان نباشد، وزن‌ها بر اساس یک فرمول به‌روزرسانی می‌شوند. این به‌روزرسانی وزن‌ها ممکن است با استفاده از یک الگوریتم یادگیری مشخص (مانند گرادیان کاهشی) انجام شود. این عملیات به معنای به‌روزرسانی وزن‌ها به گونه‌ای است که خطای پیش‌بینی کاهش یابد تا به نتایج بهتری دست پیدا کنیم.

به طور خلاصه، این توضیح توصیف می‌کند چگونه یک شبکه عصبی به عنوان مدل یادگیری عمیق، با مقداردهی اولیه وزن‌ها، محاسبه ورودی و خروجی، و به‌روزرسانی وزن‌ها، برای مسائل دسته‌بندی و یادگیری عمل می‌کند.

حال برای مجموعه‌داده‌ای به شکل داده‌های ماه شکل می‌خواهیم طبقه‌بندی را انجام دهیم.



شکل ۲۵: داده های مصنوعی به صورت ماه شکل

برای پیاده‌سازی این الگوریتم کلاس مربوط به Madaline را تعریف می‌کنیم، که قابلیت fit شدن به مجموعه‌داده ما را دارد و در آن الگوریتم گفته شده در مرحله‌ی قبل را پیاده‌سازی کرده‌ایم.

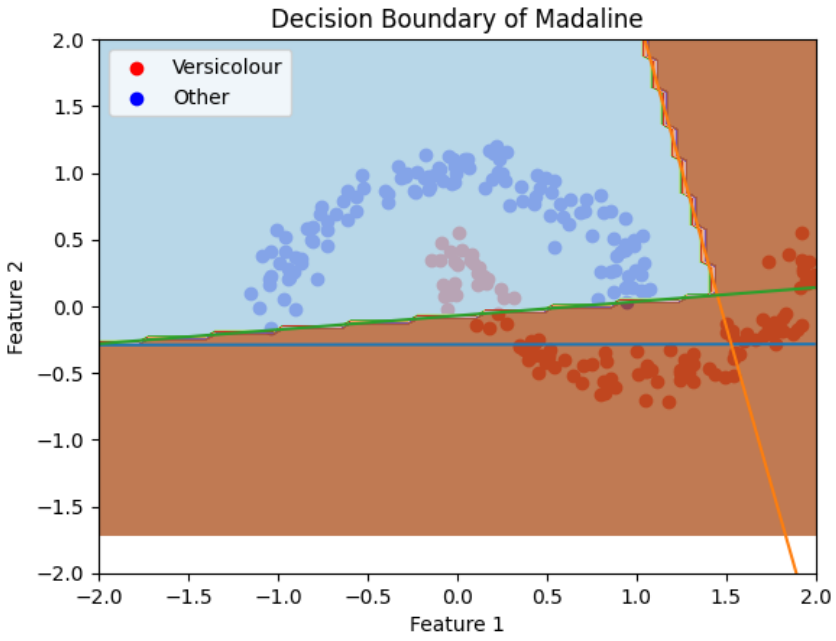
```python
class Madeline():
  def __init__(self,input_size,hidden_size):
    self.weights1 = np.random.rand(input_size, hidden_size)  # Weights for the
    self.biases1 = np.zeros((1, hidden_size))
    self.weights2 = np.random.rand(hidden_size, 1)  # Weights for the hidden to
    self.biases2 = np.zeros((1, 1))
    self.lr = None
    self.hidden_size=hidden_size
    self.history={"train_acc":[],"train_loss":[],"val_acc":[],"val_loss":[]}
  def fit(self, x, t, epochs, lr, error_func, val_split=0.1):
    # Setting the parameter n as the number of hidden neurons
    n = self.hidden_size
    # Defining parameters related to the hidden layer
    self.biases1 = np.random.random([1, n])
    self.z = np.zeros([1, n])
    self.weights1 = np.random.random([2, n])
    # Defining parameters related to the output layer
    self.weights2 = np.ones([n, 1])
    self.biases2 = np.array([n - 1])
    # Defining parameters related to the network
    index = list(range(len(t)))
    epoch = 0
    alpha = 0.1
    accuracy = 0
    history=[]
    # Implementing the Madaline algorithm
    while (epoch < epochs):
        np.random.shuffle(index)
        # Updating the weights and bias
        for i in index:
            # Calculating the input of hidden neurons
            netZ = np.matmul(x[i, :], self.weights1) + self.biases1
            for j in range(n):
                # Obtaining the output of hidden neurons
                self.z[0][j] = actv_func(netZ[0][j])
            # Achieving the input and output of Y neuron
            netY = np.dot(self.z, self.weights2) + self.biases2
```

شکل ۲۶ کلاس **Madaline**

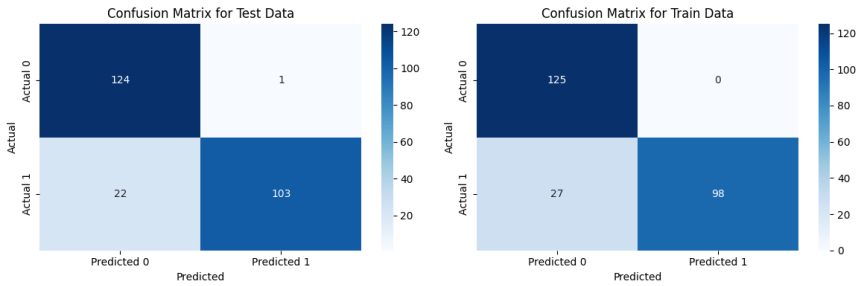
ب) حال به ترتیب برای ۳، ۵ و ۸ نورن شبکه‌ی عصبی را آموزش می‌دهیم:

برای ۳ نورون داریم:

دقت به دست آمده برابر با ۰.۸۹۲ است.



