به نام خدا

درس یادگیری عمیق

تمرین سری دوم

مدرس درس:
سرکار خانم دکتر داوودآبادی

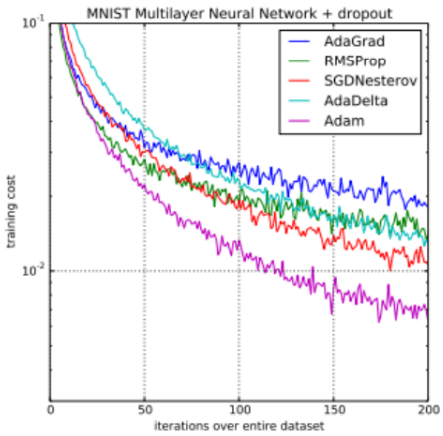
تهیه شده توسط:
الناز رضایی ۹۸۴۱۱۳۸۷

تاریخ ارسال: ۱۴۰۱/۰۸/۱۴

سوال ۱ :

نمودار زیر حاصل مقایسه انواع Optimizer بر روی دیتاست MNIST می‌باشد.

- الف) بعد از مطالعه و بررسی این لینک تحلیل خود را از نمودار زیر بیان کنید. (۱۰ نمره)



- ب) عملکرد یک Optimizer به عوامل متعددی بستگی دارد و نمی‌توان همواره یک Optimizer را بهتر از دیگری معرفی کرد. بنظر شما عملکرد یک Optimizer به چه عواملی بستگی دارد؟ چرا؟

پاسخ ۱ :

- الف) در این نمودار، نحوه عملکرد ۵ optimizer را برای loss به ازای epoch بهینه‌سازهای مختلف را بر دیتاست MNIST می‌بینیم. در ابتدا به توضیح عملکرد هر یک از این optimizer ها می‌پردازیم.

  درSGDNestrov (SGD + Momentum) optimizer، به جای گذشته به آینده نگاه می‌کنیم. از معایب این روش می‌توان به یکسان بودن learning rate در همه جهات که ممکن است موجب حرکت کم در جهت مناسب یا حرکت زیادی در featureهایی که تاثیر زیادی روی کم شدن loss دارند، بشود، اشاره کنیم. این موضوع باعث افزایش زمان همگرایی می‌شود و ممکن است نتوانیم در مدت تعیین شده در global optimum به خوبی همگرا شویم.

  در روش AdaGrad، مشکل یکسان بودن learning rate در همه جهات را حل می‌کند و با استفاده از نرخ یادگیری تطبیقی یا نرخ یادگیری بر پارامتر، باعث می‌شود در هر جهت، learning rate مناسب داشته باشیم. در نتیجه این کار، در جهت‌های نامناسب حرکت کمی داریم و بیشتر حرکتمان به سمت minimum باشد. مشکل این روش این است که مانند Momentum مفهوم سرعت و شتاب را همزمان ندارد. این موضوع باعث این می‌شود که عملکرد این روش از SGDNestrov نیز بدتر باشد؛ زیرا سریعا در local minima گیر کند.

روش RMSProp نیز شبیه روش AdaGrad می‌باشد. تفاوت این روش با AdaGrad این است که RMSProp برای تعیین نرخ یادگیری تطبیقی، از نوع خاصی از میانگین‌گیری (EMA) استفاده می‌کند. در نتیجه در این روش با توجه به گرادیان نقاط گذشته و اکنون حرکت کنیم و فقط به گرادیان حال حاضر نگاه نکنیم. بنابراین احتمال پرش از Global optimum کم می‌شود. این روش نیز مانند روش AdaGrad، از Momentum بهره نبرده است. لذا عملکردش از SGDNestrov بدتر و از AdaGrad بهتر است.
روش AdaDelta نیز مشابه RMSProp از EMA استفاده می‌کند و تفاوت آن در epochهای اولیه است؛ زیرا این روش از bias correction استفاده می‌کند و این امر موجب این می‌شود که در epochهای اولیه طول حرکت زیادی نداشته باشیم و smoothتر حرکت کنیم. در epochهای جلوتر، این روش مانند RMSProp عمل می‌کند و ضابطه آن‌ها یکی می‌شود.
روش Adam نیز از ترکیب همه روش‌های گفته شده در بالا به دست می‌آید. از Momentum به عنوان ممان اول، RMSProp به عنوان ممان دوم و اعمال bias correction بر هر دو ممان استفاده می‌کند. بنابراین عملکرد بهتری نسبت به سایر روش‌های گفته شده دارد. از دیگر مزایای این روش، می‌توان به سربار محاسباتی کم، مشخص بودن هایپر پارامترها و عدم نیاز به tuning (به جز learning rate)، پیاده‌سازی راحت و عدم نیاز به حافظه زیاد اشاره کرد.

