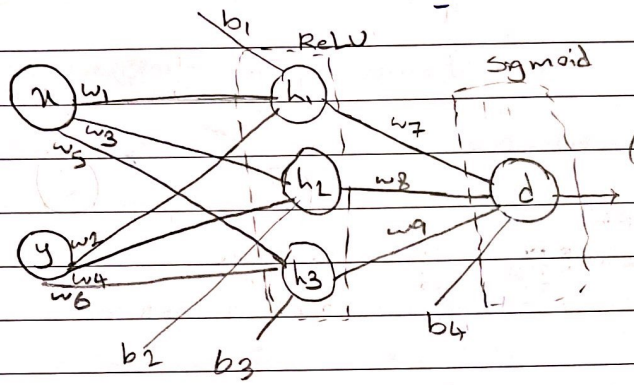
HW1 Report

Fundamentals of Computational Intelligence

Niki Nezakati
98522094
Spring 2022

(Q1

نیکی ترابی - 98522094 - تمرین سری اول



1

(دقت محاسبات تا دو رقم اعشار)

$w_1 = w_2 = w_3 = w_4 = w_5 = w_6 = w_7 = w_8 = w_9 = 0.1$

$b_1 = b_2 = b_3 = b_4 = 1$, $\eta = 0.1$ $\alpha(\text{sigmoid}) = 1$

Forward-pass:

$h_1 = ReLU(w_1 * x + w_2 * y + b_1)$

$h_2 = ReLU(w_3 * x + w_4 * y + b_2)$

$h_3 = ReLU(w_5 * x + w_6 * y + b_3)$

$d = sigmoid(w_7 * h_1 + w_8 * h_2 + w_9 * h_3 + b_4)$

Lable

Input: $x = -4$, $y = 0$, $L = 0$

$h_1 = relu(-0.4 + 1) = 0.6 = h_2 = h_3$

$d = sigmoid(0.1 \times 0.6 + 0.1 \times 0.6 + 0.1 \times 0.6 + 1) = sigmoid(1.18) = 0.76$

$Error = \frac{1}{2}(0 - 0.76)^2 = 0.28$

Backward-pass: output layer → $\Delta w_{ij} = \eta \times \text{sigmoid}' \times y_i$

Δw_9: $\frac{\delta Error}{\delta w_9} = \frac{\delta Error}{\delta d} \times \frac{\delta d}{w_9} = (d-L) \times \text{Sigmoid}(outh_3) \times h_3$

$\Rightarrow \delta w_9 : (d-L) \times 0.76(1-0.76) = 0.13 \longrightarrow \Delta w_9 = 0.13 \times 0.1 \times 0.6 = 0.007$

$w_9 = w_9 - \Delta w_9 = 0.1 - 0.007 = 0.09$ متقارن $\rightarrow w_7 = w_8 = 0.09$

$\Delta b_4 = \frac{\delta Error}{\delta d} \times \frac{\delta d}{\delta b} = (d-L) \times \text{sigmoid}' \times 1 = 0.76 \times 0.1824 = 0.13$

$\rightarrow b_4 = 1 - 0.1 \times 0.13 = 0.98$

hidden layer

$\Delta w_6 = \frac{\delta Error}{\delta h_3} \times \underbrace{\frac{\delta h_3}{\delta inh_3}}_{inh_3} \times \underbrace{\frac{\delta inh_3}{\delta w_6}}_{y} = \frac{\delta Error}{\delta d} \times \text{relu}'(outh_1) \times \frac{\delta inh_3}{\delta w_6}$

$\Rightarrow \Delta w_6 = \eta \times y_i \times \delta_6 = 0.1 \times 0 \times \delta 6 = 0 \Rightarrow w_6 = 0.1$

($y=0$ چون متقارن $\Rightarrow$ صورت متقارن) $\begin{cases} w_4 = 0.1 \\ w_2 = 0.1 \end{cases}$

$\Delta w_1 = \eta \times y_i \times \delta_1 = 0.1 \times -4 \times \delta_1$

$\Rightarrow \delta_1 = \underbrace{\text{relu}'(inh_1)}_{1} \times \sum_{k=7,8,9} \delta_k \times w_k \rightarrow (0.13) \times 0.1 \times 3 = 0.03$

$\Rightarrow \delta_1 = 1 \times 0.03 = 0.03$

$\Rightarrow \Delta w_1 = 0.1 \times -4 \times 0.03 = -0.012 \rightarrow w_1 = 0.1 + 0.012 = 0.11$

به صورت متقارن $\begin{cases} w_3 = 0.11 \\ w_5 = 0.11 \end{cases}$

$\Delta b_1 = \frac{\delta Error}{\delta b_1} = \underbrace{\delta w_7 \times w_7}_{0.09 \times 0.1} \times \underbrace{\frac{\delta w_7}{\delta b_1}}_{1} = 0.009$

$\rightarrow b_1 = 1 - 0.1 \times 0.009 = 0.99$ متقارن $\rightarrow b_2 = b_3 = b_1 = 0.99$

