به نام خدا

درس یادگیری عمیق

تمرین سری سوم

مدرس درس:
سرکار خانم دکتر داوودآبادی

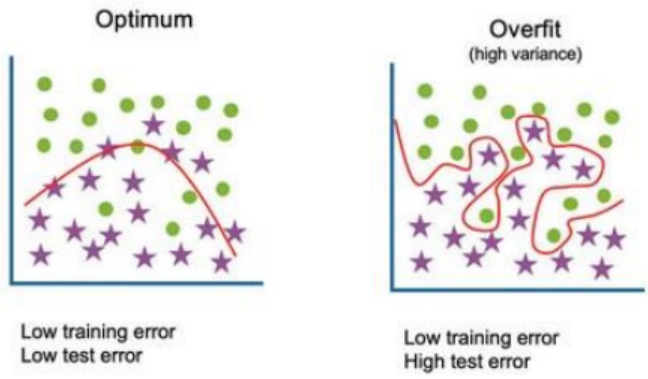
تهیه شده توسط:
الناز رضایی ۹۸۴۱۱۳۸۷

تاریخ ارسال: ۱۴۰۱/۰۹/۰۹

سوال ۱:

- الف) مفهوم overfit در شبکه‌های عصبی را توضیح دهید. چه موقع گوییم شبکه‌ی عصبی overfit شده است؟ راهکارهای خود برای جلوگیری و حل overfit شدن شبکه‌های عصبی را نام ببرید. (۱۵ نمره)

- ب) شکل شماره‌ی ۱ نمایانگر مدل overfit شده بر روی مجموعه‌ی داده است و شکل شماره‌ی ۲ مدل normal را نشان میدهد. چرا در شکل شماره‌ی ۱ مدل برای داده‌هایی که تاکنون ندیده است، خطای بیشتری نسبت به مدل شکل شمارهٔ ۲ تولید خواهد کرد؟ (۱۰ نمره)



شکل شماره‌ی ۱

شکل شماره‌ی ۲

پاسخ ۱:

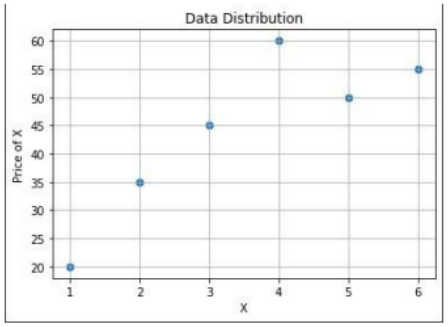
- الف) overfitting به حالتی گفته می‌شود که مدل ما روی داده‌های train عملکرد خوبی دارد اما روی داده‌های جدید (test) عملکرد ضعیفی دارد. در این حالت مدل ما روی داده‌های train دقیق شده است و ویژگی‌ها و جزئیات آن را یاد گرفته است و حتی ممکن است خطا روی داده‌های train صفر شود. از راه‌حل‌های جلوگیری از overfitting می‌توان به کاهش ابعاد شبکه، ارزیابی مدل‌های یادگیری ماشین، منظم‌سازی (L_1 و L_2)، داده‌افزایی (استفاده از برخی تبدیل‌های ریاضی مانند flip، افزودن نویز، Dropout) اشاره کرد.

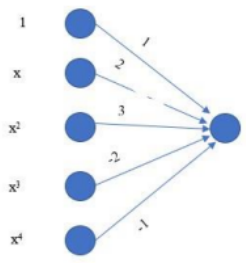
- ب) در شکل ۱ مدل روی داده‌های train، overfit شده است و بر روی جزوئیات داده‌های

train دقیق شده است و در نتیجه قابلیت تعمیم‌دهی (generalization) کمی دارد و روی داده‌های test، عملکرد ضعیفی خواهد داشت؛ اما شکل شماره ۲، مدل هم روی داده‌های train عملکرد نسبتا خوبی دارد و همچنین روی داده test عملکرد بهتری نسبت به مدل شماره ۱ دارد. چرا که قابلیت تعمیم‌دهی آن بیشتر است. در مورد شکل ۱، اتفاقی که می‌افتد به این صورت است که پس از چند تکرار، بهبود تعمیم‌دهی متوقف می‌شود و سپس شروع به تنزل می‌کند و مدل overfit می‌شود و الگوهایی را می‌آموزد که مخصوص داده‌های آموزشی است اما ارتباط درستی با مسئله مورد نظر ندارد و گمراه‌کننده است. این موضوع باعث عملکرد ضعیف آن روی داده‌های test می‌شود.

