عنوان پروژه:

تشخیص دستخط به کمک شبکه عصبی MLP

آموزشکده آزاد فنی و حرفه ای سما تهران

**پروژه کاردانی الکترونیک**

**استاد راهنما:**

**نوشین مهدوی**

**دانشجو:**

**احمد افکنده**

**شماره دانشجویی:**

**890810561**

**فهرست**

[فصل اول](#_Toc383807031) 1

[1-1 شبکه عصبی زیستی: 2](#_Toc383807032)

[1-2 نرون طبیعی: 2](#_Toc383807033)

[1-3 تعریف شبکه عصبی مصنوعی: 3](#_Toc383807034)

[1-4 نرون‌های مصنوعی: 3](#_Toc383807035)

[1-4-1 سلول عصبی تک ورودی : 4](#_Toc383807036)

[1-4-2 سلول عصبی چند ورودی: 4](#_Toc383807037)

[1-5 توابع فعال ساز(Activation Function): 5](#_Toc383807038)

[1-5-1 تابع انتقال Hard Limit: 5](#_Toc383807039)

[1-5-2 تابع انتقال خطی Pure Linear: 5](#_Toc383807040)

[1-5-3 تابع انتقال : Log Sigmoid 6](#_Toc383807041)

[1-5-4 تابع انتقال Tan Sigmoid: 6](#_Toc383807042)

[1-6 ویژگی‌های شبکه عصبی مصنوعی: 6](#_Toc383807043)

[1-6-1 قابلیت یادگیری1: 7](#_Toc383807044)

[1-6-2 قابلیت تعمیم2: 7](#_Toc383807045)

[1-6-3 پردازش موازی: 7](#_Toc383807046)

[1-6-4 مقاوم بودن: 7](#_Toc383807047)

[1-6-5 تصویر کردن ورودی - خروجی: 8](#_Toc383807048)

[1-7 معایب شبکه‌های عصبی: 8](#_Toc383807049)

[1-8 ساختار‌های مختلف شبکه‌های عصبی مصنوعی: 9](#_Toc383807050)

[1-8-1 شبکه‌های پسخور: 9](#_Toc383807051)

[1-8-2 شبکه‌های پیش خور: 9](#_Toc383807052)

[1-8-3 شبکه‌های پیش خور تک لایه: 9](#_Toc383807053)

[1-8-4 شبکه‌های پیش خور چند لایه: 9](#_Toc383807054)

[1-9 پرسپترون: 10](#_Toc383807055)

[1-10 یادگیری پرسپترون به روش برداری: 10](#_Toc383807056)

[فصل دوم](#_Toc383807057) 11

[2-1 نرم افزار MATLAB: 12](#_Toc383807058)

[2-2 شبکه‌های عصبی در متلب: 12](#_Toc383807059)

[2-3 ‌ایجاد Feed Forward MLP: 12](#_Toc383807060)

[2-4 آموزش شبکه عصبی: 14](#_Toc383807061)

[2-5 دریافت وزن‌های نرون‌ها: 14](#_Toc383807062)

[2-5-1 دریافت ماتریس وزن‌های لایه ورودی: 14](#_Toc383807063)

[2-5-2 دریافت ماتریس وزن‌های لایه‌های میانی و خروجی: 15](#_Toc383807064)

[2-5-3 دریافت ماتریس وزن‌های بایاس تمامی‌لایه‌ها: 15](#_Toc383807065)

[2-6 الگوریتم PCA: 16](#_Toc383807066)

[2-7 کاربرد الگوریتم PCA در شبکه‌های عصبی: 16](#_Toc383807067)

[2-8 الگوریتم PCA در متلب: 17](#_Toc383807068)

[2-9 کتابخانه MNIST: 18](#_Toc383807069)

[2-10 ساخت پروژه در متلب: 19](#_Toc383807070)

[2-10-1 اجرا کردن الگوریتم PCAروی داده‌های ورودی: 19](#_Toc383807071)

[2-10-2 کاهش بُعد نمونه‌ها: 19](#_Toc383807072)

[2-10-3 ساخت شبکه عصبی: 20](#_Toc383807073)

[2-10-4 دریافت وزن‌های شبکه عصبی : 20](#_Toc383807074)

[2-11 فایل mapminmax.m: 21](#_Toc383807075)

[2-11-1 تابع apply: 22](#_Toc383807076)

[2-11-2 تابع reverse 23](#_Toc383807077)

[2-11-3 دریافت ساختار settings از شبکه عصبی 24](#_Toc383807078)

[فصل سوم](#_Toc383807079) 25

[3-1 ساخت نرون در C# 26](#_Toc383807080)

[3-2 ساخت توابع مربوط به MapMinMax 27](#_Toc383807081)

[3-3 ساختار LayerOutputFunction 28](#_Toc383807082)

[3-4 ساختار Settings 28](#_Toc383807083)

[3-5 ساخت شبکه عصبی 28](#_Toc383807084)

[3-6 ساخت پنل ورودی 30](#_Toc383807085)

[3-7 الگوریتم تغییر سایز تصویر و وسط چین کردن آن 30](#_Toc383807086)

[3-7-1 الگوریتم ریسایز کردن تصویر: 30](#_Toc383807087)

[3-7-2 الگوریتم پیدا کردن مختصات و ابعاد تصویر: 31](#_Toc383807088)

[3-8 تبدیل عکس 28×28 به ماتریس 784×1 32](#_Toc383807089)

[3-9 الگوریتم ضرب ماتریس PCT در تصویر 32](#_Toc383807090)

[3-10 تست شکبه عصبی 33](#_Toc383807091)

[3-11 نمونه‌هایی از تست عملکرد پروژه و خروجی‌های آن 34](#_Toc383807092)

**مقدمه:**

شبکه‌های عصبی مصنوعی1 از مباحث جدیدی است که دانشمندان علوم کامپیوتر به آن علاقمند شده‌اند و برای پیشرفت هرچه بیشتر علوم کامپیوتر وقت و هزینه زیادی را صرف آن کرده و می‌کنند.‌این موضوع با‌ایده گرفتن از سیستم عصبی بدن انسان و با هدف شبیه سازی هرچه بیشتر کامپیوتر به انسان شکل گرفت و تابحال به خوبی پیش رفته است.

**معرفی پروژه:**

باتوجه به پیشرفت و حرکت تکنولوژی به سوی ساخت دستگاه‌های هوشمند، همانند موبایل‌های هوشمند که پردازش‌های بسیاری منجلمه تشخیص دستخط را در خود انجام می‌دهند و یا روبات‌هایی که بدون حضور انسان کارهای بسیار سخت و خطرناکی را با دقت فراوان انجام میدهند؛ نیاز بر آن دیدم که قدمی‌در‌این راه نهاده و‌این پروژه را سر آغازی برای ورود به دنیای زیبای هوش مصنوعی قرار دهم.

هدف‌این پروژه آنست که به کمک شبکه عصبی چند لایه MLP1 ، اعداد دست‌نویس را تشخیص داده و معادل کاراکتری آن عدد را اعلام کند.

فصل اول به ارائه مختصر اطلاعاتی در مورد شبکه عصبی اختصاص داده شده است.

