Deep Learning
Dr. DavoodAbadi
Fall 2022
Hoorieh Sabzevari - 98412004
HW2

۱.

الف) در مدل های شبکه عصبی از چندین بهینه ساز می توان استفاده کرد. بهینه‌سازadam یک الگوریتم است که بهینه‌سازی غیرمحدب را به خوبی انجام می‌دهد. این بهینه‌ساز مزایایی مانند: پیاده سازی سریع، بهره‌ی محاسباتی و حافظه‌ی موردنیاز کمتری دارد و برای داده های پراکنده و با شیب زیاد و اهداف غیر ثابت مناسب است. این بهینه ساز معمولا به تنظیم کمی نیاز دارد. به علاوه می‌توان گفت که این بهینه‌ساز ترکیبی از adaGrad و RMSProp است پس مزایای هر دو این موارد را دارد. این الگوریتم یک میانگین نمایی از گرادیان و مشتق گرادیان محاسبه می‌کند. همچنین پارامترهای beta1 و beta2 میزان فروپاشی این میانگین‌های متحرک را کنترل می‌کنند. طبق توضیحات بالا و نمودار سوال، RMSProp و Adadelta و Adam در شرایط مشابه عملکرد خوبی دارند. مزیت اصلی AdaDelta این است که ما نیازی به تنظیم نرخ یادگیری پیش فرض نداریم اما از نظر محاسباتی گران است. در RMS-Prop نرخ یادگیری به طور خودکار تنظیم می شود و نرخ یادگیری متفاوتی را برای هر پارامتر انتخاب می کند اما یادگیری آهسته است. تصحیح bias در بهینه ساز adam باعث می‌شود که adam از RMSProp کمی بهتر باشد، به این دلیل که گرادیان‌ها پراکنده‌تر می‌شوند. به همین دلیل در حالت کلی adam بهترین گزینه است و هزینه آموزش کمتری دارد. گزینه جایگزین برای آن می‌تواند Momentum Nestrov + SGD باشد. در حالت کلی SGD به آرامی به سمت همگرایی و جواب حرکت می‌کند و نوسان زیادی دارد ولی فضای کمی نیاز دارد و از آنجا که در هر زمان فقط یک نمونه آپدیت می‌شود برای داده‌های بزرگ بهتر است اما هزینه‌ی آپدیت زیاد است. پس از آن Adagrad نوسان در خلاف جهت جواب دارد اما با سرعت بیشتری از SGD به جواب می‌رسد. نرخ یادگیری به صورت تطبیقی با تکرارها تغییر

می‌کند. همچنین قادر به آموزش داده های پراکنده است. در رده بالایی RMSProp قرار دارد که هم نوسان کمتری از مورد قبل دارد هم با زمان بهتری به سمت همگرایی حرکت می‌کند. اما دو بهینه‌ساز Momentum + SGD و SGD + Nestrov عملکرد متفاوتی دارند. به علت اینکه Momentum را در آپدیت متغیرها وارد می‌کنند زودتر به جواب می‌رسد اما در ابتدای روند نوسان نسبتا شدیدی دارند که کمتر از SGD است ولی با سرعت بالایی از رقبای خود پیشی می‌گیرند و در طول زمان نوسان کم و کمتر می‌شود. پس مقایسه کلی: RMSProp و Adagrad نوسان کمی دارند و با روند تقریبا ثابتی به سمت هدف حرکت می‌کنند زیرا Momentum در آپدیت موثر نیست و سرعت تغییر آنچنانی نمی‌کند. اما در Momentum نوسان در ابتدا زیاد است و مسیر خوبی طی نمی‌شود اما چون سرعت در جهت هدف بیشتر می‌شود در ادامه از این دو پیشی می‌گیرند. Adam نیز از آنجا که ترکیب RSMProp و Adagrad است هم نوسان کم و هم سرعت خوبی دارد. پس در داده های پراکنده استفاده از RMSProp و Adam و Adagrad پیشنهاد می‌شود. این سه موارد یکسان زیادی دارند و اغلب نتایج مشابهی دارند. هر چه پراکندگی داده بیشتر شود adam از RMSProp بهتر عمل می‌کند. Adadelta هم در ابتدا ضعیف عمل می‌کند اما در نهایت از RMSProp نتیجه بهتری را در داده های پراکنده ثبت می‌کند. با کم شدن گرادیان، Adam بهتر از RMSprop عمل خواهد کرد. (لینک)

ب) چند عامل اصلی دخیل هستند: (۱) همگرایی global (۲) همگرایی محلی، (۳) نقش روش بهینه‌سازی اساسی، (۴) نقش بازگشت چندشبکه، (۵) ویژگی های مدل بهینه سازی

تعداد دیتا هم مهم است. مثلا برای داده‌های بزرگ از SGD استفاده می‌شود. زیرا هزینه‌ی محاسباتی کمتری دارد. همچنین پراکندگی دیتا هم موثر است. مثلا RMSProp و Adam بهتر روی این نوع کار می‌کنند. برخی بهینه‌سازهای خاص بر روی داده‌هایی با ویژگی‌های پراکنده عملکرد فوق‌العاده خوبی دارند و برخی دیگر ممکن است زمانی که مدل بر روی داده‌های دیده نشده قبلی اعمال می‌شود بهتر عمل کنند. برخی از بهینه سازها با اندازه‌های دسته بزرگ بسیار خوب کار می کنند، در حالی که برخی دیگر با تعمیم ضعیف به minimum ها همگرا می‌شوند. (لینک)

