Problems&solutions table

this color means "have reproduced"

Problem:	Dying RELU	Gradient Vanish	Exploding Gradients
Solutions:	1.SELU--robust SNN.i improved version of leaky relu *	1. Unsaturated activation function, like Relu(May lead to Dying RELU)(Leaky ReLU、PReLU、RReLU）*、（DRelu). *(case 16 need an example to show its effect)	1.Redesign the model(fewer layers\smaller batches\truncated Backpropagation through time) LSTM ——The Long Short-Term Memory (LSTM) memory units and perhaps related gated-type neuron structures.[1] Resnet——allow direct backprop to earlier layers through a "shortcut" or "skip connection"
	2. Keras allows gradient extraction directly to check each Neuron. *	2. Xavier initialization——set an appropriate weight as a start. (He initialization is another one)(also in case 16, can improve the result)*	
	3. randomized asymmetric initialization—— an improvement for He initialization. (in case 14 test didn't work very well)	3.Layerwise training:training the network greedily layer by layer reference(case17)	3.Gradient Clipping(or clipnorm)——clipping gradients if their norm exceeds a given threshold.(reproduced in case15)*
	4.leaking relu(help to solve the gradient problems) (in case 14 seems not so helpful like selu)	4.batch normalize or using drop out.(case 16)	4.Weight regularization—— apply a penalty to loss function with the big weight.*
		5.unsupervised pre-training(case 17)	5.batch norm[2](didn't work on case 15)

[1] https://www.dlology.com/blog/how-to-deal-with-vanishingexploding-gradients-in-keras/

[2] https://stackoverflow.com/questions/55486395

			6.data Preprocessing.(case 15,19)

Problem:	Unstable	Not converge
Solutions:	1.Incorporate momentum based stochastic gradient descent(SGD, comparing with adam, can performs better)	
	2.Regularisation (add regularisation loss to your data loss and try to minimise that). didn't have obviously effect in case20, this solution usually works on overfitting.	2.suitable Parameters() 1)enlarge learning rate(too little "lr" will lead to too slow converge and even stop)(case24)
	3.less learning rate.(case 20)	3.heuristic approach like "resillient backprop
	4.batch size	4. Change Adam to SGD.#no good examples now.

red * means can be used in Keras API.

Case7:
problem: loss unstable
reason:initialize too late.

```
 32/801 [>.............................] - ETA: 0s - loss: 5895.7041
801/801 [==============================] - 0s 19us/sample - loss: 6084.6799
Epoch 499/500

 32/801 [>.............................] - ETA: 0s - loss: 5879.3301
801/801 [==============================] - 0s 17us/sample - loss: 6057.1412
Epoch 500/500

 32/801 [>.............................] - ETA: 0s - loss: 6124.3047
801/801 [==============================] - 0s 19us/sample - loss: 5996.5908
```

```
 16/801 [..............................] - ETA: 0s - loss: 1.3200
801/801 [==============================] - 0s 36us/sample - loss: 1.4412
Epoch 499/500

 16/801 [..............................] - ETA: 0s - loss: 1.5271
801/801 [==============================] - 0s 34us/sample - loss: 1.4400
Epoch 500/500

 16/801 [..............................] - ETA: 0s - loss: 1.2175
801/801 [==============================] - 0s 31us/sample - loss: 1.4402
```

case 9:
problem: loss seems not to converge.
reason: In that model, accuracy isn’t the right way to measure the training effect.

case11 :

```
        #print  batch_y.shape
        _,loss,  temp_feat  =  sess.run([train_op,cost,  feat],feed_dict={  x:batch_x,  y:batch_y,  dropOut:0.4})


        if  total_feat  is  None:
            total_feat  =temp_feat
        else:
            total_feat  =  np.concatenate([total_feat,temp_feat],axis=0)

    #print  "epoch_gates  %f  %f  "  %  (np.amin(epoch_gates),  np.amax(epoch_gates))

        del  batch_x
        del  batch_y
    print  "np.amin(total_feat):  %f"  %  np.amin(total_feat)
    print  "np.amax(total_feat):  %f"  %  np.amax(total_feat)
```

```
np.amin(total_feat): -26.329517
np.amax(total_feat): 17.982983
np.amin(total_feat): -18.171093
np.amax(total_feat): 14.475341
np.amin(total_feat): -18.887138
np.amax(total_feat): 18.106163
np.amin(total_feat): -20.600521
np.amax(total_feat): 14.850040
np.amin(total_feat): -18.701208
np.amax(total_feat): 17.156317
np.amin(total_feat): -13.886760
np.amax(total_feat): 16.280766
np.amin(total_feat): -14.651957
np.amax(total_feat): 22.437336
np.amin(total_feat): -12.417414
np.amax(total_feat): 10.873753
np.amin(total_feat): -12.488481
np.amax(total_feat): 16.629656
np.amin(total_feat): -12.368624
np.amax(total_feat): 10.516583
np.amin(total_feat): -11.047361
np.amax(total_feat): 15.708699
np.amin(total_feat): -12.156299
np.amax(total_feat): 9.186778
np.amin(total_feat): -10.621801
np.amax(total_feat): 9.992596
np.amin(total_feat): -16.568949
np.amax(total_feat): 8.935214
```

```
np.amin(total_feat): -0.519152
np.amax(total_feat): 0.547111
np.amin(total_feat): -0.539376
np.amax(total_feat): 0.545885
np.amin(total_feat): -0.462878
np.amax(total_feat): 0.455493
```

```
    with  tf.variable_scope("GRU_forward"):
        cell  =  rnn.GRUCell(nunits)
        #cell  =  DropoutWrapper(cell,  output_keep_prob=1.0-dropOut,  state_keep_prob=1.0-dropOut)
        outputs_fw,_=  static_rnn(cell,  x_in,  dtype="float32")
```

if not comment the second line of cell, the gradient will explode over the range of [-1,1].

