به نام خدا

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

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

گزارش تمرین 2:

سوال اول:

الف) نمودار loss به ازای epoch برای optimizer های مختلف مقایسه شده که درباره هریک می‌توان به نکات زیر اشاره کرد:

SGDNesterov: در واقع همان SGD همراه با Momentum است که به‌جای گذشته به آینده نگاه می‌کند. مشکل این روش این است که طول گام (learning rate) در همه جهات یکسان است و ممکن است در فیچرهایی که تاثیر زیادی روی کم شدن loss ندارند، حرکت زیادی کنیم یا در جهت‌های مناسب حرکت کمی داشته باشیم. همه این موارد باعث می‌شود زمان همگرایی زیاد شود و در مدت تعیین شده نتوانیم به خوبی در نقطه Global Optimum همگرا شویم.

AdaGrad: این روش مشکل ثابت بودن طول قدم در جهات مختلف را حل می‌کند و با ارائه "نرخ یادگیری بر پارامتر" یا "نرخ یادگیری تطبیقی" باعث می‌شود در هر جهت طول قدم مناسبی داشته باشم. این کار باعث می‌شود در جهت‌های نامناسب حرکت کمی داشته باشیم و بیشتر حرکتمان به سمت نقطه min باشد. مشکلی که دارد در واقع این است که از کار مثبت Momentum بهره نبرده و نمی‌تواند مفهوم شتاب و سرعت را همزمان داشته باشد. در نتیجه عملکردش حتی از SGDNesterov بدتر است زیرا سریعا در نقطه Local Minima گیر می‌کند.

RMSprop: مشابه روش AdaGrad می‌باشد با این تفاوت که برای تعیین نرخ یادگیری تطبیقی از نوع خاصی از میانگین‌گیری استفاده می‌کند (EMA) این کار باعث می‌شود تنها به گرادیان حال حاضر نگاه نکنیم و با توجه به گرادیان نقاط گذشته و اکنون حرکت کنیم. در نتیجه احتمال پرش از Global Optimum کاهش می‌یابد. مشکل آن این است که از روش Momentum استفاده نمی‌کند. در نتیجه عملکردش از AdaGrad بهتر ولی از SGDNesterov بدتر است.

AdaDelta: بسیار شبیه به RMSprop می‌باشد یعنی از EMA استفاده می‌کند تنها تفاوتش در Epoch های اولیه می‌باشد زیرا در این روش از bias correction استفاده می‌کنیم. این امر باعث می‌شود در epoch های اولیه طول حرکت زیادی نداشته باشیم و smooth تر حرکت کنیم. در epoch های جلوتر و در ادامه این روش همانند روش RMSprop رفتار می‌کند و ضابطه‌ی آنها با هم برابر می‌شود.

Adam: از ترکیب تمامی روش‌های بالا به دست می‌آید. از Momentum به عنوان ممان اول، از RMSprop به عنوان ممان دوم استفاده می‌کند. همچنین bias correction که در روش AdaDelta نیز داشتیم را نیز بر روی هر دو ممان اعمال می‌کند. در نتیجه بهترین عملکرد را برای ما خواهد داشت.

مزایای دیگر Adam به صورت زیر است:

- پیاده‌سازی راحت
- سربار محاسباتی کم
- نیاز به حافظه زیادی ندارد
- هایپرپارامترهای آن مقدار مشخصی دارند و نیاز به tuning نیست (به جز learning rate)

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 optimizer ها دارای هایپرپارامترهایی هستند که باید Fine Tune شوند و عملکرد شبکه بستگی به آن دارد. اینکه بگوییم همیشه Adam نتیجه بهتری می‌دهد اشتباه است زیرا مقالات State of the art بسیاری را می‌بینیم که از گونه‌های SGD استفاده کرده‌اند. برخی عوامل تاثیر گذار که می توان به آن ها اشاره کرد:

میزان sparse بودن گرادیان: ممکن است data به شکلی باشد که میزان تغییرات گرادیان در ابعاد مختلف متفاوت باشد، optimizer های adagrad, rmsporp و adam برای این مسائل بهتر هستند چون learning rate داینامیک دارند و برای هر بعد متناسب با میزان تغییرات خودش نرخ آموزش را هم تغییر میدهند.