منابعی که برای یک پروژه در دسترس هستند نیز بر روی انتخاب بهینه‌ساز تأثیر دارند. محدودیت‌های محاسباتی یا محدودیت‌های حافظه و همچنین چارچوب زمانی پروژه می‌تواند مجموعه انتخاب‌های ممکن را محدود کند. با نگاهی مجدد به جدول زیر، می‌توانیم نیازهای حافظه مختلف و تعداد پارامترهای قابل تنظیم برای هر بهینه‌ساز را مشاهده کنیم. (لینک)

Optimizer	State Memory [bytes]	# of Tunable Parameters	Strengths	Weaknesses
SGD	0	1	Often best generalization (after extensive training)	Prone to saddle points or local minima Sensitive to initialization and choice of the learning rate α
SGD with Momentum	$4n$	2	Accelerates in directions of steady descent Overcomes weaknesses of simple SGD	Sensitive to initialization of the learning rate α and momentum β
AdaGrad	$\sim 4n$	1	Works well on data with sparse features Automatically decays learning rate	Generalizes worse, converges to sharp minima Gradients may vanish due to aggressive scaling
RMSprop	$\sim 4n$	3	Works well on data with sparse features Built in Momentum	Generalizes worse, converges to sharp minima
Adam	$\sim 8n$	3	Works well on data with sparse features Good default settings Automatically decays learning rate α	Generalizes worse, converges to sharp minima Requires a lot of memory for the state
AdamW	$\sim 8n$	3	Improves on Adam in terms of generalization Broader basin of optimal hyperparameters	Requires a lot of memory for the state
LARS	$\sim 4n$	3	Works well on large batches (up to 32k) Counteracts vanishing and exploding gradients Built in Momentum	Computing norm of gradient for each layer can be inefficient

②

	x_0	x_1	y
Data 1	3	-1	1
Data 2	1	-2	0

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

Forwarding:
$$\begin{cases} z = w_0 x_0 + w_1 x_1 + b \\ L(y, \sigma) = (y-\sigma)^2 \end{cases}$$

$w_0 = 2$
$w_1 = 1$
$b = 2$

Backwarding:
$$\begin{cases} \frac{\partial L}{\partial L} = 1 \ , \ \frac{\partial L}{\partial \sigma} = -2(y-\sigma) \\ \frac{\partial L}{\partial z} = -2\sigma(1-\sigma)(y-\sigma) \\ \frac{\partial L}{\partial b} = -2\sigma(1-\sigma)(y-\sigma) \\ \frac{\partial L}{\partial w_0} = -2\sigma x_0(1-\sigma)(y-\sigma) \\ \frac{\partial L}{\partial w_1} = -2\sigma x_1(1-\sigma)(y-\sigma) \end{cases}$$

update:
$$\begin{cases} w_0 = w_0 - \alpha \frac{\partial L}{\partial w_0} \\ w_1 = w_1 - \alpha \frac{\partial L}{\partial w_1} \end{cases} \qquad b = b - \alpha \frac{\partial L}{\partial b}$$

در ایپاک اول ابتدا Data1 را وارد می‌کنیم:

$$z = 6 + (-1) + 2 = 7$$

$$\sigma(7) = 0.99908894 \qquad L = (1-\sigma(7))^2 = 0.0000008$$

$$\frac{\partial L}{\partial b} = -2 \times (0.9990889488)(1-0.9990889488)(1-0.9990889488) = -0.00000166$$

$$\frac{\partial L}{\partial w_0} = -2 \times (0.999088948) \times 3 \times (1 - 0.999088948 \times (1 - 0.999088948)) = -0.00000498$$

$$\frac{\partial L}{\partial w_1} = 0.00000166$$

$$\begin{cases} w_0 = 2 - 0.001(-0.00000498) = 2.000000000498 \\ w_1 = 1 - 0.001(0.00000166) = 0.999999999834 \\ b = 2 - 0.001(0.00000166) = 2.000000000166 \end{cases}$$

ایپاک 2 : خروجی‌ها مانند epoch 1 هستند.

$$z = 7.000000001811$$

$$\sigma(z) = 0.999088948$$

$$L = 0.00000083$$

$$\frac{\partial L}{\partial b} = -0.00000166 \qquad \frac{\partial L}{\partial w_0} = -0.0000049$$

$$\frac{\partial L}{\partial w_1} = 0.00000166$$

$$\begin{cases} w_0 = 2.000000000988 \\ w_1 = 0.999999999668 \\ b = 2.000000000232 \end{cases}$$

ایپاک 3 : دنیای دوم را وارد می‌کنیم.

$z = 2.0000001884$ $L = 0.7758034 95$

$\delta(z) = 0.8807970799$

$\frac{\partial L}{\partial b} = 0.2698747771$ $\frac{\partial L}{\partial w_1} = -0.5397495542$

$\frac{\partial L}{\partial w_0} = 0.2698747771$

$w_0 = 1.9997301 3510$

$b = 1.9997301 2754$

$w_1 = 1.0005397 4623$

ایپاک 4 :

$z = 1.9983807 7018$

$L = 0.7755038504$

$\delta(z) = 0.8806269644$

$\frac{\partial L}{\partial b} = 0.2700267624$ $\frac{\partial L}{\partial w_1} = -0.54005352$

$\frac{\partial L}{\partial w_0} = 0.2700267624$

وزن w_0 و بایاس بروزی :

$$\begin{cases} w_0 = 1.99946 \\ w_1 = 1.001079 \\ b = 1.99946 \end{cases}$$

۳. در این سوال ابتدا hyperparameter ها را مقداردهی می‌کنیم. ابعاد تصاویر دیتاست mnist 28×28 است. تعداد ایپاک‌ها را ۱۰ و سایز هر batch را ۶۴ در نظر می‌گیریم. تعداد کلاس‌ها نیز ۱۰ است.

