به نام خدا

دانشگاه تهران

پردیس دانشکده‌های فنی

دانشکده برق و کامپیوتر

درس سیستم‌های هوشمند

تمرین شماره ۳

نام و نام خانوادگی : سیاوش شمس

شماره دانشجویی : ۸۱۰۱۹۷۶۴۴

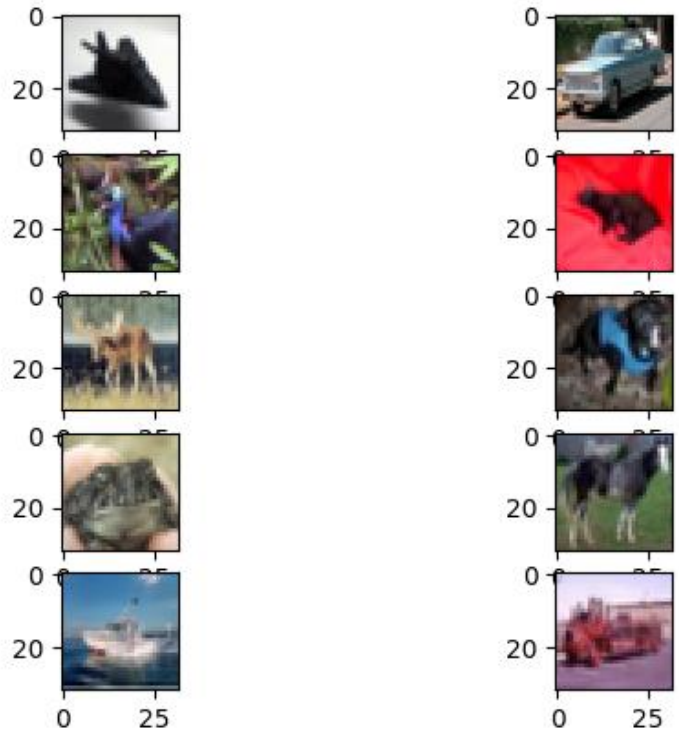
آذر ۱۴۰۰

فهرست سوالات

سوال ۱

در این سوال ابتدا تصویر یک نمونه از هر طبقه را نمایش می دهیم، سپس داده های ورودی را نرمالایز می کنیم، و در ادامه بر اساس اطلاعات داده شده در سوال و آزمون و خطا شبکه عصبی کانولوشنی با یک لایه طراحی می کنیم، در ادامه تعداد اثر لایه های مخفی، تابع فعالساز، روش بهینه سازی و روش Dropout را برسی می کنیم.

الف:



شکل ۱-۱- تصویر مختلف از هر برچسب

متوجه می شویم این دادگان شامل عکس هایی با عنوان های: هواپیما، اتومبیل، پرنده، گربه، گوزن، سگ، قورباغه، اسب، کشتی و کامیون را دارد.

به عنوان پیش پردازش[1]، داده های مربوط به هر تصویر را نرمالایز می کنیم تا یادگیری سریعتر صورت بگیرد و مدل دقیق تر باشد. این کار با تقسیم کردن عدد هر پیکسل به ۲۵۵ صورت می گیرد چون می دانیم هر پیکسل عددی بین ۰ تا ۲۵۵ را دارد.

[1] Pre-Process

چون مسئله از نوع طبقه بندی غیر باینری(۱۰ طبقه بندی مختلف) است از تابع فعالساز softmax استفاده می کنیم تا احتمال تعلق داشتن هر داده به یک طبقه را مشخص کند.

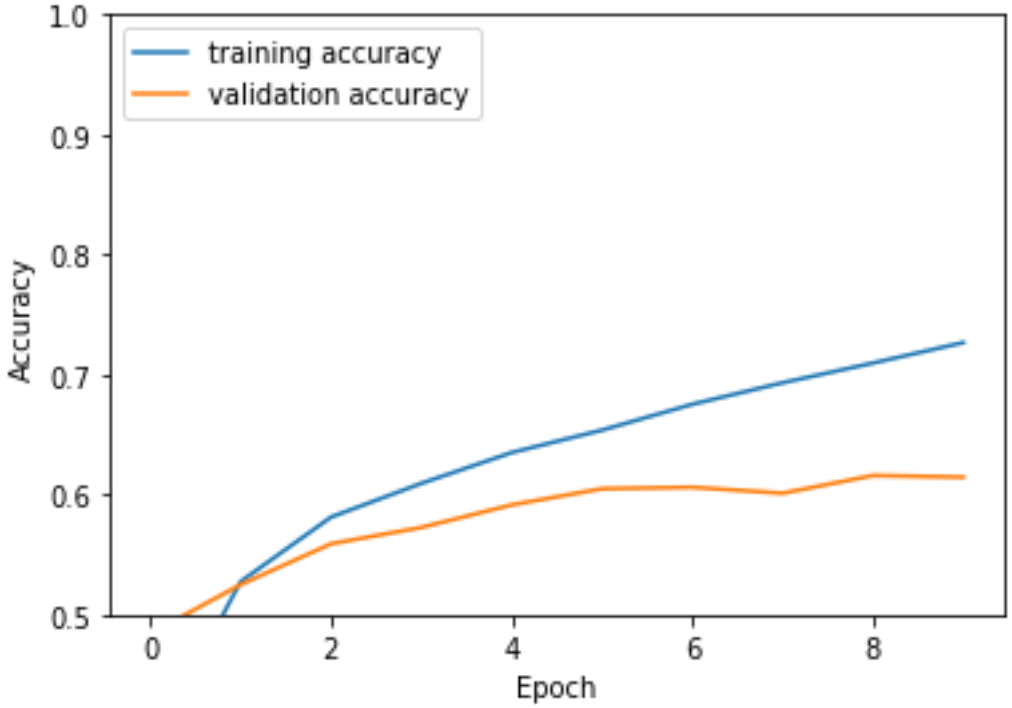
جدول ۱-۱- معماری شبکه و تعداد پارامتر های قابل یادگیری

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 conv2d_8 (Conv2D)            (None, 32, 32, 32)        416

 max_pooling2d_4 (MaxPooling   (None, 16, 16, 32)       0
 2D)

 flatten_3 (Flatten)          (None, 8192)              0

 dense_6 (Dense)              (None, 64)                524352

 dense_7 (Dense)              (None, 10)                650

=================================================================
Total params: 525,418
Trainable params: 525,418
Non-trainable params: 0
_________________________________________________________________
```

بر اساس تحقیقات و آزمون و خطا یک لایه کانولوشنی با اندازه هسته[۱] ۲ در ۲ و فعالساز ReLU و لایه گذاری حاشیه ای[۲] و طول گام برداشتن[۳] ۱ در ابتدای شبکه قرار دادیم سپس از یک لایه ادغامی بر اساس بیشینه [۴] با بعد ۲ استفاده کردیم، در ادامه یک لایه تماما متصل [۵] با ۶۴ گره با فعالساز ReLU را قرار دادیم و یک لایه با ۱۰ گره با فعالساز softmax را به عنوان لایه خروجی تعیین کردیم.

همچنین برای آموزش از روش گرادیان نزولی تصادفی [۶] با تکانه [۷]و اندازه بسته ۶۴ و گام ثابت 0.01 استفاده کردیم و مدل را در ۱۰ تکرار[۸] آموزش دادیم.

[1] Kernel
[2] Padding
[3] Stride
[4] Max-Pooling
[5] Dense
[6] Stochastic Gradient Descent
[7] Momentum
[8] Epoch



شکل۱-۲- مقایسه دقت روی داده های آموزش و ارزیابی به ازای هر تکرار

```
accuracy on test data: 0.6144999861717224
```

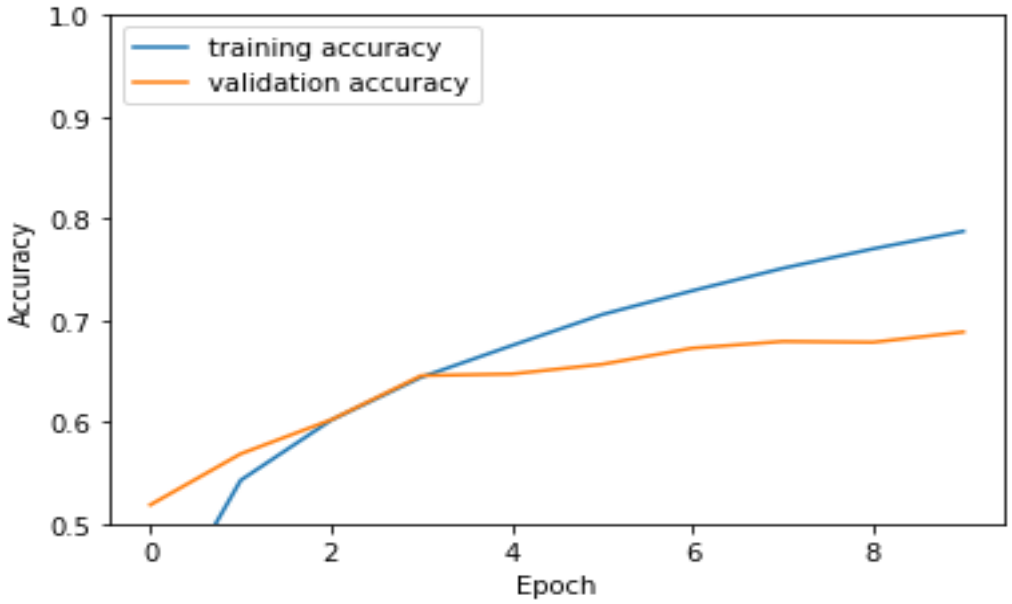
```
loss on test data: 1.2722398042678833
accuracy on train data: 0.7171000242233276
loss on train data: 0.797485888004303
```

شکل۱-۳- دقت و خطای نهایی روی داده های تست و آموزش

ب:

جدول ۱-۲- معماری شبکه جدید و تعداد پارامتر های قابل یادگیری با دو لایه مخفی[1]

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 conv2d_9 (Conv2D)            (None, 32, 32, 32)        416

 max_pooling2d_5 (MaxPooling  (None, 16, 16, 32)        0
 2D)

 conv2d_10 (Conv2D)           (None, 16, 16, 64)        8256

 max_pooling2d_6 (MaxPooling  (None, 8, 8, 64)          0
 2D)

 flatten_4 (Flatten)          (None, 4096)              0

 dense_8 (Dense)              (None, 64)                262208

 dense_9 (Dense)              (None, 10)                650

=================================================================
Total params: 271,530
Trainable params: 271,530
Non-trainable params: 0
_________________________________________________________________
```