در فصل دوم نرم افزار MATLAB و استفاده از توابع مورد نیاز برای اجرای پروژه معرفی شده، همچنین روش PCA2 و کتابخانه MNIST که در اجرای پروژه از آن استفاده کردم را معرفی خواهم کرد.

پس از آن به نحوه ساخت شبکه عصبی و اجرا کردن الگوریتم PCA روی داده‌ها، گرفتن داده‌های مورد نیاز، اعمال آنها به شبکه عصبی و آموزش شبکه عصبی و در پایان گرفتن وزن‌های هر نرون از شبکه جهت ساخت پروژه می‌پردازم.

در فصل سوم به ساخت شبکه عصبی در C#،‌ایجاد یک پنل برای رسم عدد در برنامه، تبدیل عدد رسم شده به ‌اندازه اعداد موجود در کتابخانه MNIST، منتقل کردن عدد به وسط تصویر جهت همخوانی با کتابخانه MNIST ، اعمال عکس به شبکه عصبی و گرفتن خروجی آن خواهم پرداخت.

# فصل اول

**شبکه‌های عصبی**

## 1-1 شبکه عصبی زیستی:

این شبکه‌ها مجموعه‌ای بسیار عظیم از پردازشگرهایی موازی به نام نرون1‌اند که بصورت هماهنگ برای حل مسئله عمل میکنند و توسط سیناپس2‌ها اطلاعات را منتقل میکنند. در‌این شبکه‌ها اگر یک سلول آسیب ببیند بقیه‌ی سلول‌ها می‌توانند نبود آنرا جبران کرده و نیز در بازسازی آن سهیم باشند.

این شبکه‌ها قادر به یادگیری‌اند؛ مثلا با اعمال سوزش به سلولهای عصبی لامسه، سلول‌ها یاد می‌گیرند که به طرف جسم داغ نروند و با‌این الگوریتم سیستم می‌آموزد که خطای خود را اصلاح کند. یادگیری در‌این سیستم‌ها به صورت تطبیقی صورت میگیرد، یعنی با استفاده از مثال‌ها وزن سیناپس‌ها به گونه‌ای تغییر میکند که در صورت دادن ورودی‌های جدید سیستم پاسخ درستی تولید کند.

## 1-2 نرون طبیعی:

نرون‌ها درواقع المان‌های پردازش و بسیار ساده می‌باشند. هر نرون شامل یک سوما که بدنه نرون می‌باشد، یک آکسون و چند دندریت می‌باشد.

نرون‌ها دریافت‌هایی را از سایر نرون‌ها و از طریق دندریت‌ها دارند. وقتی که مقدار سیگنال ورودی بیشتر از یک حد آستانه خاص گردید نرون اصطلاحا فعال میشود؛ درواقع یک واکنش شیمیایی حادث شده و یک پالس الکتریکی که پتانسیل فعالیتی نامیده می‌شود و به آکسون که خروجی نرون می‌باشد، فرستاده می‌شود. از آنجا و از طریق سیناپس‌هایی که متصل به نرون هستند به دندریت‌های سایر نرون‌ها منتقل می‌گردد.

## 1-3 تعریف شبکه عصبی مصنوعی:

شبکه‌های عصبی مصنوعی الگو‌هایی برای پردازش اطلاعات هستند که با تقلید از شبکه عصبی زیستی ساخته شده‌اند.

شبکه‌های عصبی مصنوعی، ساختاری متشکل از تعدادی نرون مصنوعی هستند که در داخل شبکه به یکدیگر متصل شده‌اند. هر نرون دارای یک مشخصه ورودی/خروجی (I/O) می‌باشد و محاسبه یا عملی جزئی را اجرا می‌کند. خروجی هر نرون، با توجه به مشخصه (I/O) آن، اتصالات درونیش به سایر نرون‌ها و ورودی‌های خارجی تعیین می‌گردد.

## 1-4 نرون‌های مصنوعی:

شبکه‌های عصبی براساس مغز انسان مدل شده‌اند، و شامل تعدادی نرون‌های مصنوعی میشوند. نرون‌ها در شبکه‌های عصبی مصنوعی سعی بر‌این دارند تا تعداد پیوند‌های کمتری را نسبت به نرون‌های طبیعی داشته باشند.

هر نرون در یک شبکه عصبی تعدادی ورودی دریافت می‌نماید. تابعی تحت عنوان تابع فعال ساز1 بر روی‌این مقادیر ورودی اعمال میشود که نتیجه آن تحت عنوان سطح فعال سازی نرون به عنوان مقدار خروجی نرون در نظر گرفته میشود. چند تابع ممکنه وجود دارند که می‌توانند برای نرون‌ها مورد استفاده قرار گیرند که در ادامه به آنها پرداخته خواهد شد.

### 1-4-1 سلول عصبی تک ورودی :



در مدل فوق p همان ورودی شبکه است، w وزن ورودی و b مقدار آستانه یا Threshold می‌باشد و F همان تابع فعال ساز و a خروجی نهایی نرون است.

### 1-4-2 سلول عصبی چند ورودی:

عموما یک سلول عصبی چندین ورودی دارد که هرکدام از ورودی‌ها وزن متناظر با خود را دارد.

در مدل زیر بردار p ورودی‌ها و بردار W وزن‌ها میباشند.



به طور کلی رفتار هر نرون را از نظر ریاضی می‌توان به صورت زیر مدل کرد:

که در آن Wi ماتریس وزن ورودی‌های نرون، Xi ماتریس ورودی‌های نرون و b مقدار آستانه یا bias می‌باشد.

## 1-5 توابع فعال ساز(Activation Function):

از توابع فعال ساز یا توابع انتقال برای تعیین خصوصیات نرون در راستای حل مسائل مختلف استفاده میشود.  
برخی از توابع انتقال رایج در شبکه‌های عصبی عبارتند از:

### 1-5-1 تابع انتقال Hard Limit:

این تابع انتقال درصورتی که n<0 باشد خروجی صفر و در صورتی که n≥0 باشد خروجی 1 می‌دهد. از‌این نوع تابع انتقال برای دسته بندی کردن ورودی‌ها به دو کلاس محدود استفاده می‌شود. از‌این تابع عمدتا در نرون‌های سازنده شبکه پرسپترون استفاده میشود.

### 1-5-2 تابع انتقال خطی Pure Linear:

نرون‌هایی که از‌این تابع استفاده می‌کنند برای تقریب خطی به کار میروند.‌این تابع همان مقدار ورودی را به عنوان خروجی بر می‌گرداند.

### 1-5-3 تابع انتقال : Log Sigmoid

از‌این تابع انتقال در شبکه‌های «پس انتشار»‌این تابع انتقال مقادیر ورودی را روی محدوده ±∞ دریافت کرده و مقدار خروجی بین 0 و 1 تولید میکند.

### 1-5-4 تابع انتقال Tan Sigmoid:

این تابع نیز همانند تابع Log Sigmoid ورودی در محدوده ±∞ دریافت کرده و خروجی بین -1 تا 1میدهد.