- ب) به نوع دیتاست بستگی دارد برای مثال Adam بهترین عملکرد را در دیتاهای با Noise بالا و گرادیان Sparse دارد. همچنین هر کدام از این بهینه‌سازها دارای هایپرپارامترهایی هستند که باید Tune Fine شوند و عملکرد شبکه بستگی به آن دارد. اینکه بگوییم همیشه Adam نتیجه بهتری می‌دهد اشتباه است زیرا مقالات art the of State بسیاری را می‌بینیم که از گونه‌های SGD استفاده کرده‌اند. همچنین در مقاله‌ای که لینک آن در ادامه آورده شده یک روش Hybrid پیشنهاد شده که ابتدا با Adam آموزش را شروع کنیم سپس هنگامی که یک شرطی رخ داد آن را به SGD تغییر دهیم. این کار باعث می‌شود که Generalization بهتری داشته باشیم. یکی دیگر از عوامل تاثیرگذار متریک و هدف ما می باشد. برای مثال اگر Convergence برای ما اهمیت بیشتری دارد بهتر است از Adam استفاده کنیم. اما اگر هدف Generalization است بهتر است از SGDNesterov استفاده کنیم.
لینک‌های استفاده شده در این سوال:

`https://towardsdatascience.com/deep-learning-optimizers-436171c9e23f`
`https://arxiv.org/abs/1712.07628`

سوال ۲:

یک شبکه با دو ورودی (x_1, x_0) و سه پارامتر (b, w_1, w_2) است که شامل یک لایه خطی با رابطه $w_0x_0 + w_1x_1 + b$ و تابع فعال‌ساز Sigmoid میباشد. اگر برای آموزش شبکه از تابع ضرر میانگین مربعات خطا استفاده شود، برای این شبکه عملیات Backpropagation را در دو epoch با بهینه‌ساز SGD برای دو داده ورودی زیر انجام دهید (در هر تکرار فقط یک داده را وارد شبکه کنید ، یعنی در مجموع ۴ بار وزن‌های شبکه به روز خواهند شد). پس از هر گام بهینه‌سازی، خروجی شبکه را برای دو داده محاسبه کنید. (۲۵ نمره)

	x_0	x_1	y
Data1	3	-1	1
Data2	1	-2	0

(برای شروع، پارامترهای $w_0 = 2$، $w_1 = 1$ و $b = 2$ را در نظر بگیرید.)
*تابع فعال‌ساز sigmoid: $\sigma(x) = \frac{1}{1+e^{-x}}$

پاسخ ۲:

- Forward pass:
  $z = w_0x_0 + w_1x_1 + b$
  $\hat{y} = \sigma(z) = a$
  $L(y, a) = (y - a)^2$

- Backward pass:
  $\frac{\partial L}{\partial L} = 1$
  $\frac{\partial L}{\partial a} = -2 * (y - a)$
  $\frac{\partial L}{\partial z} = \frac{\partial L}{\partial a} \cdot \frac{\partial a}{\partial z} = -2a(1 - a)(y - a)$
  $\frac{\partial L}{\partial b} = \frac{\partial L}{\partial z} \cdot \frac{\partial z}{\partial b} = -2a(1 - a)(y - a)$
  $\frac{\partial L}{\partial w_0} = \frac{\partial L}{\partial z} \cdot \frac{\partial z}{\partial w_0} = -2ax_0(1 - a)(y - a)$
  $\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial z} \cdot \frac{\partial z}{\partial w_1} = -2ax_1(1 - a)(y - a)$

- Update parameters with SGD:
  $w_0 = w_0 - \eta \frac{\partial L}{\partial w_0}$
  $w_1 = w_1 - \eta \frac{\partial L}{\partial w_1}$
  $b = b - \eta \frac{\partial L}{\partial b}$

۱. Data1:
حال با توجه به روابط نوشته شده در بالا، برای Data1 شروع به آپدیت کردن پارامترها می‌کنیم.

epoch1)

- Forward pass:
  $z = (2 * 3) + (1 * (-1)) + 2 = 7$
  $a = \sigma(7) = 0.9990889488$
  $L(1, 0.9990889488) = (1 - 0.9990889488)^2 = 0.00000083$
- Backward pass:
  $\frac{\partial L}{\partial b} = -2(0.9990889488)(1-0.9990889488)(1-0.9990889488) = -0.00000166$

$\frac{\partial L}{\partial w_0} = -2(0.9990889488)(3)(1-0.9990889488)(1-0.9990889488) = -0.00000498$
$\frac{\partial L}{\partial w_1} = -2(0.9990889488)(-1)(1-0.9990889488)(1-0.9990889488) = 0.00000166$

- Update parameters with SGD:
  $w_0 = 2 - 0.001(-0.00000498) = 2.00000000498$
  $w_1 = 1 - 0.001(0.00000166) = 0.99999999834$
  $b = 2 - 0.001(-0.00000166) = 2.00000000166$

epoch2)