Set hyperparameters

```
[ ]  IMG_WIDTH = 28
     IMG_HEIGHT = 28
     EPOCHS = 10
     BATCH_SIZE= 64
     n_classes = 10
```

سپس دیتاها را لود می‌کنیم.

```
data = np.load('mnist.npz')
[x_train, y_train, x_test, y_test] = data['x_train'], data['y_train'], data['x_test'], data['y_test']
```

در مرحله‌ی بعد مدل خود را تعریف می‌کنیم. همانطور که در صورت سوال ذکر شد، از سه لایه استفاده می‌کنیم و در لایه‌ی میانی از تابع فعالساز relu و در لایه‌ی نهایی از تابع فعالساز softmax استفاده می‌کنیم.

Define model

```
[ ]  model = tf.keras.models.Sequential([
       Flatten(input_shape=(28, 28)),
       Dense(128, activation='relu'),
       Dense(10, activation='softmax')
     ])
```

حال مدل را با بهینه‌ساز adam کامپایل و فیت می‌کنیم.

Compile and fit model , Optimizer = Adam

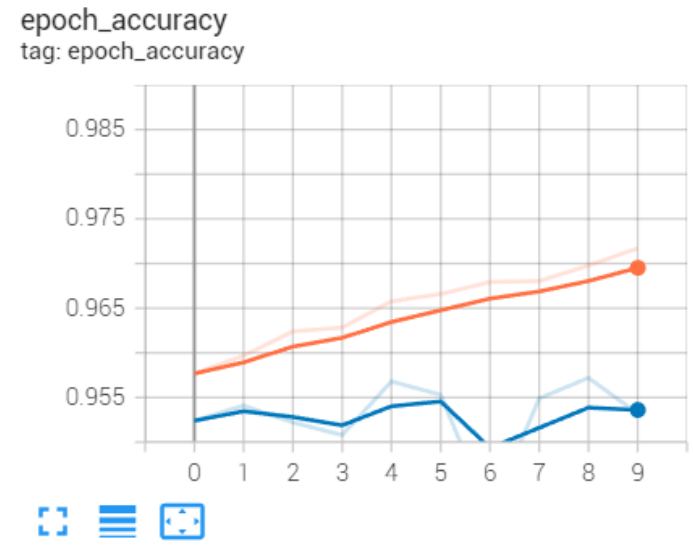
```
[ ]  # Compile model
     model.compile(
         optimizer = 'adam',
         loss = 'sparse_categorical_crossentropy',
         metrics = ['accuracy']
     )

     log_dir = "logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
     tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1)

     # Fit model
     history = model.fit(
         x = x_train,
         y = y_train,
         batch_size = BATCH_SIZE,
         epochs = EPOCHS,
         validation_data=(x_test, y_test),
         callbacks=[tensorboard_callback],
     )
```

دقت بدست آمده برای ۱۰ ایپاک:

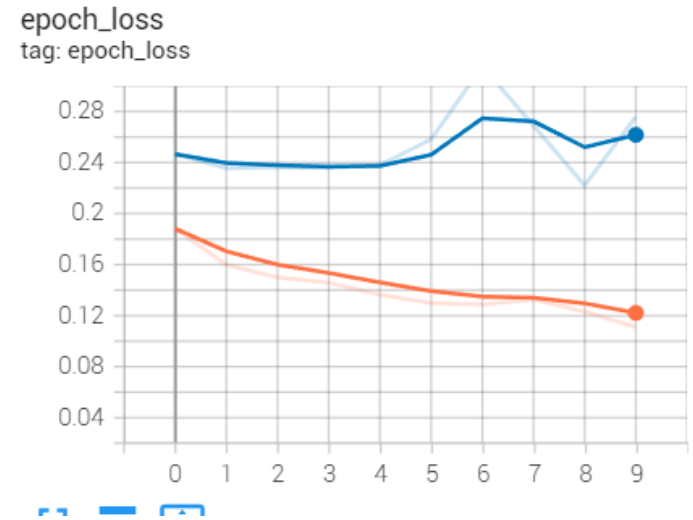
```
Epoch 1/10
938/938 [==============================] - 3s 3ms/step - loss: 0.1882 - accuracy: 0.9577 - val_loss: 0.2466 - val_accuracy: 0.9524
Epoch 2/10
938/938 [==============================] - 3s 3ms/step - loss: 0.1598 - accuracy: 0.9597 - val_loss: 0.2354 - val_accuracy: 0.9541
Epoch 3/10
938/938 [==============================] - 3s 3ms/step - loss: 0.1499 - accuracy: 0.9624 - val_loss: 0.2362 - val_accuracy: 0.9522
Epoch 4/10
938/938 [==============================] - 3s 3ms/step - loss: 0.1458 - accuracy: 0.9628 - val_loss: 0.2356 - val_accuracy: 0.9508
Epoch 5/10
938/938 [==============================] - 3s 3ms/step - loss: 0.1364 - accuracy: 0.9658 - val_loss: 0.2381 - val_accuracy: 0.9568
Epoch 6/10
938/938 [==============================] - 3s 3ms/step - loss: 0.1298 - accuracy: 0.9666 - val_loss: 0.2582 - val_accuracy: 0.9553
Epoch 7/10
938/938 [==============================] - 3s 3ms/step - loss: 0.1287 - accuracy: 0.9679 - val_loss: 0.3155 - val_accuracy: 0.9419
Epoch 8/10
938/938 [==============================] - 3s 3ms/step - loss: 0.1327 - accuracy: 0.9680 - val_loss: 0.2689 - val_accuracy: 0.9549
Epoch 9/10
938/938 [==============================] - 3s 3ms/step - loss: 0.1231 - accuracy: 0.9698 - val_loss: 0.2221 - val_accuracy: 0.9572
Epoch 10/10
938/938 [==============================] - 3s 3ms/step - loss: 0.1111 - accuracy: 0.9717 - val_loss: 0.2760 - val_accuracy: 0.9532
```

