

عبدی جمال حواه

۸۱۰۱۰۰۱۱

لرین جمارم

با ذکری مائبن

سوال ① :

الف) هدف تابع فعال ساری در شبکه های MLP ایجاد تابع های غیر خطی است.

ابتدا در نظر بگیرید که شبکه های MLP بک ورودی (عموله بردار) دارند یعنی بک تابع خستند. تمام مملبات های موجود در بین شبکه های

اعماق ضرب و جمع خطی است به هر تابع فعال ساری که تواند غیر خطی باشد و به طور

خطی تابع های شبکه را غیر خطی کند.

اگر در MLP از تابع فعال ساز استفاده ننمایم بلکه شبکه خالی خواهیم داشت این یعنی

یک طبقه بندی یا رله سور خطی خواهیم داشت مرچه هم بر آن نوردن ولا به اضافه ننمایم

همچنان خطی می شاند، زیرا ترکیب خطی بکسری ترک خطی، خطی می باشد. یعنی

کل آن شبکه رایی نهایم یا بک نوردن حاصلین ننمایم و قدرت شبکه بسیار کاملاً

بیش از آن چون نوایع و مرز تغییر های غیر خطی را نمی نواند به غوبی مدل کند.

ب) مسئله اصلی نایع فعال ساز Sigmoid این است که در محدوده کم را بایان

دارد و وقتی از نقطه صفر دوری شویم، را بایان آن سریع به صفر میل می لند و این بد است

برای بامضای می شود را بایان به خوبی به پایین شبله منتقل نشود و دقتی به خوبی

را بایان منتقل نشود، وزن ها کسر آپیت می شوند (تفیر آها در آکیدت کام است) وزنیان

را افزایش می دهد. برای حل مسئله عموماً از تابع ReLU استفاده می کند

زیرا این نایع از سمت راست یعنی از صفر تا مثبت نهایت را بایان دارد و به عویض

را بایان را به لایه های زیرین شبله منتقل می لند اما در مسما منفی ها همچنان مسئله

صفر بودن را بایان را دارد اما در عمل نتیجه خوبی می دهد و از لحاظ از زیست نیز با نورون

های معز سار کار است یعنی اگر وردی از یک آستانه ای رد شود fire می کند.

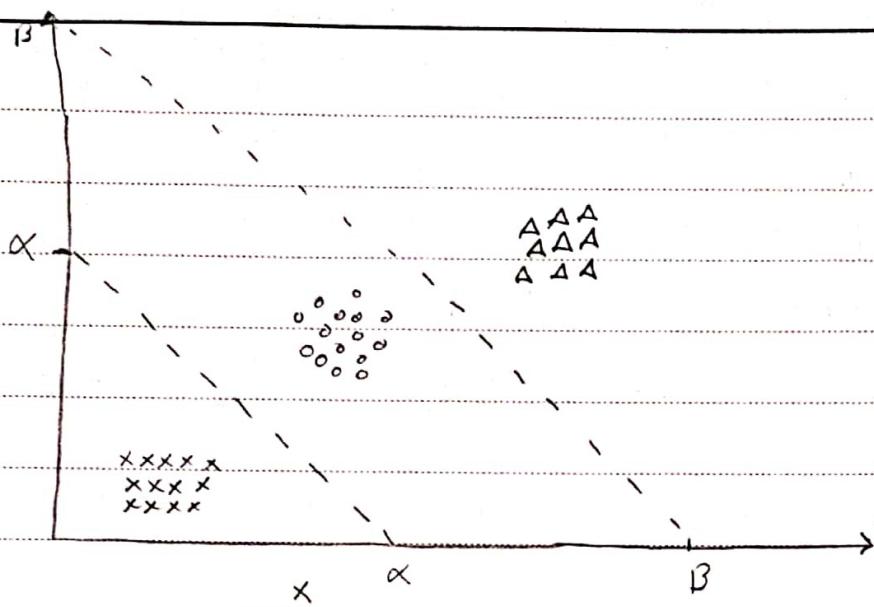
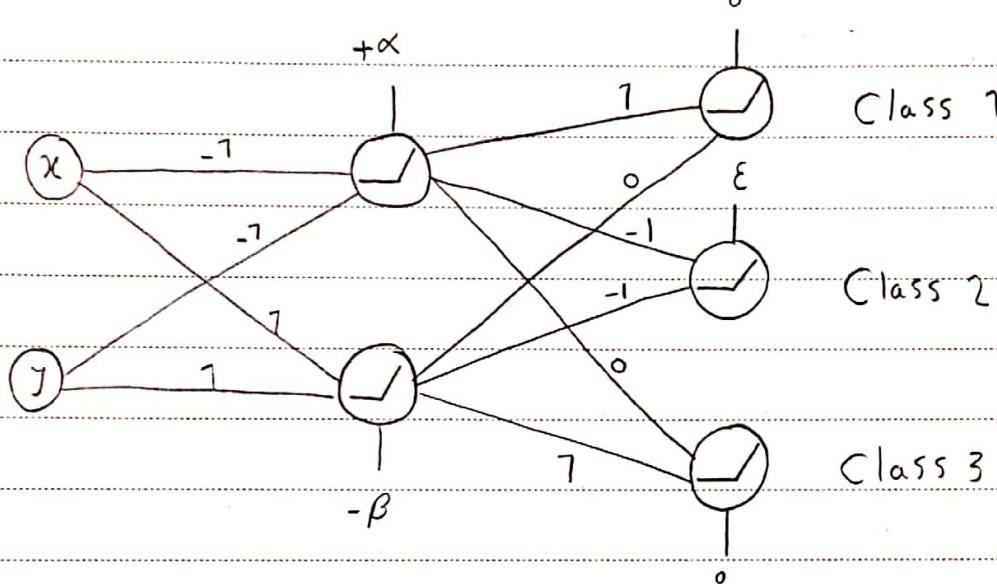
یعنی این inductive bias که سمتا جیباً نایع فعال ساز صفر باشد ناحد خوبی