شکل۱-۴- مقایسه دقت روی داده های آموزش و ارزیابی به ازای هر تکرار با دو لایه مخفی

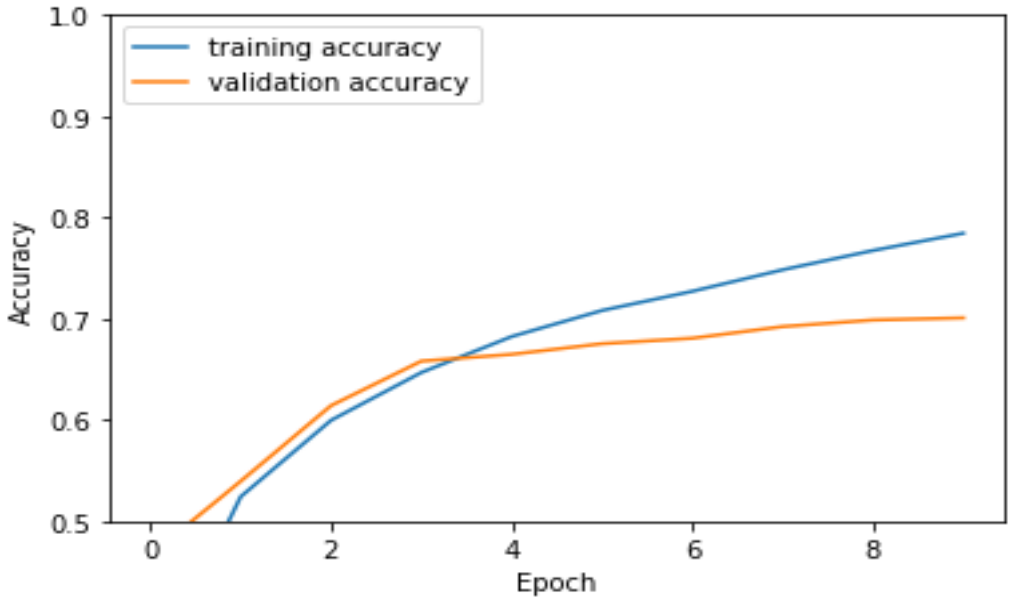
```
accuracy on test data: 0.6862999796867371
loss on test data: 0.9647068381309509
accuracy on train data: 0.8350800275802612
loss on train data: 0.4805280566215515
```

شکل۱-۵- دقت و خطای نهایی روی داده های تست و آموزش با دو لایه مخفی

[1] Hidden Layer

جدول ۱-۳- معماری شبکه جدید و تعداد پارامتر های قابل یادگیری با سه لایه مخفی

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 conv2d_11 (Conv2D)           (None, 32, 32, 32)        416

 max_pooling2d_7 (MaxPooling  (None, 16, 16, 32)        0
 2D)

 conv2d_12 (Conv2D)           (None, 16, 16, 64)        8256

 max_pooling2d_8 (MaxPooling  (None, 8, 8, 64)          0
 2D)

 conv2d_13 (Conv2D)           (None, 8, 8, 128)         32896

 max_pooling2d_9 (MaxPooling  (None, 4, 4, 128)         0
 2D)

 flatten_5 (Flatten)          (None, 2048)              0

 dense_10 (Dense)             (None, 64)                131136

 dense_11 (Dense)             (None, 10)                650

=================================================================
Total params: 173,354
Trainable params: 173,354
Non-trainable params: 0
_________________________________________________________________
```



شکل۱-۶- مقایسه دقت روی داده های آموزش و ارزیابی به ازای هر تکرار با سه لایه مخفی

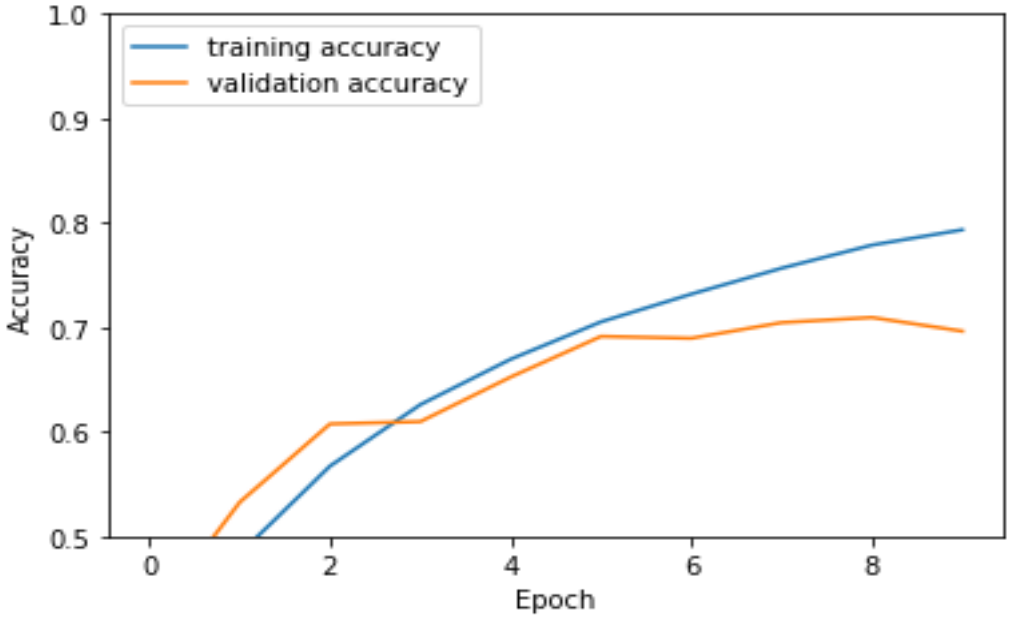
```
accuracy on test data: 0.7114999890327454
loss on test data: 0.8686661124229431
accuracy on train data: 0.8216599822044373
loss on train data: 0.5163847804069519
```

شکل۱-۷- دقت و خطای نهایی روی داده های تست و آموزش با سه لایه مخفی

جدول ۱-۴- معماری شبکه جدید و تعداد پارامتر های قابل یادگیری با چهار لایه مخفی

Layer (type)	Output Shape	Param #
conv2d_14 (Conv2D)	(None, 32, 32, 32)	416
max_pooling2d_10 (MaxPooling2D)	(None, 16, 16, 32)	0
conv2d_15 (Conv2D)	(None, 16, 16, 64)	8256
max_pooling2d_11 (MaxPooling2D)	(None, 8, 8, 64)	0
conv2d_16 (Conv2D)	(None, 8, 8, 128)	32896
max_pooling2d_12 (MaxPooling2D)	(None, 4, 4, 128)	0
conv2d_17 (Conv2D)	(None, 4, 4, 256)	131328
max_pooling2d_13 (MaxPooling2D)	(None, 2, 2, 256)	0
flatten_6 (Flatten)	(None, 1024)	0
dense_12 (Dense)	(None, 64)	65600
dense_13 (Dense)	(None, 10)	650

Total params: 239,146
Trainable params: 239,146
Non-trainable params: 0



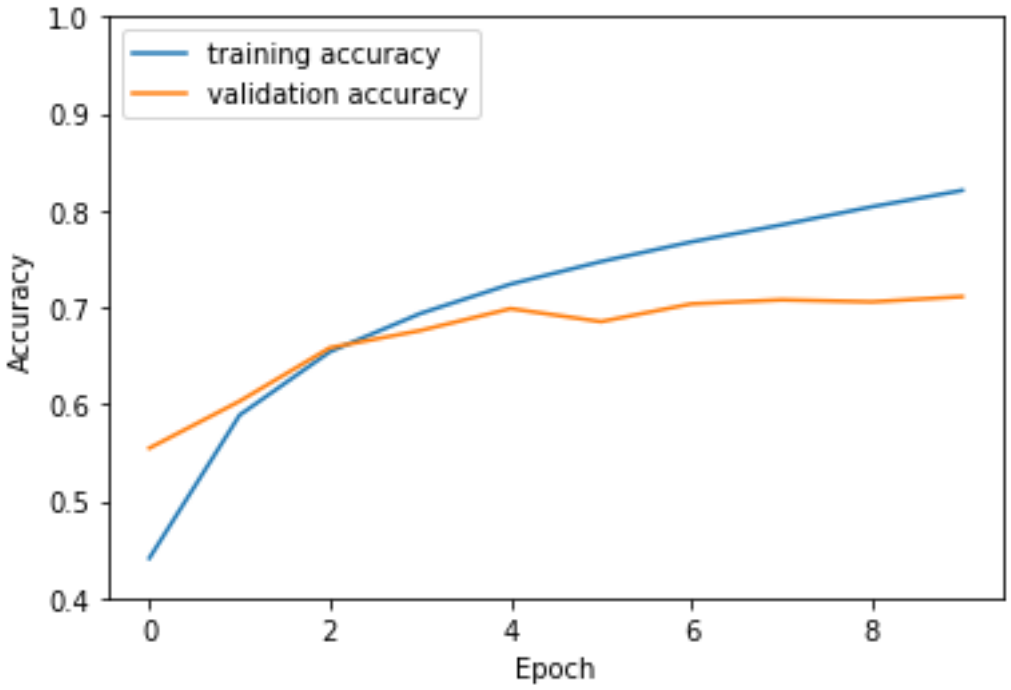
شکل۱-۸- مقایسه دقت روی داده های آموزش و ارزیابی به ازای هر تکرار با چهار لایه مخفی