همانطور که می‌بینیم دقت به تقریبا به ۹۷٪ در داده‌های آموزشی و به ۹۵٪ در داده‌های validation رسیده است.

نمودار دقت و خطا برای بهینه‌ساز adam :



epoch_loss

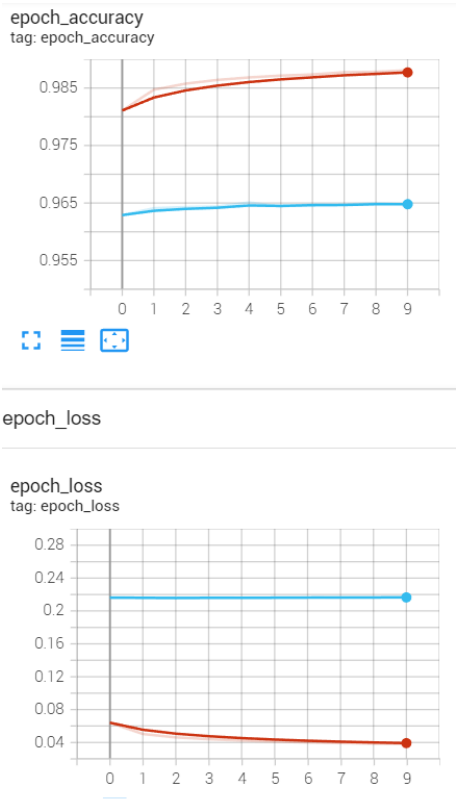


حال مدل را با بهینه‌ساز adagrad کامپایل و فیت می‌کنیم.

دقت بدست آمده برای ۱۰ ایپاک:

```
Epoch 1/10
938/938 [==============================] - 3s 3ms/step - loss: 0.0641 - accuracy: 0.9811 - val_loss: 0.2163 - val_accuracy: 0.9629
Epoch 2/10
938/938 [==============================] - 3s 3ms/step - loss: 0.0503 - accuracy: 0.9847 - val_loss: 0.2160 - val_accuracy: 0.9641
Epoch 3/10
938/938 [==============================] - 3s 3ms/step - loss: 0.0462 - accuracy: 0.9858 - val_loss: 0.2156 - val_accuracy: 0.9643
Epoch 4/10
938/938 [==============================] - 3s 3ms/step - loss: 0.0440 - accuracy: 0.9864 - val_loss: 0.2163 - val_accuracy: 0.9644
Epoch 5/10
938/938 [==============================] - 3s 3ms/step - loss: 0.0423 - accuracy: 0.9868 - val_loss: 0.2161 - val_accuracy: 0.9651
Epoch 6/10
938/938 [==============================] - 3s 3ms/step - loss: 0.0411 - accuracy: 0.9872 - val_loss: 0.2163 - val_accuracy: 0.9644
Epoch 7/10
938/938 [==============================] - 3s 3ms/step - loss: 0.0401 - accuracy: 0.9873 - val_loss: 0.2165 - val_accuracy: 0.9648
Epoch 8/10
938/938 [==============================] - 3s 3ms/step - loss: 0.0393 - accuracy: 0.9877 - val_loss: 0.2164 - val_accuracy: 0.9647
Epoch 9/10
938/938 [==============================] - 3s 3ms/step - loss: 0.0386 - accuracy: 0.9878 - val_loss: 0.2166 - val_accuracy: 0.9650
Epoch 10/10
938/938 [==============================] - 3s 3ms/step - loss: 0.0380 - accuracy: 0.9881 - val_loss: 0.2166 - val_accuracy: 0.9648
```

نمودار دقت و خطا برای بهینه‌ساز adagrad :

