به نام خدا

نام: فاطمه زهرا بخشنده **استاد: دکتر مرضیه داود آبادی**

شماره دانشجویی: 98522157 **درس مبانی یادگیری عمیق**

گزارش تمرین 3:

سوال اول:

الف) Overfit زمانی اتفاق می افتد که یک مدل دیتای آموزشی را به خوبی یاد گرفته و در واقع حفظ می کند، اما دقت آن روی داده validation و test خیلی خوب نیست و با دقت آن روی داده train فاصله زیادی دارد (واریانس بالایی داریم). در واقع مدل جزئیات داده آموزشی را حفظ می کند و سعی کرده روندی را در داده ها پیش بینی کند که بیش از حد نویزی است و مختص داده آموزشی است. اما این قضیه روی عملکرد مدل تاثیر منفی دارد چون واقعیت موجود در همه داده ها را منعکس نمی کند و عمومیت آن کم است. overfit شدن یک شبکه می تواند یک سری دلیل داشته باشد. مثلا:

- سایز دیتاست کم است.
- دیتا های آموزشی cleaned نیستند و نویزی هستند.
- تعداد لایه ها و نورون ها زیاد هستند و شبکه برای مسئله ما بیش از حد پیچیده است.

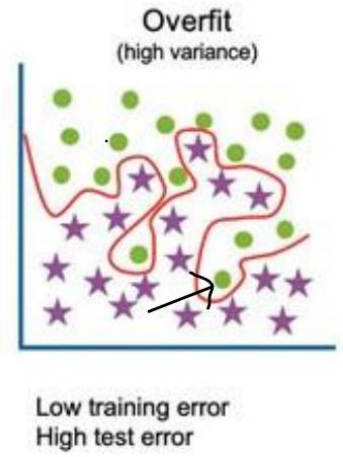
اگر شبکه ما دچار overfit شود و واریانس بالایی داشته باشیم می‌توانیم از روش های زیر استفاده کنیم:

- استفاده از دیتای بیتشر
- Regularization، که خود، روش های متفاوتی را شامل می شود:
  - استفاده از روش جریمه L1 یا L2 که قبل از انجام به روزرسانی معمولی مبتنی بر گرادیان، بردار وزن را در هر گام با یک ضریب ثابت کاهش می دهند. و منجر به اضافه کردن هزینه به Loss function برای آپدیت وزن ها می شود و وزن ها کوچکتر می شوند. (Weight decay). وزن هایی که تغییر کمتری در تابع ضرر ایجاد می کنند، اهمیت کمتری دارند و بیشتر کاهش می یابند.
  - استفاده از Drop out که در هر iteration نود های مختلفی صفر می شوند و در واقع در هر iteration شبکه ما به یک صورت متفاوت کوچکتر می شود. (در موقع test، Drop out نداریم).

- Data augmentation، یعنی ایجاد تغییراتی در دیتا مثلا چرخاندن یا زاویه دادن تصاویر، flip، افزودن نویز و ...، و اضافه کردن این دیتاها به دیتاست.
- Early stopping: یک جایی زودتر تا w ها خیلی بزرگتر نشده اند و شبکه overfit نشده است learning را متوقف کنیم. اما مشکل این کار این است که دو هدف optimize کردن cost function و overfit نشدن را باهم mix کرده ایم و دیگر به صورت مستقل آن ها را انجام نمی دهیم.

ب) در شکل شماره 1 واریانس بالایی داریم. در واقع همان مشکل overfitting روی داده های آموزشی اتفاق افتاده است. یعنی در واقع مدل جزئیات داده آموزشی را حفظ کرده و روندی را در داده ها پیش بینی کرده که بیش از حد نویزی است و مختص داده آموزشی است. اما این قضیه روی عملکرد مدل تاثیر منفی گذاشته چون واقعیت موجود در همه داده ها را منعکس نمی کند و تعمیم دهی خوبی ندارد.

برای مثال داده ای که در شکل زیر به آن اشاره کردم نویزی است و دارای الگویی است که مخصوص داده آموزشی بوده و ارتباطی با مسئله اصلی ندارد و گمراه کننده است.



