تمرین ششم یادگیری عمیق

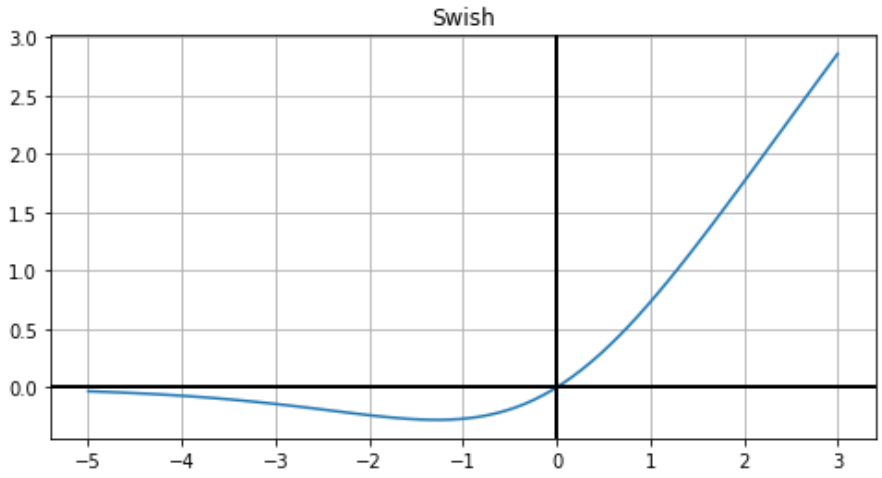
محمدعلی فراهت ۹۷۵۲۱۴۲۳

سوال ۱)

الف) تابع فعالساز Swish :

$$f(x) = x.\ \sigma(x)$$

نمودار رسم شده با matplotlib :



نمودار موجود در مقاله:

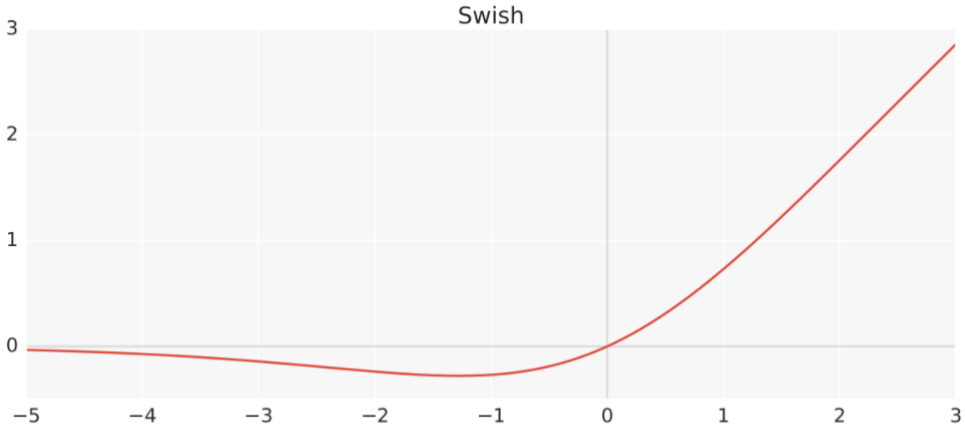
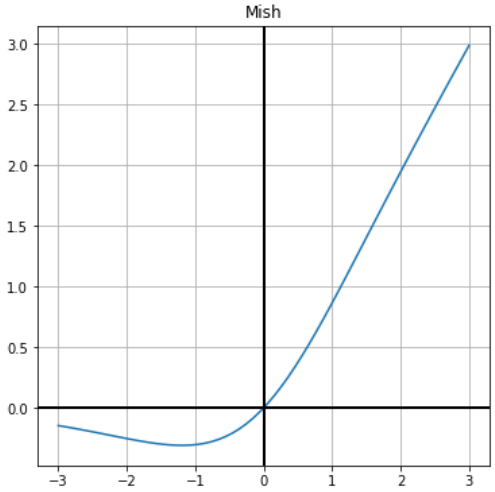