## 1-6 ویژگی‌های شبکه عصبی مصنوعی:

شبکه عصبی قدرت محاسباتی خود را در درجه اول از ساختار گسترده موازی حجیم خود و در درجه دوم از توانایی یادگیری و تعمیم دهی بدست می‌آورد. خاصیت تعمیم دهی به‌این معناست که شبکه عصبی برای ورودی‌هایی که در طول آموزش با آنها مواجه نشده است خروجی‌های قابل قبولی تولید میکند.

این دو قابلیت پردازش اطلاعات‌این امکان را به شبکه عصبی میدهد که به مسائل پیچیده و دارای مقیاس وسیع که تا کنون غیر قابل بررسی بوده‌اند، پاسخ دهد؛ با‌این وجود، عملا شبکه‌های عصبی به تنهایی نمیتوانند پاسخ همه مسائل را بدهند بلکه باید با یک روش مهندسی سیستم پایدار ترکیب شوند. درواقع مساله پیچیده مورد نظر به تعدادی مساله نسبتا ساده‌تر تجزیه میشود و شبکه عصبی وظیفه‌ای را که با قابلیت‌های ذاتی هر زیر مجموعه هماهنگ باشد، اختصاص میدهد.

از برخی ویژگی‌ها و مزایای شبکه عصبی میتوان موارد زیر را نام برد:

### 1-6-1 قابلیت یادگیری1:

قابلیت یادگیری یعنی توانایی تنظیم پارامتر‌های شبکه (یا همان ماتریس وزن‌های هر نرون)، در زمانی که شبکه شرایط جدیدی را تجربه می‌کند. تنظیم پارامتر‌های شبکه با‌این هدف صورت میگیرد که اگر شبکه برای یک وضعیت خاص آموزش دید و تغییر کوچکی در شرایط محیطی رخ داد، بتواند با آموزشی مختصر و یا بدون آموزش مجدد در شرایط جدید نیز کارآمد باشد.

### 1-6-2 قابلیت تعمیم2:

پس از آنکه شبکه مثال‌های اولیه را آموزش دید، شبکه میتواند در برابر یک ورودی آموزش داده نشده قرار گیرد و یک خروجی مناسب ارائه نماید.‌این خاصیت شبکه را قادر می‌سازد که تنها با برخورد با تعداد محدودی از یک نمونه، قانون کلی آنرا بدست آورده و نتایج‌این آموخته‌ها را به موارد جدید نیز تعمیم دهد. قابلیتی که با نبود آن، سیستم باید بینهایت واقعیت‌ها و روابط را بخاطر بسپارد.

### 1-6-3 پردازش موازی:

در شبکه‌های عصبی، نرون‌ها بطور همزمان به ورودی‌های یک تراز پاسخ میدهند؛‌این ویژگی باعث افزایش سرعت پردازش و محاسبات میگردد.

### 1-6-4 مقاوم بودن:

در یک شبکه عصبی هر نرون به طور مستقل عمل میکند و رفتار کلی شبکه، برآیند رفتار‌های نرون‌های متعدد است.‌این ویژگی باعث میشود تا خطاهای محلی تاثیر کمتری بر خروجی نهایی بگذارند.

### 1-6-5 تصویر کردن ورودی - خروجی:

یک روش اساسی رایج در یادگیری که با نام یادگیری با مربی یا یادگیری با نظارت شناخته می‌شود عبارت است از اصلاح وزن‌های سیناپسی یک شبکه عصبی با اعمال مجموعه‌ای از نمونه‌های یادگیری. در‌این روش یک نمونه به طور تصادفی از مجموعه انتخاب و به شبکه عصبی ارائه میشود و وزن‌های سیناپسی شبکه به گونه‌ای اصلاح می‌شوند که تفاوت بین پاسخ مطلوب و پاسخ واقعی شبکه را که در اثر اعمال سیگنال ورودی‌ایجاد شده است از نظر آماری به حداقل برساند. آموزش شبکه برای تمامی‌نمونه‌های مجموعه تکرار می‌شود تا زمانی که شبکه به حالت پایداری برسد و هیچ تغییر قابل توجهی در وزن‌های سیناپسی بوجود نیاید. نمونه‌های قبلی میتوانند در طول دوره آموزش با تغییر در ترتیب مجدداً به کار روند. بدین ترتیب شبکه با یادگیری از نمونه‌ها یک تصویر ورودی – خروجی‌ایجاد میکند و مساله مورد نظر را یاد می‌گیرد.

## 1-7 معایب شبکه‌های عصبی:

در کنار مزایای ذکر شده، شبکه عصبی مشکلاتی نیز دارد:

* مشخص نیست که بهترین تعداد واحد‌های محاسبه گر و ارتباطات یک شبکه از چه رابطه منطقی پیروی میکنند.
* در برخی موارد زمان محاسبه طولانی برای آموزش یک شبکه نیاز است.
* نیاز به کاهش پیچیدگی شبکه
* نیاز به ترکیب شبکه‌های عصبی مصنوعی با سایر روند‌های محاسباتی برای ساخت یک سیستم هیبرید.
* ابهام در چگونگی رسیدن یک شبکه آموزش دیده به داده‌های خروجی.

## 1-8 ساختار‌های مختلف شبکه‌های عصبی مصنوعی:

شبکه‌های عصبی چیزی بجز نرون‌های به هم پیوسته نیستند. شکل اتصال نرون‌ها به یکدیگر یکی از متغیر‌های اصلی در طراحی شبکه‌های عصبی مصنوعی و تنظیم نوع شبکه است.

### 1-8-1 شبکه‌های پسخور:

چنانچه بتوان در شبکه‌ای حداقل یک حلقه پیدا کرد که در آن سیگنالی از یک نرون به خودش یا نرون‌های همان لایه برگردد،‌این شبکه دارای پسخور یا فیدبک است.

### 1-8-2 شبکه‌های پیش خور:

چنانچه در یک شبکه عصبی، خروجی هر نرون تنها به نرون‌های لایه بعد وصل شود، به آن پیش خور یا Feed Forward Network میگویند. شبکه عصبی پیش خور را کاملا متصل گویند، اگر هر سلول در هر لایه از همه نرون‌های لایه قبل ورودی گرفته باشد. اما اگر بعضی از اتصالات سیناپسی حذف شده باشند، شبکه حاصل را متصل جزئی گویند. بر‌این اساس تقسیم بندی‌های زیر برای ساختار‌های مختلف شبکه‌های عصبی مصنوعی انجام شده است.

### 1-8-3 شبکه‌های پیش خور تک لایه:

در‌این شبکه‌ها لایه‌های میانی وجود ندارد. شبکه پیش خور تک لایه از یک لایه ورودی و یک لایه خروجی تشکیل شده است.

### 1-8-4 شبکه‌های پیش خور چند لایه:

این نوع از شبکه‌ها، با اضافه نمودن یک یا چند لایه پنهان به شبکه‌های پیش خور تک لایه بدست می‌آیند. شبکه‌های عصبی پیش خور چند لایه دارای توانایی بیشتری نسبت به شبکه‌های تک لایه‌اند؛ به طوری که شبکه‌های عصبی پیش خور دو لایه‌ای با توابع سیگموئید در لایه اول، قادرند هر تابعی را با دقت

دلخواه تقریب بزنند. شبکه‌های عصبی تک لایه، از چنین توانایی برخوردار نیستند. اگرچه در نگاه اول به نظر می‌رسد تعداد درجات آزادی ( مثلا تعداد ورودی‌ها، خروجی‌ها و نرون‌های هر لایه ) برای طراحی یک شبکه چند لایه پیش خور زیاد است، ولی باید توجه داشت تعداد ورودی‌های شبکه و تعداد خروجی‌های آن بر مساله خاصی که شبکه قرار است حل کند تاثیر دارد.