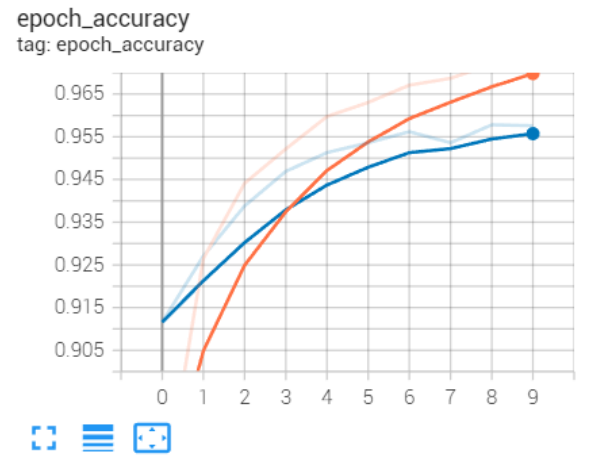


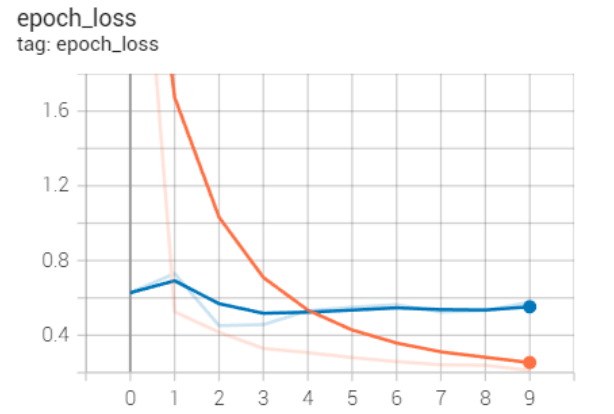
همانطور که می‌بینیم دقت به تقریبا به ۹۸٪ در داده‌های آموزشی و به ۹۶٪ در داده‌های validation رسیده است.

حال مدل را با بهینه‌ساز RMSprop کامپایل و فیت می‌کنیم.

```
Epoch 1/10
938/938 [==============================] - 6s 6ms/step - loss: 3.5868 - accuracy: 0.8688 - val_loss: 0.6278 - val_accuracy: 0.9116
Epoch 2/10
938/938 [==============================] - 5s 5ms/step - loss: 0.5259 - accuracy: 0.9265 - val_loss: 0.7304 - val_accuracy: 0.9271
Epoch 3/10
938/938 [==============================] - 5s 5ms/step - loss: 0.4162 - accuracy: 0.9441 - val_loss: 0.4516 - val_accuracy: 0.9388
Epoch 4/10
938/938 [==============================] - 5s 5ms/step - loss: 0.3297 - accuracy: 0.9521 - val_loss: 0.4571 - val_accuracy: 0.9469
Epoch 5/10
938/938 [==============================] - 5s 5ms/step - loss: 0.3072 - accuracy: 0.9597 - val_loss: 0.5314 - val_accuracy: 0.9513
Epoch 6/10
938/938 [==============================] - 5s 5ms/step - loss: 0.2807 - accuracy: 0.9631 - val_loss: 0.5501 - val_accuracy: 0.9536
Epoch 7/10
938/938 [==============================] - 5s 5ms/step - loss: 0.2593 - accuracy: 0.9671 - val_loss: 0.5642 - val_accuracy: 0.9562
Epoch 8/10
938/938 [==============================] - 5s 5ms/step - loss: 0.2421 - accuracy: 0.9687 - val_loss: 0.5243 - val_accuracy: 0.9536
Epoch 9/10
938/938 [==============================] - 5s 5ms/step - loss: 0.2396 - accuracy: 0.9720 - val_loss: 0.5318 - val_accuracy: 0.9578
Epoch 10/10
938/938 [==============================] - 5s 5ms/step - loss: 0.2117 - accuracy: 0.9744 - val_loss: 0.5783 - val_accuracy: 0.9576
```

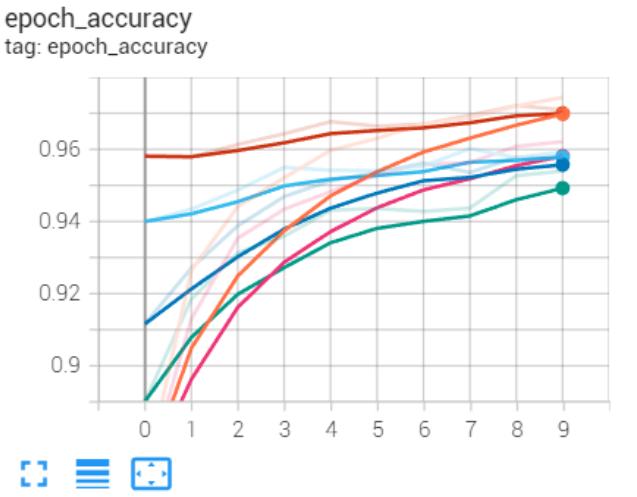
نمودار دقت و خطا برای بهینه‌ساز RMSprop :



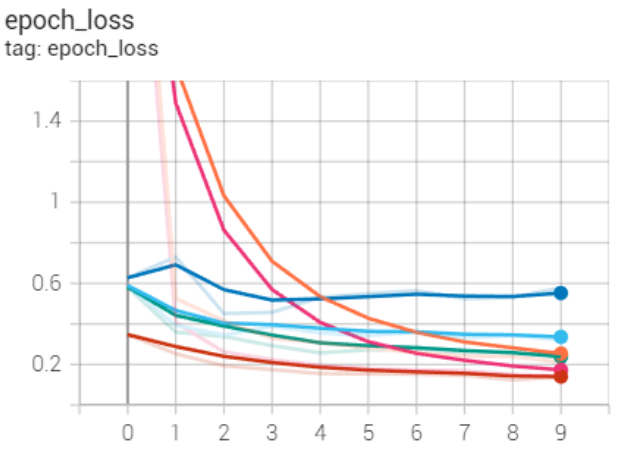


همانطور که می‌بینیم دقت به تقریبا به ۹۷٪ در داده‌های آموزشی و به ۹۵٪ در داده‌های validation رسیده است.