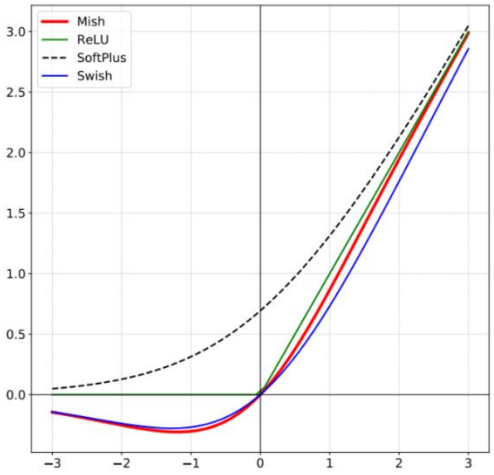
Figure 1: The Swish activation function.

تابع فعالساز Mish :

$$f(x) = x.\tanh(\text{softplus}(x)) = x.\tanh(\ln(1 + e^{x}))$$

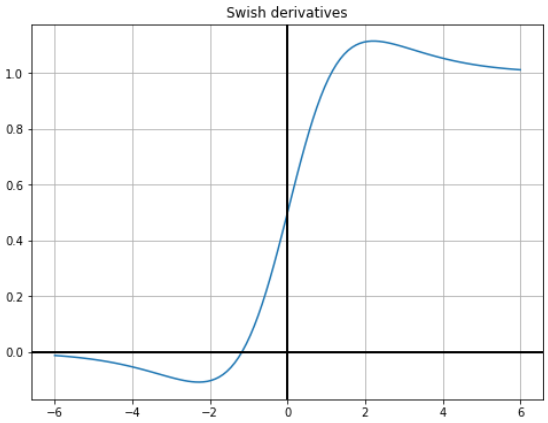
نمودار رسم شده با matplotlib (سمت چپ) و نمودار موجود در مقاله (رنگ قرمز mish است) :





ب) مشتق تابع Swish :

$$f'(x) = f(x) + \sigma(x).(1 - f(x))$$



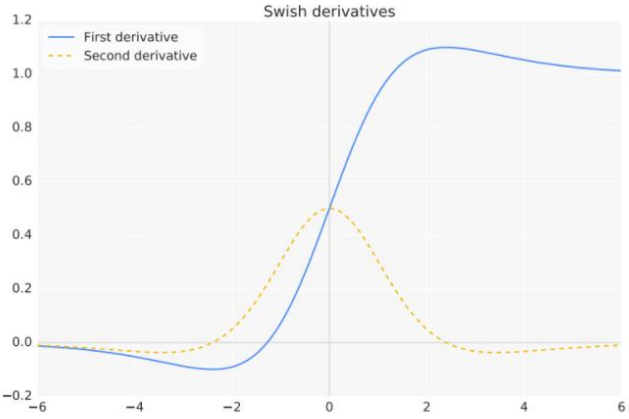
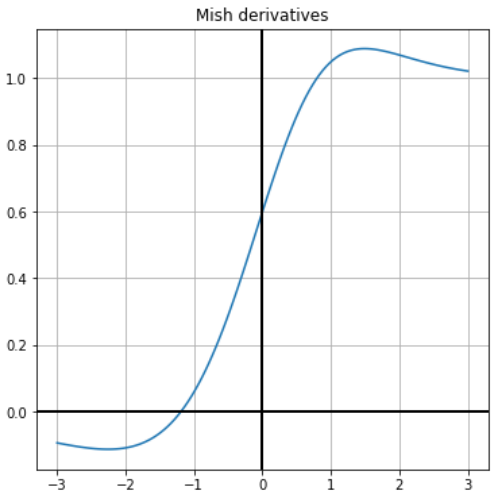
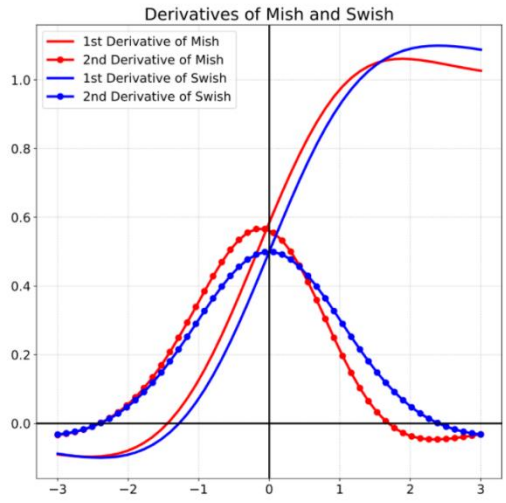


Figure 2: First and second derivatives of Swish.