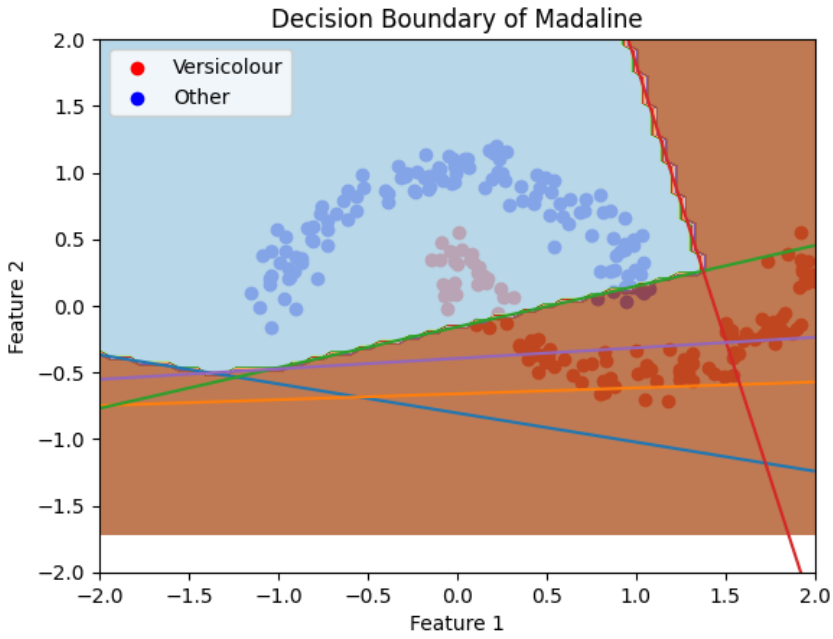
شکل ۲۷: مرز تصمیم در برای ۳ نورون



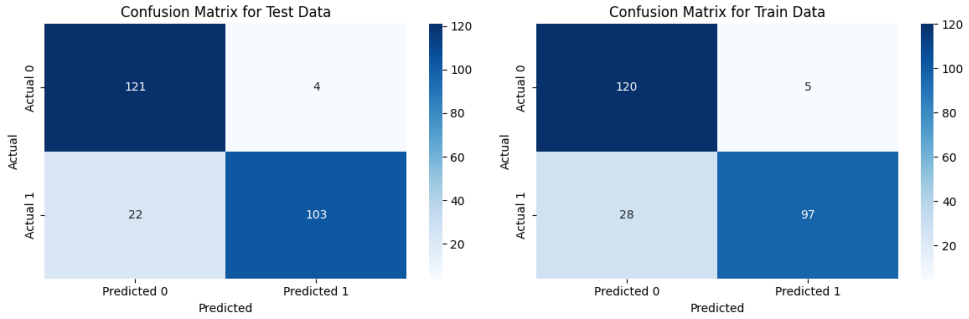
شکل ۲۸: ماتریس درهمریختگی برای ۵ نورون

برای ۵ نورون داریم:

دقت به دست آمده برابر با ۰.۸۹۲ است.



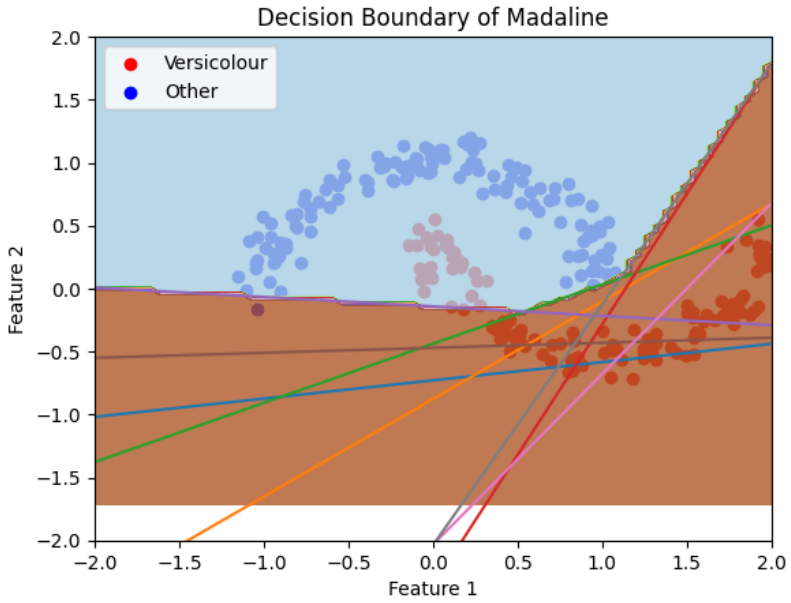
شکل ۲۹: مرز تصمیم در برای پنج نورون



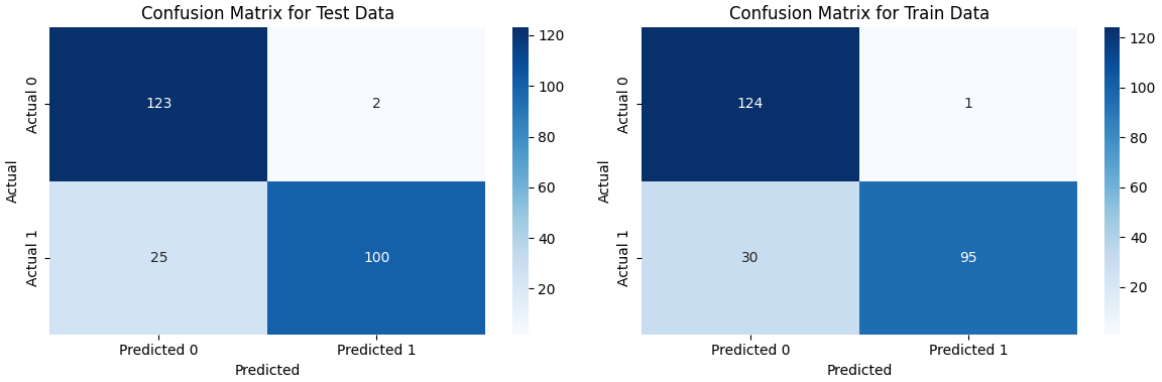
شکل ۳۰: ماتریس درهمریختگی برای ۵ نورون

برای ۸ نورون داریم:

دقت به دست آمده برابر با ۰.۸۹۲ است.