case12:

```
[ ]  t  =  tf.random.normal((2,  224,  224,  3))
    optimizer  =  tf.keras.optimizers.SGD()

    mrf  =  MRF_Block(12,  2)
    with  tf.GradientTape()  as  tape:
            preds  =  mrf([t,  t])
            loss  =  tf.keras.losses.BinaryCrossentropy(from_logits=True)(tf.random.normal((2,  224,  224,  12)),  preds)
    gradients  =  tape.gradient(loss,  mrf.trainable_variables)
    optimizer.apply_gradients(zip(gradients,  mrf.trainable_variables))
```

```
WARNING:tensorflow:Gradients do not exist for variables ['mrf__block_6/conv2d_transpose_12/kernel:0', 'mrf__block_6/conv2d_transpose_12/bias:0'] when minimizing the loss.
<tf.Operation 'SGD_6/SGD/AssignAddVariableOp' type=AssignAddVariableOp>
```

```
mrf  =  MRF_Block(12,  2)
with  tf.GradientTape()  as  tape:
        preds  =  mrf([t,  t])
        loss  =  tf.keras.losses.BinaryCrossentropy(from_logits=True)(tf.random.normal((2,  224,  224,  12)),  preds)
gradients  =  tape.gradient(loss,  mrf.trainable_variables)
gradients  =  [grad  if  grad  is  not  None  else  tf.zeros_like(var)
for  var,  grad  in  zip(mrf.trainable_variables,  gradients)]
optimizer.apply_gradients(zip(gradients,  mrf.trainable_variables))
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow_core/python/ops/resource_variable_ops.py:1630: calling BaseResourceVariable.__init__ (from tensorflow.python.ops.resource_variable_ops) with constraint is deprecated and will be rem
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow_core/python/ops/nn_impl.py:183: where (from tensorflow.python.ops.array_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
<tf.Operation 'SGD/SGD/AssignAddVariableOp' type=AssignAddVariableOp>
```

adding
”gradients = [grad if grad is not None else tf.zeros_like(var)
for var, grad in zip(mrf.trainable_variables, gradients)]
”
can only exclude the “None” gradient, but can’t solve the problem here.

[]

cost(from little to large)	solution:(px.y means the *y*th solution for problem x)
1(only check)	p1.2
2(work on training)	p1.1,p1.4||p2.1,p2.4||p3.2,p3.3,p3.4||p4.1,p4.2||P4.3
3.(work on the model, also include the training data)	p1.3||p2.2,p2.3||p3.1||P4.1,P4.2

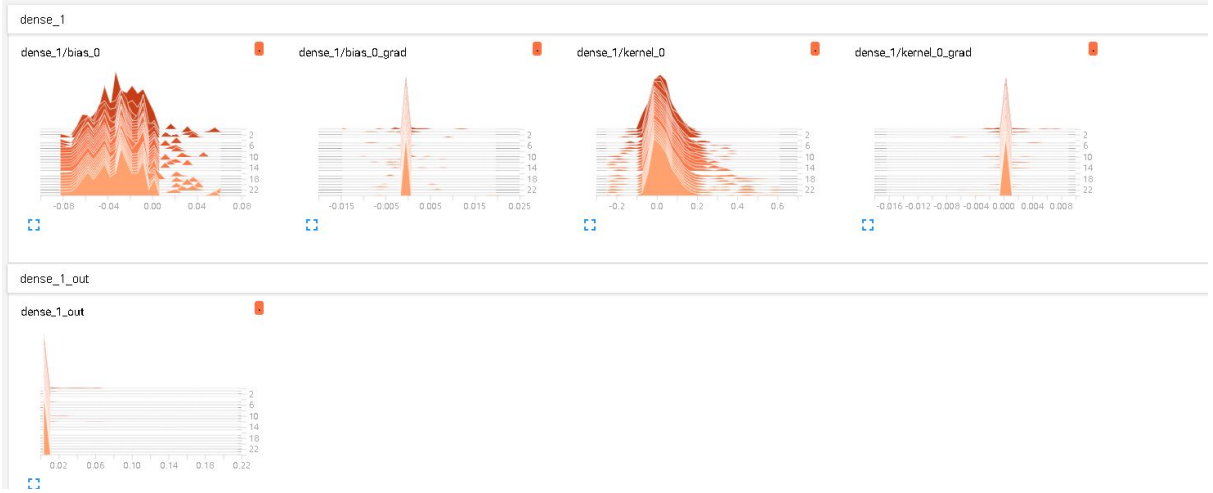
diffiluties:
1.not all solutions can be reproduce--only reproduce 5 examples from github and stackoverflow.(relu still have no example, but some papers can show how to solve it.)
2.