- Forward pass:
  $z = (2.00000000498*3)+(0.99999999834*(-1))+2.00000000166 = 7.00000001811$
  $a = \sigma(7.00000001811) = 0.9990889488$
  $L(1, 0.9990889488) = (1 - 0.9990889488)^2 = 0.00000083$
- Backward pass:
  $\frac{\partial L}{\partial b} = -2(0.9990889488)(1-0.9990889488)(1-0.9990889488) = -0.00000166$
  $\frac{\partial L}{\partial w_0} = -2(0.9990889488)(3)(1-0.9990889488)(1-0.9990889488) = -0.0000049$
  $\frac{\partial L}{\partial w_1} = -2(0.9990889488)(-1)(1-0.9990889488)(1-0.9990889488) = 0.00000166$
- Update parameters with SGD:
  $w_0 = 2.00000000498 - 0.001(-0.0000049) = 2.00000000988$
  $w_1 = 0.99999999834 - 0.001(0.00000166) = 0.99999999668$
  $b = 2.00000000166 - 0.001(-0.00000166) = 2.00000000232$

۲. Data2:

سپس با استفاده از وزن‌های جدید به دست آمده توسط داده اول، با استفاده از داده دوم، شروع به آپدیت کردن دوباره‌ی وزن‌ها می‌کنیم.

epoch3)

- Forward pass:
  $z = (2.00000000988*1)+(0.99999999668*(-2))+2.00000000232 = 2.00000001884$
  $a = \sigma(2.00000001884) = 0.8807970799$
  $L(0, 0.8807970799) = (0 - 0.88079707998)^2 = 0.7758034959$
- Backward pass:
  $\frac{\partial L}{\partial b} = -2(0.7758034959)(1-0.7758034959)(0-0.7758034959) = 0.2698747771$
  $\frac{\partial L}{\partial w_0} = -2(0.7758034959)(1)(1-0.7758034959)(0-0.7758034959) =$

0.2698747771
$\frac{\partial L}{\partial w_1} = -2(0.7758034959)(-2)(1-0.7758034959)(0-0.7758034959) = -0.5397495542$

- Update parameters with SGD:
  $w_0 = 2.00000000988 - 0.001(0.2698747771) = 1.99973013510$
  $w_1 = 0.99999999668 - 0.001(-0.5397495542) = 1.00053974623$
  $b = 2.00000000232 - 0.001(0.2698747771) = 1.99973012754$

epoch4)

- Forward pass:
  $z = (1.99973013510*1)+(1.00053974623*(-2))+1.99973012754 = 1.99838077018$
  $a = \sigma(2.00000001884) = 0.8806269644$
  $L(0, 0.8806269644) = (0 - 0.8806269644)^2 = 0.7755038504$
- Backward pass:
  $\frac{\partial L}{\partial b} = -2(0.7755038504)(1-0.7755038504)(0-0.7755038504) = 0.2700267624$
  $\frac{\partial L}{\partial w_0} = -2(0.7755038504)(1)(1-0.7755038504)(0-0.7755038504) = 0.2700267624$
  $\frac{\partial L}{\partial w_1} = -2(0.7755038504)(-2)(1-0.7755038504)(0-0.7755038504) = -0.5400535247$
- Update parameters with SGD:
  $w_0 = 1.99973013510 - 0.001(0.2700267624) = 1.99946010834$
  $w_1 = 1.00053974623 - 0.001(-0.5400535247) = 1.00107979975$
  $b = 1.9997301275 - 0.001(0.2700267624) = 1.99946010074$

وزن‌های نهایی به‌ازای ۲ epoch برای دو داده ۱ و ۲ به شکل زیر در می‌آیند:

$w_0 = 1.99946010834$
$w_1 = 1.00107979975$
$b = 1.99946010074$

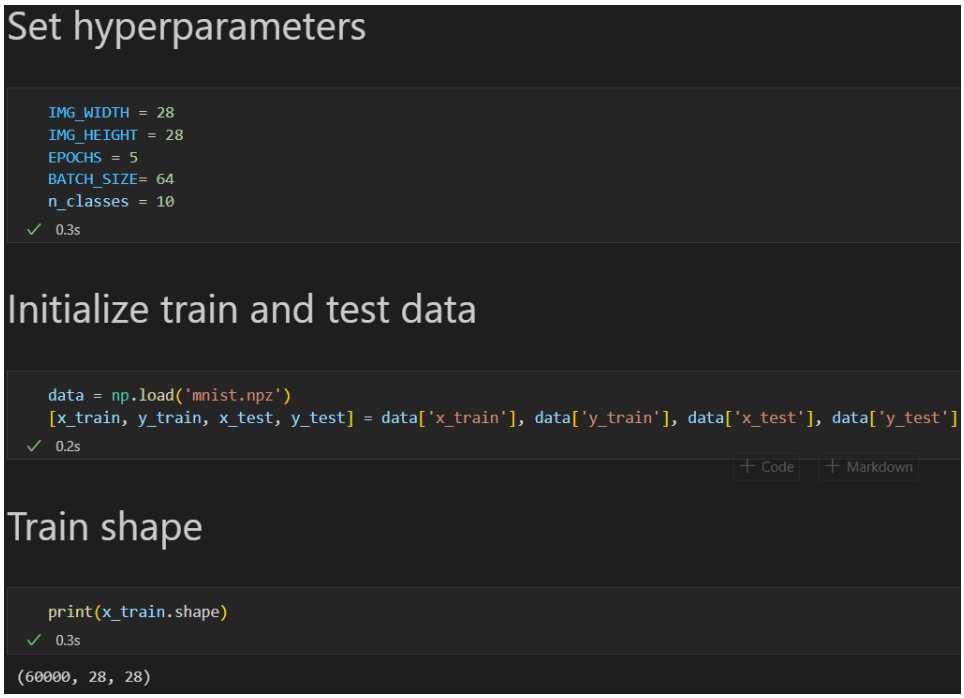
سوال ۳:

مجموعه داده MNIST به فایل تمرین دوم ضمیمه شده است. در این سوال می‌خواهیم شبکه‌ای طراحی کنید (که می‌تواند تنها دارای ۳ لایه Dense باشد) و این دیتاست را بر روی آن آموزش دهید به طوری که بهینه‌سازهای مختلفی که در درس آموختید (RMSProp, AdaGrad, Adam) را تست کنید و نتایج آنها را با هم مقایسه کنید. معیار Accuracy را برای هر آزمایش بدست آورید و tensorboard را برای آن‌ها رسم کنید و در آخر مدل آموزش دیده‌ای که بهترین دقت را دارد ذخیره

کنید. (۳۵ نمره)
* از نوتبوکی که برای این سوال در اختیارتان قرار گرفته است استفاده نمایید.

پاسخ ۳:

در ابتدا، Hyper parameterهای مدل خود را تعیین می‌کنیم. طول و عرض تصویر را با استفاده از x_train.shape به دست می‌آوریم که اولین خروجی این تابع، تعداد نمونه‌ها و خروجی‌های دوم و سوم مربوط به طول و عرض هر نمونه می‌باشد. چون دیتاست MNIST می‌باشد، می‌دانیم که تعداد کلاس‌هایش ۱۰ تا و مربوط به ارقام ۰ تا ۹ است. تعداد epoch و batch size هم در اینجا ۵ و ۶۴ در نظر گرفتیم.



سپس مدل خود را به صورت Functional API تعریف می‌کنیم. ابتدا لایه ورودی را با توجه به سایز ورودی تعریف کرده و آن را Flatten می‌کنیم. تعداد لایه‌های میانی را هم با توجه به صورت سوال، ۳ در نظر می‌گیریم و activation function را relu می‌گذاریم. تعداد نورون‌های لایه آخر را هم به تعداد کلاس‌ها و activation function آن را softmax در نظر می‌گیریم.

Define model

```python
def build_model(input_shape, num_classes):
    # Input Layer
    input = Input(input_shape)
    model = Flatten()(input)

    # Hidden Layer
    model = Dense(3, activation="relu")(model)

    # Output Layer
    output = Dense(num_classes, activation="softmax")(model)

    return Model(inputs = input, outputs = output)
```

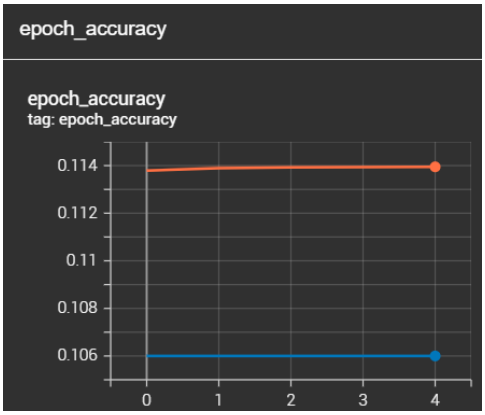
✓ 0.3s

در ادامه، مدل خود را با استفاده از تابع تعریف شده در بخش قبل، می‌سازیم و شروع به امتحان بهینه‌سازهای مختلف بر آن می‌کنیم تا ببینیم کدام بهینه‌ساز accuracy بهتری دارد. در ابتدای امر، مدل خود را با استفاده از SGD، compile و fit کرده و tensorboard آن را رسم می‌کنیم.

```python
model_SGD = build_model(
    input_shape= (IMG_WIDTH, IMG_HEIGHT),
    num_classes= n_classes
)
# Compile model
model_SGD.compile(
    optimizer = SGD(learning_rate=0.01),
    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_SGD.fit(
    x = x_train,
    y = y_train,
    batch_size = BATCH_SIZE,
    epochs = EPOCHS,
    validation_split = 0.2,
    callbacks=[tensorboard_callback]
)
```

✓ 13.8s

```
Epoch 1/5
750/750 [==============================] - 3s 3ms/step - loss: 2.3811 - accuracy: 0.1138 - val_loss: 2.3019 - val_accuracy: 0.1060
Epoch 2/5
750/750 [==============================] - 2s 3ms/step - loss: 2.3012 - accuracy: 0.1140 - val_loss: 2.3019 - val_accuracy: 0.1060
Epoch 3/5
750/750 [==============================] - 2s 3ms/step - loss: 2.3011 - accuracy: 0.1140 - val_loss: 2.3020 - val_accuracy: 0.1060
Epoch 4/5
750/750 [==============================] - 2s 3ms/step - loss: 2.3010 - accuracy: 0.1140 - val_loss: 2.3021 - val_accuracy: 0.1060
Epoch 5/5
750/750 [==============================] - 4s 5ms/step - loss: 2.3010 - accuracy: 0.1140 - val_loss: 2.3021 - val_accuracy: 0.1060
```

