عنوان : گزارش تمرین شبکه عصبی

نگارنده : سحر داستانی اوغانی

شماره دانشجویی : ۹۹۱۱۲۱۰۸

دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

دانشکده ریاضی و علوم کامپیوتر

۳

در ابتدا داده‌ها توسط کتابخانه‌ی idx2numpy از فایلی که در تمرین شبکه عشبی قرار داشت، فراخوانی شده است. این داده‌ها ۳ بعدی هستند و هر سطر از داده‌های این مجموعه شامل یک تصویر با اندازه‌ی ۲۸×۲۸ می‌باشد.

مجموعه‌ی train حاوی ۶۰,۰۰۰ ردیف و مجموعه‌ی test شامل ۱۰,۰۰۰ ردیف می‌باشد.

```
1 X_train.shape, Y_train.shape, X_test.shape, Y_test.shape
```

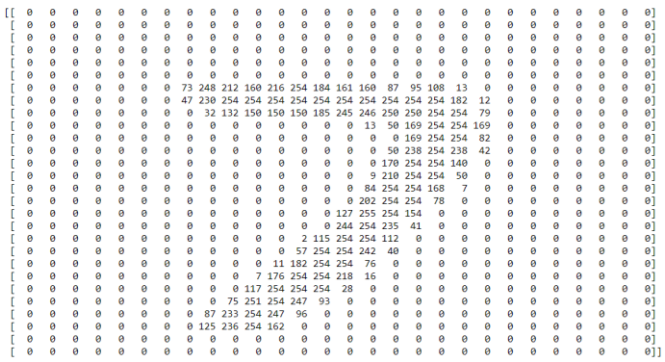
```
((60000, 28, 28), (60000,), (10000, 28, 28), (10000,))
```

طبق گفته‌ی سوال، ۴۰۰۰ داده‌ی اول مجموعه‌ی train را به عنوان مجموعه داده‌ی آموزشی خود در نظر می‌گیریم. من در این مرحله کاری اضافه‌تر از خواسته‌ی سوال انجام دادم و آن این است که باقی مجموعه‌ی train را به مجموعه‌ی validation اختصاص دادم.

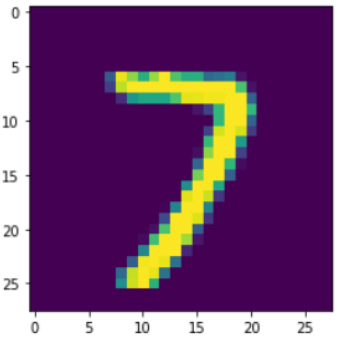
```
1 X_train.shape, Y_train.shape, X_cross.shape, Y_cross.shape
```

```
((4000, 28, 28), (4000,), (56000, 28, 28), (56000,))
```

این دیتاست شامل نمونه ارقام دست‌نویس می‌باشد، بنابراین هر نمونه‌ی آن یک رقم از ۰ تا ۹ است. برای مثال نمونه‌ی ۲۶۷۳-ام



نمایش‌دهنده‌ی عدد



۷ است.

برای این‌که یک مدل دسته بندی کننده ارائه دهیم، نیاز داریم داده‌ها را به ۲ بعد تبدیل کنیم. برای این کار از Flattens استفاده میکنیم. این کار داده‌ها را از ۳ بعد به ۲ بعد تبدیل می‌کند و درنهایت داده‌ها را به صورت زیر ارائه می‌دهد.

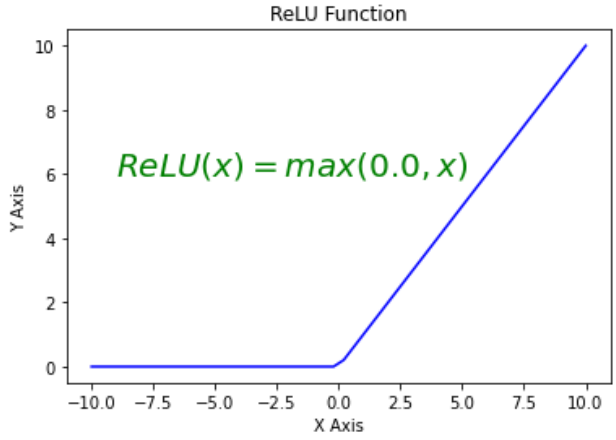
```
1 X_train.shape, Y_train.shape, X_cross.shape, Y_cross.shape, X_test.shape, Y_test.shape
```

```
((4000, 784), (4000, 10), (56000, 784), (56000, 10), (10000, 784), (10000, 10))
```

برای ارائه‌ی یک مدل دسته‌بندی کننده می‌توان از دو طریق اقدام کرد:

- یا توابعی برای انجام این کار به صورت دستی تعریف کرد.
- یا از ماژول MLPClassifier در Sklearn.neural_network استفاده کرد.

برای فیت کردن مدل در این قسمت، از MLPClassifier استفاده کرده‌ام که تابع مورد استفاده‌ی آن به صورت پیش فرض، تابع RELU می‌باشد.