سوال ۲:

- الف) شبکهٔ عصبی و مجموعه داده‌های زیر را در نظر بگیرید که قرار است در آن قیمت x ها را پیش‌بینی کنیم. نمی‌خواهیم این شبکه بر روی مجموعه دادهٔ داده شده overfit شود. می‌خواهیم با اعمال Regularization term ،یادگیری داده‌ها را برای آن سخت کنیم. با انتخاب یک نمونه داده به‌دلخواه، ابتدا آن را در شبکهٔ عصبی منتشر کنید (forward feed) و سپس هنگام به‌روزرسانی پارامترهای شبکه (propagation back) به‌روزرسانی را با regularization L2 و $\lambda = 0.9$ انجام دهید. به خاطر داشته باشید که این کار باید به‌صورت دستی انجام گیرد و سعی کنید مراحل را کامل ذکر کنید. همچنین تنها یک دور کافی است. (۱۵ نمره)





در این شبکه تابع فعال‌ساز خطی (linear) و نرخ یادگیری ۰.۱ است. تابع خطا را از نوع MSE به فرمول زیر در نظر بگیرید:

$$loss = \frac{1}{2m}\sum_{i=1}^{m}(y-\hat{y})^2$$

- ب) با تغییر مقدار λ چه تغییراتی در میزان خطای تولید شده و نحوه‌ی آموزش پارامترهای شبکه‌های عصبی به وجود خواهد آمد؟ (۱۰ نمره)

پاسخ ۲:

- الف)

  - Forward pass:

    $z = w_1 i_1 + w_2 i_2 + w_3 i_3 + w_4 i_4 + w_5 i_5$

    $\hat{y} = z$

    $L(y,z) = (y-z)^2 + \frac{\lambda}{2}(\sum_{i=1}^{5} w_i^2)$

  - Backward pass:

    $\frac{\partial L}{\partial z} = -2(y-z)$

    $\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial z}.\frac{\partial z}{\partial w_1} = -2i_1(y-z)$

    $\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial z}.\frac{\partial z}{\partial w_2} = -2i_2(y-z)$

    $\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial z}.\frac{\partial z}{\partial w_3} = -2i_3(y-z)$

    $\frac{\partial L}{\partial w_4} = \frac{\partial L}{\partial z}.\frac{\partial z}{\partial w_4} = -2i_4(y-z)$

    $\frac{\partial L}{\partial w_5} = \frac{\partial L}{\partial z}.\frac{\partial z}{\partial w_5} = -2i_5(y-z)$

  - Update parameters with GD:

    $w_1 = (1-\eta\lambda)w_1 - \eta\frac{\partial L}{\partial w_1}$

    $w_2 = (1-\eta\lambda)w_2 - \eta\frac{\partial L}{\partial w_2}$

    $w_3 = (1-\eta\lambda)w_3 - \eta\frac{\partial L}{\partial w_3}$

    $w_4 = (1-\eta\lambda)w_4 - \eta\frac{\partial L}{\partial w_4}$

    $w_5 = (1-\eta\lambda)w_5 - \eta\frac{\partial L}{\partial w_5}$

Data 1 (x=1 , y=20):

- Forward pass:

  $z = (1 * 1) + (2 * 1) + (3 * (1)^2) + (-2 * (1)^3) + (-1 * (1)^4) = 3$

  $\hat{y} = 3$

  $L(y, z) = (20-3)^2 + \frac{0.9}{2}((1)^2+(2)^2+(3)^2+(-2)^2+(-1)^2) = 297.55$

- Backward pass:

  $\frac{\partial L}{\partial z} = -2(20 - 3) = -34$

  $\frac{\partial L}{\partial w_1} = -2(1)(20 - 3) = -34$

  $\frac{\partial L}{\partial w_2} = -2(1)(20 - 3) = -34$

  $\frac{\partial L}{\partial w_3} = -2(1)^2(20 - 3) = -34$

  $\frac{\partial L}{\partial w_4} = -2(1)^3(20 - 3) = -34$

  $\frac{\partial L}{\partial w_5} = -2(1)^4(20 - 3) = -34$

- Update parameters with GD:

  $w_1 = (1 - (0.1 * 0.9))1 - 0.1(-34) = 4.31$

  $w_2 = (1 - (0.1 * 0.9))2 - 0.1(-34) = 5.22$

  $w_3 = (1 - (0.1 * 0.9))3 - 0.1(-34) = 6.13$

  $w_4 = (1 - (0.1 * 0.9))(-2) - 0.1(-34) = 1.58$

  $w_5 = (1 - (0.1 * 0.9))(-1) - 0.1(-34) = 2.49$

بنابراین، با توجه به محاسبات بالا، وزن‌های نهایی شبکه به حالت زیر درمی‌آیند:

$w_1 = 4.31$

$w_2 = 5.22$

$w_3 = 6.13$

$w_4 = 1.58$

$w_5 = 2.49$

- ب) طبق فرمول، هر چه مقدار λ بیشتر باشد، میزان خطا بیشتر افزایش پیدا می‌کند و وزن‌ها

به مقدار بیشتری تغییر می‌کنند؛ اما اگر λ کوچک‌تر باشد، میزان خطا کمتر افزایش پیدا کرده و وزن‌ها با نرخ کمتری تغییر می‌کنند.
همچنین افزایش λ ممکن است واریانس بالا (نشانه‌ای از overfitting) را با تشویق وزن‌های کوچک‌تر برطرف کند، که منجر به یک نمودار مرزی تصمیم‌گیری می‌شود که با انحنای کمتر ظاهر می‌شود. به طور مشابه، کاهش λ ممکن است با تشویق وزن‌های بزرگ‌تر، بایاس بالا (نشانه‌ای از underfitting) را برطرف کند، که به طور بالقوه منجر به مرز تصمیم‌گیری پیچیده‌تر می‌شود.

سوال ۳:

- الف) یکی از راه‌کارهای برای متعادل‌سازی پیچیدگی میان مدل و داده‌ها برای حالتی که مدل پیچیده‌تر از داده‌ها است، افزایش داده‌های آموزشی است. با این کار به مدل این فرصت داده می‌شود تا با دیدن حاالت مختلف داده‌های آموزشی بتواند تعمیم‌پذیری بیشتری داشته باشد و در نتیجه بهتر آموزش ببیند. اما این راهکار دارای چالش‌هایی است. یکی از چالش‌های اساسی این است که توزیع داده‌ها به هم نخورد هم چنین، تعداد کمی دادهٔ برچسب‌دار دررابطه با بسیاری از مسائل وجود دارند که باعث می‌شود فرایند افزایش تعداد داده‌های آموزشی، فرایندی بسیار هزینه بر هم از نظر زمانی و هم از نظر مالی باشد. راهکارهای data augmentation برای حل این مشکل در افزایش داده‌ها با هزینهٔ کمتر ارائه شدند. ما در DataAug.ipynb قرار است که برای مجموعهٔ تصاویر داده شده، عملیات data augmentation را انجام دهیم و سپس نتیجهٔ آن را ببینیم. برای این کار ابتدا یک مدل MLP داده شده است که باید بر روی داده‌های خام آموزشش دهید. سپس مجموعهٔ دادهٔ augmented را ایجاد کرده و همان مدل را بر روی مجموعهٔ دادهٔ جدید دوباره از اول آموزش دهید. برای انجام این تمرین مراحل موجود در DataAug.ipynb را انجام دهید و آن را تکمیل کنید. تحلیل خود از نتایج به‌دست‌آمده برای دو آموزش مدل یکسان با مجموعه داده‌های متفاوت را بنویسید. دربارهٔ overfit شدن یا نشدن مدل‌ها صحبت کنید. دلیل خود از استدلال‌هایتان را نیز بیاورید. (۲۵ نمره)
- ب) (امتیازی) با استفاده از کتابخانهٔ keras مراحل مربوط به data augmentation را انجام دهید و مجموعهٔ دادهٔ تولید شدهٔ خود را با مجموعهٔ دادهٔ قبلی مقایسه کنید. (۱۵ نمره)