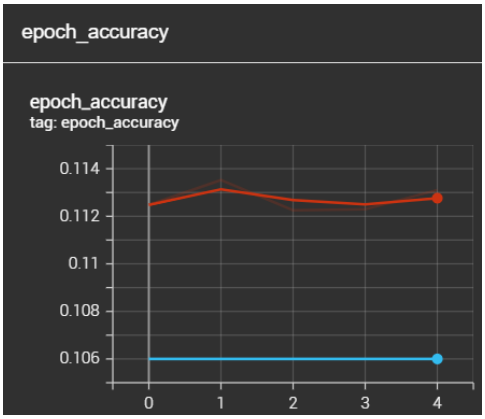


حال optimizer خود را Adam گذاشته و مراحل قبل را تکرار می‌کنیم.

```
model_Adam = build_model(
    input_shape= (IMG_WIDTH, IMG_HEIGHT),
    num_classes= n_classes
)
# Compile model
model_Adam.compile(
    optimizer = Adam(learning_rate=0.01),
    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_Adam.fit(
    x = x_train,
    y = y_train,
    batch_size = BATCH_SIZE,
    epochs = EPOCHS,
    validation_split = 0.2,
    callbacks=[tensorboard_callback]
)
```

✓ 4.7s

```
Epoch 1/5
750/750 [==============================] - 1s 1ms/step - loss: 2.5955 - accuracy: 0.1125 - val_loss: 2.3022 - val_accuracy: 0.1060
Epoch 2/5
750/750 [==============================] - 1s 1ms/step - loss: 2.3019 - accuracy: 0.1135 - val_loss: 2.3026 - val_accuracy: 0.1060
Epoch 3/5
750/750 [==============================] - 1s 1ms/step - loss: 2.3019 - accuracy: 0.1123 - val_loss: 2.3029 - val_accuracy: 0.1060
Epoch 4/5
750/750 [==============================] - 1s 1ms/step - loss: 2.3019 - accuracy: 0.1123 - val_loss: 2.3024 - val_accuracy: 0.1060
Epoch 5/5
750/750 [==============================] - 1s 1ms/step - loss: 2.3020 - accuracy: 0.1131 - val_loss: 2.3026 - val_accuracy: 0.1060
```

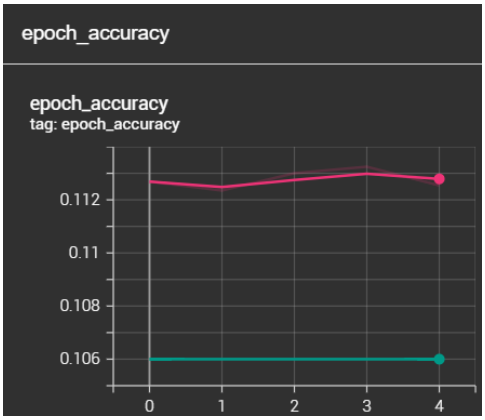


اکنون optimizer را RMSProp گذاشته و مراحل قبل را تکرار می‌کنیم.

```
model_RMSProp = build_model(
    input_shape= (IMG_WIDTH, IMG_HEIGHT),
    num_classes= n_classes
)
# Compile model
model_RMSProp.compile(
    optimizer = RMSprop(learning_rate=0.01),
    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_RMSProp.fit(
    x = x_train,
    y = y_train,
    batch_size = BATCH_SIZE,
    epochs = EPOCHS,
    validation_split = 0.2,
    callbacks=[tensorboard_callback]
)
```

✓ 5.2s

```
Epoch 1/5
750/750 [==============================] - 2s 1ms/step - loss: 2.3598 - accuracy: 0.1127 - val_loss: 2.3031 - val_accuracy: 0.1060
Epoch 2/5
750/750 [==============================] - 1s 1ms/step - loss: 2.3018 - accuracy: 0.1124 - val_loss: 2.3044 - val_accuracy: 0.1060
Epoch 3/5
750/750 [==============================] - 1s 1ms/step - loss: 2.3018 - accuracy: 0.1130 - val_loss: 2.3038 - val_accuracy: 0.1060
Epoch 4/5
750/750 [==============================] - 1s 1ms/step - loss: 2.3020 - accuracy: 0.1133 - val_loss: 2.3043 - val_accuracy: 0.1060
Epoch 5/5
750/750 [==============================] - 1s 2ms/step - loss: 2.3020 - accuracy: 0.1125 - val_loss: 2.3022 - val_accuracy: 0.1060
```

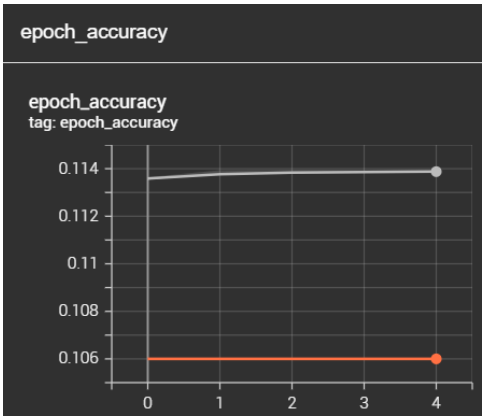


و در آخر این مراحل را برای Adagrad optimizer تکرار می‌کنیم.