$\delta w_7 = \delta w_8 = \delta w_9$

$w_7 = w_8 = w_9$

وزن‌های جدید:
$w_1 = w_3 = w_5 = 0.11$
$w_2 = w_4 = w_6 = 0.1$
$w_7 = w_8 = w_9 = 0.09$

بایاس‌های جدید:
$b_1 = b_2 = b_3 = 0.99$
$b_4 = 0.98$

■■●

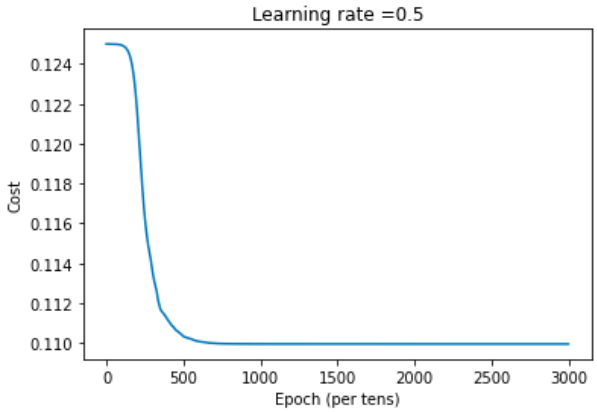
آیا این مسئله با شبکه Adaline قابل حل است؟

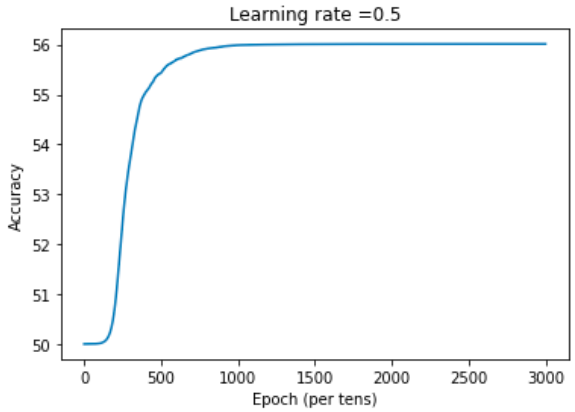
خیر؛ برای حل مسئله با شبکه Adaline نیاز است تا توان classification را با استفاده از یک خط انجام داد ولی در شکل مشاهده می‌شود که جداسازی کلاس‌های 1 و 2 با استفاده از یک خط امکان پذیر نیست پس به یک لایه میانی حداقل نیاز داریم (یعنی 2 خط برای جداسازی) پس باید از مدل MLP استفاده کنیم.

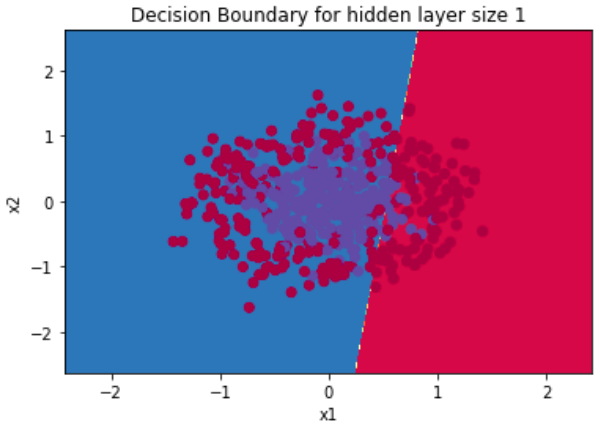
Q2) پرسپترون چند لایه (MLP)

با امتحان کردن مقادیر مختلف براي تعداد لایه های hidden و تعداد نورون های هر لایه میخواهیم عملکرد شبکه عصبی را بررسی کنیم. ابتدا با یک لایه میانی با یک نورون شروع میکنیم. پس از 3000 Epoch عملکرد به صورت زیر است:

cost: 0.1099547873999455 accuracy%: 56.011651148909586



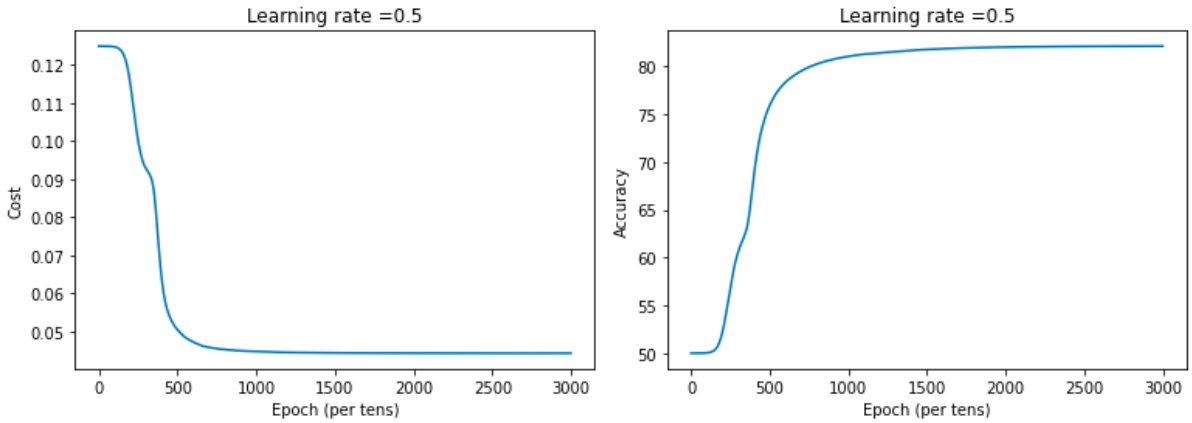


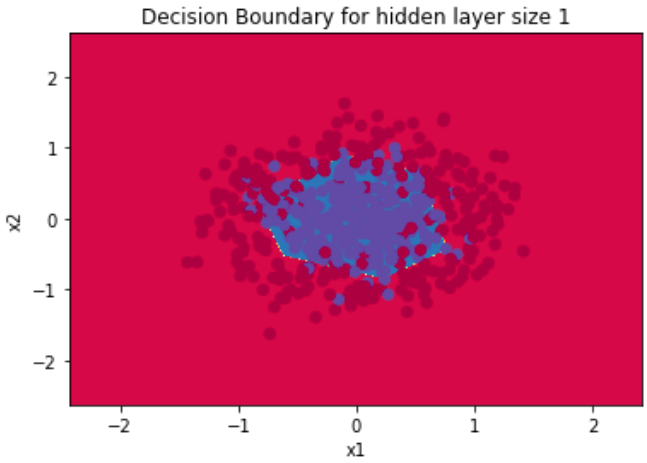


همانطور که مشاهده میشود شبکه ما دقت خوبی ندارد.

حال تعداد نورون های لایه میانی را 3 قرار میدهیم. پس از 3000 Epoch عملکرد به صورت زیر است:

cost: 0.04422747398777807 accuracy%: 82.10041569100254

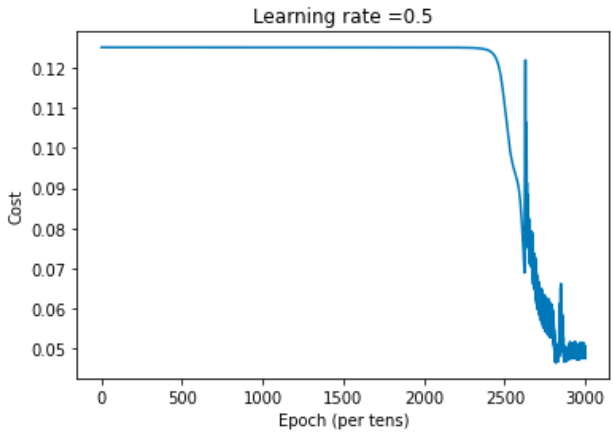


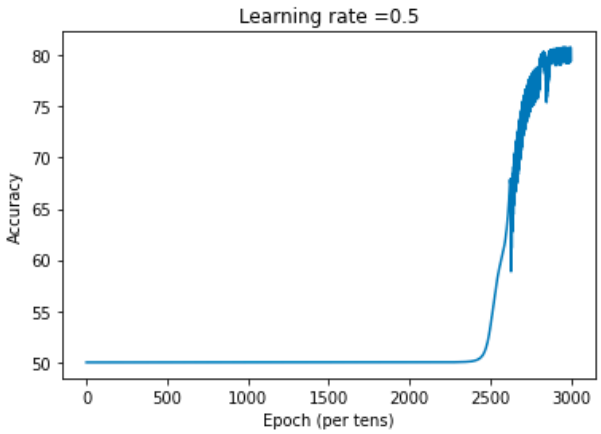


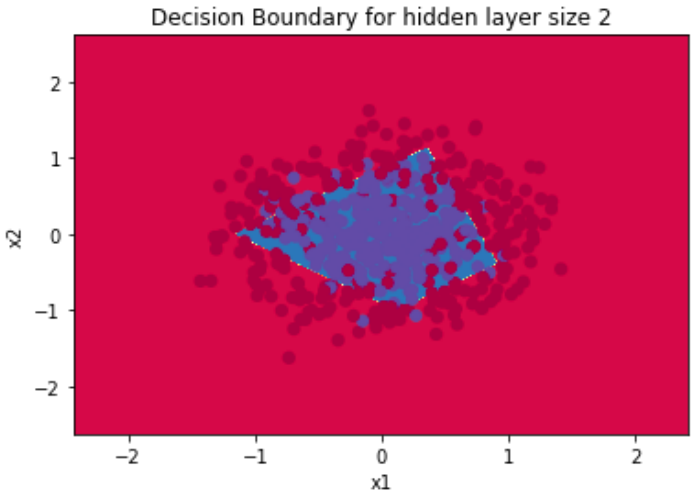
یک لایه میانی با 3 نورون عملکرد خوبی دارد. دقت به 82.1% رسیده و cost نیز به 0.04 کاهش پیدا کرده و decision boundry به خوبی نقاط را از هم تفکیک کرده است.