را واقعیت ساز کار است هر چند نایع های فعال سازی مثل ReLU Gelu مستدل

ReLU

Gelu

را کمی تغییر داده است و می نداند هر چند میل لند

 $x \Rightarrow \text{Class 1}$ $o \Rightarrow \text{Class 2}$ $A \Rightarrow \text{Class 3}$ 

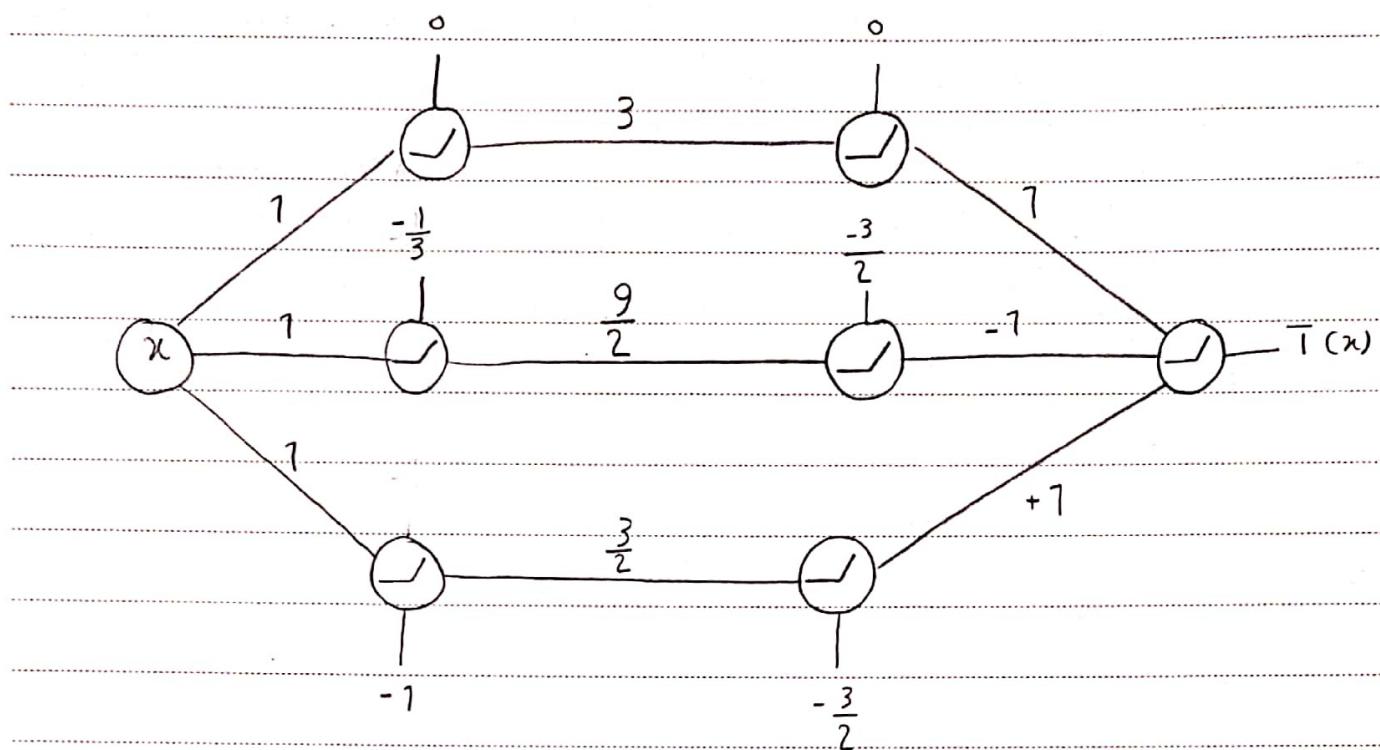
* منقول ر ۴ بک مندار خیلی لوچل است و برای این فوارکته است که فقط آن هر دو دادی

آن صفر بود activate شود.

در نهایت هر نورونی از لایه آخر معال شده بود نمونه متعلق به آن است

(به لونه ای طایی شده که دنبیغاً بکی معال می شود.)

()



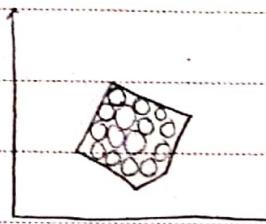
لایه اول مشخص می‌شود که در درجه بازه‌ای است بزرگتر از صفر، بزرگتر از $\frac{1}{3}$ مقدار مطلق.

و بزرگتر از ۱ سپس لایه دوم مقدار ۷ تفاقت خرمب با بازه قبل را تولید می‌کند.