نمودار کلی دقت برای ۳ بهینه‌ساز به صورت زیر است:



epoch_loss



همانطور که واضح است دقت ما هنگام استفاده از بهینه‌ساز adagrad در بالاترین مقدار خود است.

Get Reports

```
# get report of metrics
# get report of metrics
print("accuracy:")
print("train:", eTrain[1])
print("test:", eTest[1])
print("**************************")
print("loss:")
print("train:", eTrain[0])
print("test:", eTest[0])
```

```
accuracy:
train: 0.9671000242233276
test: 0.9539999961853027
**************************
loss:
train: 0.11245177686214447
test: 0.20877638459205627
```

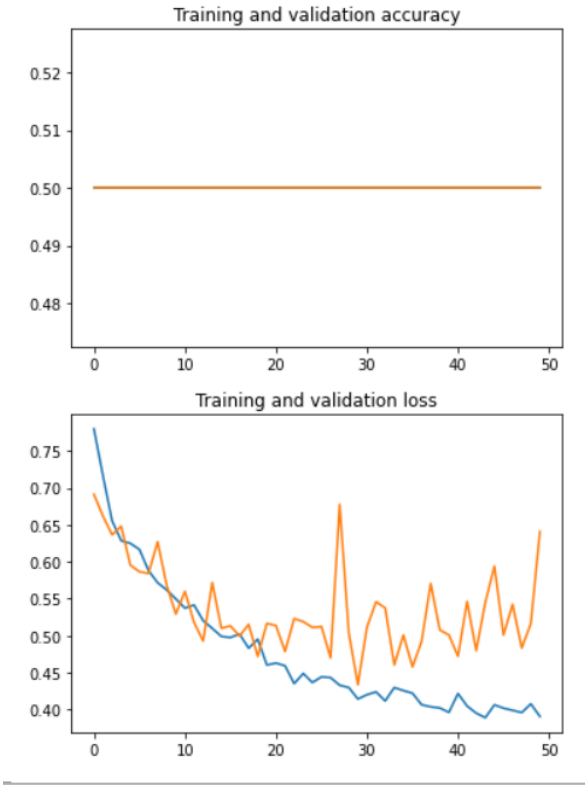
۴. در این سوال ابتدا با پارامترهای زیر مدل را اجرا می‌کنیم. تابع ضرر را برابر با binary_crossentropy قرار می‌دهیم زیرا مسئله‌ی ما classification دو کلاسه است.

```
## Set These Parameters
last_layer_neurons = 1
last_layer_activation = 'softmax'
loss_function = 'binary_crossentropy'
```

پس از ۵۰ ایپاک دقت زیر بدست آمد:

```
100/100 [==============================] - 19s 186ms/step - loss: 0.3952 - acc: 0.5000 - val_loss: 0.4798 - val_acc: 0.5000
Epoch 44/50
100/100 [==============================] - 20s 196ms/step - loss: 0.3890 - acc: 0.5000 - val_loss: 0.5451 - val_acc: 0.5000
Epoch 45/50
100/100 [==============================] - 19s 187ms/step - loss: 0.4064 - acc: 0.5000 - val_loss: 0.5941 - val_acc: 0.5000
Epoch 46/50
100/100 [==============================] - 20s 197ms/step - loss: 0.4019 - acc: 0.5000 - val_loss: 0.5009 - val_acc: 0.5000
Epoch 47/50
100/100 [==============================] - 19s 188ms/step - loss: 0.3990 - acc: 0.5000 - val_loss: 0.5425 - val_acc: 0.5000
Epoch 48/50
100/100 [==============================] - 19s 189ms/step - loss: 0.3959 - acc: 0.5000 - val_loss: 0.4833 - val_acc: 0.5000
Epoch 49/50
100/100 [==============================] - 20s 197ms/step - loss: 0.4077 - acc: 0.5000 - val_loss: 0.5163 - val_acc: 0.5000
Epoch 50/50
100/100 [==============================] - 19s 187ms/step - loss: 0.3907 - acc: 0.5000 - val_loss: 0.6410 - val_acc: 0.5000
```

سپس نمودار دقت و خطا را پلات کردیم.



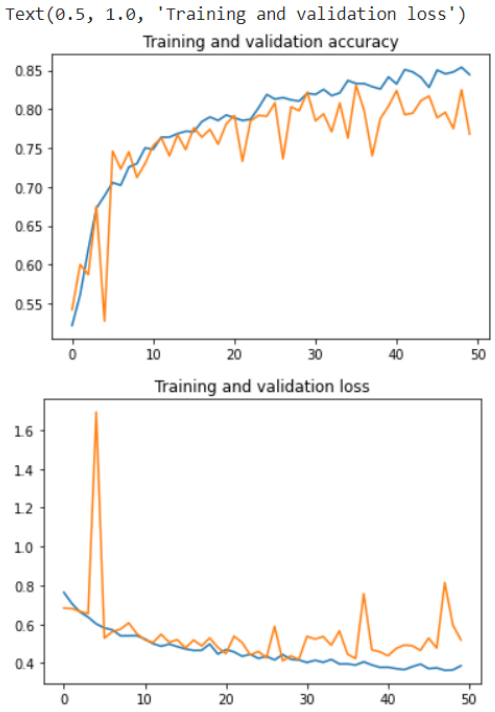
همانطور که می‌بینیم دقت داده‌های train و validation برابر ۵۰٪ است که عدد خوبی نیست. این به این دلیل است که مسئله‌ی ما binary classification است و در لایه‌ی خروجی باید از ۲ نورون استفاده کنیم. لذا در حالت بعدی با همان تابع ضرر و تابع فعال‌ساز تعداد نورون لایه‌ی خروجی را برابر ۲ قرار می‌دهیم.

```
## Set These Parameters
last_layer_neurons = 2
last_layer_activation = 'softmax'
loss_function = 'binary_crossentropy'
```