## 1-9 پرسپترون:

پرسپترون1 یک نرون ساده است که می‌تواند هر تعداد ورودی داشته باشد و ورودی‌هایش به یک یا دو دسته مجزا کلاس بندی می‌کند.

## 1-10 یادگیری پرسپترون به روش برداری:

اگر ورودی یک پرسپترون را به صورت بردار بنویسیم می‌توانیم الگوریتم مورد بحث را به شکل برداری مورد نظر قرار دهیم. بردار X دارای n مولفه است و بنابراین n بعدی شناخته میشود. ما حداکثر قادر به تصور سه بعد هستیم، لیکن میتوان آنچه در ابعاد بالاتر نیز می‌گذرد را تا حدی احساس کرد. اگر ضرایب وزنی را به صورت بردار دیگری بنویسیم آنگاه میتوانیم به جای جمع وزنی حاصل ضرب داخلی دو بردار X وW را محاسبه کنیم.

*الگوریتم فراگیری پرسپترون تضمین میکند که ضرایب وزنی هربار در جهت تقلیل خطا تعدیل یابند.*

# فصل دوم

***ساخت پروژه در***

***MATLAB***

## 2-1 نرم افزار MATLAB:

نرم افزار متلب یکی از پرکاربرد ترین نرم افزار‌های محاسبات ریاضی است.‌این نرم افزار پرقدرت کاربرد‌های بسیاری در رشته برق و کامپیوتر دارد.

این نرم افزار همچنین از قدرت بالایی جهت ساخت و آموزش شبکه‌های عصبی برخوردار است که تنها دلیل انتخاب‌این نرم افزار جهت اجرای پروژه می‌باشد.

## 2-2 شبکه‌های عصبی در متلب:

متلب قادر است انواع شبکه‌های عصبی را ساخته و آموزش دهد، اما از آنجا که هدف‌این پروژه‌ایجاد شبکه عصبی چند لایه پرسپترون (MLP) است، فقط ساخت‌این شبکه معرفی می‌شود.

## 2-3 ‌ایجاد Feed Forward MLP:

همانگونه که در فصل اول به شبکه‌های عصبی پیش خور اشاره شد،‌اینگونه شبکه‌ها با تعداد ورودی کافی قادرند تقریبا هر مساله‌ای را حل کنند، با توجه به‌این نکته و همچنین سادگی نرون پرسپترون،‌این شبکه یک شبکه مناسب برای اجرای پروژه دیده شد.

با نوشتن کد زیر در متلب میتوان‌این شبکه را ‌ایجاد کرد:

net = newff ( P , T , S , TF );

net : متغیری که شبکه ساخته شده را در خود نگهداری می‌کند

P : ماتریس ورودی‌های شبکه

T : ماتریس جواب‌های شبکه به ازای هر ستون از ماتریس ورودی‌ها

S :‌این ماتریس تعداد نرون‌های هر لایه را مشخص می‌کند

TF : تابع انتقال خروجی هر لایه را مشخص می‌کند.

نکته قابل توجه‌ اینست که تعداد سطر‌های ماتریس P تعداد ورودی‌های شبکه و تعداد ستون‌های آن تعداد نمونه‌هایی است که به شبکه جهت آموزش داده می‌شوند.

مثال:‌ایجاد یک شبکه جهت اجرای تابع XOR:

ماتریس ورودی‌های شبکه

ماتریس خروجی‌ها یا ماتریس پاسخ‌های شبکه

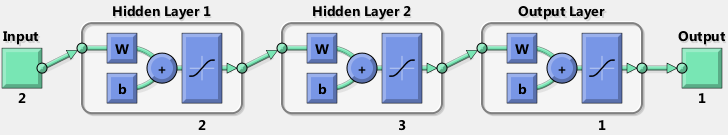
net = newff (P , T , {2,3}, { 'tansig' , 'tansig' , 'tansig' } );

در‌این مثال یک شبکه پیش خور با 2 ورودی برای هر نرون، دارای 2 نرون در لایه ورودی که هر کدام 2 ورودی میگیرند ، 3 نرون در لایه میانی که هرکدام 2 ورودی میگیرند و یک نرون در لایه خروجی که 3 ورودی میگیرد ساختیم، که خروجی همه لایه‌ها از تابع tansig عبور میکنند.

تعداد ورودی‌های نرون‌های لایه ورودی را ماتریس ورودی و خروجی‌های نرون‌های لایه خروجی را ماتریس خروجی شبکه تعیین میکند.

شبکه ساخته شده در متلب را با دستور زیر میتوان دید:

**net.view;**



نکته قابل توجه‌اینکه همه ورودی‌ها به همه نرون‌های لایه ورودی وصل می‌شوند.

## 2-4 آموزش شبکه عصبی:

دستور آموزش شبکه :

net1 = train ( net , P , T );

این دستور شبکه ساخته شده و ماتریس‌های ورودی و پاسخ‌های شبکه را دریافت کرده شبکه را آموزش داده و شبکه آموزش دیده را در متغیر 1net قرار می‌دهد.

نکته‌ای که باید به آن توجه ویژه‌ای شود‌این است که متلب برای آموزش شبکه عصبی وزن‌های اولیه را کاملا تصادفی انتخاب کرده و ممکن است برای آموزش صحیح شبکه عصبی لازم باشد چندین بار عمل آموزش تکرار شود تا به نتیجه مطلوب برسیم.

## 2-5 دریافت وزن‌های نرون‌ها:

متلب به تعداد ورودی‌های هر نرون، یک ماتریس وزن‌ایجاد میکند.

به عنوان مثال اگر لایه ورودی 5 نرون داشته باشد و 10 ورودی داشته باشیم ماتریس وزن‌های ورودی به صورت زیر میشود:

پس از آنکه شبکه آموزش داده شد، وزن‌های نرون‌ها و وزن‌های بایاس را که در فصل اول به آنها اشاره شد باید از شبکه دریافت کرد؛ تا بتوان شبکه را در خارج از محیط متلب و برای استفاده در محیط واقعی ساخته و تست کرد.

### 2-5-1 دریافت ماتریس وزن‌های لایه ورودی:

در متلب ماتریس وزن‌های لایه ورودی را با دستور IW میتوان گرفت و کاربرد آن به صورت:

var = net.IW;

## 2-5-2 دریافت ماتریس وزن‌های لایه‌های میانی و خروجی:

ماتریس وزن تمامی‌لایه‌های میانی (hidden layer) و لایه خروجی را با دستور LW میتوان گرفت و کاربرد آن بصورت :

var = net.LW;

## 2-5-3 دریافت ماتریس وزن‌های بایاس تمامی‌لایه‌ها:

وزن‌های بایاس تمامی‌لایه‌ها در ماتریس b قرار دارد که به صورت زیر قابل دسترسی‌اند:

var = net.b;

## 2-6 الگوریتم PCA:

PCA مخفف Principal Component Analysis، یک روش تحلیل آماری است که به بررسی اجزای اصلی در یک مجموعه از داده‌ها می‌پردازد.‌این الگوریتم کمک میکند که مهمترین مولفه‌ها در مجموعه‌های داده شده را پیدا کرده و ارزش هرکدام از مولفه‌ها را میدهد.