و در نهایت بسته به این که مقدار آن ارزش بازه قبل بیشتر یا کمتر مقدار مطلق

تفاقات به ترتیب صواب اثناه بیان می‌شود.

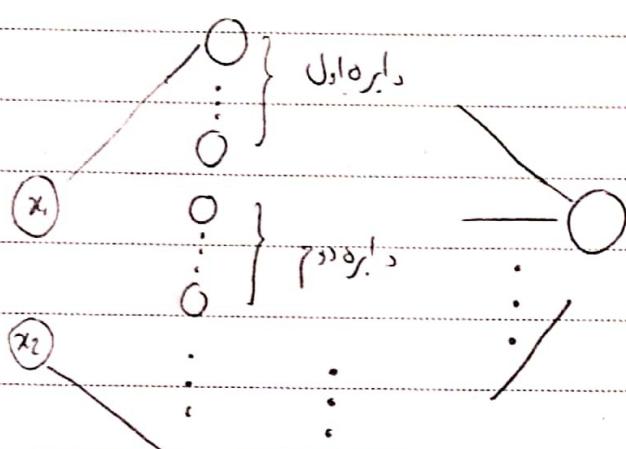
(۵) برای شبکه Classification مثال زیر را در نظر بگیرید.



با بیان این نمونه دیگر لایه هیئت‌نوازی بک داره را

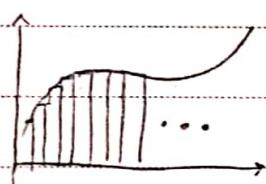
نمی‌خواهیم برای نیم زیرا هم نورون بک خط (صلع) رسمی کند و داره بیان این صلع دارد

سپس شبکه مورد نظر را با داره های توصیفی کنیم یعنی با دو لایه هیئت‌نوازی هر شغل



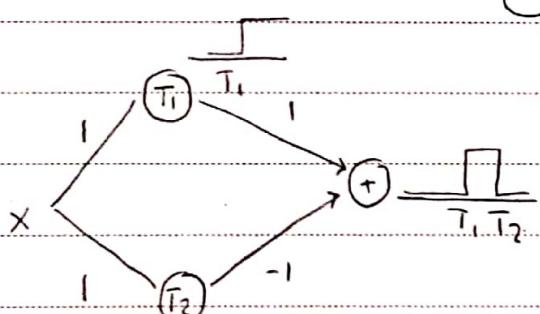
با وجود بادارم:

برای شبکه regression نیز مثال زیر را در نظر بگیرید



با شکله ای به شبکه زیر هیئت‌نوازی بک تابع به شبکه

درست کنیم و می‌توانیم هر تابع را ببینیم



از این تابع تغییل کنیم در نسبت یا جمع دهن

همی‌آنها تابع مورد نظر بدهیم آید



سپس MLP می‌تواند هر تابع را ایجاد کند

سوال ②:

(الف) برای سادگی در نوشت و فهم بسیار تر، به صورت مردمانه می‌بینیم:

خوبی مطلوب که انتظار داریم را با ۰ سیان می‌دهیم:
 $O = \begin{bmatrix} O_1 \\ O_2 \end{bmatrix}$ = خوبی

و بدی ثابع فعال سازی را با Z و خوبی T را با a سیان می‌دهیم.

$$\text{Loss} = - \sum_{i=1}^2 y_i \log O_i$$

که مردد ردار هستند

$$\frac{\partial \text{Loss}}{\partial Y} = \log(O)$$

$$\frac{\partial \text{Loss}}{\partial Z^{(3)}} = \frac{\partial Y}{\partial Z^{(3)}} \frac{\partial \text{Loss}}{\partial Y} = \nabla_{\text{Softmax}}(Z^{(3)}) \log(O)$$

$$\frac{\partial \text{Loss}}{\partial Z^{(2)}} = \frac{\partial a^{(2)}}{\partial Z^{(2)}} \frac{\partial Z^{(3)}}{\partial a^{(2)}} \frac{\partial \text{Loss}}{\partial Z^{(3)}} = \nabla_{\text{Sigmoid}}(Z^{(2)}) W^{(3)} \frac{\partial \text{Loss}}{\partial Z^{(3)}}$$

$$\frac{\partial \text{Loss}}{\partial Z^{(1)}} = \frac{\partial a^{(1)}}{\partial Z^{(1)}} \frac{\partial Z^{(2)}}{\partial a^{(1)}} \frac{\partial \text{Loss}}{\partial Z^{(2)}} = \nabla_{\text{Sigmoid}}(Z^{(1)}) W^{(2)} \frac{\partial \text{Loss}}{\partial Z^{(2)}}$$

$$\frac{\partial \text{Loss}}{\partial w^{(1)}} = \frac{\partial Z^{(1)}}{\partial w^{(1)}} \frac{\partial \text{Loss}}{\partial Z^{(1)}} = \frac{\partial \text{Loss}}{\partial Z^{(1)}} X^T, \quad \frac{\partial \text{Loss}}{\partial b^{(1)}} = \frac{\partial \text{Loss}}{\partial Z^{(1)}}$$

$$\frac{\partial \text{Loss}}{\partial w^{(1)}} = \nabla_{\text{Sigmoid}}(Z^{(1)}) W^{(2)} \nabla_{\text{Sigmoid}}(Z^{(2)}) W^{(3)} \nabla_{\text{Softmax}}(Z^{(3)}) \log(O) X^T$$

$$\frac{\partial \text{Loss}}{\partial b^{(1)}} = \nabla_{\text{Sigmoid}}(Z^{(1)}) W^{(2)} \nabla_{\text{Sigmoid}}(Z^{(2)}) W^{(3)} \nabla_{\text{Softmax}}(Z^{(3)}) \log(O)$$