مشتق تابع Mish :

$$f'(x) = \tanh(\ln(e^x+1)) + \frac{x e^x sech^2(\ln(e^x+1))}{e^x+1}$$





ت) مزایای ReLU نسبت به tanh و sigmoid :

یکی از مزایای ReUL نسبت به دو تابع دیگر، سرعت محاسبه آن است. ReLU یک تابع ساده است و به راحتی میتوان آن را محاسبه کرد. علاوه بر این ، ReLU به جایی محدود نمی‌شود و بنابراین گرادیان آن کاهش نمیابد. چون در دو تابع دیگر در x های زیاد، گرادیان به قدری کوچک میشود که عملا یادگیری را متوقف میکند.

دو تابع Swish و Mish هردو از این نظر که در سمت چپ به صفر میل می‌کنند و از راست نامحدود هستند، شبیه به ReLU هستند.
اما این دو تابع در مقادیر منفی ناگهان صفر نمیشوند و این خوب است چون باعث میشود گرادیان در مقادیر منفی هم کار کند و اثر Dying ReLU کم اثر میشود. چون تابع تغییر ناگهانی زیادی ندارد، باعث این می‌شود که وزن‌های اولیه تاثیر کمتری داشته باشند. همچنین سریع‌تر به نقطه بهینه میرسد.

ث) در این تابع یک پارامتر β برای کنترل بیشتر روی تابع استفاده میشود، اگر صفر باشد، خروجی یک تابع خطی است y = x است و اگر به سمت بینهایت برود خروجی به ReLU نزدیک میشود.

ج) دلتا در این تابع باعث می‌شود تا خروجی تغییر ناگهانی نداشته باشد و این مزیت است. چون باعث میشود بهینه سازی سریعتر و بهتر انجام شود.

سوال ۲)

الف)

اگر وزن‌های اولیه را ۰.۵ در نظر بگیریم که حد متوسط است، آنگاه مقدار خطا برای Binary cross entropy برابر خواهد بود با :

$$-\ln(0.5) = 0.7$$

و برای MSE هم به همین صورت برابر خواهد بود با:

$$(0.5)^2 = 0.25$$

بنابر این مقدارهای اولیه برای این نمودار منطقی است.

ب)

تابع مربوط به cross entropy در حال overfit شدن است، دلیل آن هم این است که آموزش با این تابع ضرر سریع تر انجام میشود، زیرا برخلاف MSE این تابع میتواند مقادیر بیشتر از ۱ در خروجی خود داشته باشد و محدود نیست.

ج)

از آنجایی که مدل سمت راست درحال overfit شدن است و با افزایش epoch مقدار loss در داده validation آن بیشتر میشود، بهتر است در نقطه مینیمم آن را نگه داشت، یعنی جایی حدود ۵۰ تا ۶۰ . ولی باید دقت کنیم که تصمیم گیری اصلی باید بر اساس نمودار accuracy باشد.
در نمودار سمت چپ، بهتر است epoch ها را حتی بیشتر از شکل هم ادامه دهیم، چون هنوز درحال یادگیری است و loss در هر دو داده درحال کاهش است.