کاربرد اصلی آن در‌اینست که داده‌های کم ارزش و یا بدون ارزش را مشخص کرده و با حذف آنها باعث کاهش داده‌های ورودی گردد.

## 2-7 کاربرد الگوریتم PCA در شبکه‌های عصبی:

همانگونه که اشاره شد‌این الگوریتم از بین تعداد بسیاری نمونه درون ماتریسی به ابعاد m\*n (ماتریس Data) نقاط مهم آنها را پیدا کرده و یک ماتریس n\*n به عنوان خروجی الگوریتم (ماتریس PCAout) تولید خواهد کرد. علاوه بر آن یک ماتریس m\*1 نیز برای مشخص کردن اهمیت هر یک از سطر‌های ماتریس خروجی تولید کرده (ماتریس lattent)، که ارزش هر سطر را با یک عدد مشخص میکند.

اگر از ماتریس خروجی (ماتریس PCAout) ، سطر‌های با ارزش بالاتر را جدا کنیم در (ماتریس PCT) قراردهیم ماتریسی به ابعاد z\*n بدست می‌آید، و اگر‌این ماتریس را در هر یک از سطرهای ماتریس Data ضرب کنیم، ماتریس حاصله (ماتریس Y) ماتریسی با ابعاد m\*z خواهد شد.

ماتریس Y همان ماتریسی خواهد بود که تمامی‌مشخصات ماتریس Data را خواهد داشت اما با ابعاد کمتر؛ در نتیجه‌این کاهش بُعد میتوان شبکه عصبی‌ای ساخت که همه آن نمونه‌ها (ماتریس Data) را بتواند یاد بگیرد اما شبکه بسیار کوچکی نیز باشد.

به عنوان مثال اگر ماتریس Data ماتریسی با ابعاد 60000×784 باشد، 60000 نمونه با 784 داده متفاوت خواهیم داشت، اگر برای‌این نمونه‌ها بخواهیم شبکه عصبی درست کنیم باید لایه ورودی آن 784 نرون داشته باشد که باعث میشود شبکه عصبی بسیار بزرگ و محاسبات و آموزش آن بشدت زمان‌گیر باشد.

اما اگر روی همین ماتریس الگوریتم PCA اجرا کنیم با توجه به ماتریس lattent میبینیم که سطرهای پایینی‌این ماتریس عملا صفر بوده و هیچ ارزشی در شبکه عصبی نخواهند داشت، ولی چند سطر اول در‌این ماتریس ارزش عددی بسیار بالایی دارند پس اگر از ماتریس PCAout به‌اندازه همان چند سطر با ارزش جدا کنیم، در‌اینصورت تعداد نرون‌های لایه ورودی شبکه به تعداد همان چند سطر خواهد بود.

در ادامه با ساختن پروژه‌این مورد را بیشتر توضیح خواهم داد.

## 2-8 الگوریتم PCA در متلب:

در متلب الگوریتم PCA با دستور زیر قابل دسترسی است:

[ PCAout , Score , lattent ] = princomp ( Data ) ;

PCAout : ماتریس خروجی الگوریتم

Score : ماتریس امتیاز‌های هر یک از داده‌ها (ابعاد‌این ماتریس با ابعاد ماتریس ورودی برابر است)

Lattent : ارزش هریک از سطر‌های ماتریس خروجی در‌این ماتریس ریخته می‌شود

Data : ماتریس ورودی که نمونه‌ها در‌این ماتریس می‌باشند

## 2-9 کتابخانه MNIST:

این کتابخانه شامل 60000 عکس با ابعاد 28×28 می‌باشد، که همگی این عکس‌ها عکس‌های کوچک شده از اعداد دست‌نویس هستند.

همه اعداد درون عکس‌های این کتابخانه در وسط عکس قرار دارند. این کتابخانه از 2 فایل تشکیل شده که فایل اول عکس‌ها و فایل دوم عددی که نمایانگر معنای عکس‌ها می‌باشد را در خود نگهداری می‌کند.

توضیحاتی که در این سایت در مورد کتابخانه داده شده است به شرح ذیل می‌باشند:

**TRAINING SET LABEL FILE (train-labels-idx1-ubyte):**

[offset] [type] [value] [description]   
0000 32bit integer  0x00000801(2049) magic number (MSB first)   
0004 32 bit integer  60000            number of items   
0008 unsigned byte   ??               label   
0009 unsigned byte   ??               label   
........   
xxxx unsigned byte   ??               label

The labels values are 0 to 9.

**TRAINING SET IMAGE FILE (train-images-idx3-ubyte):**

[offset] [type] [value] [description]   
0000 32bit integer  0x00000803(2051) magic number   
0004 32bit integer  60000            number of images   
0008 32bit integer  28               number of rows   
0012 32bit integer  28               number of columns   
0016 unsigned byte   ??               pixel   
0017 unsigned byte   ??               pixel   
........   
xxxx unsigned byte   ??               pixel

Pixels are organized row-wise. Pixel values are 0 to 255. 0 means background (white), 255 means foreground (black).

همانگونه که در توضیحات آمده است عکس‌ها همگی سیاه و سفید بوده و بصورت پیکسل به پیکسل ذخیره شده اند. در فایل عکس‌ها، اطلاعات از شانزدهمین بایت شروع شده و در فایل حاوی برچسب‌های هر عکس، اطلاعات از هشتمین بایت شروع می‌شوند.

## 2-10 ساخت پروژه در متلب:

### 2-10-1 اجرا کردن الگوریتم PCAروی داده‌های ورودی:

پس از وارد کردن عکس‌ها و لیبل‌های آنها در متلب، اولین قدم اجرا کردن الگوریتم PCA روی عکس‌ها است. نام ماتریس عکس‌ها را Data و نام ماتریس لیبل‌ها را L گذاشتم.

[PCAout , Score , Lattent ] = princomp(Data);

با توجه به ماتریس lattent، 18 سطر اول از ماتریس PCAout را انتخاب کرده و با کد زیر در ماتریس PCT کپی کردم:

PCT = PCAout( 1:18 , : );

نکته: در قسمت اول علامت ":" به معنای سطرهای یک تا 18 و در قسمت دوم به معنای کل داده‌های هر\_ سطر میباشد.

نکته: در کتابخانه MNIST رنگ سیاه با صفر و رنگ سفید با 255 نشان داده میشود، اما در ماتریس Data این پروژه رنگ سیاه را صفر و رنگ سفید را با 1 جایگزین کردم.

از اینجا به بعد فقط به ماتریس PCT نیاز خواهیم داشت و به دیگر ماتریس‌های حاصله از الگوریتم PCA نیازی نیست.