```
accuracy on test data: 0.6995999813079834
loss on test data: 0.9185530543327332
accuracy on train data: 0.8180800080299377
loss on train data: 0.5153340101242065
```

شکل۱-۹- دقت و خطای نهایی روی داده های تست و آموزش با چهار لایه مخفی

با توجه به نتایج بالا متوجه می شویم بهترین تعداد لایه مخفی برابر ۳ می باشد چون دقت در این حالت بیشترین مقدار را دارد.

ج:



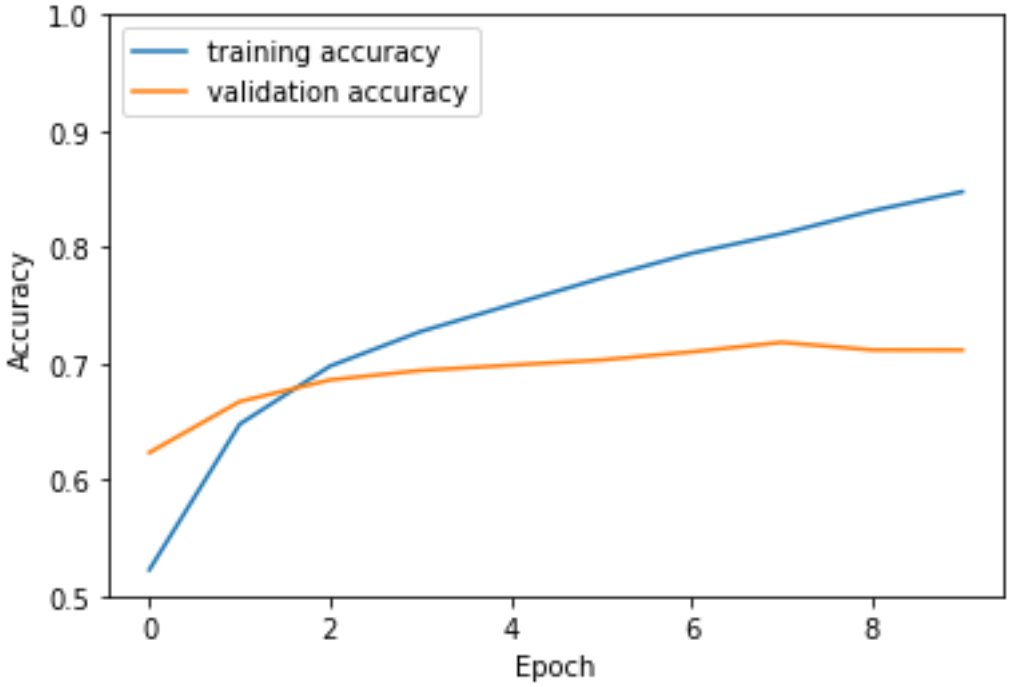
شکل۱-۱۰- مقایسه دقت روی داده های آموزش و ارزیابی به ازای هر تکرار با سه لایه مخفی و تابع فعالساز tanh

```
accuracy on test data: 0.7110000252723694
loss on test data: 0.886150598526001
accuracy on train data: 0.8516799807548523
loss on train data: 0.43746286630630493
```

شکل۱-۱۱- دقت و خطای نهایی روی داده های تست و آموزش با سه لایه مخفی و تابع فعالساز tanh

با مقایسه دقت مربوط به دو مدل متوجه می شویم که مدل با تابع فعالساز Tanh تفاوت چشمگیری در دقت نسبت به تابع فعالساز ReLU ایجاد نمی کند.

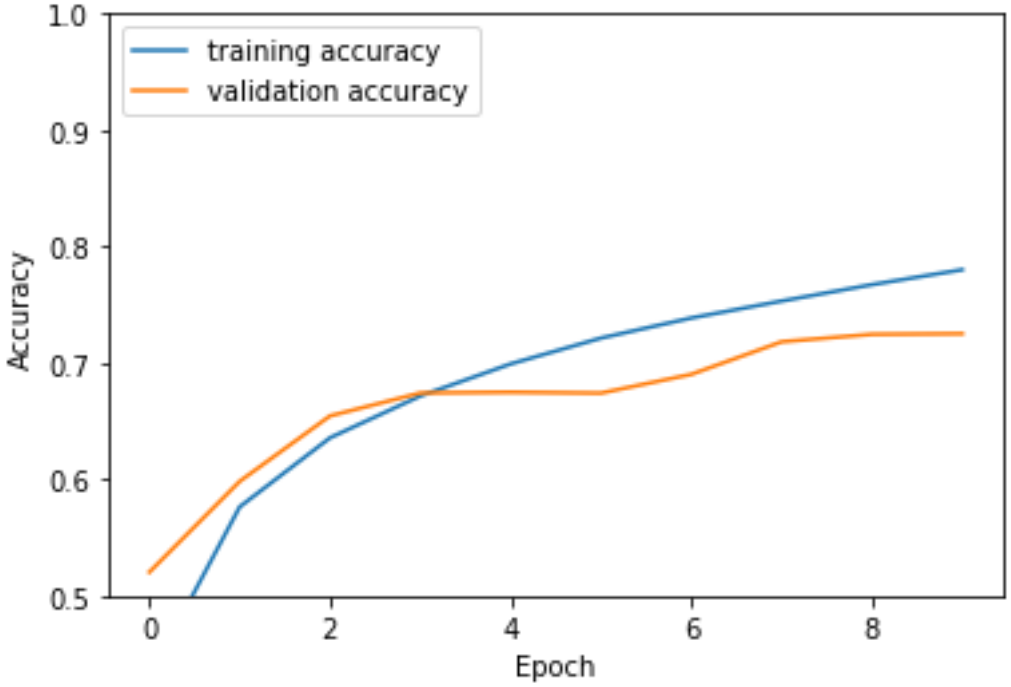
د:



شکل۱-۱۲- مقایسه دقت روی داده های آموزش و ارزیابی به ازای هر تکرار با سه لایه مخفی و روش بهینه سازی ADAM و تابع فعالساز Tanh

```
accuracy on test data: 0.7081999778747559
loss on test data: 0.9052157402038574
accuracy on train data: 0.8607800006866455
loss on train data: 0.41380277276039124
```

شکل۱-۱۳- دقت و خطای نهایی روی داده های تست و آموزش با سه لایه مخفی و روش بهینه سازی ADAM و تابع فعالساز Tanh



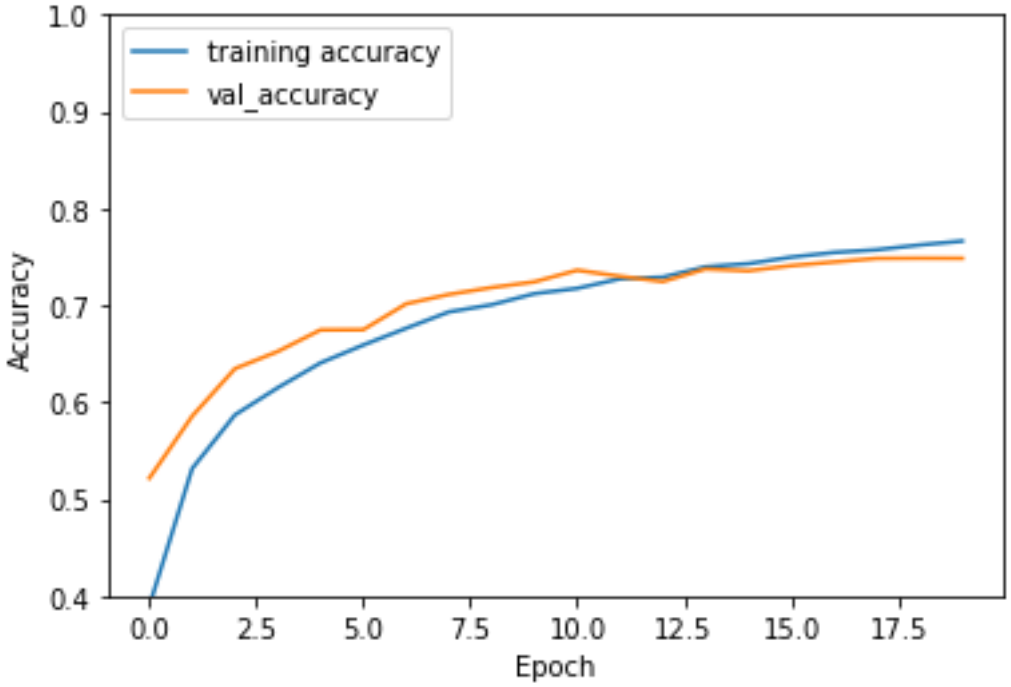
شکل۱-۱۴- مقایسه دقت روی داده های آموزش و ارزیابی به ازای هر تکرار با سه لایه مخفی و روش بهینه سازی ADAM و تابع فعالساز ReLU

```
accuracy on test data: 0.7226999998092651
loss on test data: 0.8100827932357788
accuracy on train data: 0.8049799799919128
loss on train data: 0.5721380114555359
```

شکل۱-۱۵- دقت و خطای نهایی روی داده های تست و آموزش با سه لایه مخفی و روش بهینه سازی ADAM و تابع فعالساز ReLU

با مقایسه نتایج نتیجه می گیریم روش بهینه سازی ADAM در ترکیب با تابع فعالساز ReLU دقت را تا حد چشمگیری افزایش می دهد اما این روش بهینه سازی زیاد تغییری در دقت مدل با تابع فعالساز Tanh ایجاد نمی کند.

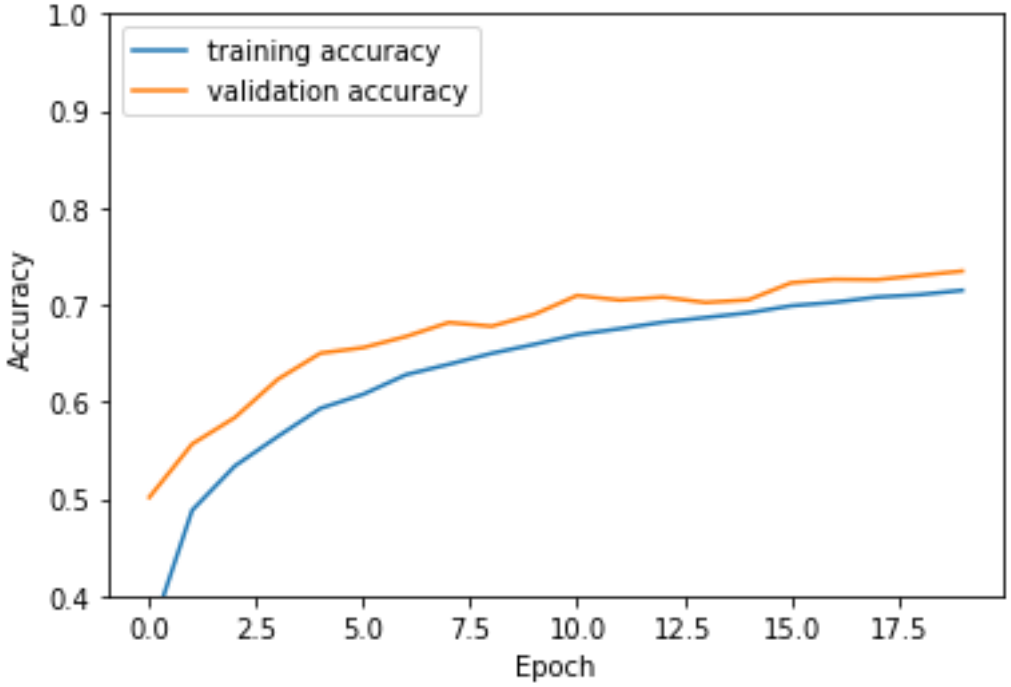
:۵



شکل۱-۱۶- مقایسه دقت روی داده های آموزش و ارزیابی به ازای هر تکرار با سه لایه مخفی و روش بهینه سازی ADAM و تابع فعالساز ReLU همرا با Dropout 10%