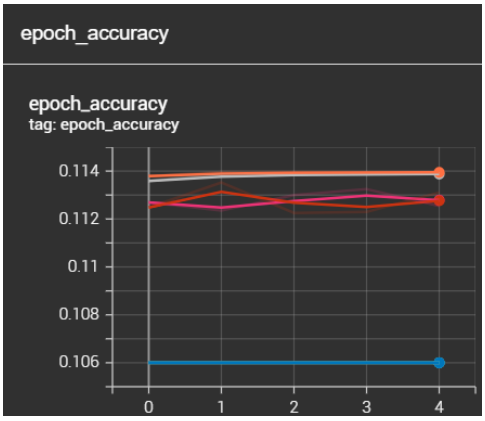
```
model_AdaGrad = build_model(
    input_shape= (IMG_WIDTH, IMG_HEIGHT),
    num_classes= n_classes
)
# Compile model
model_AdaGrad.compile(
    optimizer = Adagrad(learning_rate=0.01),
    loss = "sparse_categorical_crossentropy",
    metrics = ['accuracy']
)
log_dir = "logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S") +"/"
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1)
# Fit model
history = model_AdaGrad.fit(
    x = x_train,
    y = y_train,
    batch_size = BATCH_SIZE,
    epochs = EPOCHS,
    validation_split = 0.2,
    callbacks=[tensorboard_callback]
)
```

✓ 4.4s

```
Epoch 1/5
750/750 [==============================] - 1s 1ms/step - loss: 2.4038 - accuracy: 0.1136 - val_loss: 2.3075 - val_accuracy: 0.1060
Epoch 2/5
750/750 [==============================] - 1s 1ms/step - loss: 2.3013 - accuracy: 0.1139 - val_loss: 2.3071 - val_accuracy: 0.1060
Epoch 3/5
750/750 [==============================] - 1s 1ms/step - loss: 2.3010 - accuracy: 0.1139 - val_loss: 2.3071 - val_accuracy: 0.1060
Epoch 4/5
750/750 [==============================] - 1s 1ms/step - loss: 2.3010 - accuracy: 0.1139 - val_loss: 2.3071 - val_accuracy: 0.1060
Epoch 5/5
750/750 [==============================] - 1s 1ms/step - loss: 2.3009 - accuracy: 0.1139 - val_loss: 2.3071 - val_accuracy: 0.1060
```



اگر بخواهیم نمودار تمامی optimizerها را در کنار هم مشاهده کنیم تا ببینیم کدام دقت بیشتری دارد، از تصویر زیر استفاده می‌کنیم.



نکته‌ای که در تمامی این نمودارها وجود دارد، این است که مدل ما دقت کمی روی داده‌های validation دارد و به نوعی overfitting رخ داده است و این می‌تواند به دلیل تعداد کم لایه‌های پنهانی باشد. به علاوه، چون در صورت سوال خواسته شده تا مدلی که بیشترین accuracy را دارد ذخیره کنیم، می‌بینیم که optimizerهای SGD و Adagrad، عملکرد بهتری داشته‌اند. چون در صورت سوال به SGD اشاره نشده بود، من Adagrad را به عنوان بهترین بهینه‌ساز در نظر گرفتم. البته اگر تعداد epochها را بیشتر در نظر می‌گرفتم، ممکن بود نتایج متفاوت باشد. سپس مدل معرفی شده به عنوان بهترین مدل را load کرده و آن را compile و evaluate می‌کنیم.

Load and compile best model

```
# load best model
best_model = model_AdaGrad
# compile best model
best_model.compile(
    optimizer = Adagrad(learning_rate=0.01),
    loss = "sparse_categorical_crossentropy",
    metrics = ['accuracy']
)
```

✓ 0.3s

Evaluate best model

```
# evaluate best model
train_evaluate = best_model.evaluate(x_train, y_train, batch_size=BATCH_SIZE)
test_evaluate = best_model.evaluate(x_test, y_test, batch_size=BATCH_SIZE)
print("Train evaluation is: ", train_evaluate)
print("Test evaluation is: ", test_evaluate)
```

✓ 0.9s

```
938/938 [==============================] - 1s 735us/step - loss: 2.3012 - accuracy: 0.1124
157/157 [==============================] - 0s 772us/step - loss: 2.3034 - accuracy: 0.1136
Train evaluation is:  [2.301168441772461, 0.11236666887998581]
Test evaluation is:  [2.3033528327941895, 0.1136000007390976]
```

سوال ۴:

نوتبوک این سوال برای دسته‌بندی تصاویر مربوط به دو کلاس سگ و گربه نوشته شده است. ابتدا کد را بررسی کنید و سپس با تعیین پارامترهایی مانند تعداد نورون‌های لایه آخر، تابع فعال‌سازی لایه آخر و تابع ضرر، کد را اجرا کنید تا نتیجه دسته‌بندی بدست آید. حالت‌های مختلف را ارزیابی نمایید و نتایج را مقایسه و تحلیل نمایید.