دقت بدست آمده پس از ۵۰ ایپاک:

```
Epoch 45/50
100/100 [==============================] - 19s 187ms/step - loss: 0.3952 - acc: 0.8280 - val_loss: 0.4657 - val_acc: 0.8170
Epoch 46/50
100/100 [==============================] - 18s 179ms/step - loss: 0.3719 - acc: 0.8505 - val_loss: 0.5291 - val_acc: 0.7890
Epoch 47/50
100/100 [==============================] - 18s 179ms/step - loss: 0.3756 - acc: 0.8455 - val_loss: 0.4776 - val_acc: 0.7960
Epoch 48/50
100/100 [==============================] - 19s 194ms/step - loss: 0.3635 - acc: 0.8480 - val_loss: 0.8150 - val_acc: 0.7750
Epoch 49/50
100/100 [==============================] - 18s 180ms/step - loss: 0.3652 - acc: 0.8540 - val_loss: 0.5964 - val_acc: 0.8250
Epoch 50/50
100/100 [==============================] - 18s 184ms/step - loss: 0.3867 - acc: 0.8445 - val_loss: 0.5190 - val_acc: 0.7680
```

نمودار دقت و خطای بدست آمده:



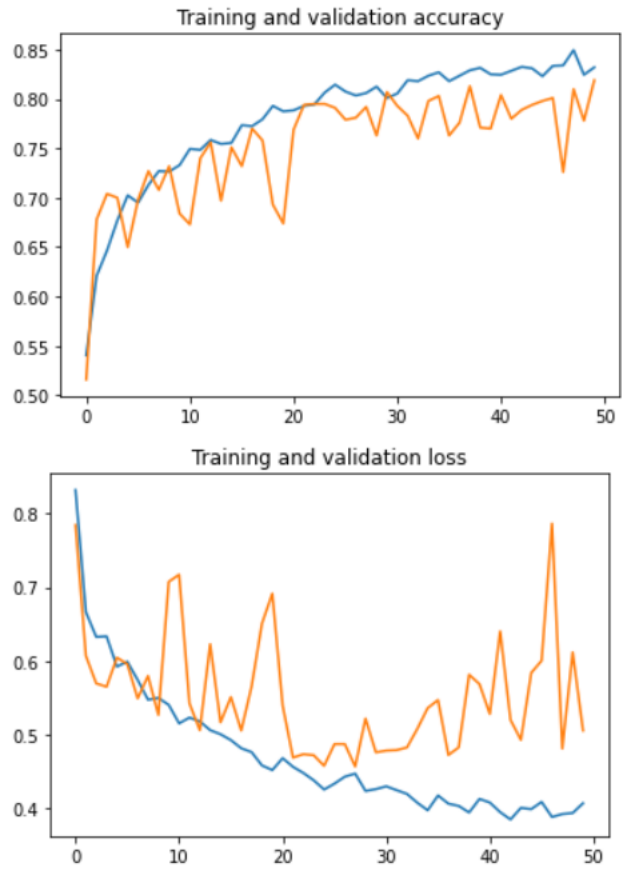
همانطور که می‌بینیم این بار دقت به تقریبا به ۸۴٪ در داده‌های آموزشی و به ۷۶٪ در داده‌های validation رسیده است.

مراحل قبل را برای پارامترهای زیر نیز تکرار می‌کنیم.

```
## Set These Parameters
last_layer_neurons = 2
last_layer_activation = 'sigmoid'
loss_function = 'binary_crossentropy'
```

```
Epoch 45/50
100/100 [==============================] - 18s 178ms/step - loss: 0.3991 - acc: 0.8230 - val_loss: 0.5839 - val_acc: 0.7980
Epoch 46/50
100/100 [==============================] - 18s 184ms/step - loss: 0.4089 - acc: 0.8335 - val_loss: 0.6003 - val_acc: 0.8010
Epoch 47/50
100/100 [==============================] - 18s 181ms/step - loss: 0.3884 - acc: 0.8340 - val_loss: 0.7863 - val_acc: 0.7260
Epoch 48/50
100/100 [==============================] - 18s 180ms/step - loss: 0.3924 - acc: 0.8495 - val_loss: 0.4811 - val_acc: 0.8100
Epoch 49/50
100/100 [==============================] - 18s 176ms/step - loss: 0.3938 - acc: 0.8245 - val_loss: 0.6117 - val_acc: 0.7780
Epoch 50/50
100/100 [==============================] - 18s 176ms/step - loss: 0.4071 - acc: 0.8320 - val_loss: 0.5056 - val_acc: 0.8190
```

```
Text(0.5, 1.0, 'Training and validation loss')
```