تعداد لایه های میانی را به 2 لایه افزایش میدهیم. لایه اول با 3 نورون و لایه دوم با 2 نورون. پس از 3000 Epoch عملکرد به صورت زیر است:

cost: 0.047511674995534665 accuracy%: 80.55976582033921



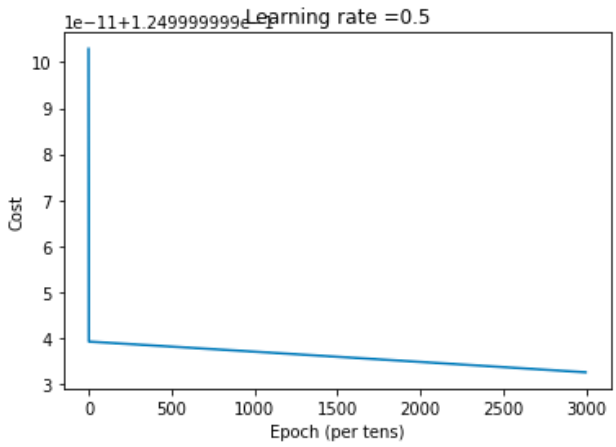


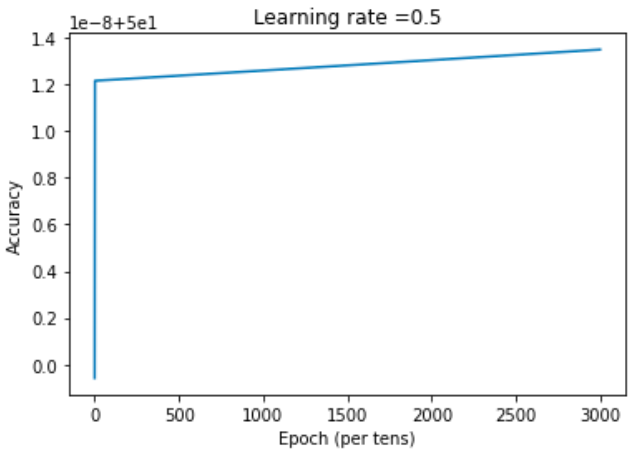


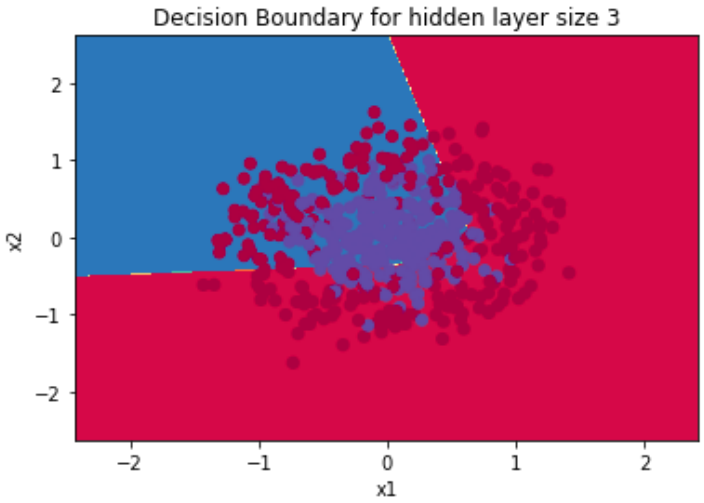
مدل همچنان عملکرد نسبتا خوبی دارد ولی دقت آن نسبت به مدل با یک لایه میانی دقتش کمتر شده است و از نمودار decision boundry میتوان مشاهده کرد که تا با زیاد شدن لایه میانی تا حدودی over-fit رخ داده است و general pattern تشخیص نقاط کمی اشتباه شده است.

تعداد لایه های میانی را 3 قرار میدهیم. با تعداد نورون های 2،3،4. پس از 3000 Epoch عملکرد به صورت زیر است:

cost: 0.12499999993276859 accuracy%: 50.00000001344628



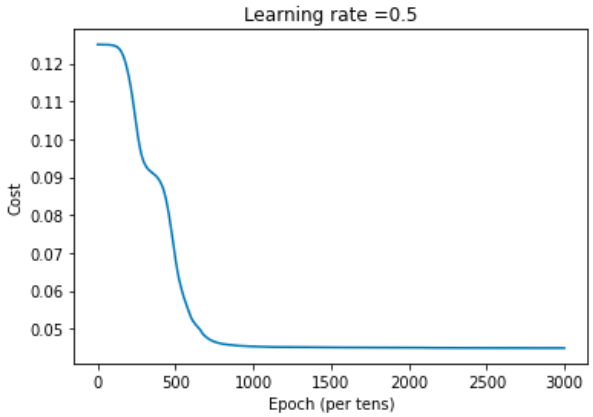


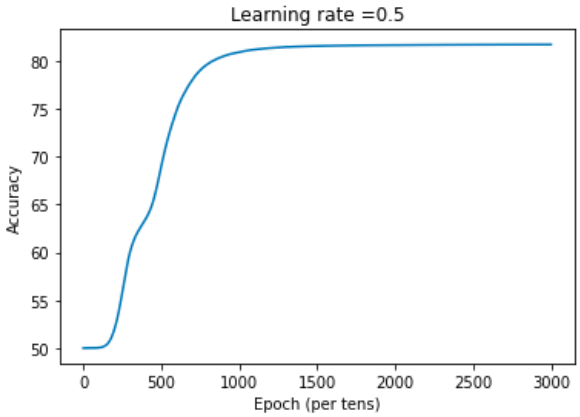


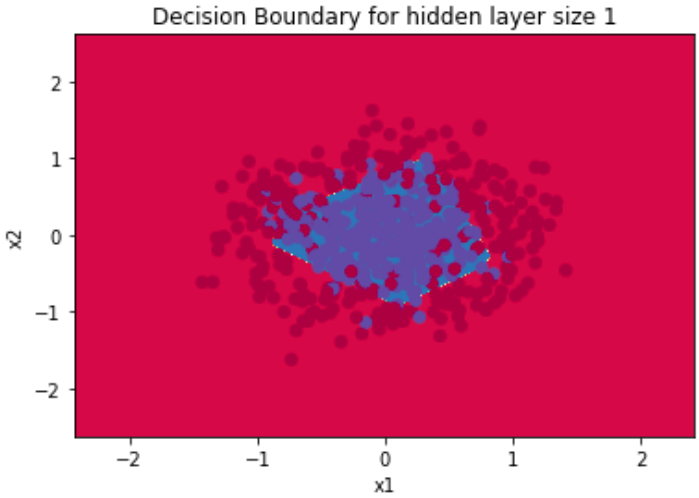
دقت مدل کاهش زیادی داشته و با رخ دادن over-fit مدل نتوانسته decision boundry درستی برای تفکیک نقاط ارائه دهد.

تا اینجا بهترین عملکرد برای مدلی با یک لایه میانی و 3 نورون بود. میخواهیم ببینیم اگر تعداد نورون ها را در همین لایه به 5 نورون افزایش بدهیم عملکرد چگونه خواهد بود. پس از 3000 Epoch :

cost: 0.044893979016888144 accuracy%: 81.71172620451111







میبنیم که دقت مدل نسبت به حالت 3 نورون کاهش یافته است و از نمودار deciosion boundry هم میتوان مقداری over-fit را مشاهده کرد.

پس بهترین مدل برای این سوال یک شبکه با یک لایه میانی و 3 نورون در آن لایه بود.

Keras (Q3

The CIFAR-10 small photo classification problem is a standard dataset used in computer vision and deep learning. CIFAR is an acronym that stands for the Canadian Institute For Advanced Research and the CIFAR-10 dataset was developed along with the CIFAR-100 dataset by researchers at the CIFAR institute. The dataset is comprised of 60,000 32×32 pixel color photographs of objects from 10 classes, such as frogs, birds, cats, ships, etc. The class labels and their standard associated integer values are listed below.

- 0: airplane
- 1: automobile
- 2: bird
- 3: cat
- 4: deer
- 5: dog
- 6: frog
- 7: horse
- 8: ship
- 9: truck

These are very small images, much smaller than a typical photograph, and the dataset was intended for computer vision research.

CIFAR-10 is a well-understood dataset and widely used for benchmarking computer vision algorithms in the field of machine learning. The problem is "*solved*." It is relatively straightforward to achieve 80% classification accuracy. Top performance on the problem is achieved by deep learning convolutional neural networks with a classification accuracy above 90% on the test dataset.

We know some things about the dataset. For example, we know that the images are all pre-segmented (e.g. each image contains a single object), that the images all have the same square size of 32×32 pixels, and that the images are color. Therefore, we can load the images and use them for modeling almost immediately.

```
# load dataset
(trainX, trainY), (testX, testY) = cifar10.load_data()
```

We also know that there are 10 classes and that classes are represented as unique integers. We can, therefore, use a one hot encoding for the class element of each sample, transforming the integer into a 10 element binary vector with a 1 for the index of the class value. We can achieve this with the *to_categorical()* utility function.

```
# one hot encode target values
trainY = to_categorical(trainY)
testY = to_categorical(testY)
```

The *load_dataset()* function implements these behaviors and can be used to load the dataset.

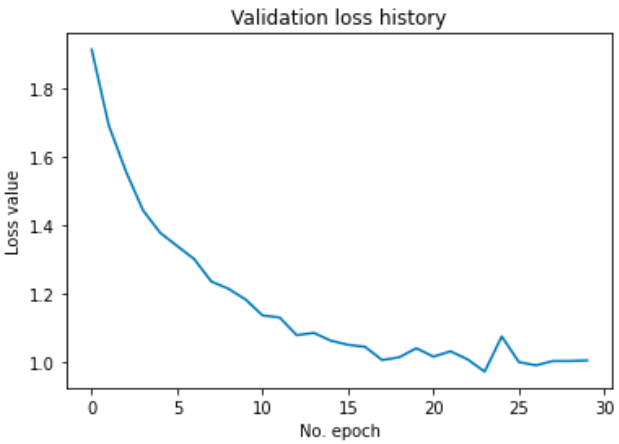
```
# load train and test dataset
def load_dataset():
    # load dataset
    (trainX, trainY), (testX, testY) = cifar10.load_data()
    # one hot encode target values
    trainY = to_categorical(trainY)
    testY = to_categorical(testY)
    return trainX, trainY, testX, testY
```