پاسخ ۳:

- الف) در ابتدا، فایل zip داده شده را با دستور "unzip <path>" از حالت فشرده خارج می‌کنیم. سپس تصاویر را از directory خوانده و در یک لیست آن‌ها را ذخیره می‌کنیم. همچنین برای تعیین کلاس آن‌ها، به تصاویر گربه بیت ۰، و به تصاویر سگ بیت ۱ را اختصاص می‌دهیم.

```
# First we should read all the images from the provided directory,

# 1. first unzip the .zip file attached with homeworks
!unzip "drive/MyDrive/data/HW3.zip"

# 2. read images from directory
images = []
dataset = os.listdir("./data_aug")

# 3. create labels array
labels=[]
for image in dataset:
    images.append(cv2.imread(os.path.join("./data_aug", image)))
    label = image.split('.')[0]
    if label == 'cat':
        labels.append(0)
    else:
        labels.append(1)
```

حال تصاویر را با تقسیم کردن بر ۲۵۵، نرمالیزه می‌کنیم تا عددی بین ۰ و ۱ به ما بدهد.

```
[ ]  # Normalize your dataset in a way to have a image dataset with pixles in range (0 to 1)

     images = np.divide(images, 255)
```

با استفاده از دستور cv2.resize، تصاویر خود را به سایز (۲۸ * ۲۸) در می‌آوریم.

```
[ ]  # Resize your images to (28 * 28)

     x_train = []
     for image in images:
       x_train.append(cv2.resize(image, (28, 28)))
```

گام بعد، flatten کردن x_train و y_train می‌باشد که چون از نوع list هستند، باید ابتدا به array تبدیل کرده و بعد flatten کنیم.

```
# Flatten your dataset images

x_train = np.array(x_train)
y_train = np.array(labels)
x_train = x_train.flatten().reshape(100, 2352)
y_train = y_train.flatten()
```

برای چک کردن سایز x_train و y_train، از دستور shape استفاده کرده و سایز آن‌ها را پرینت می‌کنیم.

```
# Check your dataset and labels to be a numpy ndarray of sizes (100, 28 * 28 * 3) and (100, 1) respectively.

print(x_train.shape)
print(y_train.shape)
```

```
(100, 2352)
(100,)
```

مدل خود را به صورت sequential تعریف می‌کنیم.

```
# create a simple model of Keras MLP with desired number of dense layers and units and activation function
# don't forget to shuffle the data
# this is a binary classification problem

model_temp_1 = Sequential()
model_temp_1.add(Dense(512, input_shape=(2352, ), activation='relu'))
model_temp_1.add(Dense(16, activation='relu'))
model_temp_1.add(Dense(1, activation='sigmoid'))
```

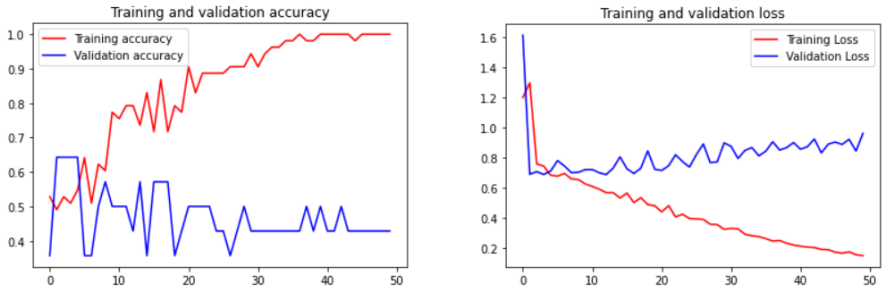
سپس با استفاده از دستور train_test_split دیتاست خود را به داده‌های test و train تقسیم کردیم. ورودی‌ها این تابع به ترتیب، داده، label مربوط به آن و نسبت test به کل داده می‌باشد که در اینجا ۰.۳۳ را به عنوان داده test و مابقی را برای train در نظر گرفتیم.

```
# Split your dataset to test and train with your desired implementation
# use sklearn.model_selection.train_test_split

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x_train, y_train, test_size=0.33)
```