شکل ۳۱: مرز تصمیم برای ۸ نورون



شکل ۳۲: ماتریس درهمریختگی برای ۸ نورون

داده‌های دو کلاس با خطوط مستقیم به طور کامل تفکیک پذیر نیستند. میبینیم که با افزایش تعداد نورون و در نتیجه تعداد خطوط نیز نتیجه بهتر نمیشود و دقت کمی کاهش می یابد. بهترین نتیجه با تعداد ۳ خط به دست آمده است. با توجه به شکل پراکندگی داده ها افزایش تعداد خط های مستقیم تاثیری در بهتر شدن نتیجه ندارد.

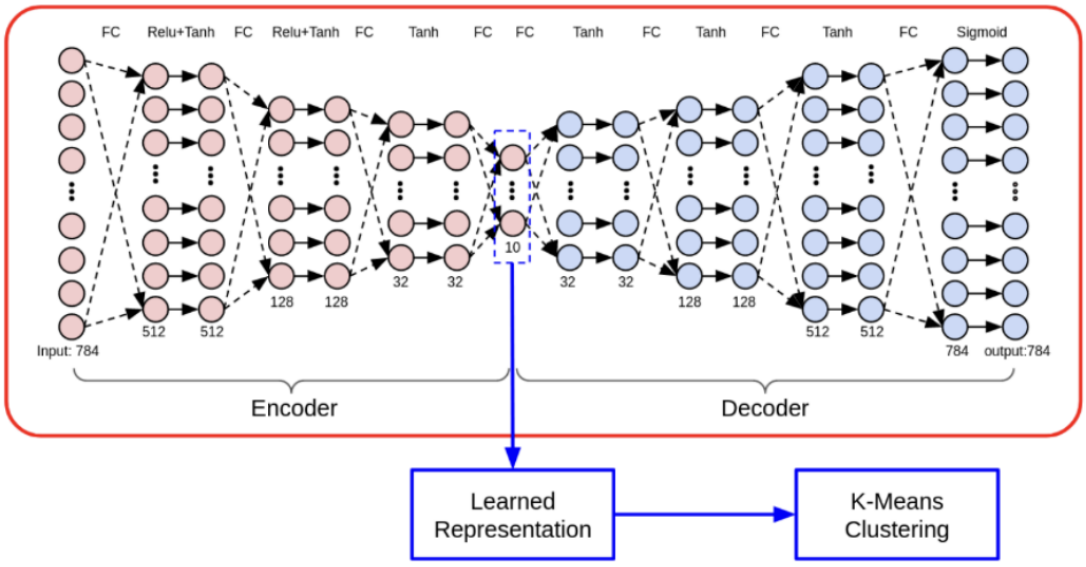
پاسخ ۳ – خوشه بندی با استفاده از Autoencoder

۳-۱. پیاده سازی Deep Autoencoder برای کاهش ابعاد داده‌ها

در این بخش با استفاده از مقاله " DAC: Deep Autoencoder-based Clustering, a General Deep Learning Framework of Representation Learning" یک Deep Autoencoder پیاده سازی خواهد شد در این ،تمرین از یک شبکه عمیق اتو انکودر (Deep Auto encoder برای کاهش ابعاد داده‌ها نمایش دهی بهتر به جهت انجام خوشه بندی با الگوریتم K-Means استفاده شده است.

برای پیاده سازی شبکه معرفی شده در این مقاله

۱. ابتدا با استفاده از کتابخانههای مورد نیاز مانند PyTorch و TensorFlow یک شبکه عمیق اتوانکودر براساس مدل ارائه شده در مقاله شکل) (۶ ایجاد کنید همچنین توجه کنید برای پیاده سازی باید از توابع فعالساز مختلفی مانند Tanha Sigmoid و ReLU استفاده نمایید.



شکل ۳۳: مدل پیشنهادی مقاله

۲. تنها از مجموعه داده MNIST برای آموزش مدل استفاده کنید و اتوانکودر را با توجه به تابع خطای ارائه در مقاله فرمول شماره (۶) آموزش دهید.

۴. پس از آموزش ،اتوانکودر از لایه نهان حاصل از اتوانکودر به عنوان ورودی برای الگوریتم K-Means و برای انجام تسک clustering استفاده کرده و خروجی نهایی را ارزیابی نمایید.

۵. نتایج بدست آمده را با استفاده از معیارهای ارزیابی مقاله مانند Adjusted Rand Index (ARI) ارزیابی کرده و نتایجی همانند جداول مقاله گزارش دهید.

پاسخ:

کتابخانه ها: در این کد، از کتابخانه Tensorflow برای درست کردن این مدل استفاده می‌کنیم.

اجزای مدل: کلاس DAC: کلاس DAC به عنوان زیر کلاس tf.keras.Model تعریف می شود. به سه پارامتر اصلی نیاز دارد:

input_dim: بعد داده های ورودی. latent_dim : بعد فضای نهفته (گلوگاه) که در آن داده ها فشرده می شوند. num_clusters: تعداد خوشه ها که مربوط به بعد خروجی آخرین لایه است. رمزگذار: رمزگذار به عنوان دنباله ای از لایه های متراکم تعریف می شود که به تدریج ابعاد ورودی را کاهش می دهد:

لایه اول دارای ۵۱۲ واحد است و از فعال سازی ReLU استفاده می کند. لایه دوم دارای ۱۲۸ واحد با فعال سازی ReLU است. لایه سوم دارای ۳۲ واحد با فعال سازی مماس هذلولی (tanh) است. لایه نهایی دارای ۱۰ واحد با فعال سازی tanh می باشد. رمزگشا: رمزگشا به عنوان دنباله ای از لایه های متراکم تعریف می شود که هدف آن بازسازی داده های ورودی از نمایش فشرده است:

لایه اول دارای ۳۲ واحد با فعال سازی tanh می باشد. لایه دوم دارای ۱۲۸ واحد با فعال سازی tanh می باشد. لایه سوم دارای ۵۱۲ واحد با فعال سازی tanh می باشد. لایه نهایی دارای واحدهای latent_dim است که بعد فضای پنهان است.

```python
class DAC(tf.keras.Model):
    def __init__(self, input_dim, latent_dim, num_clusters):
        super(DAC, self).__init__()
        # Encoder
        self.encoder = tf.keras.Sequential([
            tf.keras.layers.Dense(512, activation='relu', input_shape=(input_dim,)),
            tf.keras.layers.Dense(128, activation='relu'),
            tf.keras.layers.Dense(32, activation='tanh'),
            tf.keras.layers.Dense(10, activation='tanh')
        ])
        # Decoder
        self.decoder = tf.keras.Sequential([
            tf.keras.layers.Dense(32, activation='tanh', input_shape=(10,)),
            tf.keras.layers.Dense(128, activation='tanh'),
            tf.keras.layers.Dense(512, activation='tanh'),
            tf.keras.layers.Dense(latent_dim)
        ])
        # Clustering layer
        self.sigmoid1 = tf.keras.layers.Activation('sigmoid')

    def call(self, x):
        # Encoder
        encoded = self.encoder(x)
        # Decoder
        decoded = self.decoder(encoded)
        # Clustering
        outputs = self.sigmoid1(decoded)
        return outputs

# Example usage:
input_dim = 784  # For MNIST
latent_dim = 32  # You can choose an appropriate latent dimension
num_clusters = 10  # Number of clusters (output dimension of the last layer)

model = DAC(input_dim, latent_dim, num_clusters)
```

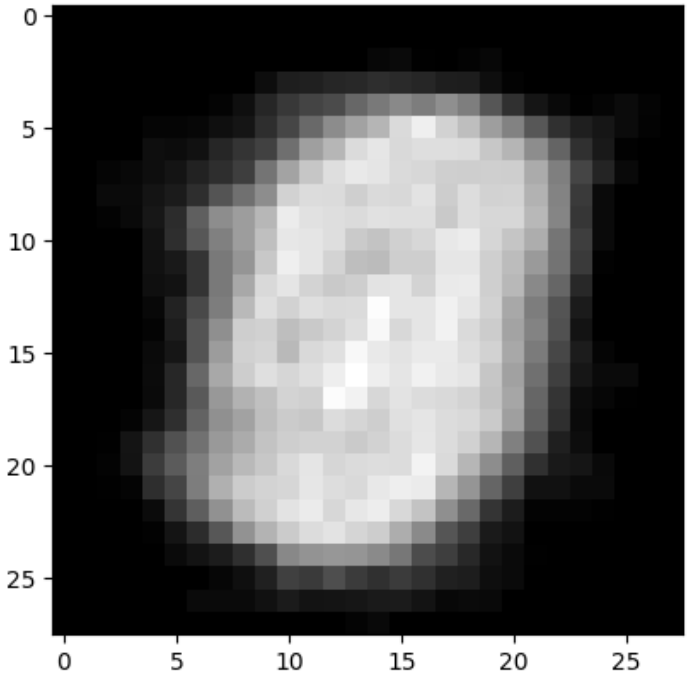
شکل ۳۴ کلاس **DAC**

از تابع compute_feautre_weights برای به دست آوردن وزن‌ها استفاده می‌کنیم. این وزن‌ها اهمیت هر ویژگی را در تشخیص خوشه های مختلف در یک مجموعه داده نشان می دهد. روش محاسباتی هم شباهت مقادیر ویژگی در یک خوشه و هم عدم تشابه بین مقادیر ویژگی در خوشه های مختلف را در نظر می گیرد.

```python
def compute_feature_weights(data, label):
    num_sampels=data.shape[0]
    num_features=data.shape[1]
    w = np.zeros(num_features)

    for i in range(num_features):
      for j in range(num_sampels):
        for k in range(num_sampels):
          if label[j] == label[k]:
            w[i] += np.exp(-(data[j,i]-data[k,i])**2)
          else:
            w[i] += 1 - np.exp(-(data[j,i]-data[k,i])**2)

    min_weight = np.min(w)
    max_weight = np.max(w)
    w = (w - min_weight) / (max_weight - min_weight)
    return w
```

شکل ۳۵ محاسبه وزن‌های هر ویژگی

.در این نقشه ، ما به صورت بصری اهمیت ویژگی ها را در مجموعه داده ای از ۵۰۰ نمونه نشان می دهیم. قابل ذکر است، ویژگی‌هایی که نزدیک‌تر به مرکز تصویر قرار دارند، اهمیت بیشتری دارند و با مرکزیت ویژگی‌ها در داده‌های MNIST همسو می‌شوند.



شکل ۳۶: نقشه حرارتی

clustering_weighted_mse_loss، خطای میانگین وزنی مربعات خطا (MSE) را بین مقادیر واقعی (y_true) و مقادیر پیش بینی شده (y_pred) را می‌دهد، همچنین از وزن‌های ویژگی‌های به دست آمده، برای بدست آمدن مقدار دقیق خطاها استفاده می‌کنیم.

```python
# Initialize the DAC model
input_dim = x_train.shape[1]
latent_dim = 784  # Choose an appropriate latent dimension
num_clusters = 10
model = DAC(input_dim, latent_dim, num_clusters)

# Define the loss function (e.g., mean squared error)
loss_fn = clustering_weighted_mse_loss

# Define the optimizer (e.g., Adam)
optimizer = tf.keras.optimizers.Adam(learning_rate=3e-4,weight_decay=0.00001)

# Training parameters
epochs = 50
batch_size = 64

# Training loop
for epoch in range(epochs):
    for i in range(0, len(x_train), batch_size):
        x_batch = x_train[i:i+batch_size]

        with tf.GradientTape() as tape:
            cluster_output = model(x_batch)
            loss = loss_fn(x_batch, cluster_output,weights)

        gradients = tape.gradient(loss, model.trainable_variables)
        optimizer.apply_gradients(zip(gradients, model.trainable_variables))

    print(f"Epoch {epoch + 1}, Loss: {loss.numpy()}")
```