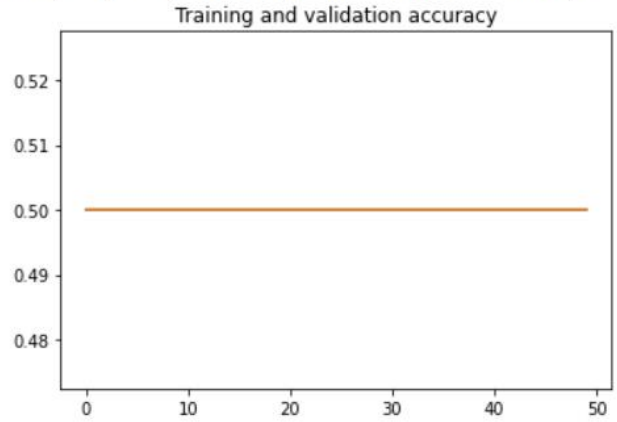
همانطور که می‌بینیم این بار دقت به تقریبا به ۸۳٪ در داده‌های آموزشی و به ۸۱٪ در داده‌های validation رسیده است.

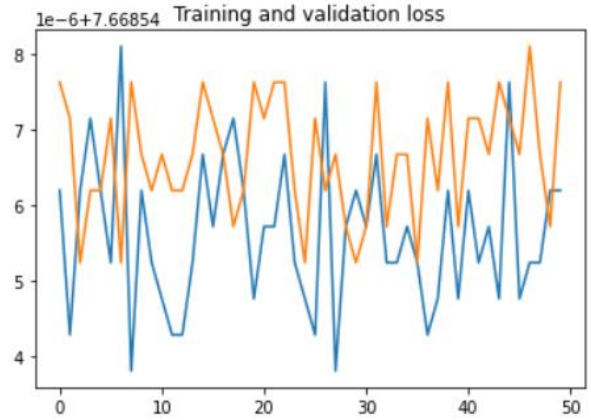
مراحل قبل را برای پارامترهای زیر نیز تکرار می‌کنیم.

```
## Set These Parameters
last_layer_neurons = 2
last_layer_activation = 'tanh'
loss_function = 'binary_crossentropy'
```

```
Epoch 45/50
100/100 [==============================] - 18s 183ms/step - loss: 7.6685 - acc: 0.5000 - val_loss: 7.6685 - val_acc: 0.5000
Epoch 46/50
100/100 [==============================] - 18s 184ms/step - loss: 7.6685 - acc: 0.5000 - val_loss: 7.6685 - val_acc: 0.5000
Epoch 47/50
100/100 [==============================] - 19s 189ms/step - loss: 7.6685 - acc: 0.5000 - val_loss: 7.6685 - val_acc: 0.5000
Epoch 48/50
100/100 [==============================] - 18s 182ms/step - loss: 7.6685 - acc: 0.5000 - val_loss: 7.6685 - val_acc: 0.5000
Epoch 49/50
100/100 [==============================] - 19s 189ms/step - loss: 7.6685 - acc: 0.5000 - val_loss: 7.6685 - val_acc: 0.5000
Epoch 50/50
100/100 [==============================] - 19s 188ms/step - loss: 7.6685 - acc: 0.5000 - val_loss: 7.6685 - val_acc: 0.5000
```

```
Text(0.5, 1.0, 'Training and validation loss')
```





همانطور که واضح است دقت باز پایین آمد و نمودار خطا نیز نوسان زیادی دارد.

حالت‌های مختلف دیگری هم تست شد اما باز نتیجه‌ی بهتری حاصل نشد و بهترین نتیجه مربوط به پارامترهای زیر است:

```
## Set These Parameters
last_layer_neurons = 2
last_layer_activation = 'sigmoid'
loss_function = 'binary_crossentropy'
```

```
Epoch 45/50
100/100 [==============================] - 18s 178ms/step - loss: 0.3991 - acc: 0.8230 - val_loss: 0.5839 - val_acc: 0.7980
Epoch 46/50
100/100 [==============================] - 18s 184ms/step - loss: 0.4089 - acc: 0.8335 - val_loss: 0.6003 - val_acc: 0.8010
Epoch 47/50
100/100 [==============================] - 18s 181ms/step - loss: 0.3884 - acc: 0.8340 - val_loss: 0.7863 - val_acc: 0.7260
Epoch 48/50
100/100 [==============================] - 18s 180ms/step - loss: 0.3924 - acc: 0.8495 - val_loss: 0.4811 - val_acc: 0.8100
Epoch 49/50
100/100 [==============================] - 18s 176ms/step - loss: 0.3938 - acc: 0.8245 - val_loss: 0.6117 - val_acc: 0.7780
Epoch 50/50
100/100 [==============================] - 18s 176ms/step - loss: 0.4071 - acc: 0.8320 - val_loss: 0.5056 - val_acc: 0.8190
```

نتیجه می‌گیریم برای مسائل کلاس‌بندی دوکلاسه بهتر است ابتدا دو نورون در لایه‌ی خروجی داشته باشیم. همچنین بهتر است از تابع فعالساز sigmoid در لایه‌ی آخر استفاده کنیم. زیرا مقدار آن بین صفر تا یک است. همچنین باز بدلیل نوع مسئله، بهتر است از تابع ضرر binary_crossentropy استفاده کنیم.