به طوری که:

$$\nabla_{\text{Softmax}}(z) = \begin{bmatrix} \frac{e^{z_1+z_2}}{(e^{z_1}+e^{z_2})^2} & -\frac{e^{z_1+z_2}}{(e^{z_1}+e^{z_2})^2} \\ -\frac{e^{z_1+z_2}}{(e^{z_1}+e^{z_2})^2} & \frac{e^{z_1+z_2}}{(e^{z_1}+e^{z_2})^2} \end{bmatrix}$$

$$\nabla_{\text{Sigmoid}}(z) = \begin{bmatrix} \sigma(z_1)(1-\sigma(z_1)) & \dots & 0 \\ 0 & \ddots & 0 \\ \vdots & \ddots & \vdots \\ 0 & 0 & \sigma(z_m)(1-\sigma(z_m)) \end{bmatrix}; \quad \sigma(z) = \frac{1}{1+e^{-z}}$$

برای بدست آوردن گرادیان $w^{(1)}, w^{(2)}$ کافیست عنصرهای مرتبه از ماتریس

آنها که توضیح داده شد انتخاب کرد و ارجوهایم آنها را به تنا عساب لتیم

(به کونه ای که سایر عناصر ماتریس را عساب نلیتم) کافیست که فقط سطر اول، اولین ماتریس

و سینه اول آنرا ماتریس درایله لفته شده لحاظ ننم:

$$\frac{\partial \text{Loss}}{\partial w_{11}} = \nabla_{\text{Sigmoid}}^{[1]}(z^{(1)}) w^{(2)} \nabla_{\text{Sigmoid}}(z^{(2)}) w^{(3)} \nabla_{\text{Softmax}}(z^{(3)}) \text{LogO}_x,$$

$$\frac{\partial \text{Loss}}{\partial b_1^{(1)}} = \nabla_{\text{Sigmoid}}^{[1]}(z^{(1)}) w^{(2)} \nabla_{\text{Sigmoid}}(z^{(2)}) w^{(3)} \nabla_{\text{Softmax}}(z^{(3)}) \text{LogO}_x$$

به طوری که ماتریس $\nabla \text{Sigmoid}$ با نویش زیر سان داده می شود:

$$\nabla \text{Sigmoid} = \begin{bmatrix} \vdots & \vdots \\ \nabla \text{Sigmoid} & \vdots \\ \vdots & \vdots \\ \nabla \text{Sigmoid} & \vdots \end{bmatrix}_{m \times m}$$

(ب)

مقدار loss را به ازای یک نمونه برآورد کنید و درجهٔ علص مسئنگ (loss) حركت می کنم ①

و بعد نمونه دوم را وارد می کنم و loss را به ازای آن برآورد کنید و درجهٔ علص مسئنگ (loss)

حركت می کنم و همین طور را نمونه های بعدی تا ایند یک نقطه می‌لارا شویم

متوجه راز حركت کردن این است که مقدار وزن های فعلی را می‌توان در آن

نقطه ضربه در یک learning rate می کنم. ایده حركت درجهٔ علص مسئنگ (loss) از آن با

می‌آید که حین مسئنگ جستار شد تابع در آن نقطه را سیگنال دهد و چون می‌توانم

مقدار loss را مینیمیم کنم بر علص آن حركت می کنم تا مقدار loss لمس شود این یک

الگوریتم حربیانه است یعنی سعی می کنم در هر مرحله هر چند نقصیم را ببریم، loss

را لمس کنم به ایند آنکه در نسبت به global minimum و یا local minimum (الگوریتم) است

PAPCO $w_{t+1} = w_t - \eta \nabla \ell(w_t)$ برسیم.

در این روش نز ازبل نقطه رندم شروع $\text{Newton-Raphson method}$ ②

جی کنم، اما این بار سعی می‌کنیم منحنی در آن نقطه را با بل منحنی درجه 2 نخوبی

برنیم (سط نیلور درجه 2) که بدل سعی اسنا و مسیریم بهستا $\min_{\text{min}} \text{سعی}$

حرکت کنم که باعث می‌شود بینه تر قدم برداریم و سرعت افزایش می‌باید.

$$\omega_{t+1} = \omega_t - \frac{\mathcal{L}(\omega_t)}{\mathcal{L}'(\omega_t)}$$

نمول آن به صورت زیر دارد:

کسیار شبیه نمول روئی نیست (قبل از این)، در واقع یک تابع دیگر به آن این اساله دارد

روئن ما learning rate می‌باشد

$$\frac{1}{\mathcal{L}'(\omega_t)}$$

بینه را اعمال می‌کنیم که برابر با

اما محاسبه $\mathcal{L}'(\omega_t)$ کار ω_t را بزرگ نمودهان می‌زرسیم

آن زیادی می‌خواهد و در بازدید آن را معلوم نمی‌کنیم که هزینه آن چه زیاد است

نهیت خاطر دار عمل معمولاً از این روئی استفاده نمی‌شود.