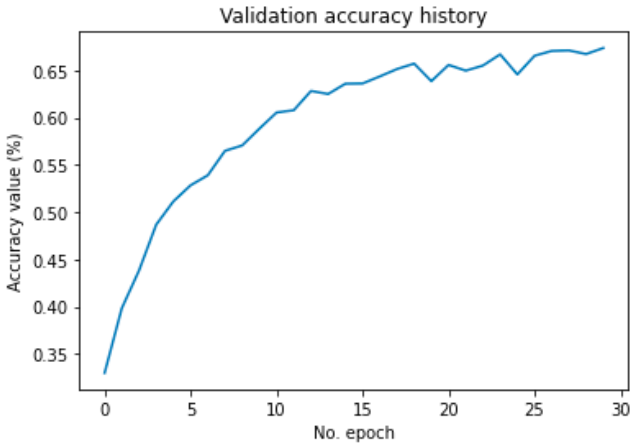
آزمایش شبکه با چند لایه مختلف:

شبکه با 3 لایه میانی با activation = relu :

Epoch 30/30: loss: 0.6071 - accuracy%: 79.13

Test loss: 1.00833261013031 / Test accuracy: 0.6658999919891357

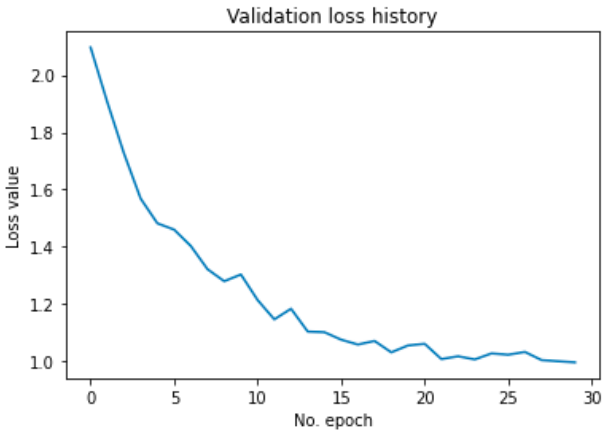


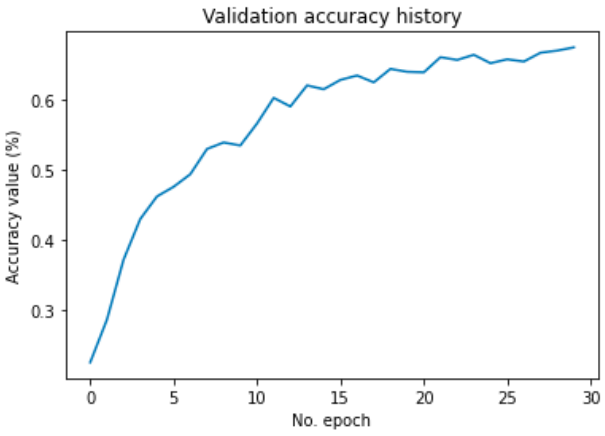


شبکه با 5 لایه میانی با activation = relu :

Epoch 30/30: loss: 0.6070 - accuracy%: 78.86

Test loss: 1.0055334568023682 / Test accuracy: 0.6747999787330627



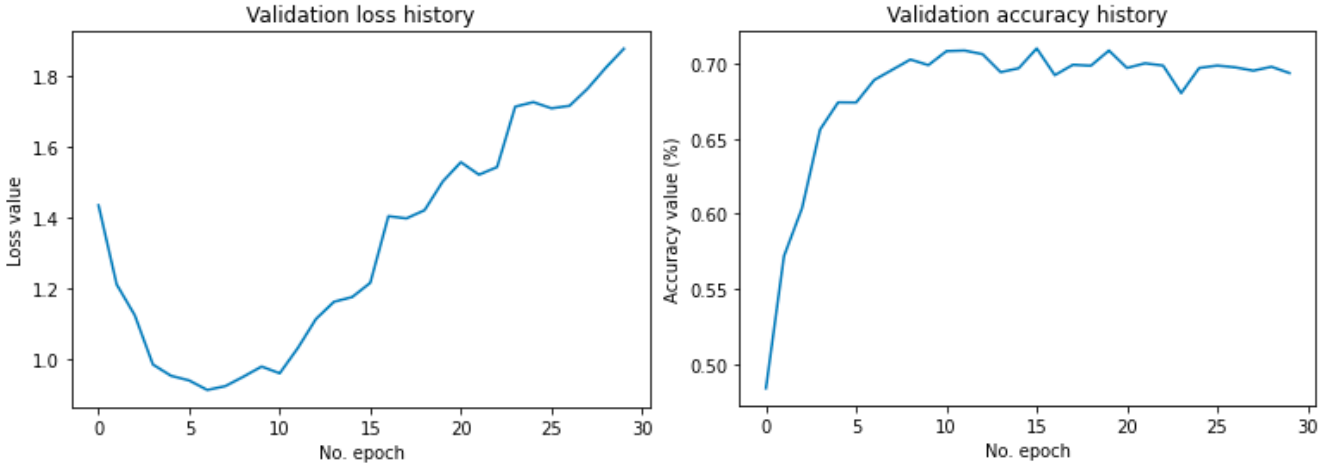


مشاهده میشود که با اضافه شدن لایه چهارم و پنجم، دقت کاهش یافته و loss نیز بیشتر شده است ولی دقت مدل روی تست ها بیشتر شده است و همانطور که از نمودارها هم مشاهده میشود با اضافه شدن این دو لایه مدل در تشخیص روند کلی و general داده ها بهبود یافته است.