سوال۳)

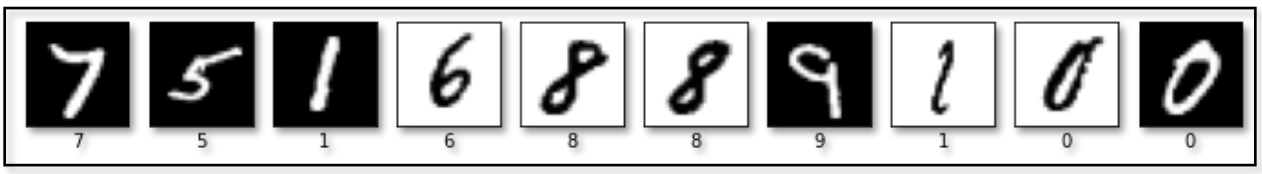
- ابتدا دیتا را با دستور mnist.load_data() دانلود می‌کنیم. سپس تعداد داده آموزش و تست را دوبرابر میکنیم، یعنی دیتاهای برعکس را هم به مجموعه خود اضافه میکنیم و آنهارا با هم مخلوط می‌کنیم و اندازه داده ها را چاپ می‌کنیم:

```
(x_train, y_train) , (x_test, y_test) = mnist.load_data()

x_train_new = np.concatenate((x_train, 255 - x_train))
y_train_new = np.concatenate((y_train, y_train))
x_test_new = np.concatenate((x_test, 255 - x_test))
y_test_new = np.concatenate((y_test, y_test))

x_train_new, y_train_new = shuffle(x_train_new, y_train_new)
x_test_new, y_test_new = shuffle(x_test_new, y_test_new)

print("shape of input data for training (x_train) : ", x_train_new.shape)
print("shape of labels for training (y_train) : ", y_train_new.shape)
print("shape of input data for testing (x_test) : ", x_test_new.shape)
print("shape of label for testing (y_test) : ", y_test_new.shape)

Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
11501568/11490434 [==============================] - 0s 0us/step
shape of input data for training (x_train) :  (120000, 28, 28)
shape of labels for training (y_train) :  (120000,)
shape of input data for testing (x_test) :  (20000, 28, 28)
shape of label for testing (y_test) :  (20000,)
```

- برای اطمینان از درستی کار، ۱۰ تا نمونه اول را با کد زیر چاپ می‌کنیم:

```
plt.figure(figsize=(12, 3))
for i in range(1, 11):
    plt.subplot(1, 10, i)
    plt.xlabel(y_train_new[i])
    plt.imshow(x_train_new[i], cmap="gray")
    plt.xticks([])
    plt.yticks([])

plt.show()
```

و خروجی ها هم به صورت زیر است:



- سپس برای آن که بتوانیم از دیتا استفاده کنیم، آن را reshape میکنیم تا به صورت زیر در بیاید:

```
x_train_new = np.reshape(x_train_new, (x_train_new.shape[0], x_train_new.shape[1], x_train_new.shape[2], 1))
x_test_new = np.reshape(x_test_new, (x_test_new.shape[0], x_test_new.shape[1], x_test_new.shape[2], 1))

print("reshaped x_train: ", x_train_new.shape)
print("reshaped x_test: ", x_test_new.shape)
```

```
reshaped x_train:  (120000, 28, 28, 1)
reshaped x_test:  (20000, 28, 28, 1)
```

- سپس برای آموزش بهتر، هر دیتا را بجای اینکه ۰ تا ۲۵۵ باشد، بین ۰ تا ۱ می‌آوریم و برای برچسب‌هایشان هم، خروجی را categorical می‌کنیم:

```
x_train_new = x_train_new / 255
x_test_new = x_test_new / 255

y_train_new = to_categorical(y_train_new, num_classes=10)
y_test_new = to_categorical(y_test_new, num_classes=10)
```

- در قسمت بعد تابعی درست می‌کنیم که مقدار alpha را می‌گیرد و مدلی مانند مدل خواسته شده، با ۴ لایه را برمی‌گرداند که در دو لایه کانولوشن اول آن، از پارامتر alpha استفاده شده:

```
def create_model(alpha):
    model = Sequential()
    model.add(Conv2D(8, 7, activation=LeakyReLU(alpha=alpha)))
    model.add(Conv2D(8, 5, activation=LeakyReLU(alpha=alpha)))
    model.add(Flatten())
    model.add(Dense(10, activation="softmax"))
    return model

model = create_model(0.85)
model.build(x_train_new.shape)
print(model.summary())
```