اگر این داده که به آن اشاره کردم وجود نداشت، مرز تصمیم عوض میشد، پس در واقع مدل ما پیچیده بوده و خیلی حساس به داده های training شده است، و با عوض شدن تنها چند داده، شکل مرز ممکن است خیلی عوض شود. پس واریانس بالاست. حالا اگر در داده validation یا test یک داده ستاره در کنار این دایره نویزی سبز وجود داشته باشد، بخاطر مرز تصمیم حساس به داده آموزشی، این داده را به عنوان داده دایره معرفی می‌کند که اشتباه است، در صورتی که اگر مدل ما مثل مدل شماره 2 بود و قابلیت تعمیم آن خوب بود، این داده را درست پیش‌بینی می‌کرد، به همین دلیل این مدل برای داده هایی که تاکنون ندیده است، خطای بیشتری نسبت به مدل شکل شماره 2 تولید خواهد کرد.

منابع: لینک

سوال دوم:

الف)

Forward Propagation:

$z = w_0 + w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4$

$\hat{y} = linear(z) = z$

$\Omega(w) = \frac{1}{2}||w||_2^2$

$L = \frac{1}{2m}\sum_1^m (y - \hat{y})^2 + \frac{\lambda}{2} w^T w = \frac{1}{2}(\hat{y} - y)^2 + \frac{\lambda}{2} w^T w \rightarrow Update\ using\ one\ example$

Backward Propagation:

$\frac{\partial L}{\partial \hat{y}} = -(y - \hat{y}),\ \frac{\partial \hat{y}}{\partial z} = = 1,\ \frac{\partial z}{\partial w_0} = 1,\ \frac{\partial z}{\partial w_1} = x,\ \frac{\partial z}{\partial w_2} = x^2,\ \frac{\partial z}{\partial w_3} = x^3,\ \frac{\partial z}{\partial w_4} = x^4$

$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial z} \times \frac{\partial z}{\partial w} + \lambda w$

$\frac{\partial L}{\partial w_0} = (\hat{y} - y) + \lambda w_0,\ \frac{\partial L}{\partial w_1} = (\hat{y} - y).x + \lambda w_1,$

$\frac{\partial L}{\partial w_2} = (\hat{y} - y).x^2 + \lambda w_2,\ \frac{\partial L}{\partial w_3} = (\hat{y} - y).x^3 + \lambda w_3,\ \frac{\partial L}{\partial w_4} = (\hat{y} - y).x^4 + \lambda w_4$

$w_0 = w_0 - \alpha\left(\frac{\partial L}{\partial w_0}\right),\ w_1 = w_1 - \alpha\left(\frac{\partial L}{\partial w_1}\right),$

$w_2 = w_2 - \alpha(\frac{\partial L}{\partial w_2}),\ w_3 = w_3 - \alpha(\frac{\partial L}{\partial w_3}),\ w_4 = w_4 - \alpha(\frac{\partial L}{\partial w_4})$

$\alpha = 0.1,\ \lambda = 0.9,\ w_0 = 1,\ w_1 = 2,\ w_2 = 3,\ w_3 = -2,\ w_4 = -1$

فرض می کنیم اندازه یک mini batch را برابر 1 گرفتیم.

Epoch 1:

Forward Propagation minibatch 1:

$data = (2, 35)$

$z = 1 \times 1 + 2 \times 2 + 3 \times 4 - 2 \times 8 - 1 \times 16 = -15$

$\hat{y} = z = -15$

$$L = \frac{1}{2}(-15-35)^2 + \frac{0.9}{2}(1+4+9+4+1) = 1250 + 8.55 = 1258.55$$

Backward Propagation minibatch 1:

$$\frac{\partial L}{\partial w_0} = (-15-35) + 0.9 \times 1 = -50 + 0.9 = -49.1$$

$$\frac{\partial L}{\partial w_1} = (-15-35) \times 2 + 0.9 \times 2 = -98.2$$

$$\frac{\partial L}{\partial w_2} = (-15-35) \times 4 + 0.9 \times 3 = -197.3$$

$$\frac{\partial L}{\partial w_3} = (-15-35) \times 8 + 0.9 \times -2 = -401.8$$

$$\frac{\partial L}{\partial w_4} = (-15-35) \times 16 + 0.9 \times -1 = -800.9$$