### 2-10-2 کاهش بُعد نمونه‌ها:

بعد از بدست آوردن ماتریس PCT لازم است که نمونه‌هایی را که هرکدام متشکل از 784 پیکسل می‌باشند را به نمونه‌هایی با 18 داده تبدیل کرد، ماتریس حاصله را (ماتریسY) به شکل زیر بدست می‌آوریم:

Y = PCT \* Data' ;

نکته: ماتریس Data به صورت 60000 سطر و 784 ستون می‌باشد و ماتریس PCT به صورت 18سطر و 784 ستون پس برای آنکه بتوان این دو ماتریس را در یکدیگر ضرب کرد لازم است ماتریس Data معکوس شود، این کار در متلب با دستور « ' » انجام می‌شود.

ماتریس Y اکنون 18 سطر و 60000 ستون خواهد داشت که هر ستون در این ماتریس نماینده سطر معادل در ماتریس Data می‌باشد.

با بدست آوردن ماتریس Y دیگر به ماتریس Data نیز نیازی نخواهیم داشت، پس در این لحظه ماتریس‌های Y ، L و PCT تنها ماتریس‌های مورد نیاز می‌باشند.

ممکن است این سوال پیش آید که چرا هنوز به ماتریس PCT نیاز است؟ پاسخ اینست که این ماتریس قرار است ابعاد داده‌های ورودی را کاهش دهد و بعد از آنکه شبکه عصبی ساخته و آموزش داده شد؛ برای تست شبکه عصبی در دنیای واقعی نیز لازم است این ماتریس در ماتریس عددی که کاربر رسم میکند ضرب شود.

### 2-10-3 ساخت شبکه عصبی:

بعد از بدست آوردن ماتریس Y، دیگر زمان ساخت و آموزش شبکه عصبی رسیده است:

Net=newff ( Y , L , { 20 , 5 } , { 'tansig' , 'tansig' , 'purelin' } );

Net = train ( Net , Y , L );

شبکه عصبی حاصله بصورت:

### 2-10-4 دریافت وزن‌های شبکه عصبی :

با ساخته شدن و آموزش دیدن شبکه عصبی، آخرین مرحله دریافت وزن نرون‌ها و وزن‌های بایاس است. همانگونه که در بخش 2-5 توضیح داده شد، ماتریس وزن‌های لایه‌ها را میگیریم.

در فصل بعد از این ماتریس‌ها برای ساخت شبکه عصبی در محیط C# استفاده میکنیم.

## 2-11 فایل mapminmax.m:

در متلب اطلاعات قبل از آنکه وارد شبکه عصبی شوند، وارد این فایل شده تغییراتی روی آنها اعمال و سپس به شبکه عصبی داده می‌شوند. همچنین خروجی شبکه قبل از آنکه توسط کاربر دیده شود یکبار دیگر وارد این فایل شده و بعد از آن نتیجه روی خروجی نشان داده می‌شود.

### 2-11-1 تابع apply:

اطلاعات ورودی شبکه عصبی وارد این تابع می‌شوند و خروجی تابع ماتریسی به همان اندازه اطلاعات ورودی می‌باشد.

کد این تابع در متلب:

function y = apply(x,settings)

xrange = settings.xrange;

xmin = settings.xmin;

gain = settings.yrange ./ xrange;

fix = find(~isfinite(xrange) | (xrange == 0));

gain(fix) = 1;

xmin(fix) = settings.ymin;

Q = size(x,2);

copyQ = ones(1,Q);

y = (x - xmin(:,copyQ)) .\* gain(:,copyQ) + settings.ymin;

end

همانگونه که در تعریف تابع آمده است، این تابع ماتریس x و ساختار settings را میگیرد و یک ماتریس y برمی‌گرداند. ماتریس x همان ورودی‌های خام می‌باشند و settings ساختاری است که متلب در زمان ساخت شبکه عصبی می‌سازد که در ادامه گرفتن این ساختار را نیز توضیح خواهم داد.

### 2-11-2 تابع reverse

ماتریس خروجی لایه‌ی خروجی شبکه عصبی به این تابع وارد شده و خروجی این تابع خروجی نهایی شبکه عصبی می‌باشد.

کد این تابع در متلب:

function x = reverse(y,settings)

xrange = settings.xrange;

xmin = settings.xmin;

gain = xrange ./ settings.yrange;

fix = find(~isfinite(xrange) | (xrange == 0));

gain(fix) = 1;

xmin(fix) = settings.ymin;

Q = size(y,2);

copyQ = ones(1,Q);

x = (y - settings.ymin) .\* gain(:,copyQ) + xmin(:,copyQ);

end

### 2-11-3 دریافت ساختار settings از شبکه عصبی

ساختار settings را با مجموعه دستورات زیر می‌توان از شبکه عصبی گرفت:

fcns = nn.subfcns(net);

pfcns = nnproc.active\_fcns(fcns.inputs(1).process);

settings = pfcns.settings;

متغیر settings اکنون ساختار settings را دارا می‌باشد و مقادیری که در دو تابع apply و reverse استفاده میشوند را از روی این ساختار کپی کرده و برای ساخت شبکه استفاده می‌شود.

# فصل سوم

**ساخت پروژه در**

**C#**

## 3-1 ساخت نرون در C#

هر نرون همانگونه که اشاره شد، از لایه قبلی‌اش یک یا چند ورودی گرفته در ماتریس وزن‌هایش ضرب کرده و با یک مقدار بایاس جمع میکند. در نهایت خروجی نرون به یک تابع انتقال داده میشود و خروجی تابع انتقال میشود ورودی لایه بعدی.

در کلاس زیر سازنده کلاس تعداد ورودی‌های نرون را میگیرد.

public class Neuron

{

public Neuron ( int SumOfInputs )

{

m\_SumOfInputs = SumOfInputs;

}

public double act(double[] Input, double[] weight, double bias, LayerOutputFunction func)

{

double tmp =bias;

for (int i = 0; i < m\_SumOfInputs; i++)

tmp +=(Input[i] \* weight[i]);

if (func == LayerOutputFunction.Tansig)

m\_output = 2 / (1 + Math.Exp(-2 \* tmp)) - 1;

else if (func == LayerOutputFunction.Purelin)

m\_output = tmp;

return m\_output;

}

public double m\_output;

private int m\_SumOfInputs;

};

تابع act همان تابعی است که ورودی‌ها را گرفته، جواب را محاسبه میکند. این تابع ماتریس Input را به عنوان ماتریس ورودی‌های نرون، ماتریس weight را به عنوان ماتریس وزن‌های نرون، عدد bias را به عنوان مقدار بایاس و ساختار LayerOutputFunction را برای تعیین تابع انتقال مورد استفاده، میگیرد.