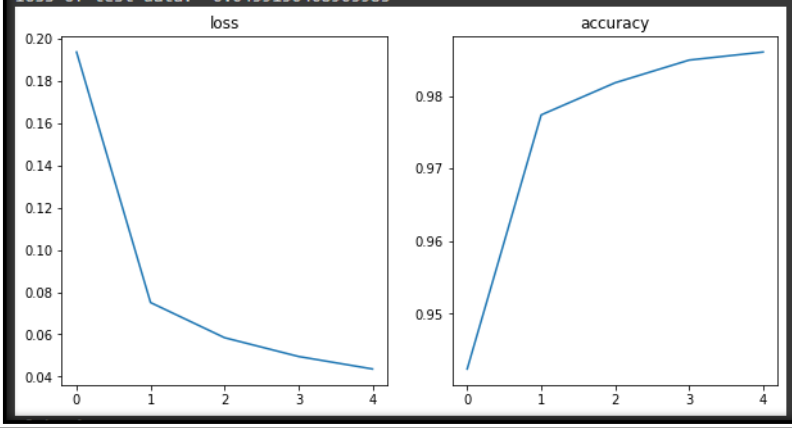
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (120000, 22, 22, 8)       400
_________________________________________________________________
conv2d_1 (Conv2D)            (120000, 18, 18, 8)       1608
_________________________________________________________________
flatten (Flatten)            (120000, 2592)            0
_________________________________________________________________
dense (Dense)                (120000, 10)              25930
=================================================================
Total params: 27,938
Trainable params: 27,938
Non-trainable params: 0
_________________________________________________________________
None
```

- حالا با ۵ مقدار برای alpha ، مدلی می‌سازیم و آن را آموزش می‌دهیم. خروجی ها و نمودارهای loss و accuracy را برای هر یک می‌بینیم:

```python
alphas = [-1, -0.5, 0, 0.5, 1]
for alpha in alphas:
    print("alpha is: ", alpha)
    model = create_model(alpha)
    model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])
    h = model.fit(x_train_new, y_train_new, epochs=5, batch_size=128)

    loss, acc = model.evaluate(x_test_new, y_test_new)
    print("accuracy of test data: ", acc)
    print("loss of test data: ", loss)

    plt.figure(figsize=(10, 5))

    plt.subplot(1, 2, 1)
    plt.plot(h.history['loss'])
    plt.title('loss')

    plt.subplot(1, 2, 2)
    plt.plot(h.history['accuracy'])
    plt.title('accuracy')
    plt.show()
```

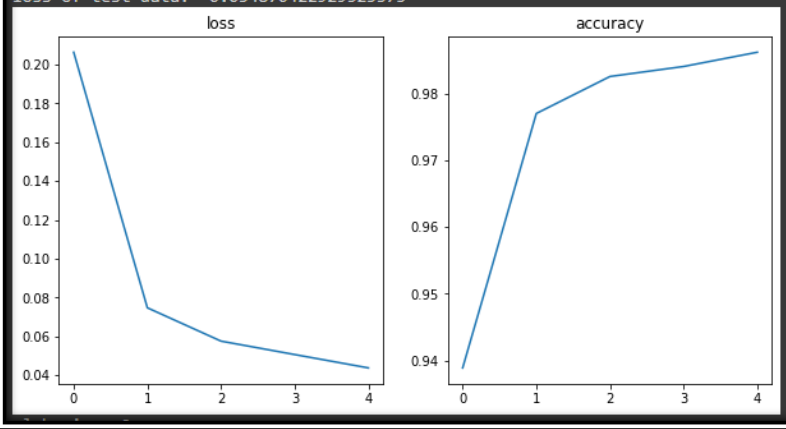
alpha = -1 :