حال مدل خود را compile و fit می‌کنیم. چون در تمرین قبل (HW2) دیدیم که optimizer Adam عملکرد بهتری نسبت به سایر بهینه‌سازها داشت، در اینجا از Adam به عنوان optimizer استفاده کردیم و به دلیل دو کلاسه بودن، از loss binary_crossentropy بهره بردیم.

```python
# Compile with your desired optimizer and loss function with 'accuracy' as metric
# # Don't forget to monitor the overfitting, you should take care of val_accuracy and val_loss

model_temp_1.compile(
    loss='binary_crossentropy',
    optimizer='adam',
    metrics=['accuracy']
)
history = model_temp_1.fit(
      x_train,
      y_train,
      epochs=50,
      validation_split=0.2,
      shuffle=True,
      verbose=2
)
```

با plot کردن مقادیر loss و accuracy داریم:

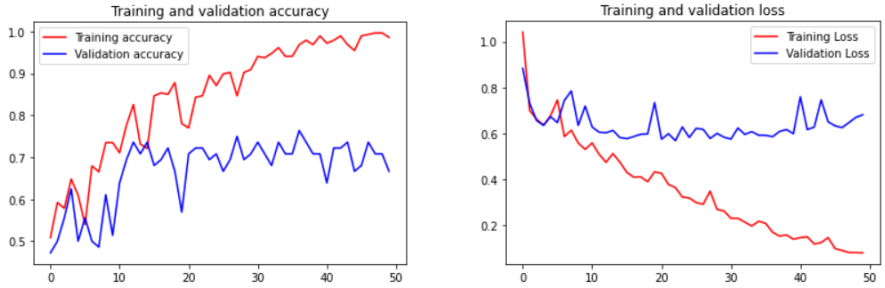


همانطور که در تصاویر بالا نیز مشخص است، مدل overfit شده است. طبق نمودار، مقدار accuracy برای داده‌های train به تدریج افزایش می‌یابد و loss آن کم می‌شود؛ اما مشکل آن‌جایی وجود دارد که روی داده‌های validation، عملکرد خوبی ندارد و دقت آن دچار تنزل می‌شود و loss آن نیز افزایش می‌یابد. از دلایل این اتفاق، می‌توان به کم بودن تعداد dataset، زیاد بودن ابعاد شبکه که موجب حفظ جزئیات داده‌های train شده است و train بیش از حد اشاره کرد. در انتها، مدل خود را evaluate می‌کنیم.

```python
[ ]  # Evaluate your model on the test data

    test_evaluate = model_temp_1.evaluate(x_test, y_test)
    print("Test evaluation is: ", test_evaluate)
```

```
2/2 [==============================] - 0s 7ms/step - loss: 1.0000 - accuracy: 0.4242
Test evaluation is:  [0.9999938607215881, 0.42424243688583374]
```

حال، از توابع تعریف شده استفاده کرده و دیتاست خود را افزایش می‌دهیم که ۸ برابر می‌شود. سپس مراحل قبل را تکرار کرده و accuracy و نمودارهای loss را plot می‌کنیم.



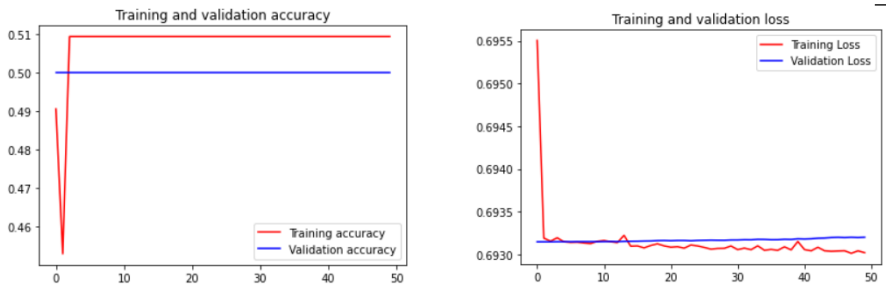
همانطور که در از نمودارها نیز مشخص است، مقدار دقت برای داده‌های validation افزایش یافت و loss آن نیز نسبت به حالت قبل کمتر می‌باشد. این موضوع نشان می‌دهد با انجام این کار (داده افزایی) تا حدی جلوی overfitting را گرفتیم و برای جلوگیری بیشتر از این اتفاق، باید عوامل دیگر نظیر Regularizaton، Dropout و غیره را نیز لحاظ کرد. در انتها مدل جدید خود را evaluate می‌کنیم.

```
# Evaluate your model on the test data

test_evaluate_2 = model_temp_2.evaluate(x_test_2, y_test_2)
print("Test evaluation is: ", test_evaluate_2)
```

```
6/6 [==============================] - 0s 6ms/step - loss: 0.6737 - accuracy: 0.6780
Test evaluation is:  [0.673651933670044, 0.6779661178588867]
```