## 3-2 ساخت توابع مربوط به MapMinMax

برای ساخت دو تابع مورد استفاده apply و reverse ابتدا لازم است تنظیمات مورد استفاده که در فصل دوم نحوه گرفتن آن توضیح داده شد را گرفته در متغیر‌های زیر ذخیره کنیم:

IXmin, IXRange, IYRange, IYMin

همچنین در سازنده کلاس نیز متغیر‌های زیر از درون ساختار Outminmaxsettings را مقدار دهی کنیم.

xmin, xRange , ymin, yRange

کلاس MapMinMax:

public class MAPMinMax{

public MAPMinMax(){

for (int i = 0; i < 9; i++){//initializing Input minmaxsettings array

Inminmaxsettings[i].xmin = IXmin[i];

Inminmaxsettings[i].xRange = IXRange[i];

Inminmaxsettings[i].ymin = IYMin;

Inminmaxsettings[i].yRange = IYRange;

}

//initializing Output minmaxsettings

Outminmaxsettings.xmin = xx;

Outminmaxsettings.xRange = xx;

Outminmaxsettings.ymin = xx;

Outminmaxsettings.yRange = xx;

}

public double applyMinMax(double input, Settings settings){

double gain = settings.yRange / settings.xRange;

double xmin = settings.xmin;

if (gain == 0 || double.IsInfinity(gain) || double.IsNaN(gain)){

gain = 1;

xmin = settings.ymin;

}

return ((input - xmin) \* gain + settings.ymin);

}

public double reverseMinMax(double y, Settings settings){

double gain = settings.xRange / settings.yRange;

double xmin = settings.xmin;

if (gain == 0 || double.IsInfinity(gain)){

gain = 1;

xmin = settings.ymin;

}

return ((y - settings.ymin) \* gain + xmin);

}

public Settings[] Inminmaxsettings = new Settings[18];

public Settings Outminmaxsettings = new Settings();

double[] IXmin, [] IXRange, IYRange, IYMin;

}

## 3-3 ساختار LayerOutputFunction

این ساختار جهت تعیین نوع تابع انتقال نرون‌ها استفاده می‌شود.

public enum LayerOutputFunction

{

Purelin = 0,

Tansig

};

## 3-4 ساختار Settings

این ساختار جهت نگهداری ساختارSettings که در فصل دوم به آن اشاره شد کاربرد دارد.

struct Settings

{

public double xRange;

public double xmin;

public double yRange;

public double ymin;

};

## 3-5 ساخت شبکه عصبی

قبل از استفاده باید همانگونه که در فصل دوم گفته شد وزن‌های لایه‌ها را گرفته در متغیر‌های زیر ذخیره کنیم:

IW, HW, OW, IB, HB, OB

public class Net

{

public Net()

{

int i;

//initializing inputLayer Neurons

for (i = 0; i < 20; i++)

InputLayer[i] = new Neuron(18);

//initializing HiddenLayer Neurons

for (i = 0; i < 5; i++)

HiddenLayer[i] = new Neuron(9);

//initializing OutputLayer

OutputLayer = new Neuron(5);

}

public double Calculate(double[] ins)

{

double[] ILay\_Outputs = new double[20]; متغیری که پاسخ‌های لایه ورودی را نگهداری میکند

double[] HLay\_Outputs = new double[5];متغیری که پاسخ‌های لایه میانی را نگهداری میکند

double[] inputs = new double[18];

int i;

/////////////////////////////////////////////////////////////////////////////////////

//input correction

for (i = 0; i < 18; i++)

inputs[i] = MinMax.applyMinMax(ins[i], MinMax.Inminmaxsettings[i]);

/////////////////////////////////////////////////////////////////////////////////////

//inputLayer acting

for (i = 0; i < 20; i++)

{

double[] weights = new double[18];

for (int y = 0; y < 18; y++)

weights[y] = IW[i, y];

ILay\_Outputs[i]= InputLayer[i].act(inputs, weights, IB[i],LayerOutputFunction.Tansig);

}

//HiddenLayer acting

for (i = 0; i < 5; i++)

{

double[] weights = new double[20];

for (int y = 0; y < 20; y++)

weights[y] = HW[i, y];

HLay\_Outputs[i]=HiddenLayer[i].act(ILay\_Outputs, weights,

HB[i],LayerOutputFunction.Tansig);

}

//OutputLayer acting

OutputLayer.act(HLay\_Outputs, OW, OB,LayerOutputFunction.Purelin);

/////////////////////////////////////////////////////////////////////////////////////

//output correction

return m\_NetOutput = MinMax.reverseMinMax(OutputLayer.m\_output,

MinMax.Outminmaxsettings);

}

//variables

Neuron[] InputLayer = new Neuron[20]; ساخت لایه ورودی

Neuron[] HiddenLayer = new Neuron[5]; ساخت لایه میانی

Neuron OutputLayer = new Neuron(); ساخت لایه خروجی

MAPMinMax MinMax=new MAPMinMax();

public double m\_NetOutput;

//net weights and biases

double[,] IW; متغیری که ماتریس وزن‌های لایه ورودی را نگهداری میکند

double[,] HW;متغیری که ماتریس وزن‌های لایه میانی را نگهداری میکند

double[] OW; متغیری که ماتریس وزن‌های لایه خروجی را نگهداری میکند

double[] IB; متغیری که ماتریس بایاس نرون‌های لایه ورودی را نگهداری میکند

double[] HB; متغیری که ماتریس بایاس نرون‌های لایه میانی را نگهداری میکند

double OB; متغیری که بایاس نرون لایه خروجی را نگهداری میکند

};

## 3-6 ساخت پنل ورودی

به وسیله یک PictureBox و رسم کردن عدد با حرکت دادن اشاره‌گر موس بروی آن، پنل ورودی جهت رسم عدد را ساختم.

سایز PictureBox را 90×90 گرفتم چراکه به راحتی آنرا به ابعاد 28×28 که سایز عکس‌های درون کتابخانه MNIST می‌باشد، میتوان تبدیل کرد و علاوه بر آن به اندازه کافی اطلاعات درون آن تصویر باقی می‌ماند تا بتوان آنرا به شبکه عصبی داد. همانطور که در شرح کتابخانه MNIST آمده است همه عکس‌ها وسط‌چین شده‌اند، پس الگوریتمی‌برای تغییر دادن سایز تصویر ورودی و همچنین وسط‌ چین کردن آن لازم است.

## 3-7 الگوریتم تغییر سایز تصویر و وسط چین کردن آن

از آنجایی که تمامی‌محتویات تصویر با رنگ مشکی و زمینه تصویر با رنگ سفید است و تصویر باید وسط چین شود، فقط لازم بود که عدد درون تصویر از ابعاد درون کادر 90×90 به تصویری به ابعاد 28×28 تبدیل شود؛ بنابراین به کمک تابع GetImageRectAngle مختصات و اندازه عدد را پیدا کرده و عدد را از تصویر جدا کردم و عدد جدا شده را کوچک کردم.

public struct RectAngle{ ساختاری که مختصات و اندازه عدد درون تصویر اصلی را نگهداری میکند

public int x1, x2, y1, y2;

};

### 3-7-1 الگوریتم ریسایز کردن تصویر:

public static class ResampleImage{

public static byte[,] Resample(byte[,] srcImg)

{

byte[,] newData = new byte[28, 28];

RectAngle tmpRect = GetImageRectAngle(srcImg);

int DestSizeW = (tmpRect.x2 - tmpRect.x1) / 3;

int DestSizeH = (tmpRect.y2 - tmpRect.y1) / 3;

int StartPointX = (28 - DestSizeW) / 2;

int StartPointY = (28 - DestSizeH) / 2;

for (int y = StartPointY + 1; y <= StartPointY + DestSizeH; y++)

for (int x = StartPointX - 1; x <= StartPointX + DestSizeW; x++){

int tx, ty;

if (x < 1) x = 1;

tx = x - StartPointX;

if (tx < 0) tx = 0;

ty = y - StartPointY;

if (ty < 0) ty = 0;

newData[(x - 1), (y - 1)] = srcImg[tmpRect.x1+ tx\*3 , tmpRect.y1+ ty\*3];

}

return newData;

}

نکته ای که در مورد الگوریتم ریسایز کردن تصویر وجود دارد اینست که این الگوریتم آرایه ای 90×90 از نوع بایت می‌گیرد، بنابراین لازم است تصویر PictureBoxe ابتدا به این نوع آرایه تبدیل شود سپس به این الگوریتم داده شود.