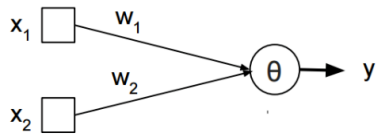
اندازه dataset و زمان: optimizer هایی مثل sgdnestrov نیاز به زمان بیشتری دارند تا بتوانند به نقاط بهنیه برسند و در صورتی که دیتاست خیلی بزرگ باشد و زمان زیادی برای train نداشته باشیم، این optimizer مؤثر نیست.

متریک و هدف ما: 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.

هموار بودن data: اگر موقعیت هایی داشته باشیم که گرادیان به شدت زیاد یا کم است، می تواند در بعضی الگوریتم ها عملکرد خوبی نداشته باشند. مثلا adagrad برای این نوع موقعیت ها عملکرد بهتری خواهد داشت.

منابع: لینک و لینک

سوال دوم:



Forward Propagation:

$z = w_0 x_0 + w_1 x_1 + b$

$\hat{y} = \delta(z) = \frac{1}{1+e^{-z}}$

$L = \frac{1}{N}\sum_{1}^{N}(\hat{y}-y)^2 = (\hat{y}-y)^2 \rightarrow Update\ using\ one\ example\ per\ iteration$

Backward Propagation:

$\frac{\partial L}{\partial \hat{y}} = -2(y-\hat{y}),\ \frac{\partial \hat{y}}{\partial z} = \delta(z)\big(1-\delta(z)\big) = \hat{y}(1-\hat{y}),\ \frac{\partial z}{\partial w_0} = x_0,\ \frac{\partial z}{\partial w_1} = x_1$

$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial z} \times \frac{\partial z}{\partial w}\ ,\quad \frac{\partial L}{\partial b} = \frac{\partial L}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial z} \times \frac{\partial z}{\partial b}$

$\frac{\partial L}{\partial w_0} = 2(\hat{y}-y) \times \hat{y}(1-\hat{y}) \times x_0\ ,\quad \frac{\partial L}{\partial w_1} = 2(\hat{y}-y) \times \hat{y}(1-\hat{y}) \times x_1$

$\frac{\partial L}{\partial b} = 2(\hat{y}-y) \times \hat{y}(1-\hat{y})$

$w_0 = w_0 - \alpha\frac{\partial L}{\partial w_0}\ ,\quad w_1 = w_1 - \alpha\frac{\partial L}{\partial w_1}\ ,\quad b = b - \alpha\frac{\partial L}{\partial b}$

$\alpha = 0.001,\quad w_0 = 2,\quad w_1 = 1,\quad b = 2,$

Epoch 1:

Forward Propagation minibatch 1:

$z = 2 \times 3 + 1 \times -1 + 2 = 7$

$\hat{y} = \delta(z) = \frac{1}{1+e^{-7}} = 0.9990889488$

$L = (0.9990889488 - 1)^2 = 8.30014289 \times 10^{-7}$

Backward Propagation minibatch 1:

$$\frac{\partial L}{\partial w_0} = 2(0.9990889488 - 1) \times 0.9990889488 \times (1 - 0.9990889488) \times 3 = -4.9755 \times 10^{-6}$$

$$\frac{\partial L}{\partial w_1} = 2(0.9990889488 - 1) \times 0.9990889488 \times (1 - 0.9990889488) \times -1 = 1.6585 \times 10^{-6}$$

$$\frac{\partial L}{\partial b} = 2(0.9990889488 - 1) \times 0.9990889488 \times (1 - 0.9990889488) \times 1 = -1.6585 \times 10^{-6}$$

$$w_0 = 2 - 0.001 \times -4.9755 \times 10^{-6} = 2.000000005$$

$$w_1 = 1 - 0.001 \times 1.6585 \times 10^{-6} = 0.9999999983$$

$$b = 2 - 0.001 \times -1.6585 \times 10^{-6} = 2.0000000017$$

Calculate the output for the data:

Data1:

$$z = 2.000000005 \times 3 + 0.9999999983 \times -1 + 2.0000000017 = 7.00000000184$$

$$\hat{y} = \delta(z) = \frac{1}{1 + e^{-z}} = 0.9990889489 \rightarrow closer\ to\ 1$$

Data2:

$$z = 2.000000005 \times 1 + 0.9999999983 \times -2 + 2.0000000017 = 2.0000000051$$

$$\hat{y} = \delta(z) = \frac{1}{1 + e^{-z}} = 0.8807970785$$

Forward Propagation minibatch 2:

$$z = 2.000000005 \times 1 + 0.9999999983 \times -2 + 2.0000000017 = 2.0000000051$$

$$\hat{y} = \delta(z) = \frac{1}{1 + e^{-z}} = 0.8807970785$$

$$L = (0.8807970785 - 0)^2 = 0.7758034944$$

Backward Propagation minibatch 2:

$$\frac{\partial L}{\partial w_0} = 2(0.8807970785 - 0) \times 0.8807970785 \times (1 - 0.8807970785) \times 1 = 0.1849560853$$

$$\frac{\partial L}{\partial w_1} = 2(0.8807970785 - 0) \times 0.8807970785 \times (1 - 0.8807970785) \times -2 = -0.3699121706$$

$$\frac{\partial L}{\partial b} = 2(0.8807970785 - 0) \times 0.8807970785 \times (1 - 0.8807970785) \times 1 = 0.1849560853$$

$$w_0 = 2.000000005 - 0.001 \times 0.1849560853 = 1.9998150489$$

$$w_1 = 0.9999999983 - 0.001 \times -0.3699121706 \ = 1.0003699105$$

$$b = 2.0000000017 - 0.001 \times 0.1849560853 = 1.9998150456$$

Calculate the output for the data:

Data1:

$$z = \ 1.9998150489 \times 3 + 1.0003699105 \times -1 + 1.9998150456 = 6.9988902818$$

$$\hat{y} = \delta(z) = \frac{1}{1 + e^{-z}} = 0.999087938156$$

Data2:

$$z = \ 1.9998150489 \times 1 + 1.0003699105 \times -2 + 1.9998150456 = 1.9988902735$$

$$\hat{y} = \delta(z) = \frac{1}{1 + e^{-z}} = 0.8806805146 \rightarrow closer\ to\ 0$$

Epoch 2:

Forward Propagation minibatch 1:

$$z = \ 1.9998150489 \times 3 + 1.0003699105 \times -1 + 1.9998150456 = 6.9988902818$$

$$\hat{y} = \delta(z) = \frac{1}{1 + e^{-z}} = 0.9990879382$$

$$L = (0.8806805146 - 1)^2 = -8.31856727 \times 10^{-7}$$

Backward Propagation minibatch 1:

$$\frac{\partial L}{\partial w_0} = 2(0.9990879382 - 1) \times 0.9990879382 \times (1 - 0.999087938) \times 3 = -4.9866 \times 10^{-6}$$

$$\frac{\partial L}{\partial w_1} = 2(0.9990879382 - 1) \times 0.9990879382 \times (1 - 0.9990879382) \times -1 = 1.6622 \times 10^{-6}$$

$$\frac{\partial L}{\partial b} = 2(0.9990879382 - 1) \times 0.9990879382 \times (1 - 0.9990879382) \times 1 = -1.6622 \times 10^{-6}$$

$$w_0 = 1.9998150489 - 0.001 \times -4.9866 \times 10^{-6} = 1.999815054$$

$$w_1 = 1.0003699105 - 0.001 \times 1.6622 \times 10^{-6} \ = 1.0003699088$$

$$b = 1.9998150456 - 0.001 \times -1.6622 \times 10^{-6} = 1.9998150439$$

Calculate the output for the data:

Data1:

$$z = 1.999815054 \times 3 + 1.0003699088 \times -1 + 1.9998150439 = 0.99908793817$$

$$\hat{y} = \delta(z) = \frac{1}{1 + e^{-z}} = 0.999087938158 \rightarrow closer\ to\ 1$$

Data2:

$$z = 1.999815054 \times 1 + 1.0003699088 \times -2 + 1.9998150439 = 1.9988902803$$

$$\hat{y} = \delta(z) = \frac{1}{1 + e^{-z}} = 0.88068051528$$

Forward Propagation minibatch 2:

$$z = 1.999815054 \times 1 + 1.0003699088 \times -2 + 1.9998150439 = 1.9988902803$$

$$\hat{y} = \delta(z) = \frac{1}{1 + e^{-z}} = 0.88068051528$$

$$L = (0.88068051528 - 0)^2 = 0.77559816999$$

Backward Propagation minibatch 2:

$$\frac{\partial L}{\partial w_0} = 2(0.88068051528 - 0) \times 0.88068051528 \times (1 - 0.88068051528) \times 1 = 0.18508794799$$

$$\frac{\partial L}{\partial w_1} = 2(0.88068051528 - 0) \times 0.88068051528 \times (1 - 0.88068051528) \times -2$$
$$= -0.37017589597$$

$$\frac{\partial L}{\partial b} = 2(0.88068051528 - 0) \times 0.88068051528 \times (1 - 0.88068051528) \times 1 = 0.18508794799$$

$$w_0 = 1.999815054 - 0.001 \times 0.18508794799 = 1.999629966$$

$$w_1 = 1.0003699088 - 0.001 \times -0.37017589597 = 1.00074008469$$

$$b = 1.9998150439 - 0.001 \times 0.18508794799 = 1.9996299559$$

Calculate the output for the data:

Data1:

$z = 1.999629966 \times 3 + 1.00074008469 \times -1 + 1.9996299559 = 6.99777976921$

$\hat{y} = \delta(z) = \frac{1}{1 + e^{-z}} = 0.9990869256635$

Data2:

$z = 1.999629966 \times 1 + 1.00074008469 \times -2 + 1.9996299559 = 1.99777975252$

$\hat{y} = \delta(z) = \frac{1}{1 + e^{-z}} = 0.8805637690 \rightarrow closer\ to\ 0$

با استفاده از یک learning rate بالاتر، تابع Loss قویتر و تعداد epoch بیشتر، می توانیم سرعت یادگیری بهتر و نتایج دقیقتری نیز داشته باشیم.

سوال سوم:

شبکه ای با 3 لایه Dense ساختم که دو لایه مخفی با 512 و 128 نورون با تابع فعال سازی relu دارد، و چون مسئله ما یک مسئله classification ده کلاسه است، لایه خروجی 10 نورون و تابع فعال سازی sotmax دارد. یک تابع compile_fit نوشتم که با ورودی گرفتن نام optimizer و دیکشنری مدل های مختلف، مدل را build می کند. و در دیکشنری models ذخیره می کند. سپس با optimizer ای که به آن دادیم مدل را compile می کند، برای آن یک tensorboard ساخته و مدل را [tensorboard] به عنوان callback و با BATCH_SIZE و EPOCHS که قبلا مشخص کرده ایم فیت می کند.

حالا از این تابع استفاده کرده و سه مدل با 3 optimizer مختلف ساخته و fit می کنیم. دقت های بدست آمده برای 3 مدل:

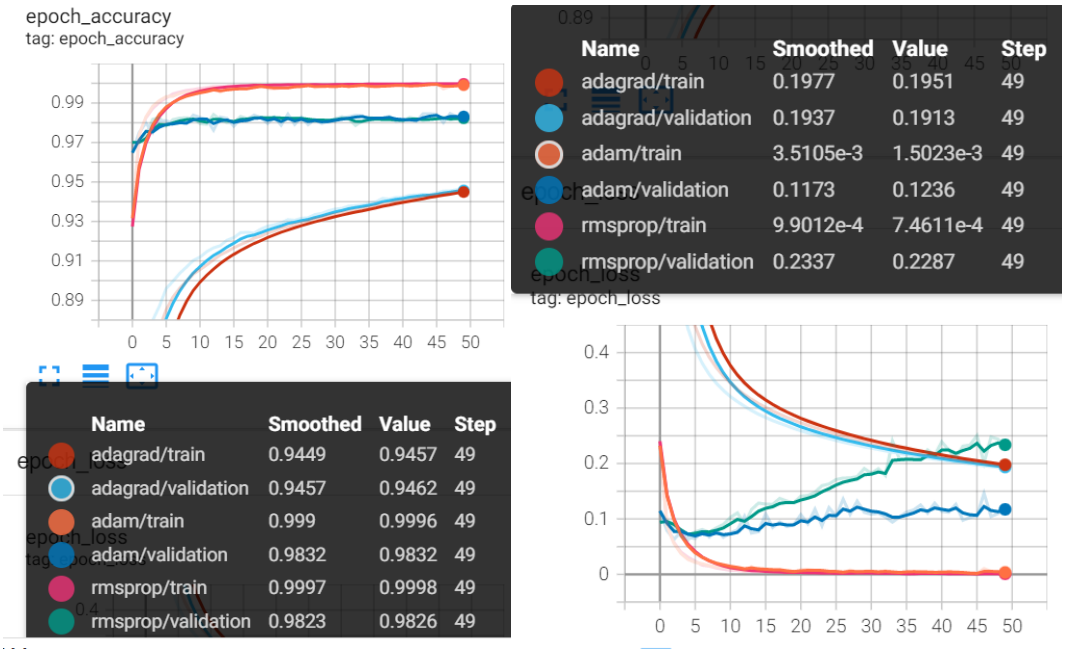
Adam: loss: 0.0015 - accuracy: 0.9996 - val_loss: 0.1236 - val_accuracy: 0.9832

AdaGrad: loss: 0.1951 - accuracy: 0.9457 - val_loss: 0.1913 - val_accuracy: 0.9462

RMSProp: loss: 7.4611e-04 - accuracy: 0.9998 - val_loss: 0.2287 - val_accuracy: 0.9826

همانطور که انتظار می رفت، Adam بهترین عملکرد را داشته است. (با توجه به val accuracy) که برخی مزیت های Adam را نسبت به بقیه optimizer ها در سوال اول ذکر کردم. RMSProp نیز بعد از Adam با اختلاف کمی جایگاه بعدی را دارد و مدل با کمترین دقت مربوط به AdaGrad است.

خروجی tensorboard، هم برای هر optimizer جداگانه و هم در آخر برای همه باهم موجود است.
خروجی مقایسه دقت و خطای همه باهم:



Adam را به عنوان بهترین مدل در best_model.h5 ذخیره می کنیم. سپس آن را Load و Compile می کنیم و روی داده های تست evaluate می کنیم که نتیجه زیر بدست می آید:

```
Test Loss: 0.1235506609082222, Test Accuracy: 0.9832000136375427
```

در بخش Get Reports هم از confusion_matrix و classification_report از sklearn.metrics استفاده کردیم که خروجی آن به صورت زیر است:

```
[[ 969    1    1    0    1    1    2    1    4    0]
 [   1 1123    6    0    0    0    2    1    2    0]
 [   2    0 1024    0    1    0    2    2    0    1]
 [   0    0   11  987    0    0    0    5    3    4]
 [   0    0    6    0  966    0    2    1    0    7]
 [   1    1    0    8    1  871    5    1    4    0]
 [   2    2    2    0    2    1  947    0    2    0]
 [   0    1   13    0    0    0    0 1010    2    2]
 [   2    0    4    5    3    0    1    3  952    4]
 [   0    2    1    3    6    4    2    3    5  983]]
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       980
           1       0.99      0.99      0.99      1135
           2       0.96      0.99      0.98      1032
           3       0.98      0.98      0.98      1010
           4       0.99      0.98      0.98       982
           5       0.99      0.98      0.98       892
           6       0.98      0.99      0.99       958
           7       0.98      0.98      0.98      1028
           8       0.98      0.98      0.98       974
           9       0.98      0.97      0.98      1009

    accuracy                           0.98     10000
   macro avg       0.98      0.98      0.98     10000
weighted avg       0.98      0.98      0.98     10000
```