از مقایسه نتیجه این بخش برای این مدل با نتیجه حاصل برای مدل قبل نیز، می‌بینیم که مقدار دقت ۰.۲۵ افزایش و loss، ۰.۳۲ کاهش یافته است.

- ب) حال با استفاده از توابع آماده keras، مراحل قبل را انجام می‌دهیم. با plot کردن accuracy و loss داریم:



همانطور که از نمودارهای بالا نیز مشخص است، در این حالت بهتر از حالت قبل جلوی overfitting گرفته شد؛ اما نکته‌ای که وجود دارد، این است که مقادیر دقت و loss تقریبا کاهش یافته است. حال مدل کنونی خود را evaluate می‌کنیم.

```
test_evaluate_3 = model_temp_3.evaluate(x_test_3, y_test_3)
print("Test evaluation is: ", test_evaluate_3)
```

```
2/2 [==============================] - 0s 9ms/step - loss: 0.6935 - accuracy: 0.4848
Test evaluation is:  [0.6935147047042847, 0.4848484992980957]
```

همانطور که مشاهده می‌شود، مقدار loss و accuracy نسبت به حالت قبل، کاهش یافته است؛ اما نسبت به حالت اولیه، loss ۰.۳ کاهش و دقت حدود ۰.۱ افزایش داشته است. بنابراین نتیجه می‌گیریم مزیت استفاده از این روش نسبت به روش قبلی، کاهش بیشتر overfitting و از معایب آن نسبت به روش قبل، می‌توان به کمتر شدن accuracy و افزایش loss اشاره کرد.

(قابل ذکر است که مقدار accuracy و loss را با مدل قبل (mode_temp_2) مقایسه کردیم؛ مگرنه نسبت به مدل اولیه شاهد افزایش accuracy و کاهش loss بودیم.)

سوال ۴:

در MLP_Overfit.ipynb یک مدل MLP و یک مجموعه‌داده به شما داده شده است. این مدل بر روی مجموعهٔ دادهٔ داده شده، overfit شده است. با استفاده از ابزارهای موجود در کتابخانهٔ keras سعی کنید از overfit شدن آن جلوگیری کنید. در نهایت نتایج به‌دست آمدهٔ خود را گزارش کرده و تحلیل کنید. (۲۵ نمره) (حتماً با دلیل ذکر کنید که چرا مدل overfit بود و چرا با تغییرات انجام داده شده مدل در راستای حل آن گام برداشته است)

(توجه: استفاده از regularization و dropout مورد انتظار اما به راهکارهای اضافه‌تر که بتوانند نتایج بهتری نیز ثبت کنند نمرهٔ امتیازی تعلق می‌گیرد.)

پاسخ ۴:

در ابتدا مدل خود را به صورت sequential طبق توضیحات داده شده، با ۳ لایه مخفی که هر کدام دارای ۵۰ unit و relu activation function هستند و یک لایه خروجی حاوی ۱ unit و sigmoid activation function می‌سازیم.

```python
# Create a Sequential MLP model with these Dense layers:
# 3 hidden layers with 50 units each and 'relu' activation
# 1 unit output with 'sigmoid' activation

model = Sequential()

# Hidden Layer
model.add(Dense(units=50, activation="relu"))
model.add(Dense(units=50, activation="relu"))
model.add(Dense(units=50, activation="relu"))

# Output Layer
model.add(Dense(units=1, activation="sigmoid"))
```

سپس با استفاده از دستور train_test_split دیتاست خود را به داده‌های test و train تقسیم کردیم. ورودی‌ها این تابع به ترتیب، داده، label مربوط به آن و نسبت test به train می‌باشد که در اینجا ۰.۳۳ را به عنوان داده test و مابقی را برای train در نظر گرفتیم.

```python
# Split your dataset to test and train with your desired implementation
# use sklearn.model_selection.train_test_split

X_train, X_test, y_train, y_test = train_test_split(sgx, sgy, test_size=0.33)
```