$$w_0 = 1 - 0.1 \times -49.1 = 5.91$$

$$w_1 = 2 - 0.1 \times -98.2 = 11.82$$

$$w_2 = 3 - 0.1 \times -197.3 = 22.73$$

$$w_3 = -2 - 0.1 \times -401.8 = 38.18$$

$$w_4 = -1 - 0.1 \times -800.9 = 79.09$$

ب) با توجه به فرمول زیر با کاهش مقدار λ مقدار جریمه وزن ها کاهش یافته، و با افزایش λ مقدار جریمه وزن ها افزایش می یابد در نتیجه وزن ها کوچکتر خواهند شد.

$$\Omega(\boldsymbol{w}) = \frac{1}{2}\|\boldsymbol{w}\|_2^2 \qquad \nabla_{\boldsymbol{w}} \tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = \alpha \boldsymbol{w} + \nabla_{\boldsymbol{w}} J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y})$$

$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = \frac{\alpha}{2}\boldsymbol{w}^T\boldsymbol{w} + J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) \qquad \boldsymbol{w} \leftarrow \boldsymbol{w} - \epsilon\big(\alpha \boldsymbol{w} + \nabla_{\boldsymbol{w}} J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y})\big)$$

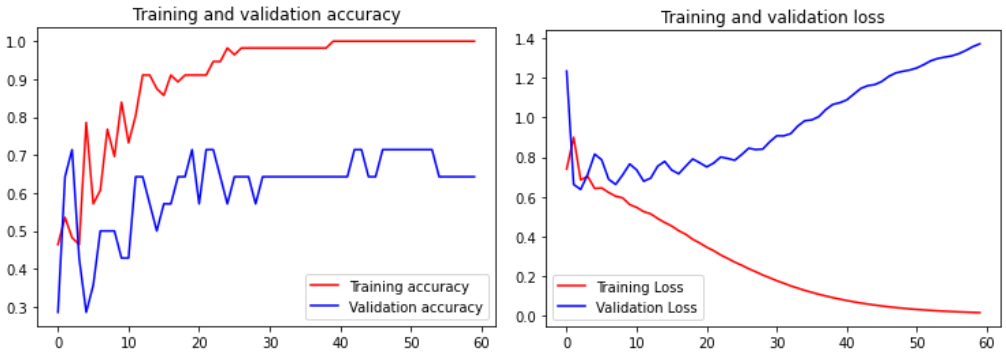
سوال سوم:

الف) ابتدا فایل دیتاست را در کولب آپلود کرده و دیتا ها را unzip می کنیم. تصاویر را می خوانیم و label ها را مشخص می کنیم. چند تا از دیتا ها را plot می کنیم که به صورت زیر هستند:

مراحل را طبق نوت های موجود در کولب انجام می دهیم و دیتاست را آماده می کنیم. مدل را میسازیم:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 128)               301184

 dense_1 (Dense)             (None, 64)                8256

 dense_2 (Dense)             (None, 1)                 65

=================================================================
Total params: 309,505
Trainable params: 309,505
Non-trainable params: 0
_________________________________________________________________
```

مدل را Compile و fit می کنیم. نتایج بدست آمده از train:

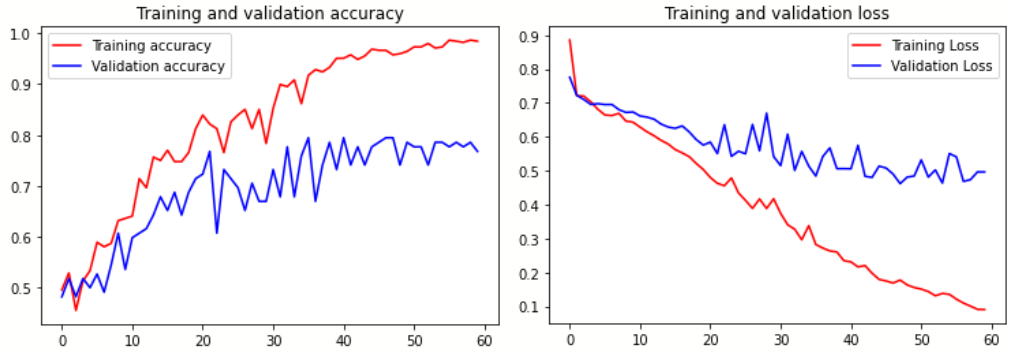


مدل را روی داده train و test، evaluate می کنیم. نتیجه:

```
Train Loss: 0.28661924600601196, Train Accuracy: 0.9285714030265808
Test Loss: 2.117107629776001, Test Accuracy: 0.5333333611488342
```

می بینیم که مدل روی داده train، overfit شده است. دقت آن روی داده train خیلی بهتر از دقت آن روی داده validation و test است.

حالا با استفاده از توابع موجود، data augmentation را انجام می دهیم. به این صورت که به ازای هر تصویر موجود در دیتاست اولیه، 7 تصویر دیگر با اعمال horizontal_shift و vertical_shift و تغییر brightness و zoom و channel_shift و horizontal_flip و vertical_flip می سازیم و خود تصویر و این 7 تصویر را به دیتاست جدید اضافه می کنیم. Label هارا نیز درست می کنیم. و دیتاست جدید را طبق مراحل آماده می کنیم. مدل را مانند مدل قبلی ساخته و روی دیتاست جدید با 800 دیتا fit می کنیم.
نتایج بدست آمده از train:



مدل را روی داده train و test جدید، evaluate می کنیم. نتیجه:

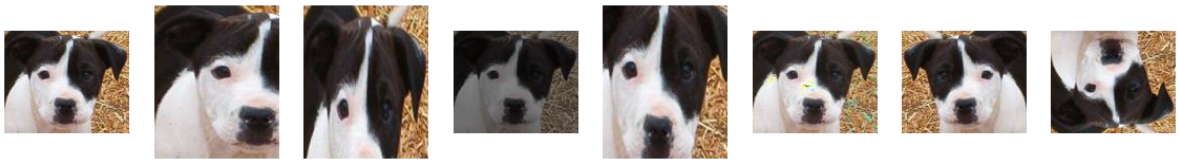
```
Train Loss: 0.17803941667079926, Train Accuracy: 0.9410714507102966
Test Loss: 0.47091227769851685, Test Accuracy: 0.7875000238418579
```

می بینیم که test accuracy از 53% به 79% رسید. val accuracy نیز از 64% به 78% رسید. یعنی با انجام data augmentation توانستیم تا حد خوبی پیشرفت کرده و از overfitting جلوگیری کنیم.

در واقع در مدل اول پس از چند تکرار، بهبود تعمیم‌دهی متوقف می‌شود و سپس شروع به تنزل می‌کند. مدل شروع به overfit شدن می‌کند و الگوهایی را می‌آموزد که مخصوص داده‌های آموزشی است اما ارتباط درستی با مسئله مورد نظر ندارد و گمراه‌کننده است. بهترین راه برای افزایش قدرت تعمیم دهی یک الگوریتم یادگیری ماشین با ظرفیت یادگیری بالا، آموزش آن بر روی داده های بیشتر است. جمع آوری داده معمولا فرآیند دشوار و خسته کننده ای است. می توانیم داده های ساختگی بسازیم و به داده های آموزشی اضافه کنیم. در اینجا نیز از این روش استفاده کردیم و نتیجه خوبی گرفتیم. اگر روش های دیگری مانند dropout نیز روی این مدل استفاده کنیم به نتایج بهتری نیز می توان رسید.

بخش امتیازی: با استفاده از کتابخانه keras مشابه کاری که در بخش قبل انجام داده ایم را روی دیتاست اولیه اعمال می کنیم. در واقع به ازای کل تصاویر دیتاست اولیه، 7 تصویر دیگر با اعمال horizontal_shift و vertical_shift و تغییر brightness و zoom و channel_shift و horizontal_flip و vertical_flip ایجاد کرده و ذخیره می کنیم. در آخر مانند روش قبلی 800 تصویر داریم. نتیجه روی یکی از تصویرهای دیتاست:

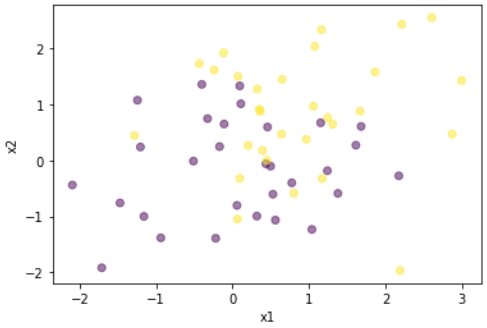
نتیجه روش قبلی روی یکی از تصاویر(قبل از resize کردن تصاویر دیتاست) :



منابع: لینک و لینک

سوال چهارم:

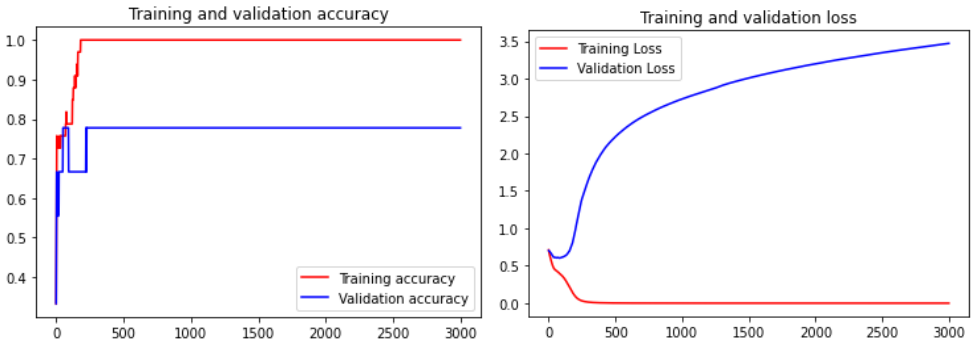
ابتدا 60 داده زیر را داریم:



مدل را میسازیم.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 50)                150

 dense_1 (Dense)             (None, 50)                2550

 dense_2 (Dense)             (None, 50)                2550

 dense_3 (Dense)             (None, 1)                 51

=================================================================
Total params: 5,301
Trainable params: 5,301
Non-trainable params: 0
```