برخی از ویژگی‌های مهم مدل فیت شده بر روی داده‌ها را در زیر مشاهده می‌کنید:

- تعداد لایه‌های پنهان = ۵۰
- تعداد تکرار الگوریتم = ۱۵۰
- تابع مورد استفاده = relu (که به صورت پیش‌فرض بر روی MLPClassifier قرار دارد)

در مرحله‌ی بعد مدل بر روی داده‌های train فیت شده است و با وجود تعریف ۱۵۰ تکرار، الگوریتم پس از ۱۱۲ تکرار متوقف شده است. دلیل آن این است که الگوریتم پس از ۱۱۲ تکرار به این نتیجه رسیده است که مقدار loss کمتر از ۰,۰۰۴۳۵۳۰۸ نمی‌شود و تکرار بیش از ۱۱۲ بار بی فایده است.

```
Iteration 112, loss = 0.00435308
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
```

```
1 model.score(X_train, Y_train)
```

```
1.0
```

```
1 model.score(X_test, Y_test)
```

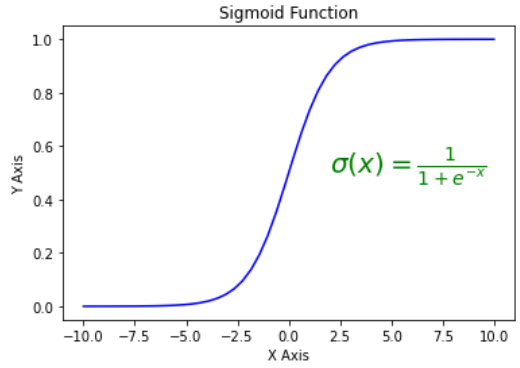
```
0.8717
```

دقت این مدل بر روی داده‌های train ۱۰۰٪، بر روی داده‌های validation، ۹۸,۲۲٪ و بر روی داده‌های test، ۹۸,۳۱٪ است.

```
Training accuracy = 100.00%
Validation accuracy = 98.22%
Test accuracy = 98.31%
```

بخش امتیازی

بررسی تاثیر انواع تابع فعالیت:



- ✓ تابع سیگموید باینری

تابع سیگموید به شرح روبرو است:

در این مرحله به الگوریتم اجازه دادیم تا ۳۰۰ تکرار جلو برود.

در نهایت الگوریتم بیش از ۲۸۴ تکرار جلو نمی‌رود زیرا که الگوریتم

پس از ۲۸۴ تکرار به این نتیجه رسیده است که مقدار loss کمتر از

۰,۰۱۵۵۰۴۰۶ نمی‌شود و تکرار بیش از این مقدار بار بی فایده است.

```
Iteration 284, loss = 0.01550406
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
```

```
1  model_sigmoid.score(X_train, Y_train)
```

```
0.99975
```

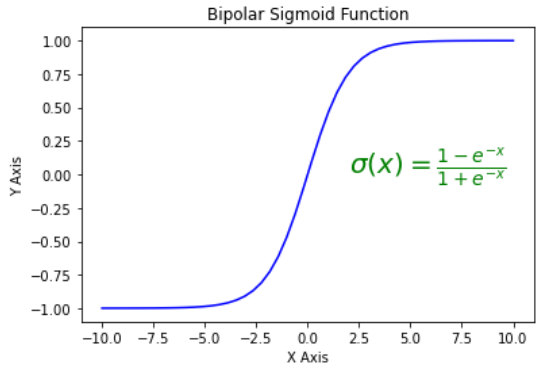
```
1  model_sigmoid.score(X_test, Y_test)
```

```
0.8833
```

دقت این مدل بر روی داده‌های train ۱۰۰٪، بر روی داده‌های validation، ۹۸,۳۶٪ و بر روی داده‌های test، ۹۸,۴۲٪ است

```
Training accuracy = 100.00%
Validation accuracy = 98.36%
Test accuracy = 98.42%
```

- ✓ تابع سیگموید دو قطبی



تابع سیگموید دو قطبی به شرح زیر است:

در این مرحله هنگام اجرای مدل ساخته شده به صورت دستی ، مقادیر argmax و max مدل به ازای مقادیر Y_test به شرح زیر می‌شود:

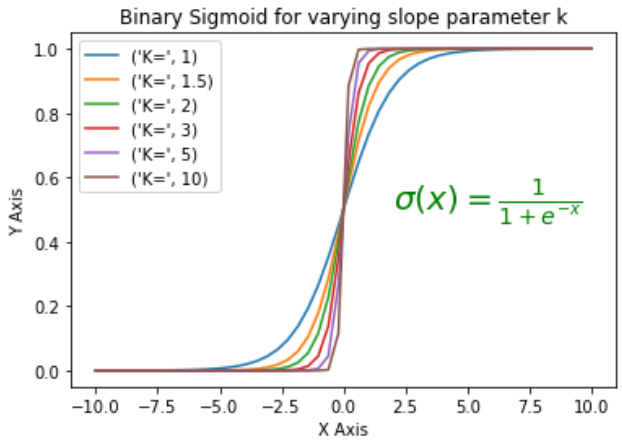
```
7 7 0.9831757848446804
2 2 0.7476519310143985
1 1 0.9513742591061877
0 0 0.9002464340336842
4 4 0.857313941308341
1 1 0.9684011972853604
4 4 0.7373282041072573
9 9 0.9491419835710652
5 6 0.32165232649445274
9 9 0.9457832669832678
0 0 0.8779424481888193
6 6 0.13472856375921438
9 9 0.9451987206006893
0 0 0.9210188640719933
1 1 0.9572121410765855
5 5 0.48390984706269846
9 9 0.8225191301439326
7 7 0.9858012142863222
3 6 0.0646256965215064
4 4 0.9314792576084738
```