```
accuracy on test data: 0.7484999895095825
loss on test data: 0.7345209121704102
accuracy on train data: 0.8431199789047241
loss on train data: 0.4618484377861023
```

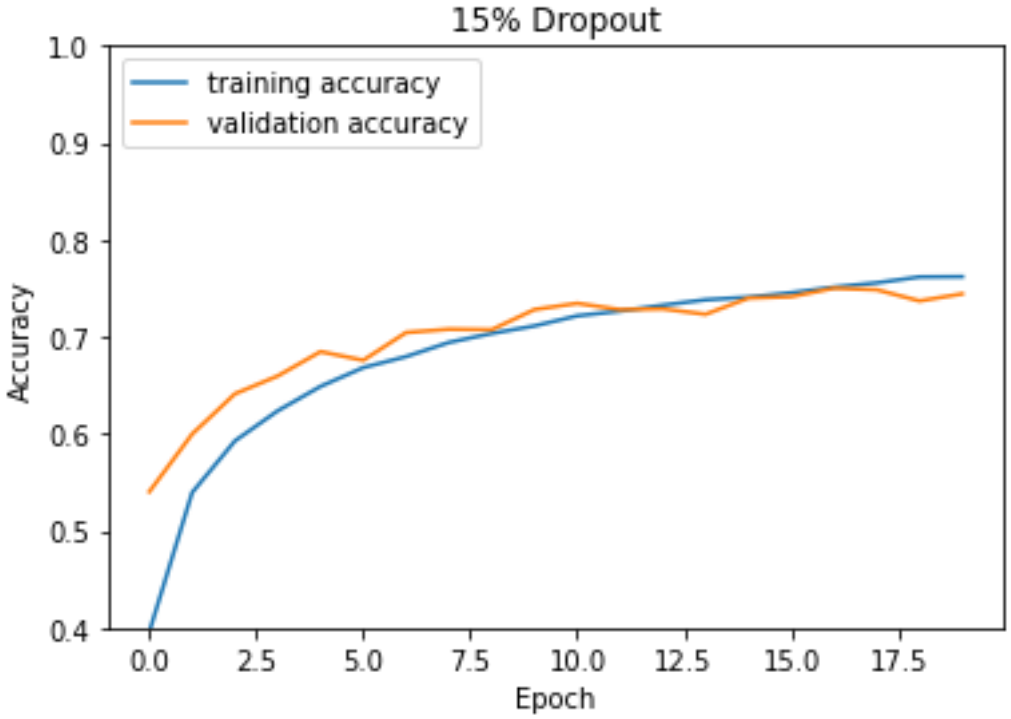
شکل۱-۱۷- دقت و خطای نهایی روی داده های تست و آموزش با سه لایه مخفی و روش بهینه سازی ADAM و تابع فعالساز ReLU همرا با Dropout 10%



شکل۱-۱۸- مقایسه دقت روی داده های آموزش و ارزیابی به ازای هر تکرار با سه لایه مخفی و روش بهینه سازی ADAM و تابع فعالساز ReLU همرا با Dropout 20%

```
accuracy on test data: 0.7347000241279602
loss on test data: 0.7561365365982056
accuracy on train data: 0.8015400171279907
loss on train data: 0.5701342821121216
```

شکل۱-۱۹- دقت و خطای نهایی روی داده های تست و آموزش با سه لایه مخفی و روش بهینه سازی ADAM و تابع فعالساز ReLU همرا با Dropout 20%



شکل۱-۲۰- مقایسه دقت روی داده های آموزش و ارزیابی به ازای هر تکرار با سه لایه مخفی و روش بهینه سازی ADAM و تابع فعالساز ReLU همرا با Dropout 15%