شکل ۳۷ آموزش شبکه **DAC**

این شبکه را با استفاده از loss function تعریف شده که حاصل ترکیب clustering_weighted_mse_loss و l۲ رگورالزیشن می‌باشد، همچنین این از optimizer برای adams برای آموزش این مدل استفاده می‌کنیم.

```
Epoch 1, Loss: 0.024853438138961792
Epoch 2, Loss: 0.01832585409283638
Epoch 3, Loss: 0.015772975981235504
Epoch 4, Loss: 0.014166552573442459
Epoch 5, Loss: 0.013006295077502728
Epoch 6, Loss: 0.01217908039689064
Epoch 7, Loss: 0.011603920720517635
Epoch 8, Loss: 0.01111545693129301
Epoch 9, Loss: 0.010699561797082424
Epoch 10, Loss: 0.010342519730329514
Epoch 11, Loss: 0.009995510801672935
Epoch 12, Loss: 0.009685509838163853
```

شکل ۳۸: **loss** در هر ایپاک

ARI یا "Adjusted Rand Index" یک معیار ارزیابی استفاده می‌شود تا اندازه‌گیری کیفیت یک الگوریتم خوشه‌بندی (مانند K-Means) روی داده‌ها انجام شود. این معیار نمایانگر انطباق بین دسته‌بندی‌های پیش‌بینی شده توسط الگوریتم و دسته‌بندی‌های واقعی (ارائه شده توسط برچسب‌های واقعی داده‌ها) است.

ARI مقداری بین -۱ تا +۱ را اختصاص می‌دهد:

- اگر ARI برابر با ۱ باشد، این نشان می‌دهد که دسته‌بندی پیش‌بینی شده تماماً با دسته‌بندی واقعی مطابقت دارد.

- اگر ARI برابر با ۰ باشد، این نشان می‌دهد که دسته‌بندی پیش‌بینی شده تصادفی است و هیچ تطابقی با دسته‌بندی واقعی ندارد.

- اگر ARI کمتر از ۰ باشد، این نشان می‌دهد که دسته‌بندی پیش‌بینی شده با دسته‌بندی واقعی متضاد است (تضاد در دسته‌بندی).

ARI به عنوان یک اندازه‌گیری از انطباق برای مسائل دسته‌بندی چند کلاسه (با دسته‌بندی‌های واقعی) مورد استفاده قرار می‌گیرد و بهترین مقدار ARI برای یک مدل خوشه‌بندی زمانی به دست می‌آید که تمام داده‌ها در یک خوشه (دسته) قرار گیرند و هیچ تباهی در دسته‌بندی نداشته باشیم.

حال ما قصد داشتیم که با استفاده از این الگوریتم representionهای بهتری برای clustering تصاویر به دست بیاوریم، به این منظور از ARI برای این مقایسه استفاده شده است.

در مقاله این مدل برای ۳۰۰ تا ایپاک آموزش دیده است، اما ما در اینجا برای ۵۰ تا دوره آموزش داده‌ایم و نتایج به صورت زیر بود:

جدول ۱ معیار **ARI**

	دقت
DAC	۰.۵۷
Kmeans	۰.۳۶

همانطور که در جدول مشاهده می‌شود تبعیه درست شده از این مدل تفاوت زیادی از kmeans بدون این مدل دارد که نشان می‌دهد استفاده از اتوانکودر تاثیر زیادی در بدست آوردن تعبیه خوب دارد. که با استفاده از این تعبیه می‌توان با دقت خوبی داده‌های Mnist را clusterکرد.

```
[142] # After training, you can use the encoder part of the model to cluster data
      num_clusters=10
      encoder_output = model.encoder(x_test)
      kmeans = KMeans(n_clusters=num_clusters, random_state=0).fit(encoder_output)
      predicted_labels = kmeans.labels_
      ari = adjusted_rand_score(y_test, predicted_labels)
      print(f"Adjusted Rand Index: {ari}")

      /usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default va
        warnings.warn(
      Adjusted Rand Index: 0.5689845557630019
```

شکل ۳۹ حاصل معیار **ARI**

پاسخ ۴ – شبکه ی Multi-Layer Perceptron

در این سوال با مجموعه داده MNIST آشنا خواهید شد و چند شبکه MLP را برای طبقه بندی این داده آموزش خواهید داد. همچنین در ادامه با بحث Knowledge Distillation آشنا خواهید شد.

برای آشنایی بیشتر با این موضوع مقاله ضمیمه شده را مطالعه کنید.

۱-۴. آشنایی و کار با مجموعه دادگان (پیش پردازش)

هدف از این قسمت آشنایی و کار کردن با مجموعه داده مورد نظر است.

الف) ابتدا مجموعه دادگان MNIST را که مجموعه ای از ارقام دست نویس است فراخوانی کنید برای این کار میتوانید از توابع موجود در کتابخانههای Tensorflow/Keras و Pytorch استفاده نمایید. تعداد و ابعاد دادههای آموزش و آزمون را گزارش کنید .

ب) یک نمونه از هر کلاس را نمایش دهید .

ج) نمودار histogram مربوط به تعداد نمونه‌های هر کلاس را رسم .کنید این نمودار برای داده‌های شده است شکل (۷ همین نمودار را برای داده‌های آموزش رسم کنید.

د) با استفاده از min-max normalization داده ها را به بازه ۰ و ۱ اسکیل کنید.

پاسخ:

کتابخانه ها: با استفاده Pytorch این کد را پیاده‌سازی می‌کنیم.

ابتدا مجموعه‌داده MNIST را دانلود می‌کنیم که یک مجموعه داده محبوب برای تشخیص رقم است، شروع می کنیم. ما همچنین یک تبدیل برای پیش پردازش داده ها تعریف می کنیم که تصاویر را به تانسور تبدیل می کند و همچنین عمل min-max normalization را انجام می‌دهیم.

```
min_value = torch.min(trainset.data.to(dtype=torch.float32))
max_value = torch.max(trainset.data.to(dtype=torch.float32))

# Define a function to normalize the data to the range [0, 1]
def min_max_normalize(data):
    return (data - min_value.numpy()) / (max_value.numpy() - min_value.numpy())
```