منابع: لینک و لینک و لینک

سوال چهارم:

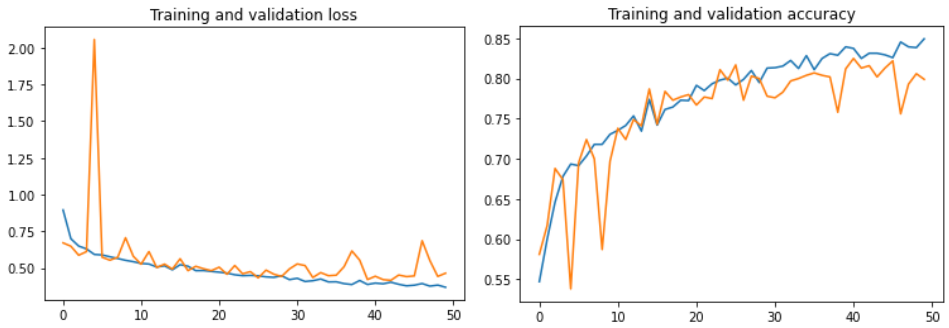
با یک مسئله Classification دو کلاسه رو به رو هستیم. چند حالت برای این سوال در نظر گرفته و نتیجه را مقایسه کرده، بهترین حالت را انتخاب می کنیم.

1- استفاده از تابع فعالسازی sigmoid و تابع ضرر binary crossentropy با یک نورون خروجی:
چون مسئله ما single label است وقتی از sigmoid استفاده می کنیم برای لایه خروجی یک نورون قرار می دهیم که خروجی بین 0 و 1 خواهد بود که احتمال سگ یا گربه بودن است، و 1 منهای این احتمال، احتمال خلاف آن است. همچنین می دانیم چون مسئله classification داریم باید از تابع ضرر crossentropy استفاده کنیم و چون در این حالت داریم از sigmoid استفاده می کنیم از binary crossentropy استفاده می کنیم.

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

نتیجه:

```
loss: 0.3683 - acc: 0.8495 - val_loss: 0.4641 - val_acc: 0.7990
```

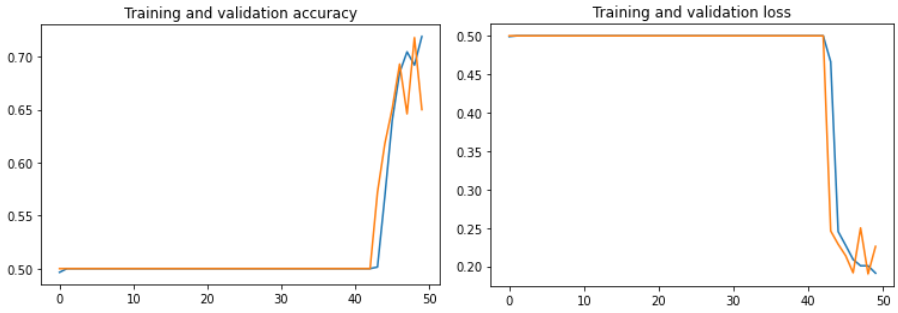


2- استفاده از تابع فعالسازی sigmoid و تابع ضرر mse با یک نورون خروجی:
مانند حالت قبل عمل کرده اما تابع ضرر آن را mse میگذاریم. می دانیم که تابع ضرر mse برای مسئله classification مناسب نیست. در این حالت مشتق تابع ضرر همواره نزدیک 0 خواهد بود و وزن ها خوب آپدیت نخواهد شد. (Vanishing Gradient)

```
## Set These Parameters
last_layer_neurons = 1
last_layer_activation = 'sigmoid'
loss_function = 'mse'
```