```
accuracy on test data: 0.7443000078201294
loss on test data: 0.7267612814903259
accuracy on train data: 0.8412799835205078
loss on train data: 0.46946483850479126
```

شکل۱-۲۱- دقت و خطای نهایی روی داده های تست و آموزش با سه لایه مخفی و روش بهینه سازی ADAM و تابع فعالساز ReLU همرا با Dropout 15%

با مقایسه نتایج بالا نتیجه می گیریم Dropout در افزایش دقت مدل ما نقش موثری داشته و درصد بهینه آن بین 10% تا 15% می باشد.

*نکته: در بعضی از موارد می بینیم که دقت روی داده های تست بیشتر از دقت روی داده های آموزش است، دلیل آن استفاده کردن از dropout می باشد، زیرا با استفاده از این روش در هنگام آموزش بعضی از گره ها غیر فعال می شوند ولی هنگام تست همه آنها فعال هستند.

سوال ۲

در این سوال در قسمت اول ابتدا به کمک قاعده پس انتشار وزن ها را بر اساس پارامتر های اولیه داده شده، برای دو تکرار به دست می آوریم. سپس در قسمت ب در مورد موارد خواسته شده در مقالات و سایت های مختلف تحقیق می کنیم، و در قسمت آخر هم یک شبکه عصبی با یک لایه را از پایه پیاده سازی می کنیم تا بتواند الگوی تابع $\sin(x+y)$ را یاد بگیرد.

الف:

$$w_1 = \begin{pmatrix} 1.44 & -0.14 & 0.4 \\ 0.24 & -1.4 & 0.43 \end{pmatrix}$$

$$w_2 = \begin{pmatrix} 1.45 & -0.54 & 0.44 \end{pmatrix}$$

$$w_3 = \begin{pmatrix} -0.4 \\ 1.4 \end{pmatrix}$$

$$b_1 = \begin{pmatrix} 0 \\ 0.41 \\ 1.4 \end{pmatrix}$$

$$b_2 = 0.1\,,\quad b_3 = 0.1$$

$$\alpha = 0.1$$

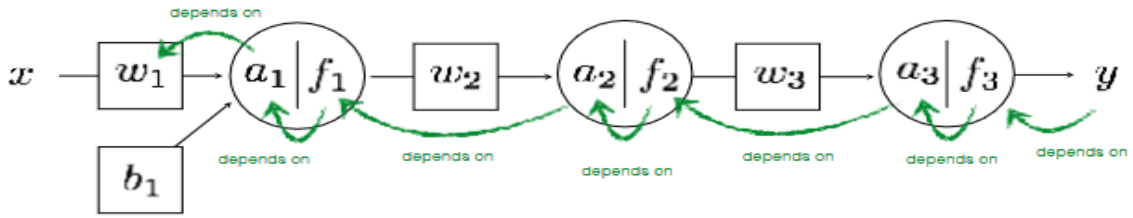
$$x = \begin{pmatrix} 2 \\ 3 \end{pmatrix}$$

$$y = 6$$

$$L = \frac{1}{2}(\hat{y} - y)^2 \rightarrow \frac{\partial L}{\partial \hat{y}} = (\hat{y} - y)$$

$$\frac{d}{dx}\tanh(x) = \frac{1}{\cosh(x)^2}$$

$$\frac{d}{dx}\mathrm{ReLu}(x) = \begin{cases} 0 & x \leq 0 \\ 1 & x > 0 \end{cases}$$



$$\frac{\partial \mathcal{L}}{\partial w_3} = \frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3}$$

$$\frac{\partial \mathcal{L}}{\partial w_2} = \frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial w_2}$$

$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial f_1}\frac{\partial f_1}{\partial a_1}\frac{\partial a_1}{\partial w_1}$$

$$\frac{\partial \mathcal{L}}{\partial b} = \frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial f_1}{\partial a_1}\frac{\partial a_1}{\partial b}$$

شکل۲-۱ – معادلات پس انتشار [1]

$$w_j^{k+1} = w_j^k - \alpha \nabla_{w_j} L$$

$$b_j^{k+1} = b_j^k - \alpha \nabla_{b_j} L$$

$$\hat{y} = a_3$$

$$a_3 = w_3 x + f_2 + b_3$$

$$f_2 = ReLu(a_2)$$

$$a_2 = w_2 f_1 + b_2$$

$$f_1 = \tanh(a_1)$$

$$a_1 = w_1 x + b_1$$

تکرار اول:

$$\hat{y} = 6.41$$

$$w_3{}^1 = w_3{}^0 - 0.1(\hat{y} - y).1.x = \begin{pmatrix} -0.48 \\ 1.28 \end{pmatrix}$$

$$w_2{}^1 = w_2{}^0 - 0.1(\hat{y} - y).1 \times 1.H(a_2).f_1 = (1.41 \quad -0.52 \quad 0.34)$$

تابع H در رابطه فوق تابع *Heaviside* می باشد.

[1] Backpropagation

$$w_1{}^1 = w_1{}^0 - 0.1(\hat{y} - y).1.1.H(a_2).w_2.\frac{1}{\cosh(a_1)^2}.x$$
$$= \begin{pmatrix} 1.43967 & -0.14006 & 0.39344 \\ 0.23951 & -1.40009 & 0.42017 \end{pmatrix}$$

$$b_3{}^1 = b_3{}^0 - 0.1(\hat{y} - y).1 \times 1 = 0.059$$
$$b_2{}^1 = b_2{}^0 - 0.1(\hat{y} - y).1 \times 1.H(a_2).1 = 0.059$$
$$b_1{}^1 = b_1{}^0 - 0.1(\hat{y} - y).1 \times 1.H(a_2).w_2.\frac{1}{\cosh(a_1)^2}.\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} -0.002 \\ 0 \\ -0.033 \end{pmatrix}$$

تکرار دوم:

$$\hat{y} = 5.51$$
$$w_3{}^2 = w_3{}^1 - 0.1(\hat{y} - y).1.x = \begin{pmatrix} -0.38 \\ 1.42 \end{pmatrix}$$