شکل ۴۰ **min-max normalization**

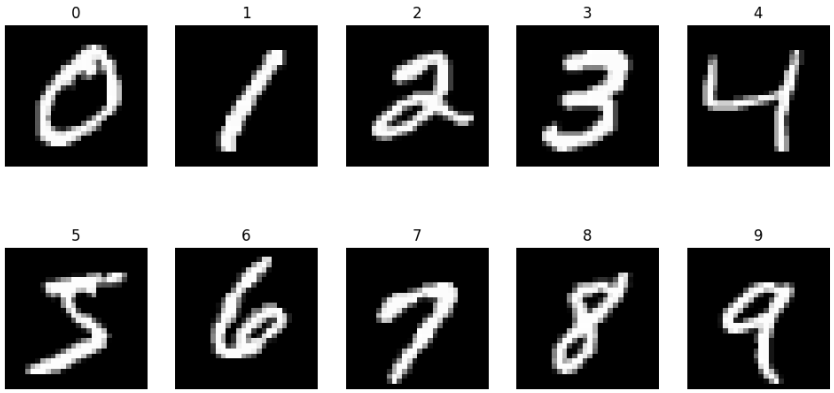
همچنین تعداد و ابعاد این داده‌ها را نشان می‌دهیم.

```
There is 60000 samples in the trainset with sizs of (28, 28)
There is 10000 samples in the testset with sizs of (28, 28)
Also we split validationset for evaluation. There is 2000 samples in the valset with sizs of (28, 28)
```

شکل ۴۱: تعداد و ابعاد داده‌ها

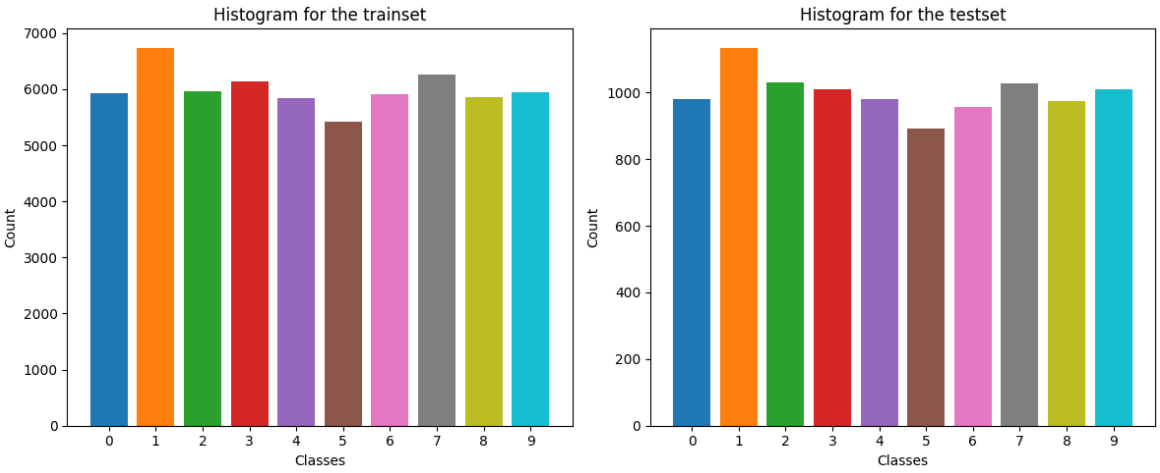
برای اینکه شبکه را ارزیابی کنیم ۲۰ درصد از داده‌ها را به عنوان validation نیز جدا می‌کنیم.

حال تابع show_random_images را تعریف می کنیم که تصاویر تصادفی را از هر کلاس در مجموعه داده MNIST نمایش می دهد. به این صورت که از هر کلاس یک تصویر تصادفی انتخاب می‌کنیم و آن را نمایش می‌دهیم.



شکل ۴۲: نمونه از داده

در این کد، یک تابع plot_histogram برای ایجاد هیستوگرام هایی که توزیع کلاس را در مجموعه داده های آموزشی و آزمایشی نشان می دهد، تعریف می‌کنیم. تعداد نمونه ها را برای هر کلاس شمارش می کند و سپس نمودار میله‌ای را برای تجسم توزیع ترسیم می کند.



شکل ۴۳: هیستوگرام داده‌های آموزش و تست

در این کد، پیش پردازش را روی مجموعه داده انجام می‌دهیم و DataLoader را برای آموزش، اعتبارسنجی و آزمایش ایجاد می‌کنیم.

```python
# Step 3: Preprocess the dataset and create DataLoaders
batch_size = 32

normalized_trainset = [(min_max_normalize(torch.tensor(np.array(image))), label) for image, label in trainset]
normalized_testset = [(min_max_normalize(torch.tensor(np.array(image))), label) for image, label in testset]

#Create validationset

trainset_size = int(0.8 * len(normalized_trainset))  # 80% for training
valset_size = len(normalized_trainset) - trainset_size  # 20% for validation
trainset, valset = torch.utils.data.random_split(normalized_trainset, [trainset_size, valset_size])

# Create DataLoaders



# Create DataLoaders for training, validation, and testing
trainloader = torch.utils.data.DataLoader(normalized_trainset, batch_size=batch_size, shuffle=True)
valloader = torch.utils.data.DataLoader(valset, batch_size=batch_size, shuffle=False)
testloader = torch.utils.data.DataLoader(normalized_testset, batch_size=batch_size, shuffle=False)
```

شکل ۴۴ درست کردن **Dataloader**

۴-۲. Teacher Network

ابتدا یک شبکه عصبی سه لایه را به ترتیب با ۱۰۲۴ و ۵۱۲ نورون در لایه پنهان به عنوان شبکه Teacher بسازید و در لایه های پنهان از تابع فعال سازی ReLU استفاده کنید شکل (۸) دقت کنید که لایه آخر به صورت خطی است و تابع فعالساز ندارد.

```
TeacherNetwork(
  (fc1): Linear(in_features=784, out_features=1024, bias=True)
  (relu1): ReLU()
  (fc2): Linear(in_features=1024, out_features=512, bias=True)
  (relu2): ReLU()
  (fc3): Linear(in_features=512, out_features=10, bias=True)
)
```

شکل ۴۵: معماری شبکه **Teacher**

حال این شبکه را با استفاده از تنظیمات زیر در ۲۰ epoch آموزش می‌دهیم. سپس نمودارهای accuracy و loss در طول آموزش را رسم کنید.