نتیجه:

```
loss: 0.1911 - acc: 0.7190 - val_loss: 0.2259 - val_acc: 0.6500
```



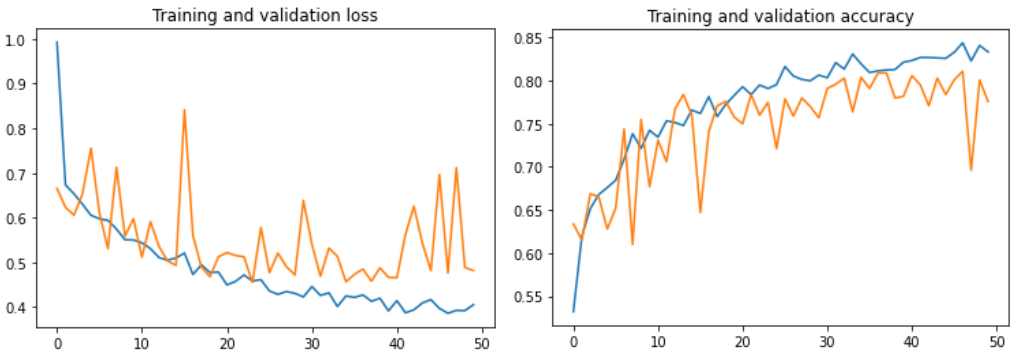
در واقع با رویکرد Likelihood Maximum اگر مسئله را بررسی کنیم تابع ضرر MSE برای تابع فعالسازی Sigmoid اصلا مناسب نیست. نمودار خطا و دقت آن نیز تا تقریبا epoch 40 عملا یک خط با شیب ثابت شده است که مقدار شیب آن تقریبا نزدیک 0 است و در این محدوده عملا بهینه سازی نداریم.

3- استفاده از تابع فعالسازی softmax و تابع ضرر categorical crossentropy با دو نورون خروجی:
در این حالت با یک مسئله multi class single label روبرو هستیم که از softmax اسفاده می کند پس باید از دو نورون برای لایه خروجی استفاده کنیم که یکی احتمال سگ بودن و یکی احتمال گربه بودن را بیان می کند. احتمال هرکدام بیشتر باشد جواب مسئله همان است. در این حالت چون مسئله ما کلاس بندی است و مسئله را چند کلاسه با تابع فعالسازی softmax در نظر گرفتیم، بهترین تابع ضرر categorical crossentropy خواهد بود.

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

نتیجه:

```
loss: 0.4054 - acc: 0.8335 - val_loss: 0.4819 - val_acc: 0.7760
```



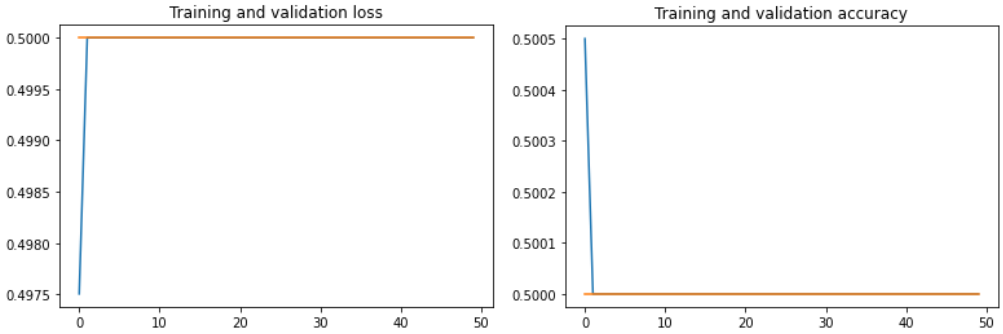
4- استفاده از تابع فعالسازی softmax و تابع ضرر mse با دو نورون خروجی:

مانند حالت قبل عمل کرده اما تابع ضرر آن را mse میگذاریم. می دانیم که تابع ضرر mse برای مسئله classification مناسب نیست. در این حالت مشتق تابع ضرر همواره نزدیک 0 خواهد بود و وزن ها آپدیت نخواهد شد. (Vanishing Gradient)

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

نتیجه:

```
loss: 0.5000 - acc: 0.5000 - val_loss: 0.5000 - val_acc: 0.5000
```



به دلیل تابع ضرر نامناسب نمودار خطا و دقت آن عملا یک خط با شیب ثابت شده است که مقدار شیب آن تقریبا نزدیک 0 است و عملا بهینه سازی نداریم.

5- استفاده از تابع فعالسازی relu و تابع ضرر mse:

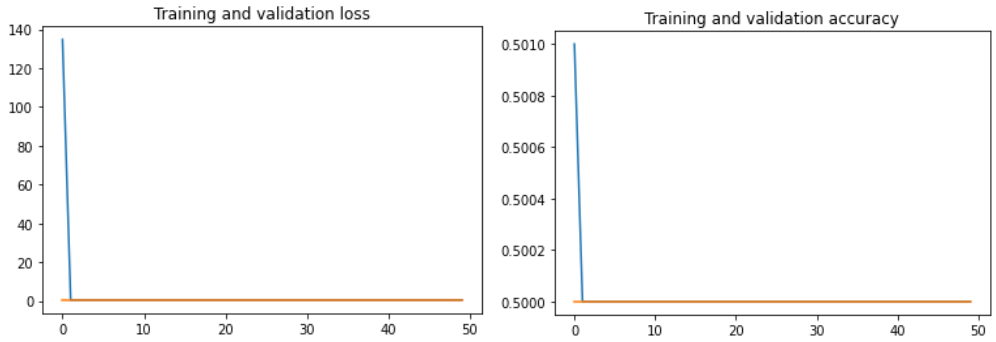
می دانیم که استفاده از تابع فعالسازی relu در خروجی مسئله classification مناسب نیست.

.

```
## Set These Parameters
last_layer_neurons = 1
last_layer_activation = 'relu'
loss_function = 'mse'
```

نتیجه:

```
loss: 0.5000 - acc: 0.5000 - val_loss: 0.5000 - val_acc: 0.5000
```



میبینیم که بهینه سازی نداریم. البته اگر 2 نورون در خروجی بگذاریم شاید نتیجه بهتری از relu هم بگیریم. چون تابع فعالسازی relu مقادیر را به بازه 0 تا 1 نمی برد تابع ضرر mse مناسب آن است. و ترکیب این دو اگر دو نورون در خروجی استفاده کنیم، می تواند عملکرد بهتری داشته باشد. اما به هرحال در آخر پاسخ ما به صورت توزیع احتمال نیست پس بجای آن از همان Softmax استفاده می کنیم.

نتیجه کلی: با توجه به اینکه هر تابع ضرر، فرمولی مخصوص خود را دارد نمی توان ملاک و Metric مقایسه را مقدار loss قرار داد و ملاک، Accuracy میباشد زیرا فرمول آن برای تمامی حالات یکسان است.

ترتیب accuracy در train:

Sigmoid, BCE > Softmax, CCE > Sigmoid, MSE > Relu, MSE >= Softmax MSE

ترتیب accuracy در validation:

Softmax, CCE > Sigmoid, BCE > Sigmoid, MSE > Relu, MSE >= Softmax MSE

می بینیم که Sigmoid, BCE و Softmax, CCE با اختلاف دقت بهتری از بقیه حالات دارند. دقت Softmax, CCE کمی از Sigmoid, BCE کمتر شده است. از آن جایی که Softmax یک تعمیم از Sigmoid برای مسئله چند کلاسه است، در واقع اثبات می‌شود که که Sofmax دو کلاسه معادل با Sigmoid است. به همین دلیل هم دقت این دو خیلی نزدیک به هم شده است.