3 Transitional Invariance in CNN

3.1 A

Invariance means that the model can recognize an object as an object, even when its appearance varies in some way. Transitional invariance refers to the ability of the model to classify the objects regardless of their position within the image. Which means if an object is shifted in the test image the network still can recognize it correctly.

3.2 B

In a Convolutional Neural Network, the convolutional layers use **small filters** to look for specific patterns in the image. These filters have the same set of weights, which means they look for the same pattern everywhere in the image.

This is different from a regular neural network, where the position of the input features is important. In a CNN, the filters can find the same pattern no matter where it is in the image, because of the **shared weights**.

However, this transitional invariance is not as strong in the fully connected layers of the CNN. In the fully connected layers, the position of the features becomes more important again.

3.3 C

Let's compare how the concept of translational invariance in CNNs versus MLPs, using the MNIST dataset. First, we make an MLP architecture with the layer represented in the following figure:

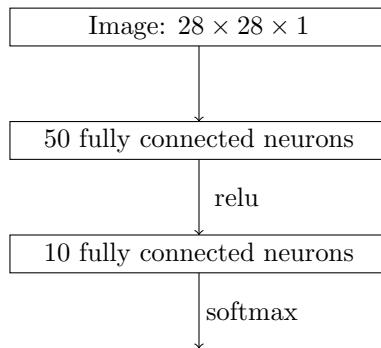


Figure 3.1: MLP architecture used for classifying the MNIST dataset.

The MLP model we previously implemented achieved an accuracy of 92% on the test dataset, which is quite good. Next, we used the LeNet CNN architecture on the same test dataset and achieved an accuracy of 97%.

Both the MLP and CNN models have shown promising results so far. Now, let's see how they perform when we use a shifted image as the input. The chosen shifted image is shown in Figure 3.2.

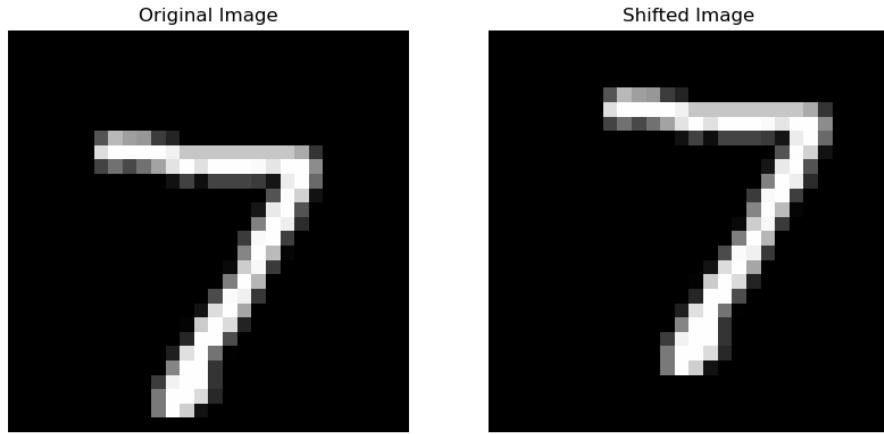


Figure 3.2: Shifted digit

	Original Image	Shifted Image
LeNet	7	7
MLP	7	2

Table 3.1: Predicted label of image in Figure 3.2

As expected, the LeNet CNN model was able to recognize the same object appearing in both the original and shifted images, despite the differences in the actual pixel values. This demonstrates the translational invariance property of CNNs.

In contrast, the MLP model is not translational invariant. This is an important distinguishing feature between the two architectures.

The translational invariance of CNNs allows them to have an improved generalization performance, as the model is better able to make accurate predictions on unseen data.

4 Q4

4.1 Hard SVM

$$x_1 = [1, 4], x_2 = [2, 3], x_3 = [4, 5], x_4 = [5, 6]$$

$$\begin{aligned} 17\lambda_1 + 14\lambda_2 - 24\lambda_3 - 29\lambda_4 &= 1 \\ 14\lambda_1 + 13\lambda_2 - 23\lambda_3 - 28\lambda_4 &= 1 \\ -24\lambda_1 - 23\lambda_2 + 41\lambda_3 + 50\lambda_4 &= 1 \\ -29\lambda_1 - 28\lambda_2 + 50\lambda_3 + 61\lambda_4 &= 1 \end{aligned}$$

$$\lambda_1 = 0, \lambda_2 = 0.25, \lambda_3 = 0.25\lambda_4 = 0$$

$$W = \sum_{i=1}^n \lambda_i x_i = 0.25x_2 + 0.25x_3 = [-0.5, -0.5]$$

$$W^T x_2 + b = 1 \rightarrow b = 3.5$$

Weights	[-0.5, 0.5]
Bias	3.5

Table 4.1: Results of Hard SVM in Python



Figure 4.1: Hard SVM in Python

4.2 Mapping

We define two kinds of maps as follows:

- **Absolute value(abs):** suppose that α is constant

$$|x - \alpha|$$

- **Euclidean distance:** suppose that v_{center} is vector as a reference for calculating the distance

$$\|x - v_{center}\|_2$$

4.2.1 Dataset #1

The data are as follows:

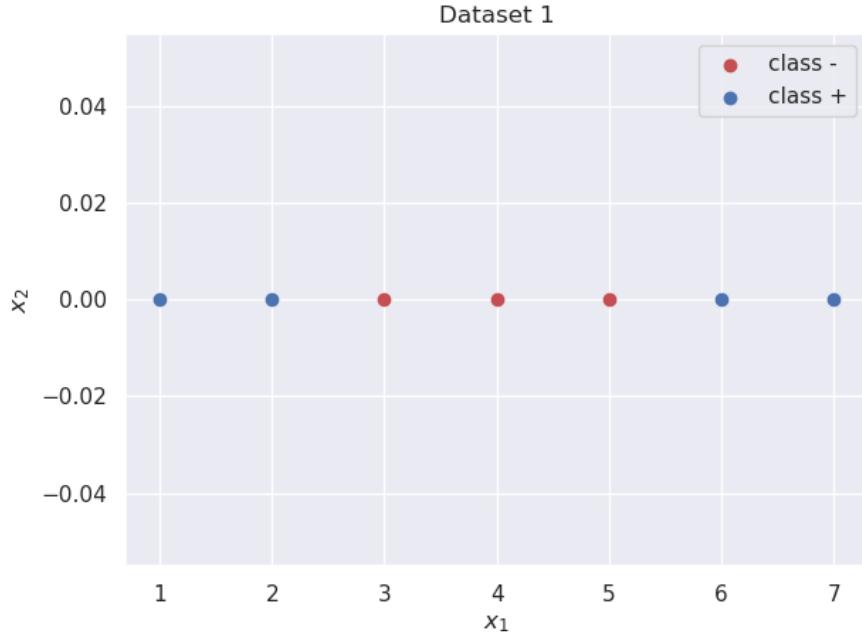


Figure 4.2: Dataset #1

We can use an ABS map with $\alpha = 3.7$ (we won't use $\alpha = 4$ to have better visualization, because if we use that, some points locate at the same position). The result is as follows, , and as you can see now they are linearly separable:

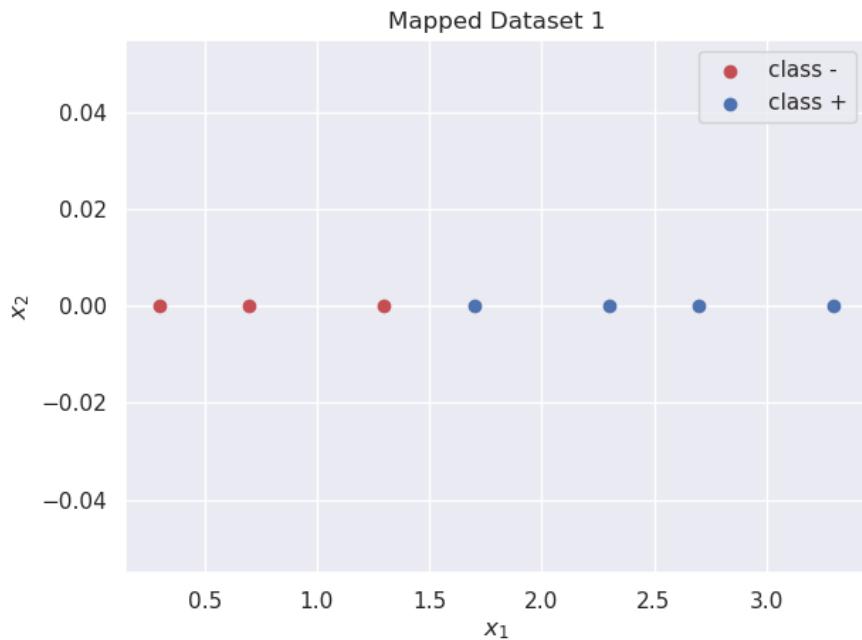


Figure 4.3: Mapped Dataset #1

4.2.2 Dataset #2

The data are as follows:



Figure 4.4: Dataset #2

We can use a Euclidean distance map with $v_{center} = [0.2, 0.2]$ (again, we won't use $v_{center} = [0, 0]$ to have better visualization, because if we use that, some points locate at the same position). The result is as follows, and as you can see now they are linearly separable:

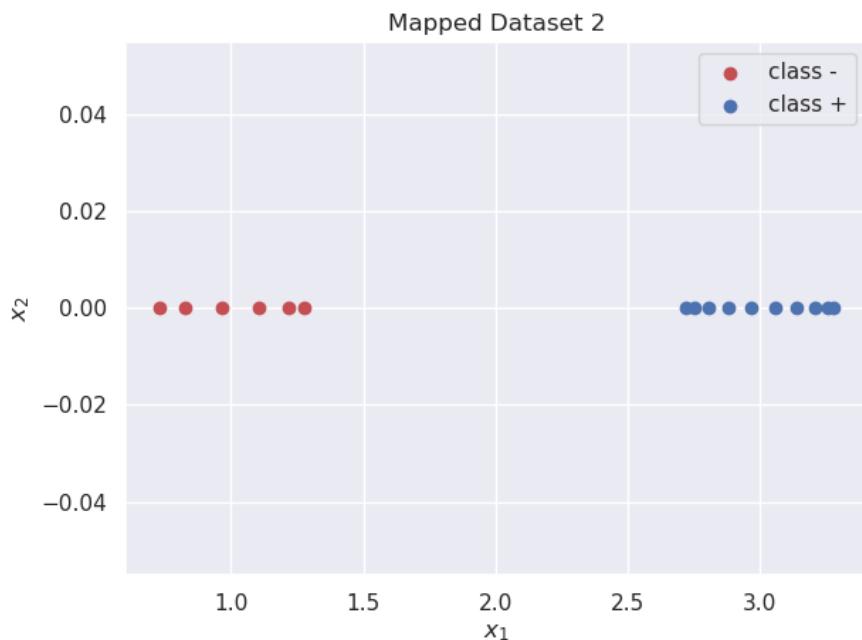


Figure 4.5: Mapped dataset #2

4.2.3 Dataset #3

The data are as follows:

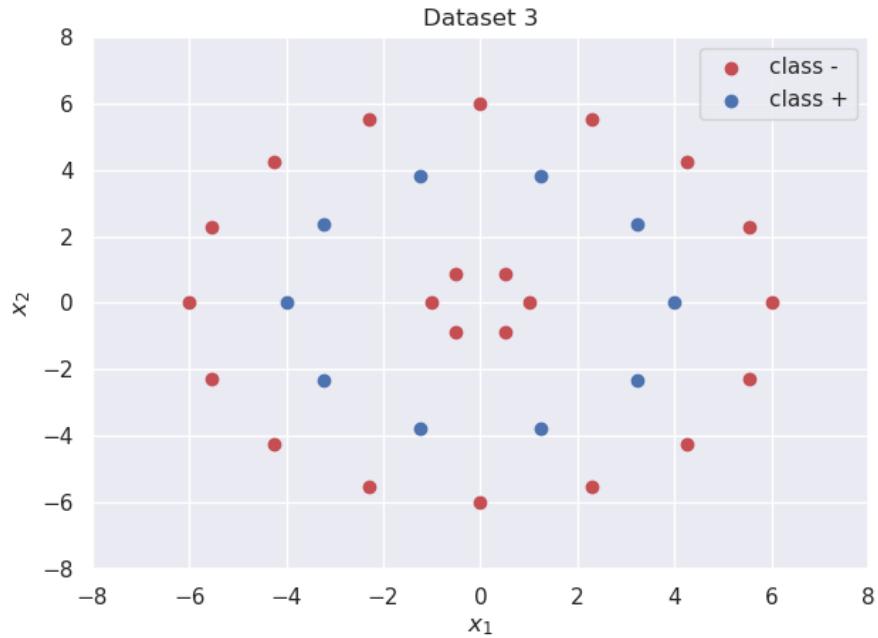


Figure 4.6: Dataset #3

We can use both mentioned maps with each other, which means first we apply an Euclidean distance map with $v_{center} = [0.4, 0.4]$ and then an ABS map with $\alpha = 4$. Actually, by applying the first map, the data will look like dataset 1, and then the ABS map will make it linearly separable, just like dataset 1. The result is as follows:

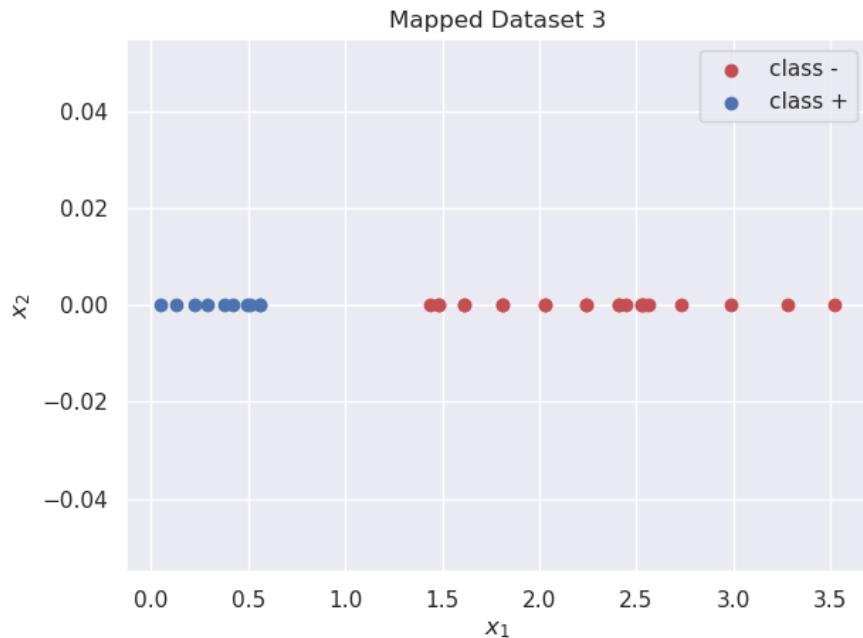


Figure 4.7: Mapped dataset #3

5 Kernel Method

5.1

Most of the classifiers we use are linear, so we tend to map our data to a new space that is linearly separable. However, when we rewrite the equations, we realize that we don't need the actual function of the mapping. Instead, the function of the inner product in the new space is sufficient. The kernel is this inner product function. One of the main benefits of the kernel trick is that it can capture complex and **nonlinear patterns** in the data without explicitly computing the features. It also saves a lot of **computational time and memory**, especially when the number of features in mapped space is very large or infinite. Moreover, it can be used as a form of **feature selection**, when the number of features in mapped space is less than the original space.