و دقت الگوریتم برای مجموعه‌های train, validation, test نیز به شرح زیر می‌گردد:

```
Training accuracy = 89.85%
Validation accuracy = 86.83%
Test accuracy = 87.80%
```

✓ تابع سیگموییید با شیب دلخواه

ظاهر این تابع به ازای k های مختلف مانند روبرو است:



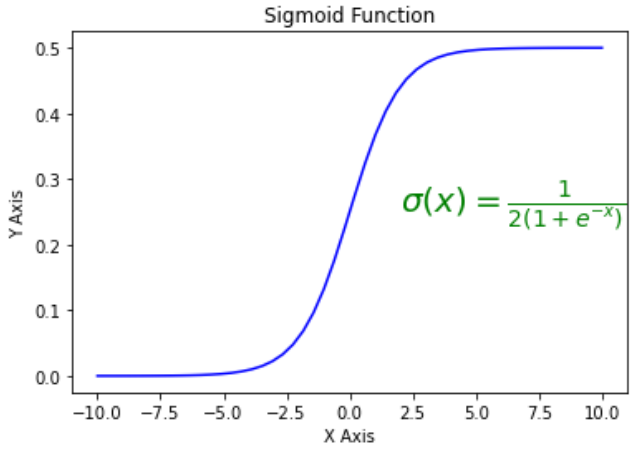
دقت الگوریتم به ازای kهای مختلف در جدول زیر شرح داده شده است.

	Training accuracy = 89.88% Validation accuracy = 86.69% Test accuracy = 87.86%	K = 1
	Training accuracy = 90.48% Validation accuracy = 87.09% Test accuracy = 87.76%	K = 5
	Training accuracy = 91.42% Validation accuracy = 87.36% Test accuracy = 88.33%	K = 10

Training accuracy = 89.22% Validation accuracy = 86.03% Test accuracy = 86.68%	K = 20
Training accuracy = 83.58% Validation accuracy = 80.03% Test accuracy = 80.72%	K = 30
Training accuracy = 85.52% Validation accuracy = 82.48% Test accuracy = 83.04%	K = 40
Training accuracy = 82.55% Validation accuracy = 78.27% Test accuracy = 78.49%	K = 50

همانطور که مشاهده می‌کنید، مقادیر دقت الگوریتم بر روی مجموعه داده‌های train,validation,test، زمانی که k از ۱ به سمت ۱۰ می‌رود، افزایش میابد ولی در مقادیر ۲۰ تا ۵۰، کاهش میابد.

✓ تابع سیگموییید باینری در بازه‌ی دلخواه



ظاهر تابع به شکل مقابل است. می‌توان بازه‌های مختلف را با ضرب، تقسیمریال جمع و تفریق یک عدد با فرمول بدست آورد.

دقت آن در نهایت نیز به شرح زیر است:

Training accuracy = 29.07%
Validation accuracy = 27.93%
Test accuracy = 28.35%

بررسی ویژگی‌هایی اضافه بر تمرین تعریف شده

یک مدل Sequential برای یک دسته ساده از لایه ها مناسب است که در آن هر لایه دقیقاً یک سنسور ورودی و یک سنسور خروجی دارد. برای داده‌های تعریف شده در بالا، این مدل تعریف شده و لایه‌های پنهان اول و دوم و لایه‌ی آخر برای آن درنظر گرفته شده است سپس با استفاده از دستور model.compile وزن‌های شبکه داده شده و در صورت نیاز به update، recompile می‌شود.