پاسخ ۴:

در اینجا ما دسته‌بندی ۲ کلاسه (باینری) داریم. بنابراین برای لایه خروجی بهتر است از ۲ نورون استفاده کنیم؛ هر نورون مربوط به یکی از کلاس‌های ما است. در نظر گرفتن بیشتر از ۲ نورون برای خروجی در این سوال، بی‌معنی می‌باشد. در مورد activation function نیز باز هم به دلیل باینری بودن، انتخاب sigmoid در لایه آخر بهتر می‌باشد تا خروجی را به صورت یک عدد بین ۰ تا ۱ به ما بدهد. برای loss function هم در مسائل ۲ کلاسه، بهتر است از binary cross entropy استفاده کنیم.

- انتخاب تعداد نورون لایه خروجی:
همانطور که در بخش قبل گفته شد، به دلیل ۲ کلاسه بودن، نیاز به ۲ نورون در لایه خروجی داریم. ۱ نورون هم می‌توانیم انتخاب کنیم اما عملکرد بسیار ضعیفی داشت و دقت آن 0.5 بود.

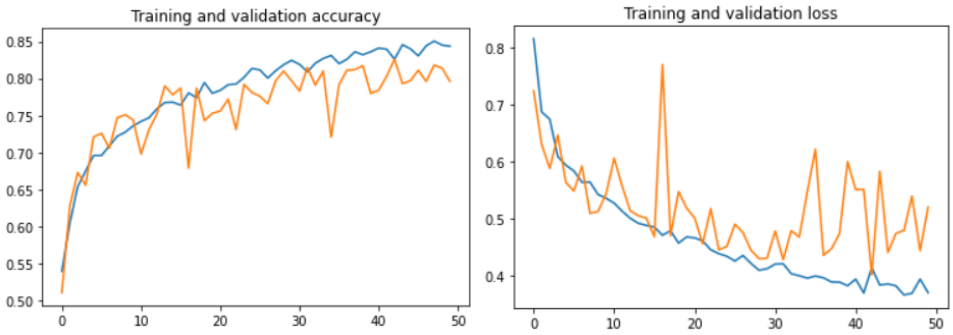
- انتخاب activation function لایه آخر:
به منظور انتخاب بهترین activation function، مقدار loss function را یک مقدار ثابت گذاشته و activation functionهای مختلف را تست می‌کنیم. در ابتدا از softmax استفاده می‌کنیم.

```
[8]  ## Set These Parameters
     last_layer_neurons = 2
     last_layer_activation = 'softmax'
     loss_function = 'categorical_crossentropy'
```

مقادیر loss و accuracy برای این تابع فعال‌سازی پس از ۵۰ epoch، مطابق شکل زیر است.

```
Epoch 50/50
100/100 [==============================] - 17s 172ms/step - loss: 0.3696 - acc: 0.8435 - val_loss: 0.5202 - val_acc: 0.7960
```

همچنین نمودارهای مربوط به loss و accuracy این تابع فعال‌سازی، در تصویر زیر نمایش داده شده‌اند.



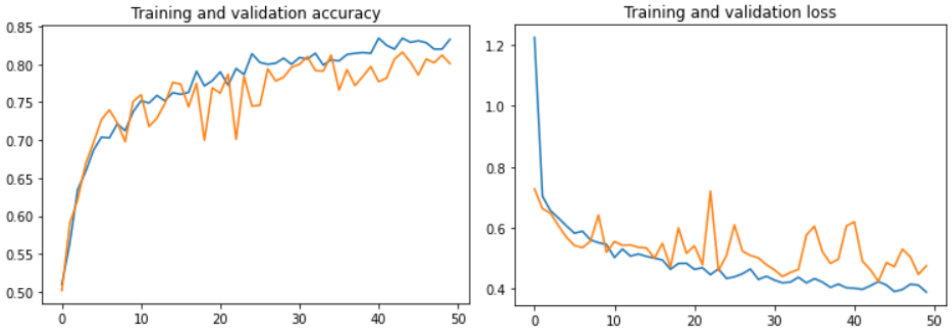
حال تابع فعال‌سازی sigmoid را امتحان می‌کنیم.

```
[41] ## Set These Parameters
     last_layer_neurons = 2
     last_layer_activation = 'sigmoid'
     loss_function = 'categorical_crossentropy'
```

مقدار loss و accuracy این مدل، با استفاده از تابع فعال‌سازی sigmoid، به شرح زیر است.

```
Epoch 50/50
100/100 [==============================] - 17s 174ms/step - loss: 0.3895 - acc: 0.8330 - val_loss: 0.4761 - val_acc: 0.8010
```

نمودارهای loss و accuracy این تابع نیز در تصویر زیر نمایش داده شده‌اند.