```
alpha is:  -1
Epoch 1/5
938/938 [==============================] - 7s 7ms/step - loss: 0.1936 - accuracy: 0.9424
Epoch 2/5
938/938 [==============================] - 6s 7ms/step - loss: 0.0751 - accuracy: 0.9774
Epoch 3/5
938/938 [==============================] - 6s 7ms/step - loss: 0.0584 - accuracy: 0.9818
Epoch 4/5
938/938 [==============================] - 6s 7ms/step - loss: 0.0495 - accuracy: 0.9850
Epoch 5/5
938/938 [==============================] - 6s 7ms/step - loss: 0.0436 - accuracy: 0.9861
625/625 [==============================] - 2s 3ms/step - loss: 0.0459 - accuracy: 0.9859
accuracy of test data:  0.9858999848365784
loss of test data:  0.0459156408905983
```



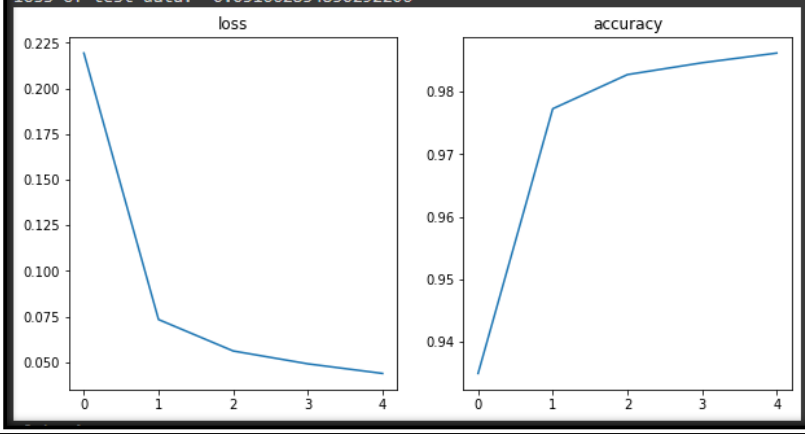
: alpha = -0.5

```
alpha is:  -0.5
Epoch 1/5
938/938 [==============================] - 7s 7ms/step - loss: 0.2064 - accuracy: 0.9389
Epoch 2/5
938/938 [==============================] - 6s 7ms/step - loss: 0.0747 - accuracy: 0.9770
Epoch 3/5
938/938 [==============================] - 6s 7ms/step - loss: 0.0575 - accuracy: 0.9825
Epoch 4/5
938/938 [==============================] - 6s 7ms/step - loss: 0.0506 - accuracy: 0.9840
Epoch 5/5
938/938 [==============================] - 6s 7ms/step - loss: 0.0437 - accuracy: 0.9862
625/625 [==============================] - 2s 3ms/step - loss: 0.0549 - accuracy: 0.9830
accuracy of test data:  0.9829999804496765
loss of test data:  0.054870422929525375
```



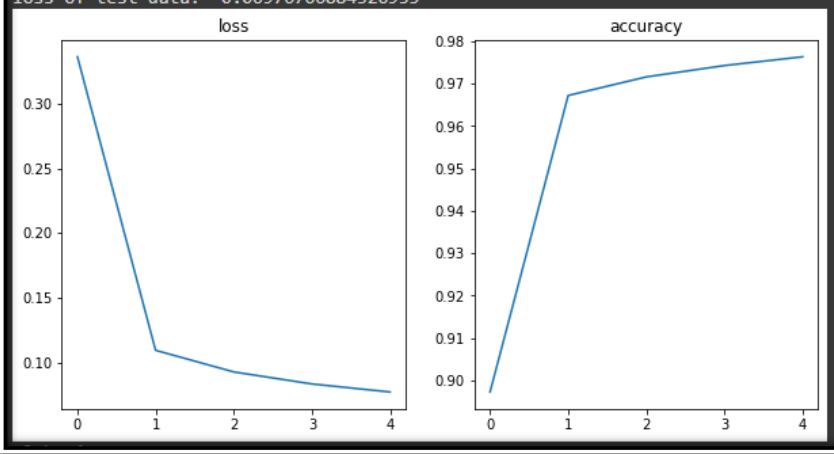
: alpha = 0