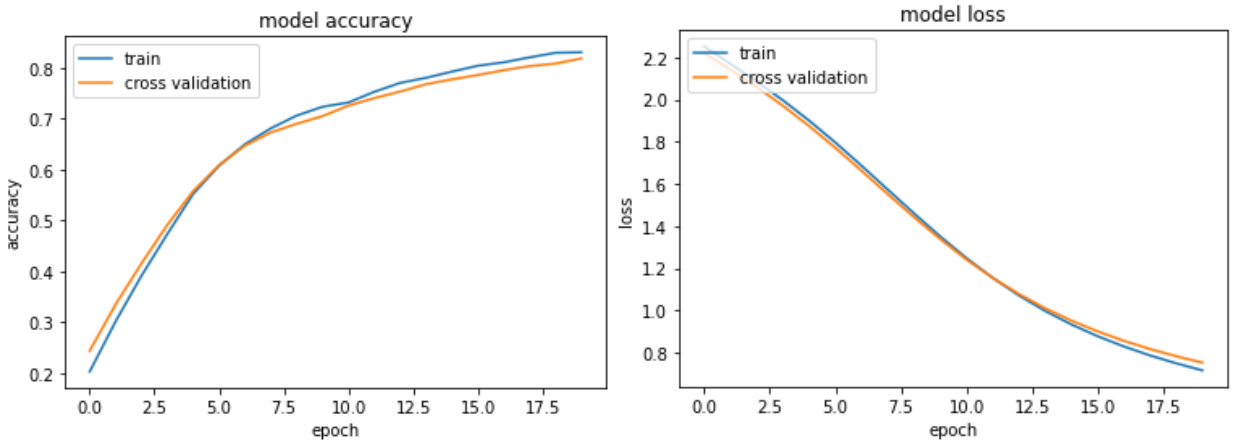
خلاصه‌ای از عمکرد مدل را در روبرو می‌توانید مشاهده کنید:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Hidden-1 (Dense)             (None, 100)               78500
_________________________________________________________________
Hidden-2 (Dense)             (None, 100)               10100
_________________________________________________________________
OutputLayer (Dense)          (None, 10)                1010
=================================================================
Total params: 89,610
Trainable params: 89,610
Non-trainable params: 0
```

در مرحله‌ی بعد، تمام داده‌ها را در history لیست می‌کنیم.

```
Epoch 1/20
20/20 [==============================] - 2s 87ms/step - loss: 2.2676 - acc: 0.1863 - val_loss: 2.2191 - val_acc: 0.2435
Epoch 2/20
20/20 [==============================] - 0s 24ms/step - loss: 2.1872 - acc: 0.2763 - val_loss: 2.1407 - val_acc: 0.3347
Epoch 3/20
20/20 [==============================] - 1s 27ms/step - loss: 2.1032 - acc: 0.3712 - val_loss: 2.0586 - val_acc: 0.4154
Epoch 4/20
20/20 [==============================] - 0s 25ms/step - loss: 2.0147 - acc: 0.4544 - val_loss: 1.9698 - val_acc: 0.4913
Epoch 5/20
20/20 [==============================] - 1s 28ms/step - loss: 1.9278 - acc: 0.5274 - val_loss: 1.8738 - val_acc: 0.5577
Epoch 6/20
20/20 [==============================] - 0s 24ms/step - loss: 1.8194 - acc: 0.5949 - val_loss: 1.7704 - val_acc: 0.6088
Epoch 7/20
20/20 [==============================] - 0s 23ms/step - loss: 1.7190 - acc: 0.6351 - val_loss: 1.6618 - val_acc: 0.6473
Epoch 8/20
20/20 [==============================] - 0s 24ms/step - loss: 1.6075 - acc: 0.6659 - val_loss: 1.5513 - val_acc: 0.6732
Epoch 9/20
20/20 [==============================] - 0s 23ms/step - loss: 1.5004 - acc: 0.6935 - val_loss: 1.4425 - val_acc: 0.6903