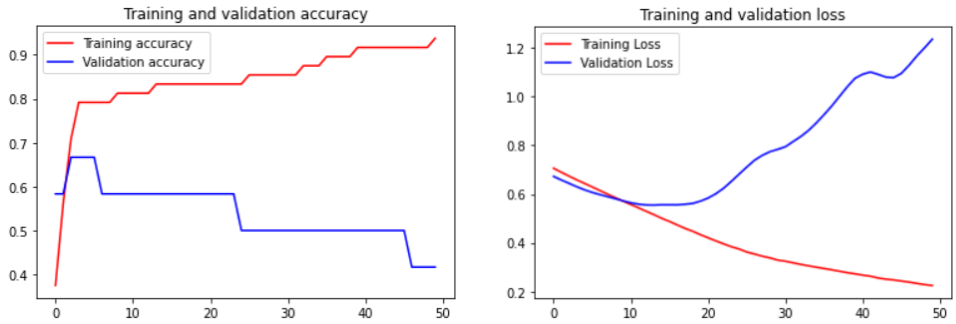
در ادامه مدل خود را با استفاده از BinaryCrossentropy loss function و Adam optimizer با learning rate = 0.1، compile می‌کنیم. (طبق توضیحات داده شده، metrics را accuracy می‌گذاریم)

```python
[95] # Compile with your desired optimizer and loss function with 'accuracy' as metric
     # Don't forget to monitor the overfitting, you should take care of val_accuracy and val_loss

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

حال مدل خود را fit کرده و validation_split را ۰.۲ (طبق گفته سوال) قرار می‌دهیم.

```python
# Start training and fit your model with desired parameters and validation_split=0.2

history = model.fit(
    x = sgx,
    y = sgy,
    epochs = 50,
    validation_split = 0.2,
    verbose=0,
)
```

با plot کردن مقادیر loss و accuracy داریم:



همانطور که در تصاویر بالا نیز مشخص است، مدل overfit شده است. طبق نمودار، مقدار accuracy برای داده‌های train به تدریج افزایش می‌یابد و loss آن کم می‌شود؛ اما مشکل آن‌جایی وجود دارد که روی داده‌های validation، عملکرد خوبی ندارد و دقت آن دچار تنزل می‌شود و loss آن نیز افزایش می‌یابد. از دلایل این اتفاق، می‌توان به کم بودن تعداد dataset، زیاد بودن ابعاد شبکه که موجب حفظ جزئیات داده‌های train شده است و train بیش از حد اشاره کرد. در انتها، مدل خود را evaluate می‌کنیم.

```
[98] # Evaluate your model with train and test data

     train_evaluate = model.evaluate(x_train, y_train)
     print("Train evaluation is: ", train_evaluate)

     test_evaluate = model.evaluate(x_test, y_test)
     print("Test evaluation is: ", test_evaluate)
```

```
2/2 [==============================] - 0s 7ms/step - loss: 0.3719 - accuracy: 0.8500
Train evaluation is:  [0.3718905448913574, 0.8500000238418579]
1/1 [==============================] - 0s 24ms/step - loss: 0.5290 - accuracy: 0.8000
Test evaluation is:  [0.5290249586105347, 0.800000011920929]
```

برای بهبود مدل و کاهش overfitting، می‌توانیم از روش‌های Dropout، L_1، L_2، اضافه کردن تعداد نمونه‌های موجود در دیتاست، کم کردن نورون‌های هر لایه و کم کردن ابعاد شبکه (تعداد لایه‌ها) اشاره کرد. برای کم کردن overfitting تمامی این روش‌ها را اعمال می‌کنیم. در ابتدا hyperparameterها را برای L_1، L_2 و Dropout تعریف می‌کنیم.

```
[4]  l1 = 0.0005
     l2 = 0.005
     p = 0.5
```

سپس به منظور جلوگیری از overfitting، تعداد نمونه‌ها را افزایش می‌دهیم؛ زیرا یکی از دلایل بروز overfitting، کم بودن تعداد نمونه‌های موجود در دیتاست می‌باشد.