$$w_2{}^2 = w_2{}^1 - 0.1(\hat{y} - y).1 \times 1.H(a_2).f_1 = (1.46 \quad -0.54 \quad 0.46)$$

تابع H در رابطه فوق تابع *Heaviside* می باشد.

$$w_1{}^2 = w_1{}^1 - 0.1(\hat{y} - y).1.1.H(a_2).w_2.\frac{1}{\cosh(a_1)^2}.x$$
$$= \begin{pmatrix} 1.44003 & -0.14 & 0.40128 \\ 0.24005 & -1.3999 & 0.43192 \end{pmatrix}$$

$$b_3{}^2 = b_3{}^1 - 0.1(\hat{y} - y).1 \times 1 = 0.108$$
$$b_2{}^2 = b_2{}^1 - 0.1(\hat{y} - y).1 \times 1.H(a_2).1 = 0.108$$
$$b_1{}^2 = b_1{}^1 - 0.1(\hat{y} - y).1 \times 1.H(a_2).w_2.\frac{1}{\cosh(a_1)^2}.\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0.41 \\ -1.4 \end{pmatrix}$$

*نتایج مربوط به این بخش برای اطمینان بیشتر از پاسخ ها در پیوست آوردیم

ب:

ب-۱:

$$\mathrm{MAE} = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j|$$

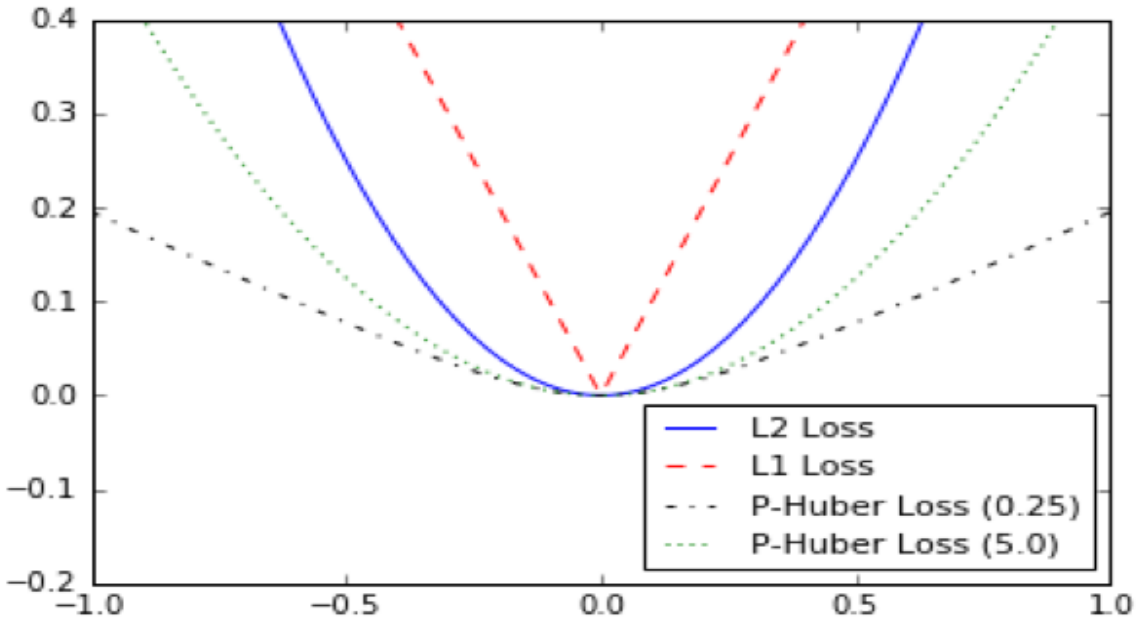
شکل ۲-۲-۱- تابع هزینه L1

$$\mathrm{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

شکل ۲-۲-۲- تابع هزینه L2

$$L_\delta(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2 & \text{for} |y - f(x)| \le \delta, \\ \delta\,|y - f(x)| - \frac{1}{2}\delta^2 & \text{otherwise.} \end{cases}$$

شکل ۲-۲-۳- تابع هزینه Huber

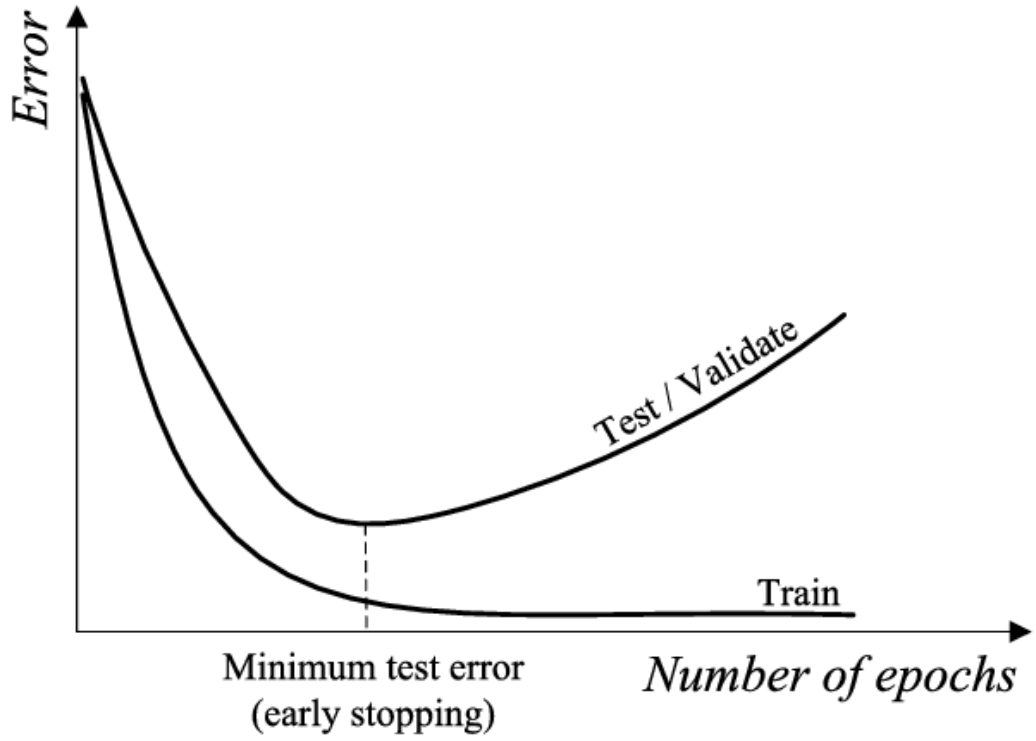


شکل ۲-۲-۴- مقایسه تابع های هزینه L1 و L2 و Huber به ازای دلتا های ۰.۲۵ و ۵

با توجه به روابط هر تابع هزینه میبینیم که تابع هزینه هوبر ترکیبی از تابع هزینه L1 و L2 می باشد و با تعیین پارامتر δ میتوانیم تابع هزینه را متناسب با نوع مسئله تعریف کنیم، تابع هزینه هوبر نزدیک به صفر شبیه به تابع هزینه L2 رفتار می کند ولی در ادامه شبیه به L1 رفتار می کند لذا معایبی که L1 و L2 به تنهایی دارند را ندارد.

ب-۲:

موقعی که مقدار خطا برای داده های آموزش و تست به هم نزدیک است به این معنی است که مدل به اندازه کافی آموزش داده شده است و کم برازش [۱] یا بیش برازش [۲] نداریم و بهتر است آموزش در این مرحله متوقف شود. شکل زیر شهودی در رابطه با خطای مدل روی داده های آموزش و تست را بر حسب میزان آموزش نشان می دهد.



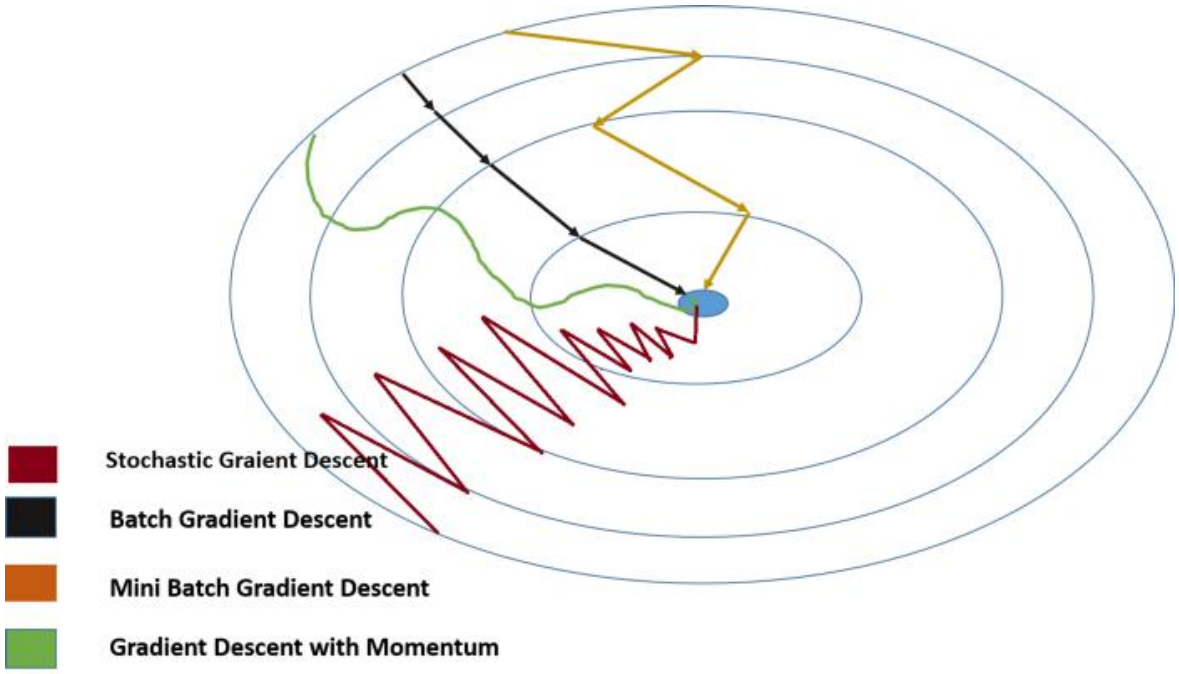
شکل ۲-۲-۵- خطای مدل روی داده های آموزش و تست بر حسب تعداد تکرار

ب-۳:

با تحقیق در منابع مختلف نتیجه میگیریم سرعت روش گرادیان نزولی تصادفی با تکانه [۳] بیشتر از روش گرادیان نزولی تصادفی است، و روش گرادیان نزولی از همه سرعت کمتری دارد، زیرا در روش گرادیان نزولی نیاز به محاسبه گرادیان بر روی کل داده ها داریم ولی در روش گرادیان نزولی تصادفی در هر مرحله

[1] Under-fitting
[2] Over-fitting
[3] Momentum

روی تعداد محدودی داده گرادیان را حساب می کنیم، همچنین در روش گرادیان نزولی تصادفی با تکانه به دلیل کاهش نویز، جهت های مناسب تری انتخاب می شود و در نتیجه سریعتر به مقدار بهینه می رسد.



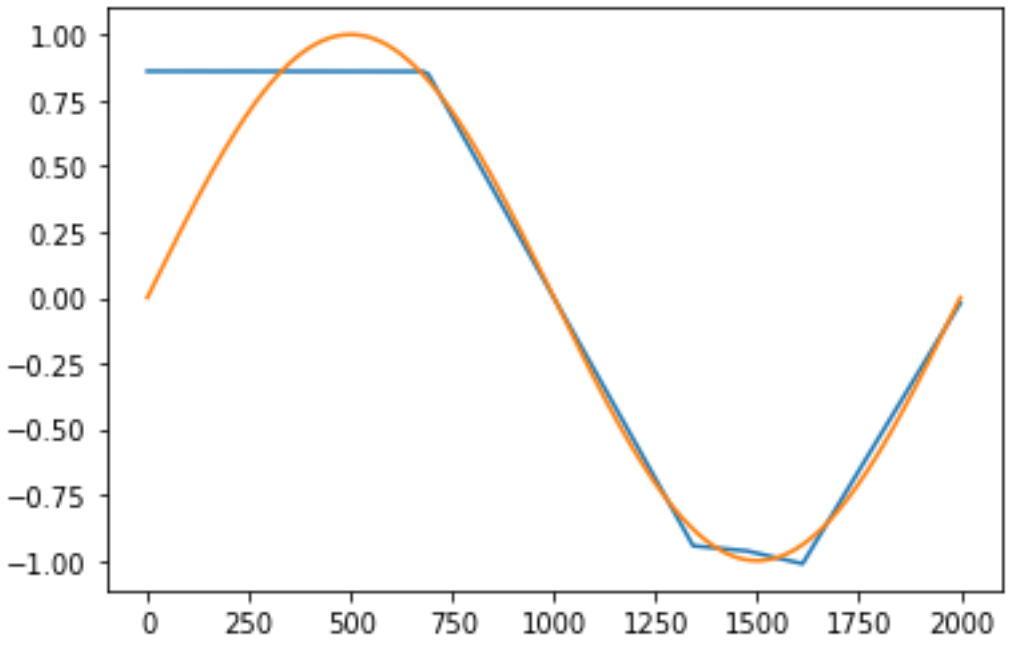
شکل ۲-۲-۶- مقایسه همگرایی سه روش

منبع:

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*.

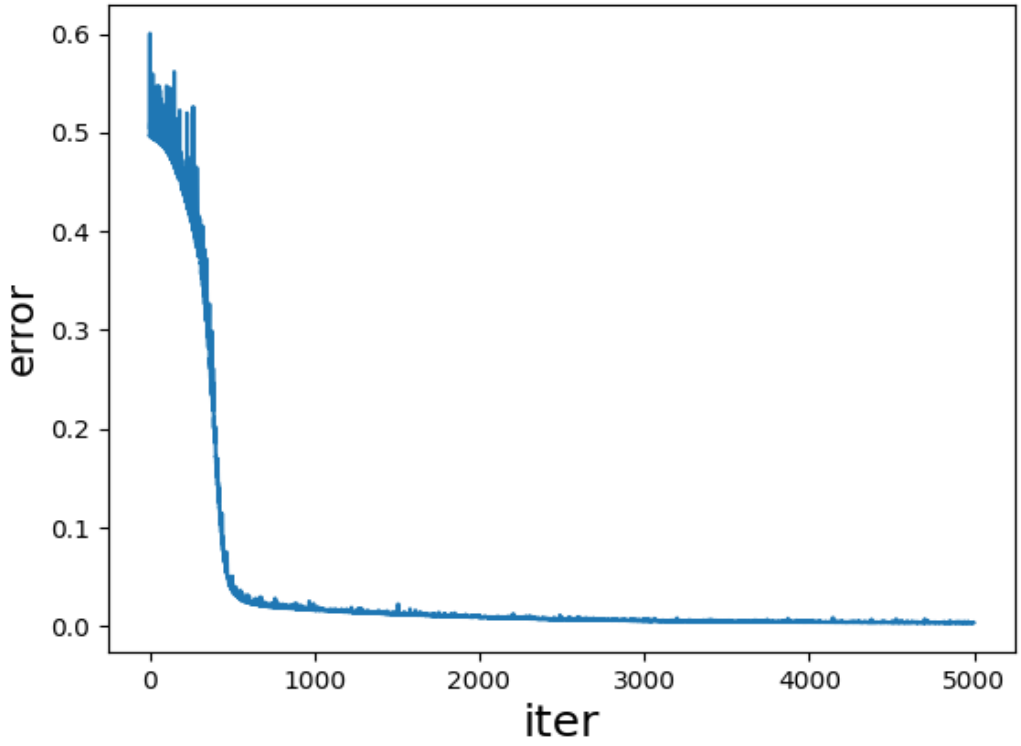
پ:

پس از پیاده سازی مدل با استفاده از تابع فعالساز ReLu نتیجه زیر حاصل می شود.

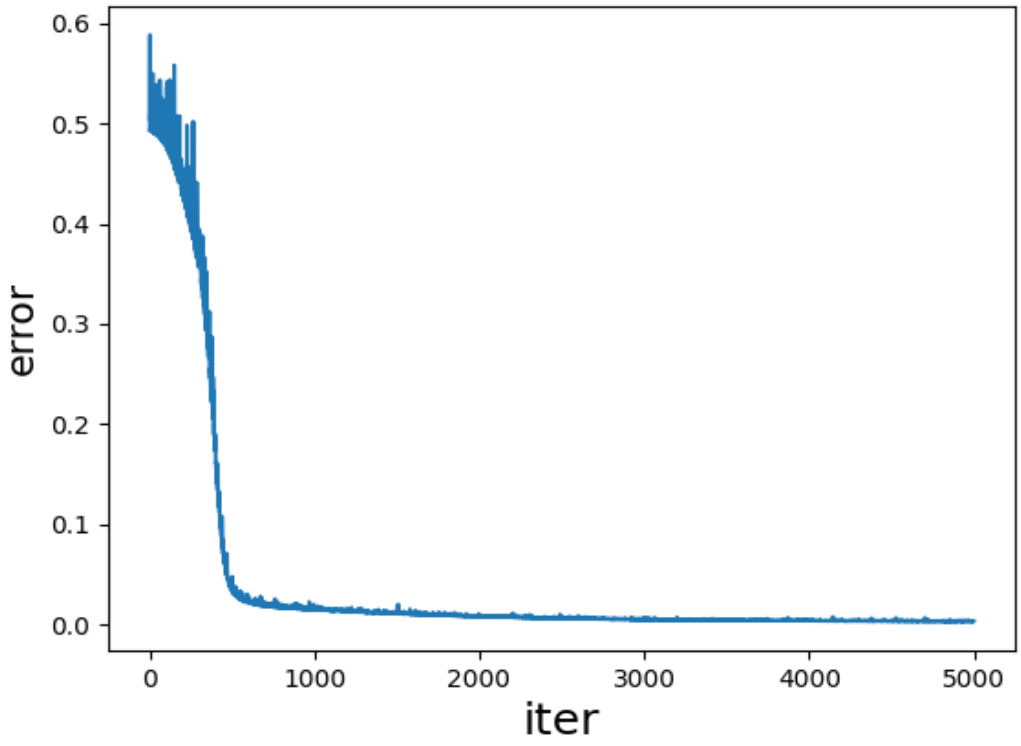


شکل ۲-۳-۱ تابع تخمین زده شده sin(x) با استفاده از تابع فعالساز ReLu در مقایسه با تابع اصلی

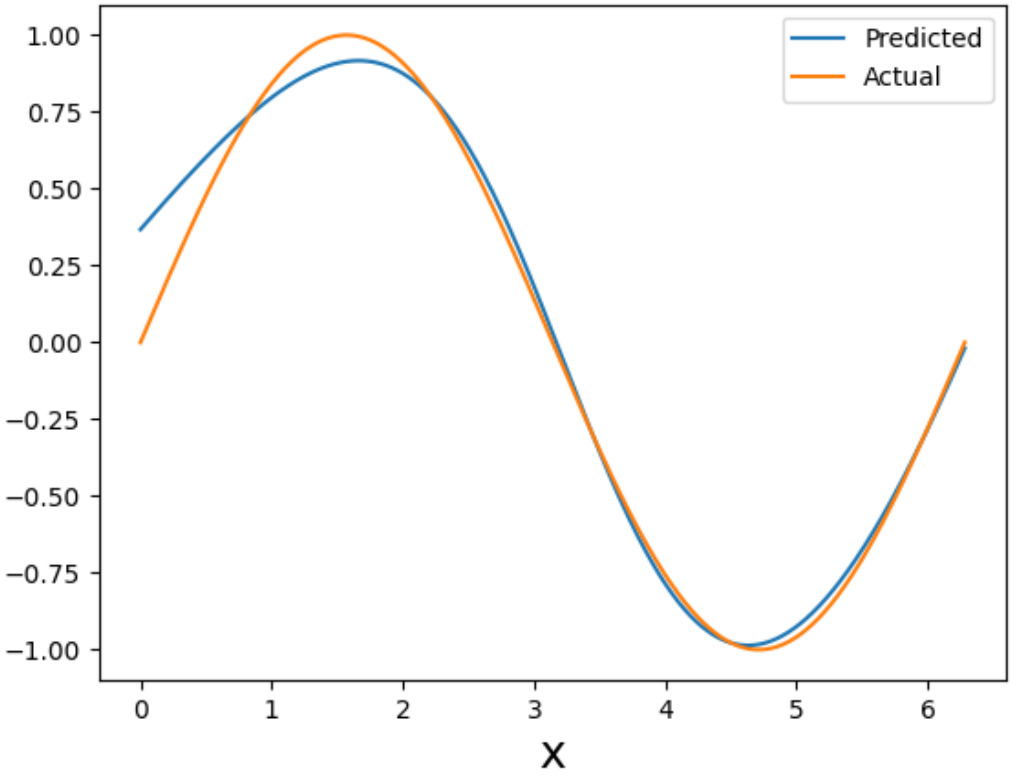
با توجه به شکل ۲-۳-۱، تصمیم می گیریم در ادامه از تابع Tanh به عنوان فعالساز استفاده کنیم.



شکل۲-۳-۲- نمودار خطای مدل بر حسب شماره تکرار برای داده ها تست



شکل۲-۳-۳- نمودار خطای مدل بر حسب شماره تکرار برای داده های آموزش



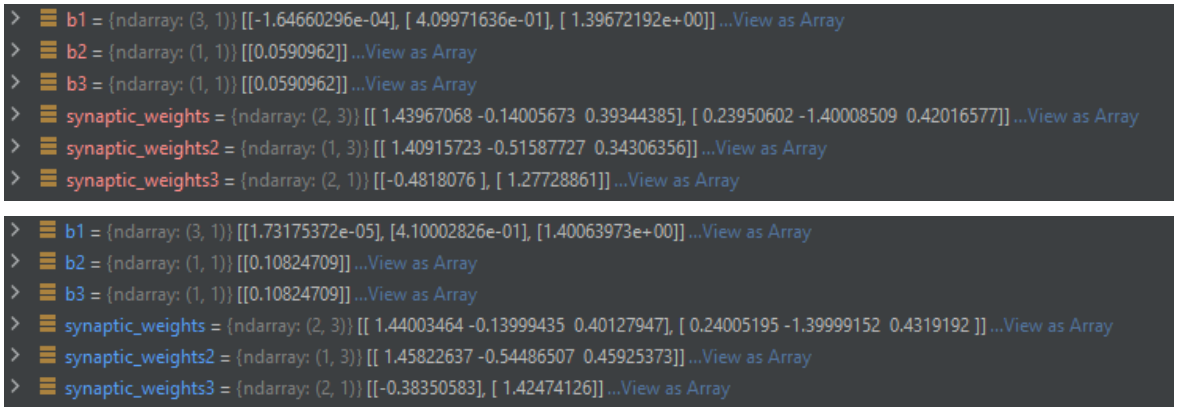
شکل ۲-۳-۴ تابع تخمین زده شده sin(x) با استفاده از تابع فعالساز Tanh در مقایسه با تابع اصلی