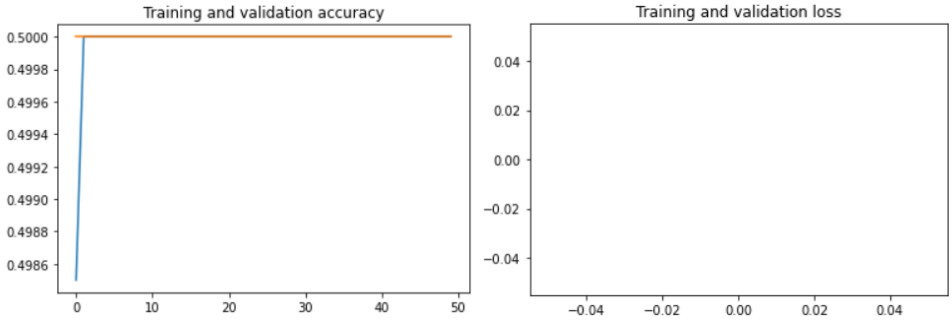
همانطور که مشخص است، sigmoid عملکرد بهتری روی داده‌های validation داشت.
حال به سراغ تابع فعال‌سازی relu می‌رویم و آن را هم برای لایه آخر تست می‌کنیم.

```
[51] ## Set These Parameters
     last_layer_neurons = 2
     last_layer_activation = 'relu'
     loss_function = 'categorical_crossentropy'
```

مقادیر loss و accuracy پس از ۵۰ epoch، به شرح زیر می‌باشد.

```
Epoch 50/50
100/100 [==============================] - 17s 171ms/step - loss: nan - acc: 0.5000 - val_loss: nan - val_acc: 0.5000
```

نمودار loss و accuracy آن نیز به شکل زیر در می‌آید.



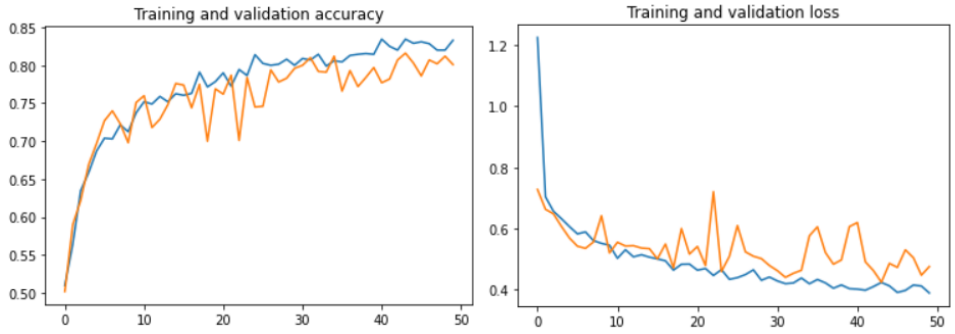
همانطور که مشخص است، این تابع اصلا عملکرد خوبی نداشت، بنابراین همان sigmoid را به عنوان بهترین activation function انتخاب می‌کنیم.

- انتخاب loss function:

عملکرد categorical_crossentropy loss function را در بخش قبل مشاهده کردیم، حال به طور مختصر در این بخش نتایج و کد مربوط به آن را می‌آوریم.

```
[41] ## Set These Parameters
     last_layer_neurons = 2
     last_layer_activation = 'sigmoid'
     loss_function = 'categorical_crossentropy'
```

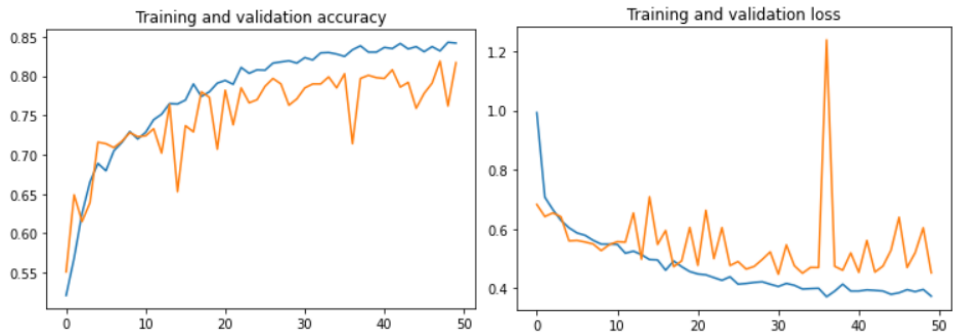
```
Epoch 50/50
100/100 [==============================] - 17s 174ms/step - loss: 0.3895 - acc: 0.8330 - val_loss: 0.4761 - val_acc: 0.8010
```



اکنون binary_crossentropy را تست می‌کنیم و کد و نتایج آن را در ادامه نمایش می‌دهیم.

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

```
Epoch 50/50
100/100 [==============================] - 18s 178ms/step - loss: 0.3735 - acc: 0.8420 - val_loss: 0.4524 - val_acc: 0.8170
```



همانطور که مشخص است، binary_crossentropy در این مسئله عملکرد بهتری داشت و مقادیر loss روی داده‌های train و test کمتر شد (به علت ۲ کلاسه بودن)؛ پس آن را به عنوان بهترین loss function در این مسئله انتخاب می‌کنیم. (loss functionهای دیگر مانند MSE, MAE, Relative crossentropy برای این مدل، نتایج جالبی نداشتند.)

نتیجه نهایی برای تعیین Hyper parameterها:

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