Loss: Cross Entropy

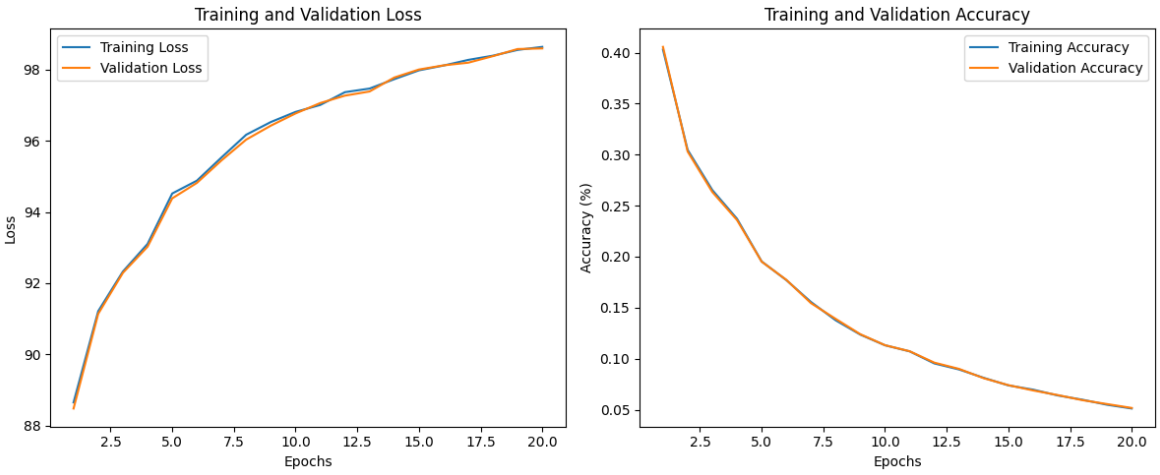
Optimizer: SGD

Batch Size: ۳۲

Learning Rate: ۰.۰۱

برای استفاده از logtit ما خروجی مدل را به عنوان argmax می‌گیریم، و عنصر مربوط به آن، شماره کلاس را نشان خواهد داد. یا همینطور می‌توانیم Softmax بگذاریم و از خروجی argmax استفاده کنیم.

یک کلاس شبکه عصبی PyTorch به نام TeacherNetwork تعریف می کنیم که به عنوان مدل معلم عمل می کند. این شبکه را ابتدا روی داده‌های آموزش برای ۲۰ دوره آموزش می‌دهیم، نتایج دقت و خطا به صورت زیر خواهد بود.

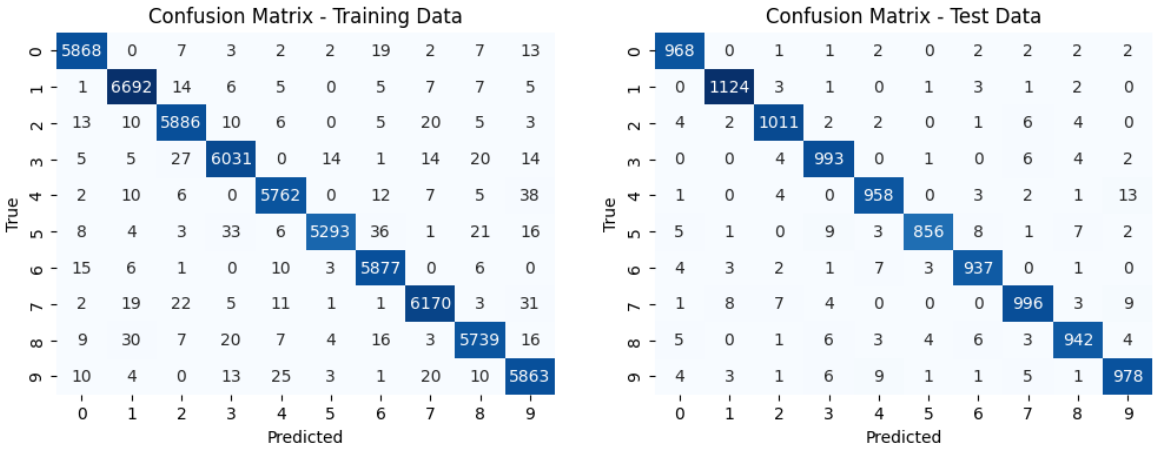


شکل ۴۶: نتایج پیاده سازی **Teacher**

حال این دقت و تعداد misclassification، برای طبقه‌بند را نشان می‌دهیم.

```
Accuracy on train data: 98.64%
Misclassifications on train data: 819
Accuracy on test data: 97.63%
Misclassifications on test data: 237
```

شکل ۴۷: تعداد پاسخ های غلط



شکل ۴۸: نتایج **confusion matrix**

۴-۳. Students Network

حال یک شبکه عصبی سه لایه را به ترتیب با ۱۲۸ و ۶۴ نورون در لایه پنهان به عنوان شبکه Student بسازید و در لایه های پنهان از تابع فعالسازی ReLU استفاده کنید شکل دقت کنید که لایه آخر به صورت خطی است و تابع فعال ساز ندارد.

```
StudentNetwork(
  (fc1): Linear(in_features=784, out_features=128, bias=True)
  (relu1): ReLU()
  (fc2): Linear(in_features=128, out_features=64, bias=True)
  (relu2): ReLU()
  (fc3): Linear(in_features=64, out_features=10, bias=True)
)
```

شکل ۴۹: معماری شبکه **student**

این شبکه را با استفاده از تنظیمات زیر در ۱۰ epoch آموزش دهید پس از پایان آموزش تعداد پیش بینیهای غلط (misclassifications) شبکه روی دادههای آزمون را گزارش کنید.