```
0.003610128752980454 MSE for validation data
0.003958287984506545 MSE for train data
0.003512591632378864 MSE for test data
```

شکل ۲-۳-۵ میانگین خطای حداقل مربعات برای داده های آموزش و تست و ارزیابی

پیوست

نتایج شبیه سازی برای بدست آوردن ضرایب در قسمت الف سوال ۲



شکل ۳-۱– نتایج به روز رسانی وزن ها پس از تکرار اول و دوم سوال ۲-الف

کد سوال ۲

```
        import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

np.random.seed(644)
xx = np.random.uniform(0, 2 * np.pi, 10000)
yy = np.random.uniform(0, 2 * np.pi, 10000)

x = (xx - min(xx)) / (max(xx) - min(xx))
y = (yy - min(yy)) / (max(yy) - min(yy))
label = np.sin(xx + yy)
df = pd.DataFrame([x, y]).transpose()
train = df.sample(frac=0.8, random_state=200)  # random state is a seed
value
test = df.drop(train.index)
validation = train.sample(frac=0.2, random_state=200)
label_validation = label[validation.index]
ttrain = train.drop(validation.index)
label_train = label[ttrain.index]
label_test = label[test.index]


class MyNeuralNet():
    def __init__(self):
        # defining weights and error
        np.random.seed(200)
        self.weights = np.random.rand(2, 10)
        self.weights2 = np.random.rand(10, 1)
        self.b1 = np.random.rand(10, 1)
        self.b2 = 1
        self.erriter = np.zeros(5000)
        self.errtest = np.zeros(5000)

    def tanh(self, x):
```

```
        return np.tanh(x)

    def tanh_derivative(self, x):
        return (1 / np.cosh(x) ** 2)

    def train(self, train, training_out, test, test_out,
training_iterations=5000):
        # training the model and recording errors in each iteration
        for i in range(training_iterations):
            for iteration in np.random.randint(len(train), size=100):
                training_data = train[iteration]
                training_labels = training_out[iteration]
                outputs = self.predict(training_data)
                output = outputs[1]
                output1 = outputs[0]
                error = training_labels - output
                weight2_update = np.dot(output1.T, error)
                weight1_update = np.dot(training_data.reshape(2, 1),
                                        error *
self.tanh_derivative(np.dot(training_data,

self.weights) + self.b1.T) * self.weights2.T)

                b2_update = error
                b1_update = error *
self.tanh_derivative(np.dot(training_data, self.weights) + self.b1.T
                                                                ) *
self.weights2.T

                self.weights2 = self.weights2 + 0.01 * weight2_update
                self.weights = self.weights + 0.01 * weight1_update
                self.b1 = self.b1 + 0.01 * b1_update.T
                self.b2 = self.b2 + 0.01 * b2_update
            # calculate MSE for train and test data and save them in
array
            predicted_train = neural_network.predict(train)[1]
            self.erriter[i] = ((predicted_train.T - training_out) **
2).mean()
            predicted = neural_network.predict(test)[1]
            self.errtest[i] = ((predicted.T - test_out) ** 2).mean()

    def predict(self, inputs):
        # passing the inputs via the neuron to get output
        inputs = inputs/1.0
        output1 = self.tanh(np.dot(inputs, self.weights) + self.b1.T)
        output = np.dot(output1, self.weights2) + self.b2
        return [output1, output]


neural_network = MyNeuralNet()
training_data = np.array(ttrain)
training_labels = np.array(label_train)
neural_network.train(training_data, training_labels, test, label_test,
5000)
plt.plot(neural_network.errtest)
plt.xlabel('iter', fontsize=18)
plt.ylabel('error', fontsize=16)
plt.show()
plt.plot(neural_network.erriter)
plt.xlabel('iter', fontsize=18)
```

```
plt.ylabel('error', fontsize=16)
plt.show()
val_pred = neural_network.predict(validation)[1]
val_error = ((val_pred.T - label_validation) ** 2).mean()
print(val_error, "MSE for validation data")
print(neural_network.erriter[4999], "MSE for train data")
print(neural_network.errtest[4999], "MSE for test data")

predsin = []
xtest = np.linspace(0, 2 * np.pi, 2000)
xtest_norm = (xtest - min(xtest)) / (max(xtest) - min(xtest))
ytest = np.zeros(2000)
data = np.array([xtest_norm, ytest])
predsin = neural_network.predict(data.T)[1]
predsin = np.array(predsin)
plt.plot(xtest, predsin, label="Predicted")
plt.plot(xtest, np.sin(xtest), label="Actual")
plt.legend(loc="upper right")
plt.xlabel('x', fontsize=18)
plt.show()
```

کد سوال ۱

```
# -*- coding: utf-8 -*-
"""Copy of cnn.ipynb

Automatically generated by Colaboratory.

Original file is located at

https://colab.research.google.com/drive/1CGVC86TsvvNscMm0j9eRwM0l_a8AcTvD

# **PART A**
"""

import tensorflow as tf
from keras import layers, models
import ssl
ssl._create_default_https_context = ssl._create_unverified_context
import numpy as np
import matplotlib.pyplot as plt
from keras.datasets import cifar10
from tensorflow.keras.optimizers import SGD

# load data
(train_data, train_label), (test_data, test_label) = cifar10.load_data()
samples = [29, 32, 24, 21, 28, 27, 25, 37, 62, 67]
# plot image of each class
plt.figure(0)
for i in range(5):
    for j in range(2):
        plt.subplot2grid((5, 2), (i, j))
        plt.imshow(train_data[samples[2 * i + j]])
plt.show()

# normalize data
train_data = train_data / 255.0
```

```
test_data = test_data / 255.0

# define our model
model = models.Sequential()
model.add(layers.Conv2D(32, (2, 2), activation='relu', input_shape=(32,
32, 3),padding="same",strides=(1,1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10,activation='softmax'))
model.summary()

# training the model
opt = SGD(learning_rate=0.01, momentum=0.9)
model.compile(optimizer=opt,

loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
fit_history = model.fit(train_data, train_label, epochs=10,
                    validation_data=(test_data, test_label),
batch_size=64)

plt.plot(fit_history.history['accuracy'], label='training accuracy')
plt.plot(fit_history.history['val_accuracy'], label='validation
accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.4, 1])
plt.legend(loc='upper left')
plt.show()
# calculate loss and accuracy on test data and print it
test_loss, test_acc = model.evaluate(test_data, test_label, verbose=2)
train_loss, train_acc = model.evaluate(train_data,  train_label,
verbose=2)
print("accuracy on test data:",test_acc)
print("loss on test data:",test_loss)
print("accuracy on train data:",train_acc)
print("loss on train data:",train_loss)

"""# **PART B**"""

# define our model
model = models.Sequential()
model.add(layers.Conv2D(32, (2, 2), activation='relu', input_shape=(32,
32, 3),padding="same",strides=(1,1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (2, 2), activation='relu', input_shape=(32,
32, 3),padding="same",strides=(1,1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10,activation='softmax'))
model.summary()