```
alpha is:  0
Epoch 1/5
938/938 [==============================] - 7s 7ms/step - loss: 0.2194 - accuracy: 0.9350
Epoch 2/5
938/938 [==============================] - 7s 7ms/step - loss: 0.0734 - accuracy: 0.9773
Epoch 3/5
938/938 [==============================] - 6s 7ms/step - loss: 0.0562 - accuracy: 0.9827
Epoch 4/5
938/938 [==============================] - 6s 7ms/step - loss: 0.0492 - accuracy: 0.9846
Epoch 5/5
938/938 [==============================] - 6s 7ms/step - loss: 0.0439 - accuracy: 0.9862
625/625 [==============================] - 2s 3ms/step - loss: 0.0517 - accuracy: 0.9837
accuracy of test data:  0.9837499856948853
loss of test data:  0.051662854850292206
```



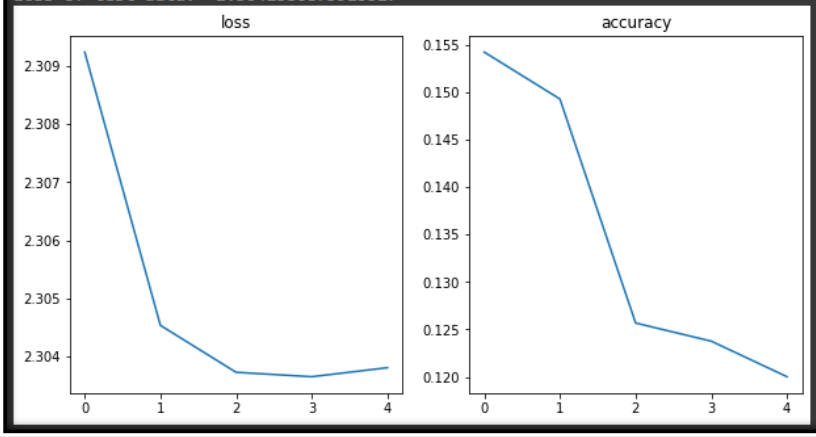
: alpha = 0.5

```
alpha is:  0.5
Epoch 1/5
938/938 [==============================] - 7s 7ms/step - loss: 0.3363 - accuracy: 0.8972
Epoch 2/5
938/938 [==============================] - 6s 7ms/step - loss: 0.1093 - accuracy: 0.9672
Epoch 3/5
938/938 [==============================] - 7s 7ms/step - loss: 0.0926 - accuracy: 0.9716
Epoch 4/5
938/938 [==============================] - 6s 7ms/step - loss: 0.0833 - accuracy: 0.9743
Epoch 5/5
938/938 [==============================] - 6s 7ms/step - loss: 0.0771 - accuracy: 0.9764
625/625 [==============================] - 2s 3ms/step - loss: 0.0698 - accuracy: 0.9781
accuracy of test data:  0.9781000018119812
loss of test data:  0.06976766884326935
```



: alpha = 1

```
alpha is:  1
Epoch 1/5
938/938 [==============================] - 7s 7ms/step - loss: 2.3092 - accuracy: 0.1542
Epoch 2/5
938/938 [==============================] - 6s 7ms/step - loss: 2.3045 - accuracy: 0.1493
Epoch 3/5
938/938 [==============================] - 6s 7ms/step - loss: 2.3037 - accuracy: 0.1257
Epoch 4/5
938/938 [==============================] - 6s 7ms/step - loss: 2.3037 - accuracy: 0.1238
Epoch 5/5
938/938 [==============================] - 6s 7ms/step - loss: 2.3038 - accuracy: 0.1200
625/625 [==============================] - 2s 3ms/step - loss: 2.3042 - accuracy: 0.1337
accuracy of test data:  0.13369999825954437
loss of test data:  2.3041586875915527
```



- اگر به نمودار آخر (alpha = 1) دقت کنید، می‌بینید که دقت به طرز عجیبی کم است و همچنین در حال کاهش است، به جای اینکه افزایش یابد. دلیل آن هم این است که در فرمول این تابع، اگر مقدار alpha برابر با ۱ باشد، تابع دیگر در جایی شکستگی ندارد و به صورت خطی عمل میکند و فرمول آن y = x خواهد بود. میدانیم که تابع خطی برای activation چندان بدرد نمی‌خورد. برای همین دقت انقدر بد است.

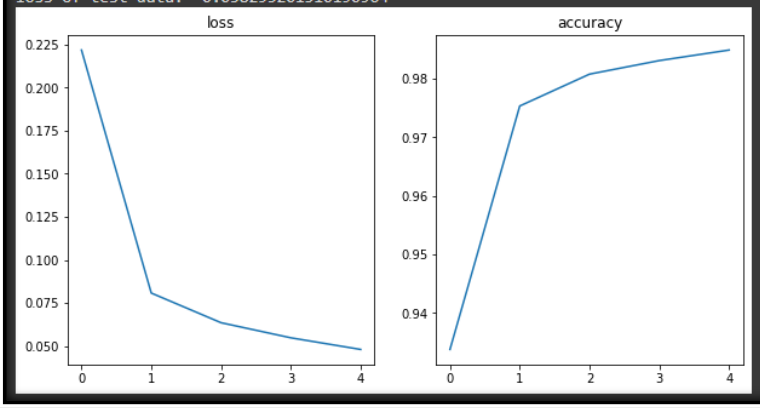
  بهترین عملکرد هم برای alpha = -1 است، چون دقت تست در آن از همه بالاتر بوده.

- حالا به جای LeakyReLU از PReLU استفاده می‌کنیم. تابع مورد نظر و کد آموزش به صورت زیر است:

```
def create_model2():
    model = Sequential()
    model.add(Conv2D(8, 7, activation=PReLU(shared_axes=[1,2])))
    model.add(Conv2D(8, 5, activation=PReLU(shared_axes=[1,2])))
    model.add(Flatten())
    model.add(Dense(10, activation="softmax"))
    return model


model2 = create_model2()
model2.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])
h = model2.fit(x_train_new, y_train_new, epochs=5, batch_size=128)

loss, acc = model2.evaluate(x_test_new, y_test_new)
print("accuracy of test data: ", acc)
print("loss of test data: ", loss)

plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.plot(h.history['loss'])
plt.title('loss')

plt.subplot(1, 2, 2)
plt.plot(h.history['accuracy'])
plt.title('accuracy')
plt.show()
```

خروجی آن هم به صورت زیر است :

```
Epoch 1/5
938/938 [==============================] - 8s 8ms/step - loss: 0.2218 - accuracy: 0.9338
Epoch 2/5
938/938 [==============================] - 7s 8ms/step - loss: 0.0807 - accuracy: 0.9754
Epoch 3/5
938/938 [==============================] - 7s 8ms/step - loss: 0.0635 - accuracy: 0.9808
Epoch 4/5
938/938 [==============================] - 7s 7ms/step - loss: 0.0548 - accuracy: 0.9831
Epoch 5/5
938/938 [==============================] - 7s 7ms/step - loss: 0.0480 - accuracy: 0.9849
625/625 [==============================] - 2s 3ms/step - loss: 0.0583 - accuracy: 0.9829
accuracy of test data:  0.9829000234603882
loss of test data:  0.058295201510190964
```



حالا پارامترهای آن را پیدا می‌کنیم تا با قسمت قبل مقایسه کنیم:

```
1 print("alpha params in first layer weights: ", model2.layers[0].get_weights()[2])
2 print("alpha params in second layer weights: ", model2.layers[1].get_weights()[2])
3

alpha params in first layer weights:  [[[-0.02322879  0.24976456  0.2727916   0.38647506  0.06843951
   -0.03745963  0.30824876 -0.06797314]]]
alpha params in second layer weights:  [[[-0.24805605 -0.21426804 -0.36600178 -0.37278825 -0.13345793
   -0.31092107  0.20969895 -0.35918495]]]
```

می‌بینیم که alpha های بهینه حدود 0.4- تا 0.4 هستند. اما در قسمت قبل چیزی که ما به آن رسیده بودیم alpha = -1 از بقیه بهتر بوده، اولا وقتی alpha = 0 بود، تفاوت زیادی در دقت وجود نداشت (حدود ۰.۰۰۲۴ تفاوت وجود داشت). دوما ممکن بود در تعداد epoch بالاتر این برتری خودش را نشان دهد.

"با تشکر فراوان"