Loss: Cross Entropy

Optimizer: SGD

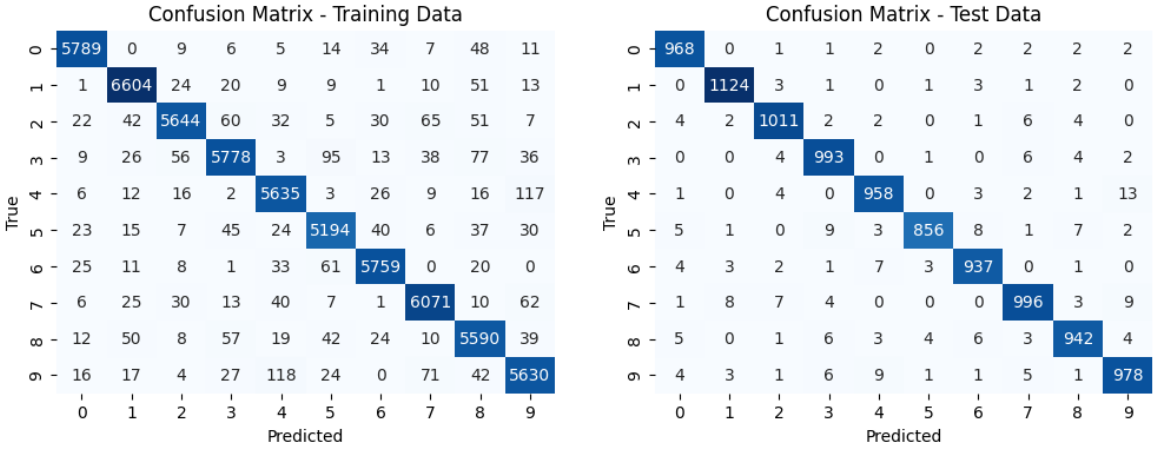
Batch Size: ۳۲

Learning Rate: ۰.۰۱

حال یک مدل Student تعریف می‌کنیم، و ابتدا این مدل را به صورت جدا بر روی داده‌ها آموزش می‌دهیم.
حال این دقت و تعداد misclassification، برای طبقه‌بند را نشان می‌دهیم.

```
Accuracy on train data: 96.16%
Misclassifications on train data: 2306
Accuracy on test data: 97.63%
Misclassifications on test data: 237
```

شکل ۵۰: تعداد پاسخ های غلط



شکل ۵۱: نتایج **confusion matrix**

۴-۴. Knowledge Distillation

حال می‌خواهیم شبکه student را با استفاده از KD آموزش دهیم. این شبکه را با استفاده از تنظیمات زیر در ۱۰ epoch آموزش دهید. پس از پایان آموزش، تعداد پیشبینی های غلط (misclassifications) شبکه روی دادههای آزمون را گزارش کنید.

Loss: MSE

Optimizer: SGD

Batch Size: ۳۲

Learning Rate: ۰.۰۱

پاسخ:

در این کد، تابع Knowledge_Distillation برای انجام انتقال دانش از مدل معلم به مدل دانش آموز تعریف شده است. این کد همچنین شامل یک تابع ارزیابی سفارشی به نام ارزیابی_distilled_model است.

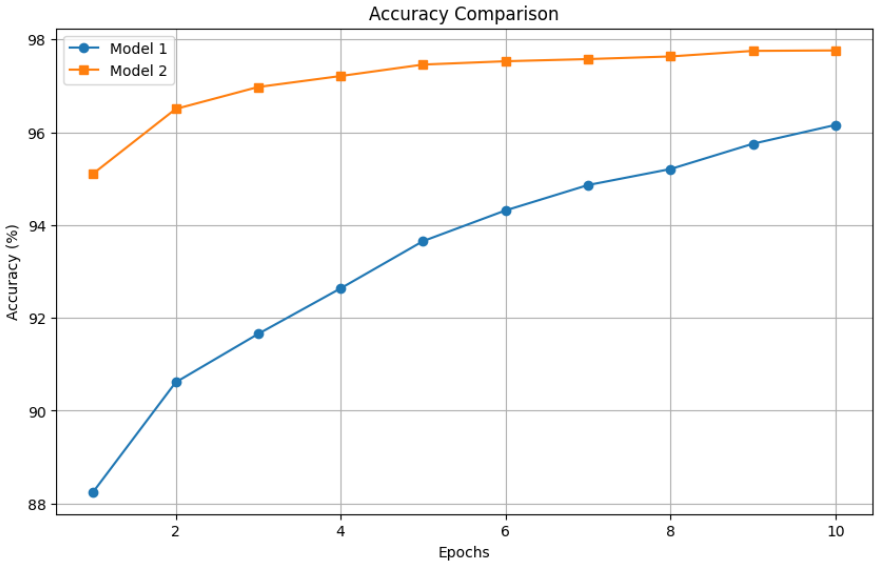
Knowledge_Distillation برای آموزش شبکه آموزش با استفاده از شبکه معلم به عنوان منبع دانش انجام می شود. شبکه دانش آموزی برای تقلید از پیش بینی های معلم بهینه شده است.

شما در حال ارزیابی عملکرد شبکه دانش آموزی در مجموعه داده های آموزشی و آزمایشی هستید. شما دقت و طبقه‌بندی‌های اشتباه را محاسبه می‌کنید و بینش‌هایی را در مورد اینکه مدل چقدر خوب یاد گرفته است، ارائه می‌کنید.

```
Accuracy on train data: 97.76%
Misclassifications on train data: 1342
Accuracy on test data: 97.12%
Misclassifications on test data: 288
```

شکل ۵۲: تعداد پاسخ های غلط

همچنین نمودار زیر دقت آموزش برای حالت آموزش دادن بدون استفاده از معلم و با استفاده از داده‌ی اصلی را نشان می‌دهیم.



شکل ۵۳: مقایسه دقت‌ها

الف)

همچنین دقت و misclassification را برای مدل Student به صورت زیر می‌باشد:

Accuracy on train data: ۹۶.۱۶٪
Misclassifications on train data: ۲۳۰۶
Accuracy on test data: ۹۷.۶۳٪
Misclassifications on test data: ۲۳۷

با استفاده از Teacher:

Accuracy on train data: ۹۷.۷۶٪
Misclassifications on train data: ۱۳۴۲
Accuracy on test data: ۹۷.۱۲٪
Misclassifications on test data: ۲۸۸

ب) با استفاده از Knowledge distillation سعی شده است که عملکرد مدل، دانش‌آموز در چند دوره شباهت به آموزش روی مجموعه‌داده اصلی پیدا کند. این مدل با اینکه با سرعت کمتری همگرا می‌شود، اما دارای generalization بیشتری نسبت به مدل اصلی است، همچنین توانسته تا حد خوبی عملکرد مدل اصلی را روی داده‌ها نشان دهد.