Epoch 10/20
20/20 [==============================] - 0s 24ms/step - loss: 1.3943 - acc: 0.7131 - val_loss: 1.3388 - val_acc: 0.7054
Epoch 11/20
20/20 [==============================] - 0s 24ms/step - loss: 1.2825 - acc: 0.7227 - val_loss: 1.2419 - val_acc: 0.7260
Epoch 12/20
20/20 [==============================] - 0s 23ms/step - loss: 1.1699 - acc: 0.7486 - val_loss: 1.1548 - val_acc: 0.7409
Epoch 13/20
20/20 [==============================] - 0s 24ms/step - loss: 1.0896 - acc: 0.7682 - val_loss: 1.0785 - val_acc: 0.7537
Epoch 14/20
20/20 [==============================] - 0s 24ms/step - loss: 1.0106 - acc: 0.7865 - val_loss: 1.0106 - val_acc: 0.7682
Epoch 15/20
20/20 [==============================] - 0s 24ms/step - loss: 0.9468 - acc: 0.7940 - val_loss: 0.9524 - val_acc: 0.7778
Epoch 16/20
20/20 [==============================] - 0s 25ms/step - loss: 0.8834 - acc: 0.8087 - val_loss: 0.9017 - val_acc: 0.7862
Epoch 17/20
20/20 [==============================] - 0s 24ms/step - loss: 0.8261 - acc: 0.8156 - val_loss: 0.8568 - val_acc: 0.7958
Epoch 18/20
20/20 [==============================] - 0s 24ms/step - loss: 0.7986 - acc: 0.8209 - val_loss: 0.8179 - val_acc: 0.8038
Epoch 19/20
20/20 [==============================] - 0s 25ms/step - loss: 0.7516 - acc: 0.8280 - val_loss: 0.7836 - val_acc: 0.8087
Epoch 20/20
20/20 [==============================] - 1s 28ms/step - loss: 0.7165 - acc: 0.8332 - val_loss: 0.7534 - val_acc: 0.8189
```

برای بررسی لیست داده‌ها، ۲۰ تکرار در نظر گرفتیم. در هر تکرار زمان اجرا، میزان تابع loss و دقت الگوریتم برای داده‌های train, validation محاسبه می‌شود. با افزایش تکرار بررسی لیست داده‌ها در history، دقت نیز بالا می‌رود و از طرفی میزان تابع loss کاهش می‌یابد. نمودار این رفتار در زیر نمایش داده شده است.



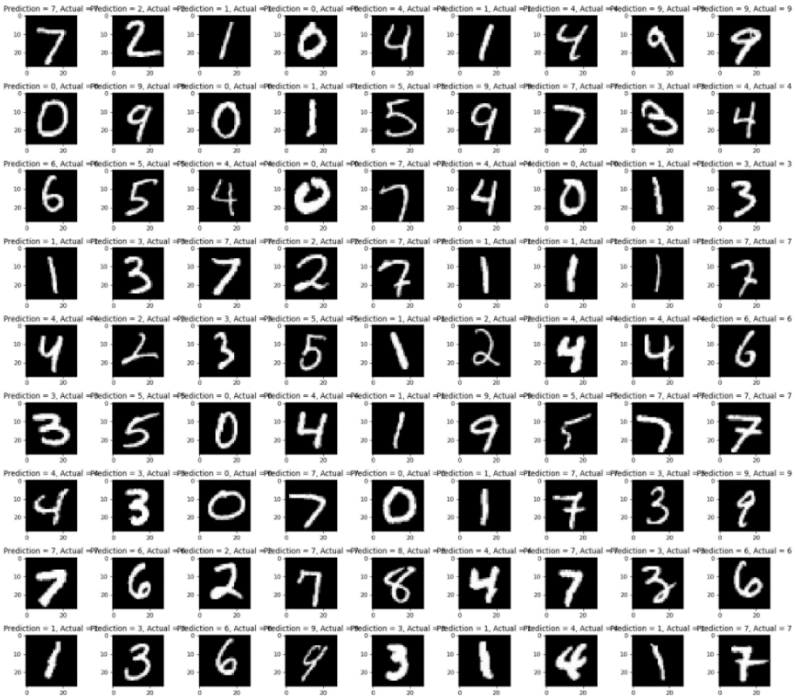
دقت در این مدل به شرح روبرو است:

```
Test accuracy: 0.823199987411499
```

در نهایت نیز نمونه‌ای از داده‌ها را می‌بینید که یک دسته به درستی و دسته‌ای دیگر به نادرستی دسته‌بندی شدند.

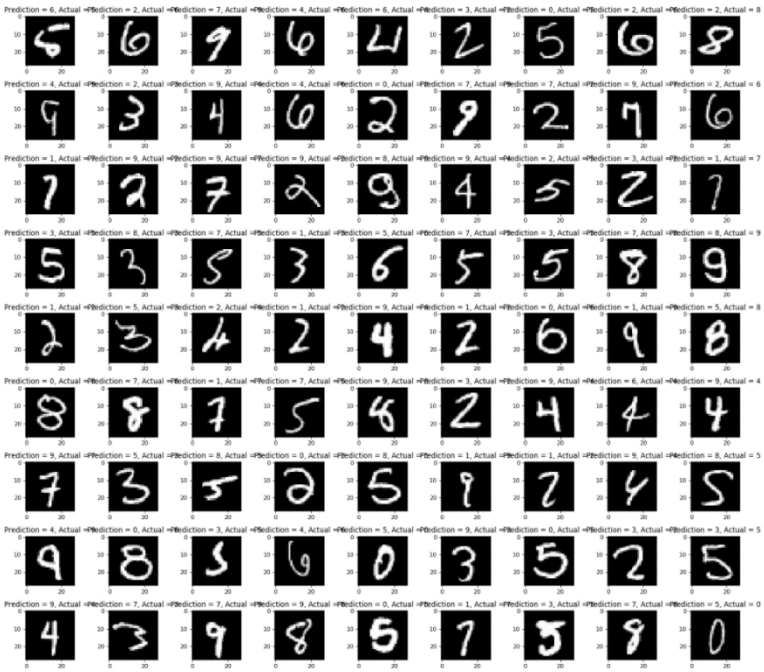
درست:

برای مثال داده‌ای را در نظر بگیرید که در بالاترین و چپ‌ترین قسمت تصویر قرار دارد، این داده نمایش‌دهنده‌ی عدد ۷ است و در prediction نیز label ۷ به آن نسبت داده شده است



نادرست:

برای مثال داده‌ای را در نظر بگیرید که در بالاترین و راست‌ترین قسمت تصویر قرار دارد، این داده نمایش‌دهنده‌ی عدد ۸ است ولی در prediction ، label ۲ به آن نسبت داده شده است که غلط است.



منابع

https://colab.research.google.com/github/CC-MNNIT/2018-19-Classes/blob/master/MachineLearning/2018_08_27_Logical-Rhythm-3/MNIST.ipynb#scrollTo=xXTcooRdQ1bA

https://www.python-course.eu/neural_network_mnist.php

https://machinelearningmastery.com/display-deep-learning-model-training-history-in-keras/

https://www.tensorflow.org/guide/keras/train_and_evaluate

https://stackoverflow.com/questions/36952763/how-to-return-history-of-validation-loss-in-keras

https://keras.io/guides/sequential_model/

https://towardsdatascience.com/fixing-the-keyerror-acc-and-keyerror-val-acc-errors-in-keras-2-3-x-or-newer-b29b52609af9

https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

https://www.tensorflow.org/install/#anaconda_installation

https://www.youtube.com/watch?v=w09WJieJ2Ao

https://github.com/tensorflow/tensorflow/issues/32147

https://data-flair.training/blogs/install-tensorflow/

https://faradars.org/courses/fvdm9406-machine-learning-in-python-first-part

پاسخ تمارین تشریحی

۱) توابع بولی $\{T,F\}^d \to \{T,F\}$ را در نظر بگیرید. اگر از صفر برای نمایش F و از یک برای نمایش T استفاده کنیم: الف) آیا هر عبارت منطقی (متشکل از متغیرها و عملگرهای منطقی) را می‌توان توسط یک شبکه عصبی با یک لایه مخفی که تابع آستانه نورون های لایه ی مخفی و خروجی آن پله‌ای باشد، مدل کرد؟ توضیح دهید.

گیت های منطقی شامل موارد زیر هستند:

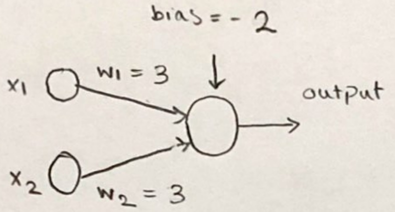
OR AND XOR NOT
NOR NAND XNOR

حال می‌خواهیم آن‌ها را با یک شبکه عصبی که دارای تابع پله‌ای است مدل کنیم:

OR: جدول x_1 or x_2 به صورت مقابل است ←

x_1	x_2	OR
0	0	0