و طبق مراحل خواسته شده پیش می رویم. چون تعداد داده ها کم است، با تنظیم batch size درست، تنها یک batch در نظر می گیریم. نتیجه:




```
Train Loss: 0.7436507344245911, Train Accuracy: 0.9523809552192688
Test Loss: 5.516526222229004, Test Accuracy: 0.6666666865348816
```

می بینیم که دقت روی train خیلی بیشتر از دقت test و validation است، یعنی مدل، روی داده train، overfit شده است. در این مدل پس از چند تکرار، بهبود تعمیم‌دهی متوقف می‌شود و سپس شروع به تنزل می‌کند، مدل شروع به overfit شدن می‌کند و الگوهایی را می‌آموزد که مخصوص داده‌های آموزشی است اما ارتباط درستی با مسئله مورد نظر ندارد و گمراه‌کننده است.

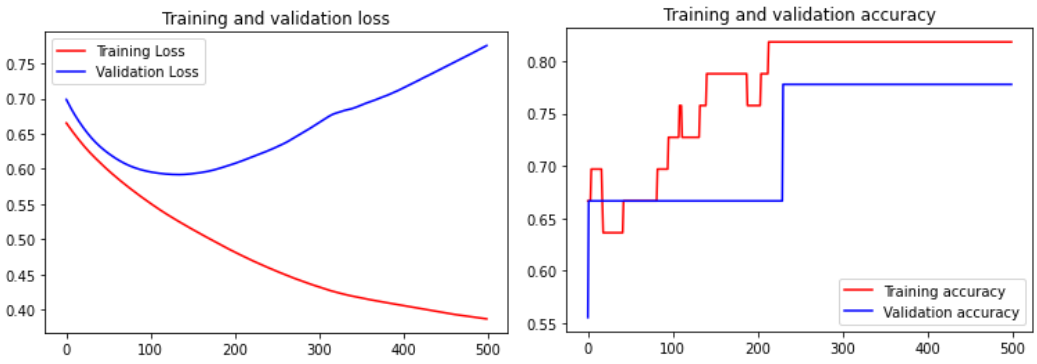
چندین راه را برای جلوگیری از overfit شدن آزمایش می کنیم.