# training the model
opt = SGD(learning_rate=0.01, momentum=0.9)
model.compile(optimizer=opt,

loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

```
fit_history = model.fit(train_data, train_label, epochs=10,
                    validation_data=(test_data, test_label),
batch_size=64)

plt.plot(fit_history.history['accuracy'], label='training accuracy')
plt.plot(fit_history.history['val_accuracy'], label='validation
accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('2 hidden layers')
plt.ylim([0.4, 1])
plt.legend(loc='upper left')
plt.show()
# calculate loss and accuracy on test data and print it
test_loss, test_acc = model.evaluate(test_data, test_label, verbose=2)
train_loss, train_acc = model.evaluate(train_data,  train_label,
verbose=2)
print("accuracy on test data:",test_acc)
print("loss on test data:",test_loss)
print("accuracy on train data:",train_acc)
print("loss on train data:",train_loss)

# define our model
model = models.Sequential()
model.add(layers.Conv2D(32, (2, 2), activation='relu', input_shape=(32,
32, 3),padding="same",strides=(1,1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (2, 2), activation='relu', input_shape=(32,
32, 3),padding="same",strides=(1,1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (2, 2), activation='relu', input_shape=(32,
32, 3),padding="same",strides=(1,1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10,activation='softmax'))
model.summary()

# training the model
opt = SGD(learning_rate=0.01, momentum=0.9)
model.compile(optimizer=opt,

loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
fit_history = model.fit(train_data, train_label, epochs=10,
                    validation_data=(test_data, test_label),
batch_size=64)

plt.plot(fit_history.history['accuracy'], label='training accuracy')
plt.plot(fit_history.history['val_accuracy'], label='validation
accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('3 hidden layers')
plt.ylim([0.4, 1])
plt.legend(loc='upper left')
plt.show()
# calculate loss and accuracy on test data and print it
test_loss, test_acc = model.evaluate(test_data, test_label, verbose=2)
train_loss, train_acc = model.evaluate(train_data,  train_label,
```

```
verbose=2)
print("accuracy on test data:",test_acc)
print("loss on test data:",test_loss)
print("accuracy on train data:",train_acc)
print("loss on train data:",train_loss)

# define our model
model = models.Sequential()
model.add(layers.Conv2D(32, (2, 2), activation='relu', input_shape=(32,
32, 3),padding="same",strides=(1,1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (2, 2), activation='relu', input_shape=(32,
32, 3),padding="same",strides=(1,1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (2, 2), activation='relu', input_shape=(32,
32, 3),padding="same",strides=(1,1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(256, (2, 2), activation='relu', input_shape=(32,
32, 3),padding="same",strides=(1,1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10,activation='softmax'))
model.summary()

# training the model
opt = SGD(learning_rate=0.01, momentum=0.9)
model.compile(optimizer=opt,

loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
fit_history = model.fit(train_data, train_label, epochs=10,
                    validation_data=(test_data, test_label),
batch_size=64)

plt.plot(fit_history.history['accuracy'], label='training accuracy')
plt.plot(fit_history.history['val_accuracy'], label='validation
accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('4 hidden layers')
plt.ylim([0.4, 1])
plt.legend(loc='upper left')
plt.show()
# calculate loss and accuracy on test data and print it
test_loss, test_acc = model.evaluate(test_data, test_label, verbose=2)
train_loss, train_acc = model.evaluate(train_data,  train_label,
verbose=2)
print("accuracy on test data:",test_acc)
print("loss on test data:",test_loss)
print("accuracy on train data:",train_acc)
print("loss on train data:",train_loss)

"""# **PART C**"""

# define our model
model = models.Sequential()
model.add(layers.Conv2D(32, (2, 2), activation='tanh', input_shape=(32,
32, 3),padding="same",strides=(1,1)))
model.add(layers.MaxPooling2D((2, 2)))
```

```
model.add(layers.Conv2D(64, (2, 2), activation='tanh', input_shape=(32,
32, 3),padding="same",strides=(1,1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (2, 2), activation='tanh', input_shape=(32,
32, 3),padding="same",strides=(1,1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='tanh'))
model.add(layers.Dense(10,activation='softmax'))
model.summary()

# training the model
opt = SGD(learning_rate=0.01, momentum=0.9)
model.compile(optimizer=opt,

loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
fit_history = model.fit(train_data, train_label, epochs=10,
                    validation_data=(test_data, test_label),
batch_size=64)

plt.plot(fit_history.history['accuracy'], label='training accuracy')
plt.plot(fit_history.history['val_accuracy'], label='validation
accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('ReLU activation')
plt.ylim([0.4, 1])
plt.legend(loc='upper left')
plt.show()
# calculate loss and accuracy on test data and print it
test_loss, test_acc = model.evaluate(test_data, test_label, verbose=2)
train_loss, train_acc = model.evaluate(train_data,  train_label,
verbose=2)
print("accuracy on test data:",test_acc)
print("loss on test data:",test_loss)
print("accuracy on train data:",train_acc)
print("loss on train data:",train_loss)

"""# **PART D**"""

# define our model
model = models.Sequential()
model.add(layers.Conv2D(32, (2, 2), activation='tanh', input_shape=(32,
32, 3),padding="same",strides=(1,1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (2, 2), activation='tanh', input_shape=(32,
32, 3),padding="same",strides=(1,1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (2, 2), activation='tanh', input_shape=(32,
32, 3),padding="same",strides=(1,1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='tanh'))
model.add(layers.Dense(10,activation='softmax'))
model.summary()

# training the model
model.compile(optimizer="adam",
```

```
loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
fit_history = model.fit(train_data, train_label, epochs=10,
                    validation_data=(test_data, test_label),
batch_size=64)

plt.plot(fit_history.history['accuracy'], label='training accuracy')
plt.plot(fit_history.history['val_accuracy'], label='val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Tanh activation & ADAM')
plt.ylim([0.4, 1])
plt.legend(loc='upper left')
plt.show()
# calculate loss and accuracy on test data and print it
test_loss, test_acc = model.evaluate(test_data, test_label, verbose=2)
train_loss, train_acc = model.evaluate(train_data,  train_label,
verbose=2)
print("accuracy on test data:",test_acc)
print("loss on test data:",test_loss)
print("accuracy on train data:",train_acc)
print("loss on train data:",train_loss)

# define our model
model = models.Sequential()
model.add(layers.Conv2D(32, (2, 2), activation='relu', input_shape=(32,
32, 3),padding="same",strides=(1,1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (2, 2), activation='relu', input_shape=(32,
32, 3),padding="same",strides=(1,1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (2, 2), activation='relu', input_shape=(32,
32, 3),padding="same",strides=(1,1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10,activation='softmax'))
model.summary()

# training the model
model.compile(optimizer="adam",

loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
fit_history = model.fit(train_data, train_label, epochs=10,
                    validation_data=(test_data, test_label),
batch_size=64)

plt.plot(fit_history.history['accuracy'], label='training accuracy')
plt.plot(fit_history.history['val_accuracy'], label='val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('ReLU activation & ADAM')
plt.ylim([0.4, 1])
plt.legend(loc='upper left')
plt.show()
# calculate loss and accuracy on test data and print it
test_loss, test_acc = model.evaluate(test_data, test_label, verbose=2)
train_loss, train_acc = model.evaluate(train_data,  train_label,
verbose=2)
```

```
print("accuracy on test data:",test_acc)
print("loss on test data:",test_loss)
print("accuracy on train data:",train_acc)
print("loss on train data:",train_loss)

"""# **PART E**"""

# define our model
model = models.Sequential()
model.add(layers.Conv2D(32, (2, 2), activation='relu', input_shape=(32,
32, 3),padding="same",strides=(1,1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.1))
model.add(layers.Conv2D(64, (2, 2),
activation='relu',padding="same",strides=(1,1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.1))
model.add(layers.Conv2D(128, (2, 2),
activation='relu',padding="same",strides=(1,1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.1))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dropout(0.1))
model.add(layers.Dense(10,activation='softmax'))
model.summary()

# training the model
opt = SGD(learning_rate=0.01, momentum=0.9)
model.compile(optimizer="adam",

loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
fit_history = model.fit(train_data, train_label, epochs=20,
                    validation_data=(test_data, test_label),
batch_size=64)

plt.plot(fit_history.history['accuracy'], label='training accuracy')
plt.plot(fit_history.history['val_accuracy'], label='validation
accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('10% Dropout')
plt.ylim([0.4, 1])
plt.legend(loc='upper left')
plt.show()
# calculate loss and accuracy on test data and print it
test_loss, test_acc = model.evaluate(test_data, test_label, verbose=2)
train_loss, train_acc = model.evaluate(train_data,  train_label,
verbose=2)
print("accuracy on test data:",test_acc)
print("loss on test data:",test_loss)
print("accuracy on train data:",train_acc)
print("loss on train data:",train_loss)

# define our model
model = models.Sequential()
model.add(layers.Conv2D(32, (2, 2), activation='relu', input_shape=(32,
32, 3),padding="same",strides=(1,1)))
model.add(layers.MaxPooling2D((2, 2)))
```

```
model.add(layers.Dropout(0.2))
model.add(layers.Conv2D(64, (2, 2),
activation='relu',padding="same",strides=(1,1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.2))
model.add(layers.Conv2D(128, (2, 2),
activation='relu',padding="same",strides=(1,1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.2))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dropout(0.2))
model.add(layers.Dense(10,activation='softmax'))
model.summary()

# training the model
model.compile(optimizer="adam",

loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
fit_history = model.fit(train_data, train_label, epochs=20,
                    validation_data=(test_data, test_label),
batch_size=64)

plt.plot(fit_history.history['accuracy'], label='training accuracy')
plt.plot(fit_history.history['val_accuracy'], label='validation
accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('20% Dropout')
plt.ylim([0.4, 1])
plt.legend(loc='upper left')
plt.show()
# calculate loss and accuracy on test data and print it
test_loss, test_acc = model.evaluate(test_data, test_label, verbose=2)
train_loss, train_acc = model.evaluate(train_data,  train_label,
verbose=2)
print("accuracy on test data:",test_acc)
print("loss on test data:",test_loss)
print("accuracy on train data:",train_acc)
print("loss on train data:",train_loss)

# define our model
model = models.Sequential()
model.add(layers.Conv2D(32, (2, 2), activation='relu', input_shape=(32,
32, 3),padding="same",strides=(1,1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.15))
model.add(layers.Conv2D(64, (2, 2),
activation='relu',padding="same",strides=(1,1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.15))
model.add(layers.Conv2D(128, (2, 2),
activation='relu',padding="same",strides=(1,1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.15))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dropout(0.15))
model.add(layers.Dense(10,activation='softmax'))
```

```
model.summary()

# training the model
model.compile(optimizer="adam",

loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
fit_history = model.fit(train_data, train_label, epochs=20,
                    validation_data=(test_data, test_label),
batch_size=64)

plt.plot(fit_history.history['accuracy'], label='training accuracy')
plt.plot(fit_history.history['val_accuracy'], label='validation
accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('15% Dropout')
plt.ylim([0.4, 1])
plt.legend(loc='upper left')
plt.show()
# calculate loss and accuracy on test data and print it
test_loss, test_acc = model.evaluate(test_data, test_label, verbose=2)
train_loss, train_acc = model.evaluate(train_data,  train_label,
verbose=2)
print("accuracy on test data:",test_acc)
print("loss on test data:",test_loss)
print("accuracy on train data:",train_acc)
print("loss on train data:",train_loss)
```