Theorem.

$$K(x, y)^2 \leq K(x, x)K(y, y)$$

Proof. suppose

$$\begin{aligned}\Phi(x) &= [a_1, \dots, a_n]^T \\ \Phi(y) &= [b_1, \dots, b_n]^T\end{aligned}$$

Then we can derive

$$\begin{aligned}\sum_{i=1}^n \sum_{j=1}^n (a_i a_j - a_j a_i)^2 &= \sum_{i=1}^n \sum_{j=1}^n (a_i^2 b_j^2 + a_j^2 b_i^2 - 2 a_i b_i a_j b_j) \\ &= \sum_{i=1}^n a_i^2 \sum_{j=1}^n b_j^2 + \sum_{i=1}^n b_i^2 \sum_{j=1}^n a_j^2 - 2 \sum_{i=1}^n a_i b_i \sum_{j=1}^n b_j a_j \\ &= 2 \left(\sum_{i=1}^n a_i^2 \right) \left(\sum_{i=1}^n b_i^2 \right) - 2 \left(\sum_{i=1}^n a_i b_i \right)^2\end{aligned}$$

Because the left-hand side of the equation is a sum of the squares of real numbers it is greater than or equal to zero, thus

$$\left(\sum_{i=1}^n a_i^2 \right) \left(\sum_{i=1}^n b_i^2 \right) \geq \left(\sum_{i=1}^n a_i b_i \right)^2$$

$$\rightarrow (\Phi(x) \cdot \Phi(x)) (\Phi(y) \cdot \Phi(y)) \geq (\Phi(x) \cdot \Phi(y))^2$$

$$\rightarrow K(x, x)K(y, y) \geq K(x, y)^2$$

5.2

$$\begin{aligned}
||\mu_\Phi|| &= \sqrt{\left\| \frac{1}{Q} \sum_{i=1}^Q \Phi(x_i) \right\|^2} \\
&= \frac{1}{Q} \sqrt{\left(\sum_{i=1}^Q \Phi(x_i) \right)^T \left(\sum_{i=1}^Q \Phi(x_i) \right)} \\
&= \frac{1}{Q} \sqrt{\left(\sum_{i=1}^Q \Phi(x_i)^T \right) \left(\sum_{i=1}^Q \Phi(x_i) \right)} \\
&= \frac{1}{Q} \sqrt{\left(\sum_{m=1}^Q \sum_{n=1}^Q \underbrace{\Phi(x_m)^T \Phi(x_n)}_{\text{kernel}} \right)} \\
&= \frac{1}{Q} \sqrt{\sum_{m=1}^Q \sum_{n=1}^Q K_\Phi(x_m, x_n)}
\end{aligned}$$

6 Soft SVM with Kernel

In this section, we will classify the MNIST dataset using a Support Vector Machine classifier with different kernel functions, including linear, RBF (Radial Basis Function), and polynomial degree 2. Then, we will compare the performance of the SVM classifiers with that of a Logistic Regression model.

6.1 Train Test Split

For this experiment, we will use a subset of the MNIST dataset to train and evaluate the models. Instead of using the entire 60,000 training and 10,000 test samples, we will use 1,000 samples as the training data and 100 samples as the test data to ensure faster training and testing.

The distribution of training data is shown in the table below:

Digit	Count
0	97
1	116
2	99
3	93
4	105
5	92
6	94
7	117
8	87
9	100

Table 6.1: Train data distribution

6.2 Linear Kernel

The best parameter and score for this classifier are as follows, which were achieved by grid search:

Best Score	0.88
C (Regularization parameter)	1e-06

Table 6.2: Linear kernel grid search

Best Score	0.91
C (Regularization parameter)	10
gamma	scale

Table 6.3: RBF kernel grid search

6.3 RBF kernel

The best parameter and score for this classifier are as follows, which were achieved by grid search:

6.4 Polynomial Kernel (degree = 2)

The best parameter and score for this classifier are as follows, which were achieved by grid search:

Best Score	0.89
C (Regularization parameter)	10
gamma	scale

Table 6.4: Polynomial kernel grid search

6.5 Best SVM

So the best SVM model was achieved using the RBF kernel with the hyperparameter in Table 6.3. The training and test accuracy of this optimized SVM model is shown in the table below:

Train accuracy	1.00
Test accuracy	0.94

Table 6.5: Best SVM model evaluation

6.6 SVM VS. Logistic Regression

We also trained a Logistic Regression classifier on the same training and test datasets used for the SVM models. The Logistic Regression model achieved an accuracy of 0.90 on the test set, which is lower than the accuracy of the optimized SVM classifier.

This result was expected, as the SVM model with the RBF kernel can learn a more complex, non-linear decision boundary, which is better suited for the MNIST dataset. In contrast, the Logistic Regression model is limited to learning linear decision boundaries.

Additionally, the SVM algorithm aims to find the maximum-margin hyperplane, which can provide better generalization performance compared to the Logistic Regression model.

For example, consider the image shown below. This particular sample was classified correctly by the optimized SVM model but was misclassified by the Logistic Regression model.

SVM Correct, Logistic Wrong

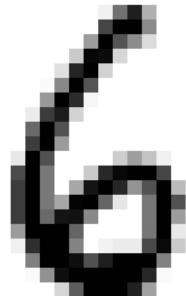


Figure 6.1: This image was classified correctly by the SVM model, but was misclassified by the Logistic Regression model.