### 3-7-2 الگوریتم پیدا کردن مختصات و ابعاد تصویر:

public static RectAngle GetImageRectAngle(byte[,] image){

RectAngle tmpRect = new RectAngle();

GETX1:

for (int x = 0; x < 90; x++)

for (int y = 0; y < 90; y++){

if (image[x, y] == 1){

tmpRect.x1 = x;

goto GETX2;

}

}

GETX2:

for (int x = 90 - 1; x > -1; x--)

for (int y = 0; y < 90; y++){

if (image[x, y] == 1){

tmpRect.x2 = x;

goto GETY1;

}

}

GETY1:

for (int y = 0; y < 90; y++)

for (int x = 0; x < 90; x++){

if (image[x, y] == 1){

tmpRect.y1 = y;

goto GETY2;

}

}

GETY2:

for (int y = 90 - 1; y > -1; y--)

for (int x = 0; x < 90; x++)

if (image[x, y] == 1){

tmpRect.y2 = y;

goto END;

}

END:

return tmpRect;

}

در مورد الگوریتم بالا X1، X2، Y1، Y2 در عکس زیر مشخص شده اند:



## 3-8 تبدیل عکس 28×28 به ماتریس 784×1

در فصل دوم اشاره شد که لازم است ماتریس PCT رانگهداری کنیم که بُعد ماتریس تصویر ورودی را کاهش دهیم. از آنجایی که خروجی الگوریتم ریسایز کردن تصویر ماتریسی با ابعاد 28×28 است و ورودی موردنیاز برای کاهش بُعد تصویر باید ماتریسی در ابعاد 784×1 باشد، لازم است که ماتریس را به اصطلاح reshape کنیم. که الگوریتم زیر این کار را انجام میدهد.

public static byte[] Reshape(byte[,] img)

{

byte[] image = new byte[784];

int counter = 0;

for (int y = 0; y < 28; y++)

for (int x = 0; x < 28; x++)

image[counter++] = img[x, y];

return image;

}

## 3-9 ضرب ماتریس PCT در تصویر

متغیر PCT حاوی مقادیر ماتریس PCT است که در فصل دوم بدست آوردم.

double[,] PCT;

public double[] Mux(double[] img){

double[] Y = new double[18];

for (int i = 0; i < 18; i++)

Y[i] = 0;

for (int j = 0; j < 18; j++)

for (int i = 0; i < 784; i++)

Y[j] += PCT[j, i] \* img[i];

return Y;

}

## 3-10 تست شکبه عصبی

با دستورات زیر شبکه عصبی را تست می‌کنیم. روش کار به ترتیب به این گونه است که تصویر PictureBox ورودی را گرفته تبدیل به ماتریسی به همان ابعاد از نوع بایت کرده، سپس آنرا ریسایز میکنیم. تصویر ریسایز شده را reshape کرده، آنرا در ماتریس PCT ضرب میکنیم و تصویر کاهش بُعد داده شده را به شبکه عصبی میدهیم.

byte[,] \_data = new byte[90, 90];

Bitmap srcImg = new Bitmap(InputPictureBox.Image);

MUXtoPCA mtp = new MUXtoPCA();

Net net = new Net();

for (int x = 0; x < 90; x++)تبدیل عکس ورودی به آرایه ای از بایت

for (int y = 0; y < 90; y++){

if (srcImg.GetPixel(x, y).R == Color.White.R)

\_data[x, y] = 0;

else

\_data[x, y] = 1;

}

byte[,] res = ResampleImage.Resample(\_data); ریسایز کردن تصویر

//reshape 28\*28 array into 1\*784 array

Byte[] reshaped = ResampleImage.Reshape(res);

//convert reshaped array from bytes into double

double[] reshapeddb = new double[784];

for (int i = 0; i < 784; i++)

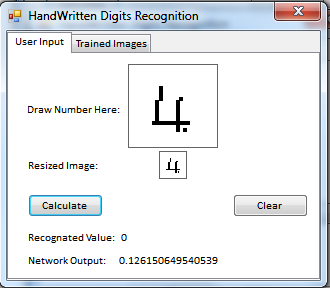
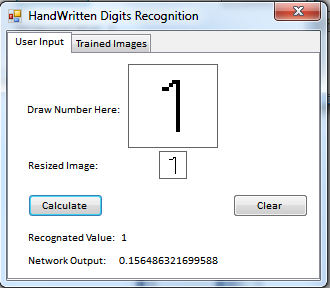
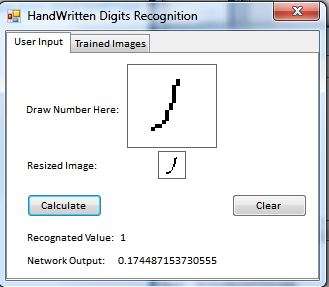
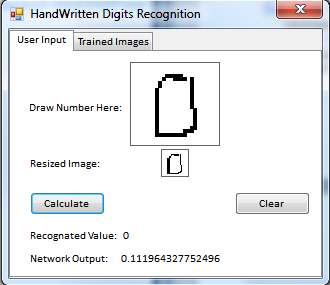
reshapeddb[i] = (double)reshaped[i];

double[] re = mtp.Mux(reshapeddb); PCT ضرب تصویر در ماتریس

net.Calculate(re); اعمال تصویر کاهش بُعد داده شده به شبکه عصبی

lbl\_NetOutput.Text = net.m\_NetOutput.ToString();

## 3-11 نمونه‌هایی از تست عملکرد پروژه و خروجی‌های آن



**منابع:**

1- پروژه آقای "مایک اونیل" برگرفته از سایت Codeproject

<http://www.codeproject.com/Articles/16650/Neural-Network-for-Recognition-of-Handwritten-Digi>

2- کتاب الکترونیکی "شبکه‌های عصبی و کاربرد آنها در الکترونیک" برگرفته از سایت Prozhe.com

3- جزوه شبکه عصبی – دانشکده برق دانشگاه خواجه نصیر

4- کتاب Principal Component Analysis نوشته I.T.Jolliffe

5- سایت NeuralNetworksAndDeepLearning.com

6- راهنمای نرم افزار متلب

**پایان**

با تشکر از اساتید عزیزم سرکار خانم نوشین مهدوی وجناب آقای مهرداد حیدرزاده، همچنین دوست عزیزم آقای رضا حسن زاده که در اجرای این پروژه با صبر و حوصله فراوان، راهنمایی‌ و کمک‌های بسیاری به من کردند.

احمد افکنده