عملکرد شبکه با فعال بودن Momentum=0.9 :

Epoch 30/30: loss: 0.1299 - accuracy%: 95.66

Test loss: 1.9896999597549438 / Test accuracy: 0.6855999827384949



در حالت فعال بودن Momentum دقت شبکه بهبود میابد و در epoch 30 از 78.86% به 95.66% میرسد که دقت بالایی است.

محاسبات Gradient descent با فعال بودن Momentum به صورت زیر است:

```
velocity = momentum * velocity - learning_rate * g
w = w + momentum * velocity - learning_rate * g
```

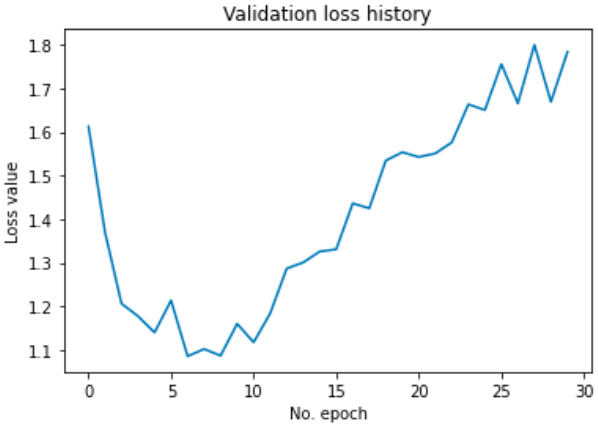
این هایپر پارامتر (Momentum) که مقداری بین صفر و یک است، قسمتی از wight update قبلی را به weight update جدید اضافه میکند. این باعث میشود که بردارهای gradient descent در جهت درست و سریع تر تغییر کنند و نوسانات آن را کاهش میدهد. در نتیجه همگرایی سریع تر اتفاق میفتد.

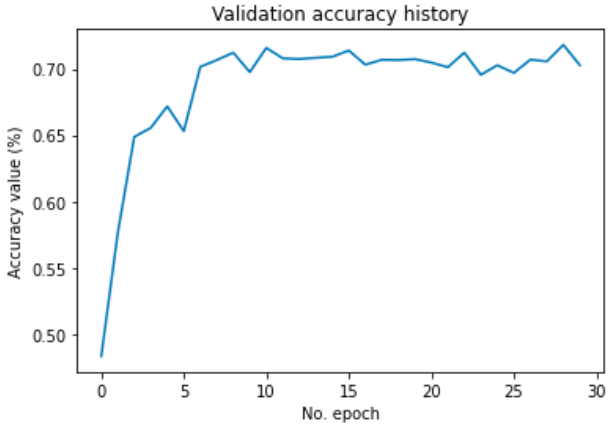
در نتیجه با فعال شدن momentum در مدل مناسب، دقت مدل ما افزایش پیدا میکند. که این اتفاق در مدل ما نیز قابل مشاهده است.

عملکرد شبکه با فعال بودن decay= l1=1e-5, l2=1e-4 در لایه های ConV2D با activation = relu :

Epoch 30/30: loss: 0.4328 - accuracy%: 96.24

Test loss: 1.8278414011001587 / Test accuracy: 0.6970999836921692





در حالت فعال بودن weight decay دقت شبکه بهبود میابد و در epoch 30 از 95.66% به 96.24% میرسد که دقت بالایی است. همچنین دقت test نیز افزایش یافته است.

محاسبات Gradient descent با فعال بودن weight decay به صورت زیر است:

loss = loss + weight decay parameter * L2 norm of the weights

از weight decay برای جلوگیری از overfitting و بزرگ شدن بیش از حد weight ها استفاده میشود. به دلیل مقدار اضافه شده به loss، در هر epoch شبکه تلاش میکند تا weight های مدل را با توجه به این مقدار اضافه شده optimize کند که همانطور که در مدل ما نیز میبینیم، این اتفاق موجب بهبود عملکرد شبکه شده است.

(Q4

4

1. مقایسه بین Perceptron و Adaline ، تفاوت اصلی در این است که perceptron برای بدست آوردن error ، آپدیت کردن وزن ها ، از binary response و نتیجه classification استفاده می‌کند ولی Adaline از continous response استفاده می‌کند. این عملکرد Adaline باعث می‌شود که update ها سریع به error واقعی نزدیک باشند به همین دلیل نیز قابلیت تعمیم Adaline از perceptron بیشتر می‌باشد. Madaline چون نسبت به Adaline تعداد لایه‌های بیشتری دارد دارد قابلیت تعمیم بیشتری نیز دارد. MLP چون از Activation function های non-linear نیز می‌تواند استفاده کند درنتیجه برای classification های پیچیده‌تر نیز قابل استفاده است پس قابلیت تعمیم MLP از همه بیشتر است.