1- کاهش ابعاد شبکه:

اگر یک شبکه فقط بتواند تعداد کمی از الگوها را حفظ کند، فرآیند بهینه‌سازی آن را مجبور می‌کند تا بر برجسته‌ترین الگوها تمرکز کند، که شانس بیشتری برای تعمیم خوب دارند. به صورت شهودی، یک مدل با پارامترهای بیشتر دارای ظرفیت حفظ بیشتری است و بنابراین می‌تواند به راحتی یک نگاشت کامل را بین نمونه‌های آموزشی و اهداف آنها بیاموزد بدون آنکه تعمیم‌دهی داشته باشد. اگر شبکه منابع محدودی برای حفظ داشته باشد، نمی‌تواند به راحتی چنین نگاشتی را یاد بگیرد و مجبور خواهد بود بازنمایی‌های فشرده‌ای را یاد بگیرد که دارای قدرت پیش‌بینی بالایی باشند. برای این کار از یک مدل با یک لایه مخفی شامل 10 نورون استفاده می کنیم و مدل را با epoch های کمتر fit می‌کنیم.

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_4 (Dense)             (None, 10)                30

 dense_5 (Dense)             (None, 1)                 11

=================================================================
Total params: 41
Trainable params: 41
Non-trainable params: 0
```

نتیجه:

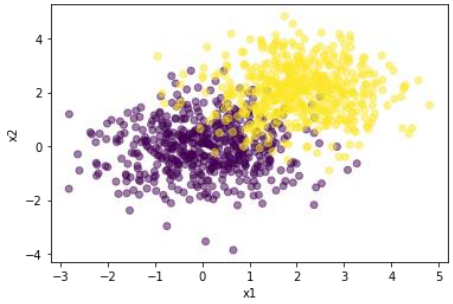


```
Train Loss: 0.47026756405830383, Train Accuracy: 0.8095238208770752
Test Loss: 0.5339462757110596, Test Accuracy: 0.7777777910232544
```

می بینیم که دقت validation و test به 78% افزایش یافته است. که پیشرفت خوبی است.

2- افزایش داده:

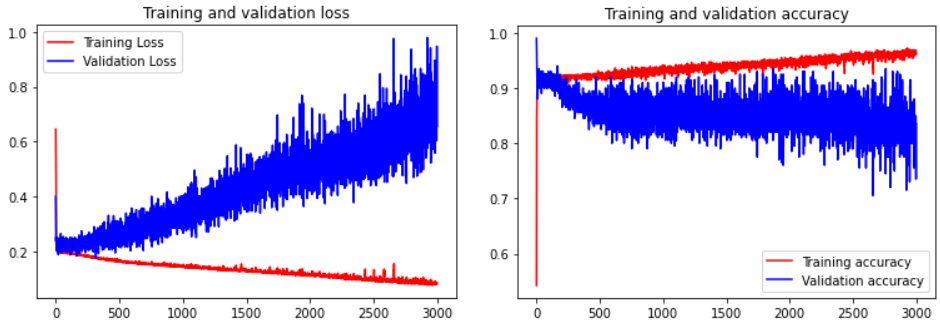
در حالت قبل به نتیجه بهتری رسیدیم اما دقت train نیز کاهش یافت. می خواهیم از راهی استفاده کنیم که هم از مدل پیچیده تری استفاده کنیم هم overfit نشویم. پس افزایش داده را امتحان می کنیم. با استفاده از تابع make_sample، 1000 داده train (500 داده از هز کلاس) و 200 داده test می سازیم. داده های train به صورت زیر هستند:



مدلمان را میسازیم.