```
sgx, sgy = make_sample(200)

plt.scatter(sgx[:,0], sgx[:,1], alpha=0.5, c=sgy)
plt.xlabel('x1')
plt.ylabel('x2')
```

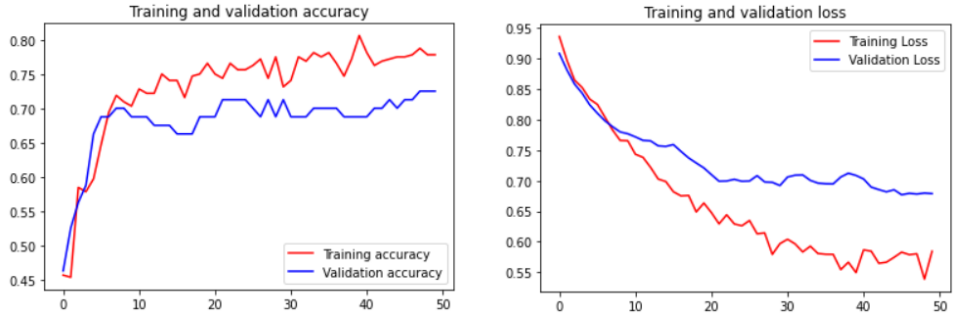
در ادامه با کم کردن ابعاد شبکه (تغییر تعداد لایه‌ها از ۴ به ۳) و کم کردن نورون‌های موجود در هر لایه، به کاهش overfitting کمک می‌کنیم.

```
model = Sequential([
  Dense(32, activation='relu', kernel_regularizer=L1L2(l1, l2)),
  Dropout(p),
  Dense(16, activation='relu', kernel_regularizer=L1L2(l1, l2)),
  Dropout(p),
  Dense(1, activation='sigmoid', kernel_regularizer=L1L2(l1, l2)),
])
```

این‌بار، بخش کوچک‌تری از دیتاست را برای test جدا می‌کنیم.

```
[71] x_train, x_test, y_train, y_test = train_test_split(sgx, sgy, test_size=0.1)
```

سپس مدل خود را compile و fit کرده و نتایج accuracy و loss آن را مشاهده می‌کنیم.



همانطور که در تصاویر بالا نیز مشخص است، مقادیر loss و accuracy برای داده‌های validation تا حد زیادی بهبود پیدا کرد و دقت و loss برای داده‌های train و validation نسبتا همگرا شد که نشان می‌دهد مدل ما از حالت overfitting تقریبا خارج شده است.

با اضافه کردن تعداد نمونه‌ها، باعث شدیم تا مدلمان روی یک سری داده محدود fit نشود و با استفاده

از نمونه‌های بیشتری تصمیم‌گیری کند. با کم کردن تعداد لایه‌ها و نورون‌ها، موجب کاهش ظرفیت یادگیری مدل شدیم تا فقط ویژگی‌های برجسته را حفظ کند و به جزئیات کاری نداشته باشد. با استفاده از جریمه کردن پارامترها (L_1 و L_2) باعث محدود کردن پارامترهای مدل، به منظور محدود کردن ظرفیت یادگیری شدیم. Dropout نیز به نوعی با استفاده از داده افزایی از overfitting جلوگیری می‌کند. سپس با کاهش تعداد نمونه‌های انتخابی برای test و اختصاص بخش بزرگ‌تر برای train باز هم موجب داده افزایی شدیم.

با evaluate کردن مدل جدیدمان، مطابق انتظار، مقدار accuracy برای داده test افزایش و مقدار loss آن کاهش می‌یابد که نشان می‌دهد خوب عمل کردیم.

```
train_evaluate = model.evaluate(x_train, y_train)
print("Train evaluation is: ", train_evaluate)

test_evaluate = model.evaluate(x_test, y_test)
print("Test evaluation is: ", test_evaluate)
```

```
12/12 [==============================] - 0s 2ms/step - loss: 0.5536 - accuracy: 0.7833
Train evaluation is:  [0.5535984635353088, 0.7833333611488342]
2/2 [==============================] - 0s 6ms/step - loss: 0.4980 - accuracy: 0.8500
Test evaluation is:  [0.49796637892723083, 0.8500000238418579]
```