0	1	1
1	0	1
1	1	1

شبکه عصبی or به صورت مقابل می‌شود:



وزن ها را به صورت random مقدار دهی می‌کنیم.

حال اگر تابع پله‌ای را به صورت مقابل در نظر بگیریم، شبکه عصبی درست عمل می‌کند

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

زیرا:

$x_1 = 0$, $x_2 = 0$ $\rightarrow -2 + 0 \times 3 + 0 \times 3 = -2 \rightarrow f = 0$ ✓

$x_1 = 0$, $x_2 = 1$ $\rightarrow -2 + 0 \times 3 + 1 \times 3 = 1 \rightarrow f = 1$ ✓

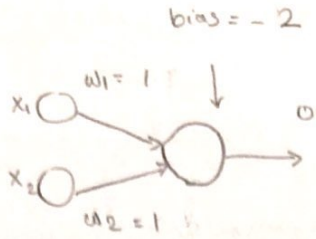
$x_1 = 1$, $x_2 = 0$ $\rightarrow -2 + 1 \times 3 + 0 \times 3 = 1 \rightarrow f = 1$ ✓

$x_1 = 1$, $x_2 = 1$ $\rightarrow -2 + 1 \times 3 + 1 \times 3 = 4 \rightarrow f = 1$ ✓

AND : جدول

x_1	x_2	and
0	0	0
0	1	0
1	0	0
1	1	1

ورودی‌ها x_1 and x_2 مقادیر متغیر است.



شبکه عصبی آن به شرح مقابل است:

وزن وصل را به ورودی‌ها مقداردهی می‌کنیم random

حال باید تابع پله‌ای را به ورودی‌ها متغیر بدهیم، شبکه عصبی درست عمل می‌کند.

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

زیرا:

$x_1 = 0$
$x_2 = 0$ → $-2 + 0\times1 + 0\times1 = -2$ → $f = 0$ ✓

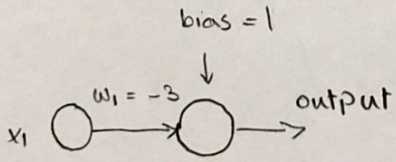
$x_1 = 0$
$x_2 = 1$ → $-2 + 0\times1 + 1\times1 = -1$ → $f = 0$ ✓

$x_1 = 1$
$x_2 = 0$ → $-2 + 1\times1 + 0\times1 = -1$ → $f = 0$ ✓

$x_1 = 1$
$x_2 = 1$ → $-2 + 1\times1 + 1\times1 = 0$ → $f = 1$ ✓

NOT : جدول NOT به صورت مقابل است:

x_1	NOT
0	1
1	0

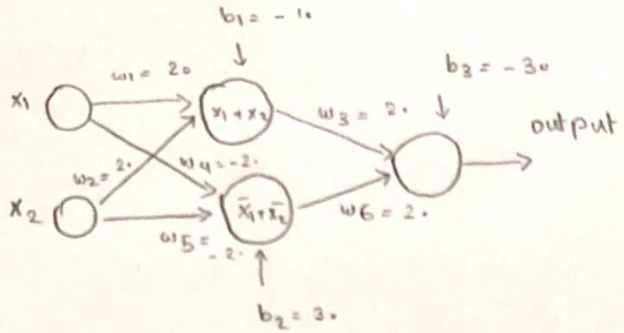


تابع ←
$$f(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases}$$

$x_1 = 0$ → $0\times-3 + 1 = 1$ ✓

$x_1 = 1$ → $1\times-3 + 1 = 0$ ✓

$x_1 \text{ XOR } x_2 = \bar{x}_1 x_2 + x_1 \bar{x}_2 = (x_1 + x_2) \wedge (\bar{x}_1 + \bar{x}_2)$: XOR



تابع : $f(n) = \begin{cases} 1 & n > 0 \\ 0 & n < 0 \end{cases}$

جدول :

x_1	x_2	XOR
0	0	0
0	1	1
1	0	1
1	1	0

$x_1 = 0$, $x_2 = 0$ $\rightarrow f\left[\left[f(0\times 20 + 0\times 20 - 10)\times 20\right] + \left[f(0\times -20 + 0\times -20 + 30)\times 20\right] + -30\right] = 0$ ✓

$x_1 = 0$, $x_2 = 1$ $\rightarrow f\left[\left(f(0\times 20 + 1\times 20 - 10)\times 20\right) + \left(f(0\times -20 + 1\times -20 + 30)\times 20\right) - 30\right] = 1$ ✓