2. وقتی یک model روی training data عملکرد خوبی دارد ولی روی test data ، data های جدید عملکرد خوبی ندارد می‌گوییم که شبکه دچار overfit شده است. در این حالت model جزئیات و نویزهای training data را یاد می‌گیرد به طوری که اثر منفی بر روی عملکرد شبکه هنگام اجرا روی داده‌های جدید می‌گذارد. راه‌های مختلف آن می‌تواند به صورت زیر باشد:

1- داده استفاده شده برای training باید cleaned باشد و شامل noise نباشد.

2- model باید پارامترهای بهینه باشد.

3- سایز training dataset کافی باشد.

4- model یکی از ساده‌ترین باید (مثل تعداد لایه‌های زیاد یا تعداد نورون زیاد در هر لایه)

3. ساده کردن مدل - متوقف کردن training در زمان کوتاه‌تر - بزرگ‌تر کردن training dataset - استفاده از regularization مثل weight decay

4. وقتی model الگوی بین training data را به خوبی یاد نگرفته است و توانایی تعمیم به داده‌های جدید را ندارد می‌گوییم underfitting رخ داده است. یک underfit model ، عملکرد ضعیفی بر روی training data دارد و prediction های خوبی نخواهد داشت.

(Q5

$y = ax_1^2 + bx_2^2 + cx_1x_2 + d$

$\begin{cases} a=-1 \\ b=1 \\ c=-1 \\ d=2 \end{cases}$

5

Batch size = 4, Learning rate = 0.01, $\rho = 0.9$

x_1	x_2	y
1	-1	10
2	0	13
-1	2	15
1	1	6

$$\Delta w_{ji} = \alpha \sum_{k=1}^{4} (d_j^k - y_j^k)\varphi'(v_j)y_i$$

$$\Rightarrow \varphi'(v_j) = 1 \rightarrow \Delta w_{ji} = \left[\alpha \sum_{k=1}^{4} (d_j^k - y_j^k)y_i\right]$$

with Momentum $\Rightarrow \Delta w_{ji}(n) = \rho \Delta w_{ji}(n-1) + \alpha \sum_{k=1}^{4} (d_j^k(n) - y_j^k(n))\, y_i(n)$

Epoch = 1:

$\Delta w_{ji}(0) = 0$

k=1: $-1\times1 + 1\times1 + (-1)(-1)(1) + 2 = 3 \Rightarrow d-y = 7$

k=2: $-1\times4 + 1\times0 + (-1)(2)(0) + 2 = -2 \Rightarrow d-y = 15$

k=3: $-1\times1 + 1\times4 + (-1)(-1)(2) + 2 = 7 \Rightarrow d-y = 8$

k=4: $-1\times1 + 1\times1 + (-1)(1)(1) + 2 = 1 \Rightarrow d-y = 5$

$\Delta a(1) = 0.9\times0 + 0.01\times(7\times1 + 15\times4 + 8\times1 + 5\times1) = 0.8$

$\Delta b(1) = 0.9\times0 + 0.01\times(7\times1 + 15\times0 + 8\times4 + 5\times1) = 0.44$

$\Delta c(1) = 0.9\times0 + 0.01(7\times(-1) + 15\times0 + 8\times(-2) + 5\times1) = -0.18$

$\Delta d(1) = 0.9\times0 + 0.01(7+15+8+5) = 0.35$

$w_{ji} = w_{ji} + \Delta w_{ji} \Rightarrow \begin{cases} a = -0.2 \\ b = 1.44 \\ c = -1.18 \\ d = 2.35 \end{cases}$

/ / /

Epoch = 2:

$k=1: -0.2\times1 + 1.44\times1 + (-1.18)(1)(-1) + 2.35 = 4.77 \rightarrow d-y = 5.23$

$k=2: -0.2\times4 + 1.44\times0 + (-1.18)\times2\times0 + 2.35 = 1.55 \rightarrow d-y = 11.45$

$k=3: -0.2\times1 + 1.44\times4 + (-1.18)\times2\times(-1) + 2.35 = 10.27 \rightarrow d-y = 4.73$

$k=4: -0.2\times1 + 1.44\times1 + (-1.18)(1)(1) + 2.35 = 2.41 \rightarrow d-y = 3.59$

$\Delta a(2) = 0.9\times0.8 + 0.01(5.23\times1 + 11.45\times4 + 4.73\times1 + 3.59\times1) = 1.31$

$\Delta b(2) = 0.9\times0.44 + 0.01\times(5.23\times1 + 11.45\times0 + 4.73\times4 + 3.59\times1) = 0.67$

$\Delta c(2) = 0.9\times(-0.18) + 0.01(5.23(-1) + 11.54\times0 + 4.73(-2) + 3.59) = -0.27$

$\Delta d(2) = 0.9\times0.35 + 0.01(5.23 + 11.45 + 4.73 + 3.59) = 0.56$

$w_{ji} = w_{ji} + \Delta w_{ji} \Rightarrow \begin{cases} a = 1.11 \\ b = 2.11 \\ c = -1.45 \\ d = 2.91 \end{cases}$