```
Model: "sequential_11"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_36 (Dense)            (None, 50)                150

 dense_37 (Dense)            (None, 50)                2550

 dense_38 (Dense)            (None, 50)                2550

 dense_39 (Dense)            (None, 1)                 51

=================================================================
Total params: 5,301
Trainable params: 5,301
Non-trainable params: 0
```

نتیجه:



```
Train Loss: 0.6224902272224426, Train Accuracy: 0.8920000195503235
Test Loss: 0.3084014356136322, Test Accuracy: 0.9049999713897705
```

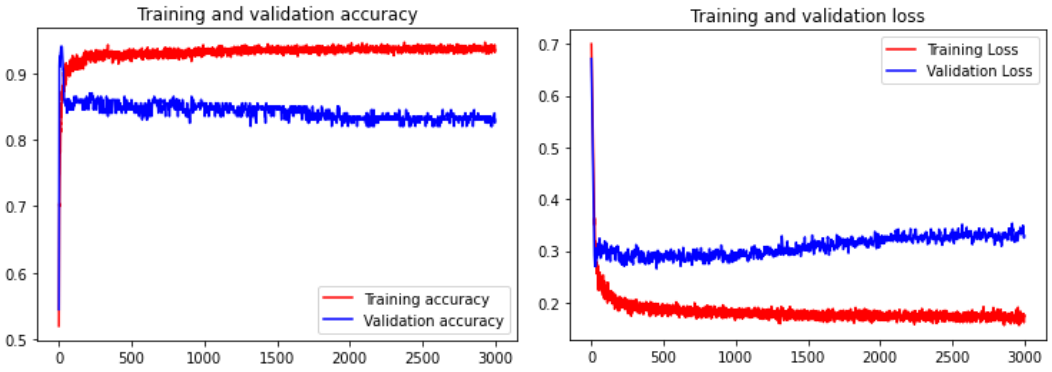
به دقت تست 90% رسیدیم! همچنین دقت داده train نیز خیلی کاهش پیدا نکرده است. پس افزایش داده تاثیر خیلی خوبی داشته است.

3- استفاده از dropout روی داده افزایش یافته:

می خواهیم همچنان دقت تست را بالاتر نیز ببریم. اگر روی همان داده کم از regularization استفاده کنیم احتمالا خیلی نتیجه بهتری نخواهیم داشت. اما با افزایش داده مطمئن هستیم که مدل منابع داده کافی دارد، حالا ظرفیت مدل را کمی پایین می آوریم و لایه های dropout را نیز اضافه می کنیم و نتیجه را می بینیم.

```
Model: "sequential_14"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_46 (Dense)            (None, 20)                60

 dropout_10 (Dropout)        (None, 20)                0

 dense_47 (Dense)            (None, 20)                420

 dropout_11 (Dropout)        (None, 20)                0

 dense_48 (Dense)            (None, 1)                 21

=================================================================
Total params: 501
Trainable params: 501
Non-trainable params: 0
```

نتیجه:




```
Train Loss: 0.19593316316604614, Train Accuracy: 0.9179999828338623
Test Loss: 0.17792600393295288, Test Accuracy: 0.9300000071525574
```

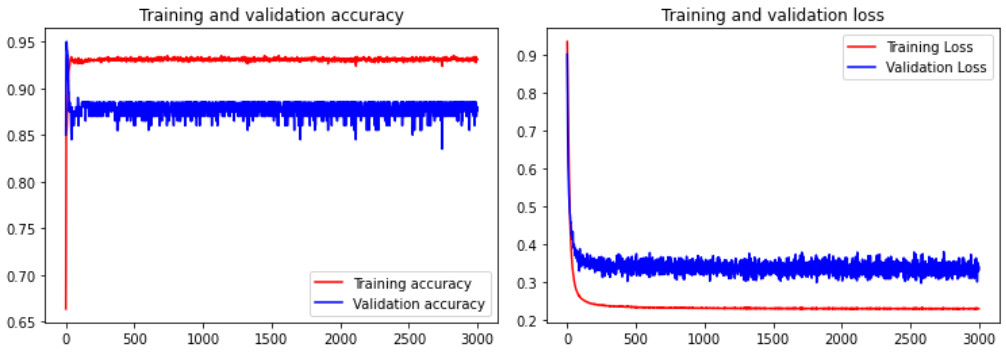
دقت تست به 93% افزایش یافت.

4- استفاده از L2 Regularization روی داده افزایش یافته:

```
model_2 = Sequential()
model_2.add(Dense(20, activation='relu', kernel_regularizer='l2', input_shape=input_shape))
model_2.add(Dense(20, activation='relu', kernel_regularizer='l2',))
model_2.add(Dense(1, activation='sigmoid', kernel_regularizer='l2'))
```

```
Model: "sequential_15"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_49 (Dense)            (None, 20)                60

 dense_50 (Dense)            (None, 20)                420

 dense_51 (Dense)            (None, 1)                 21

=================================================================
Total params: 501
Trainable params: 501
Non-trainable params: 0
```

نتیجه:




```
Train Loss: 0.2508573830127716, Train Accuracy: 0.9210000038146973
Test Loss: 0.2094368040561676, Test Accuracy: 0.9449999928474426
```

به دقت تست بیش از 94% رسیدیم.

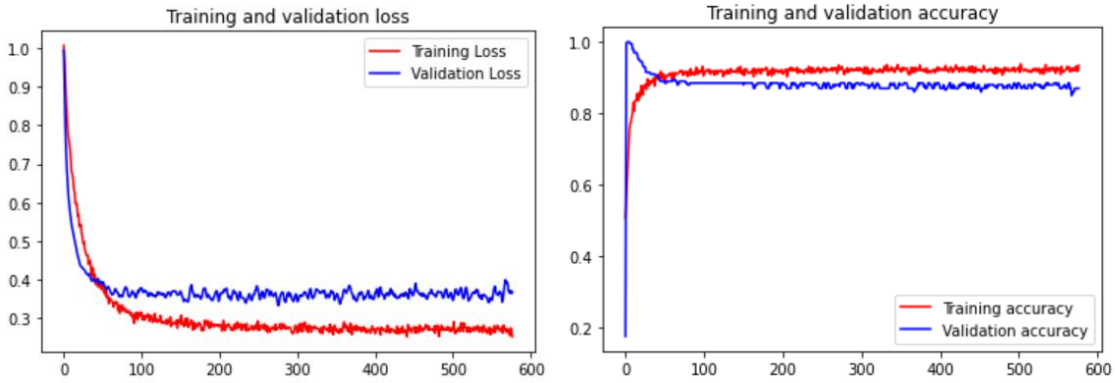
5- استفاده از Early Stopping همراه با روش های قبلی، روی داده افزایش یافته:

```
model_3 = Sequential()
model_3.add(Dense(20, activation='relu', kernel_regularizer='l2', input_shape=input_shape))
model_3.add(Dropout(0.5))
model_3.add(Dense(20, activation='relu', kernel_regularizer='l2'))
model_3.add(Dropout(0.2))
model_3.add(Dense(1, activation='sigmoid', kernel_regularizer='l2'))
```

```
from tensorflow.keras.callbacks import EarlyStopping

early_stopping = EarlyStopping(monitor='val_loss', patience=300)
history = compile_and_fit(model_3, x_train_1, y_train_1, [early_stopping])
```

```
Model: "sequential_12"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_25 (Dense)            (None, 20)                60

 dropout_4 (Dropout)         (None, 20)                0

 dense_26 (Dense)            (None, 20)                420

 dropout_5 (Dropout)         (None, 20)                0

 dense_27 (Dense)            (None, 1)                 21

=================================================================
Total params: 501
Trainable params: 501
Non-trainable params: 0
```

نتیجه:



نمودار ها smooth تر و دارای نوسان کمتری هستند. همچنین دقت validation خیلی نزدیک دقت train است.

```
Train Loss: 0.2664262354373932, Train Accuracy: 0.9179999828338623
Test Loss: 0.22229856252670288, Test Accuracy: 0.9350000023841858
```

به دقت تست تقریبا 94% رسیدیم. در این حالت روش های 2 تا 4 را ترکیب کرده و از early stopping نیز استفاده کردیم. و با تعداد epoch کمتری به دقت خیلی خوبی رسیدیم.

موارد امتیازی: کاهش ابعاد شبکه، افزایش داده، early stopping