$x_1 = 1$, $x_2 = 0$ $\rightarrow f\left(\left(f(1\times 20 + 0\times 20 - 10)\times 20\right) + \left(f(1\times -20 + 0\times -20 + 30)\times 20\right) - 30\right)$
$= 1$ ✓

$x_1 = 1$, $x_2 = 1$ $\rightarrow f\left(\left(f(1\times 20 + 1\times 20 - 10)\times 20\right) + \left(f(1\times -20 + 1\times -20 + 30)\times 20\right) - 30\right)$
$= 0$ ✓

بنابراین چون گیت XOR نیز توسط یک شبکه عصبی حاوی تابع پله‌ای قابل رسم بود، عامه تشبیات آن با گیت‌های دیگر نیز قابل رسم است.

پس می‌توان هر تابع و عبارت منطقی را با شبکه عصبی با یک لایه‌ی مخفی که خروجی آن تابع پله‌ای است نمایش داد.

ب) برای تابع NAND سه‌تا وزن های تک شبه با دو ورودی وقتی که کارکرد NAND را داشته باشد مناسب.

x_1 و x_2 را قبل گیت NAND می‌گذاریم و ورودی آن‌ها را پیاده می‌کنیم:

$$\neg(x_1 \wedge x_2) = (\neg x_1) \vee (\neg x_2)$$

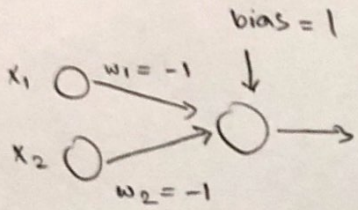
حالت مختلف NAND بصورت متقابل می‌باشد:

x_1	x_2	NOTAND
0	0	1
0	1	1
1	0	1
1	1	0

تابع مورد نظر را بصورت مقابل در نظر می‌گیریم:

$$f(n) = \begin{cases} 1 & n \geq 0 \\ 0 & n < 0 \end{cases}$$

شبکه عصبی بصورت زیر می‌شود:



$x_1 = 0$, $x_2 = 0$ $\rightarrow$ $0 \times -1 + 0 \times -1 + 1 = 1 \rightarrow f = 1$ ✓

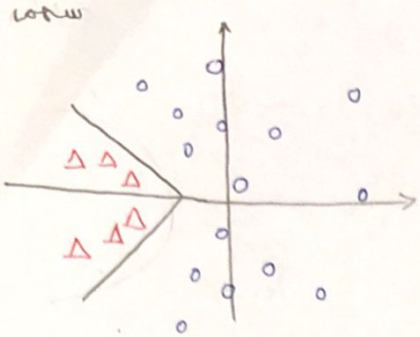
$x_1 = 0$, $x_2 = 1$ $\rightarrow$ $0 \times -1 + 1 \times -1 + 1 = 0 \rightarrow f = 1$ ✓

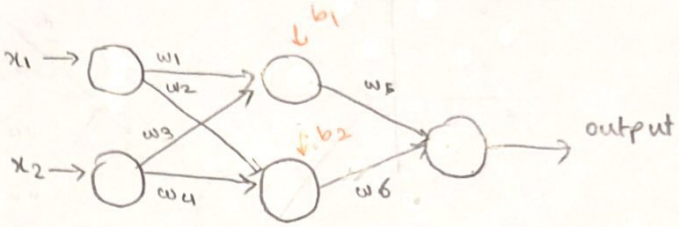
$x_1 = 1$, $x_2 = 0$ $\rightarrow$ $1 \times -1 + 0 \times -1 + 1 = 0 \rightarrow f = 1$ ✓

$x_1 = 1$, $x_2 = 1$ $\rightarrow$ $1 \times -1 + 1 \times -1 + 1 = -1 \rightarrow f = 0$ ✓

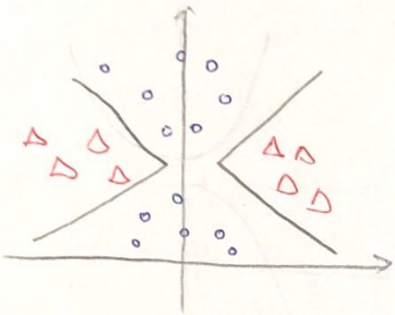
(2) برای دسته بندی مجموعه داده‌های زیر شبکه‌های MLP با حداقل تعداد نورون و لایه‌های ممکن ارائه کنید. (تابع آستانه دلخواه در نظر گرفته شود) که قابلیت جداسازی کامل نمونه‌ها را درست را داشته باشد.

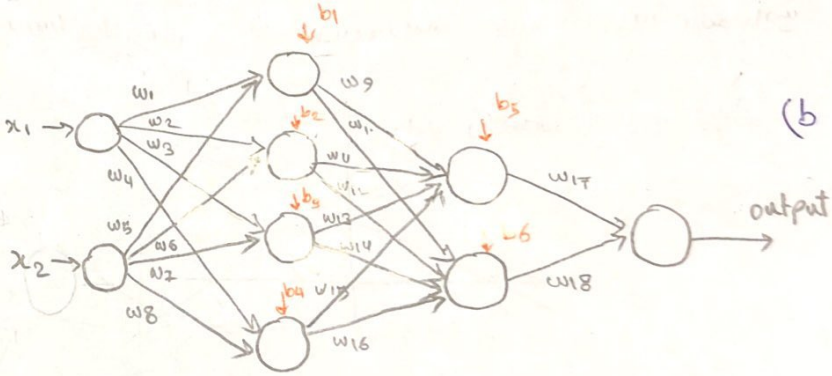
(a



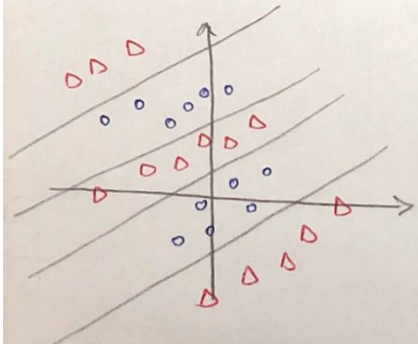


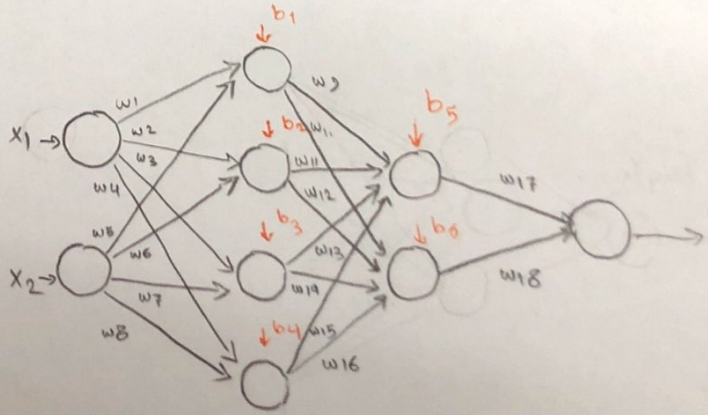
(b



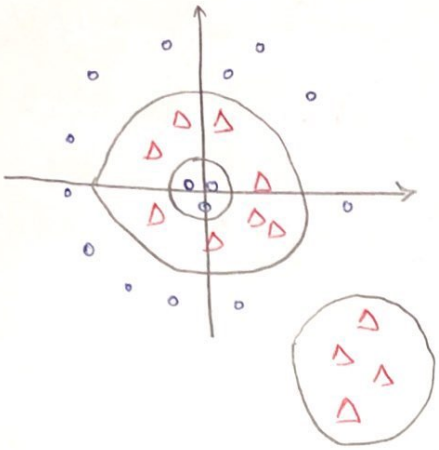


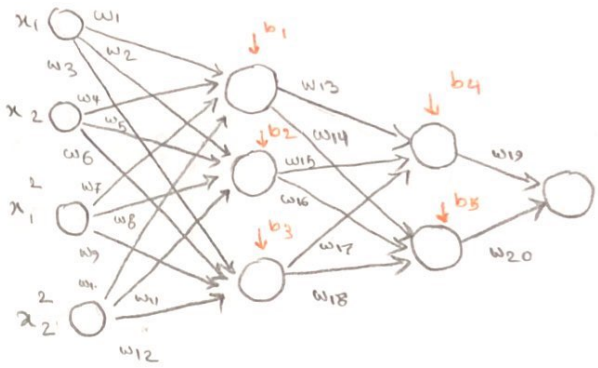
(c





d)





به‌دلیل اینکه شبکه Fully Connected هست ، تمام نورون های لایه اول به تمام نورون های لایه دوم وصل می‌شوند

ولی بعضی از آن‌ها مانند w_7 و w_{10} می‌توانند مقدار صفر بگیرند