Reproduce The Cases:

Dying relu”

case 14(relu):

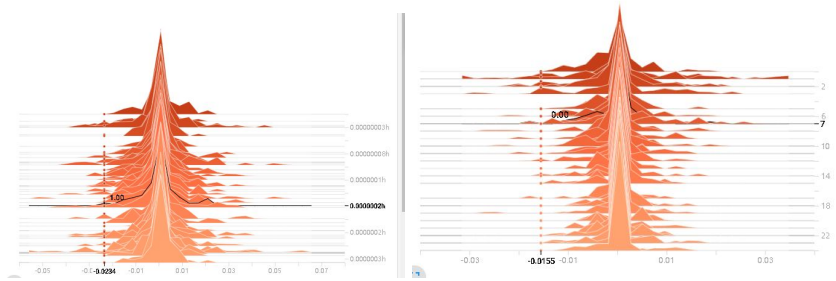
1.origin

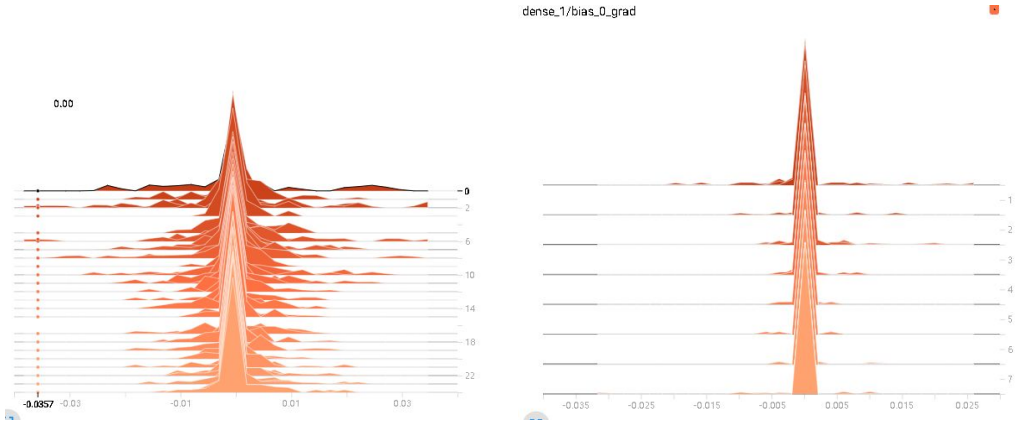


https://link.medium.com/UNwRMiEEI2

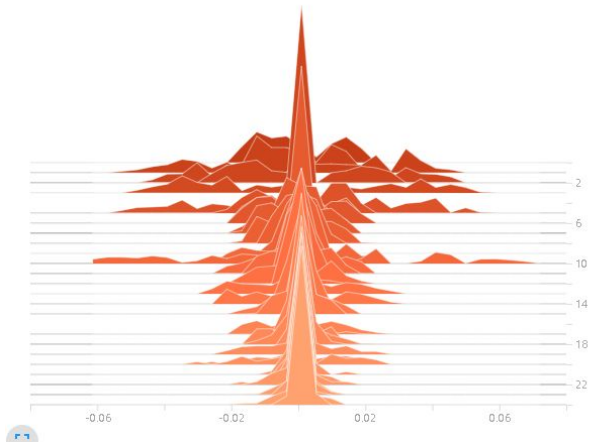
didn't get the right result when I add more layers.

2.change initialize:

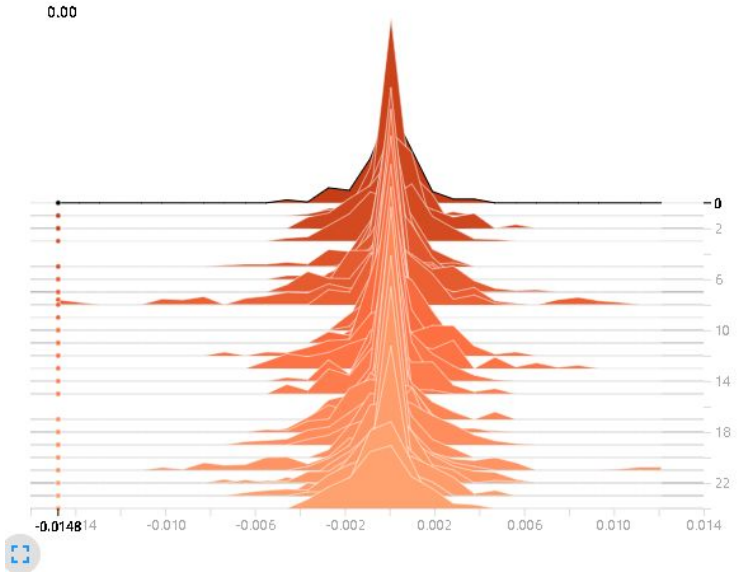




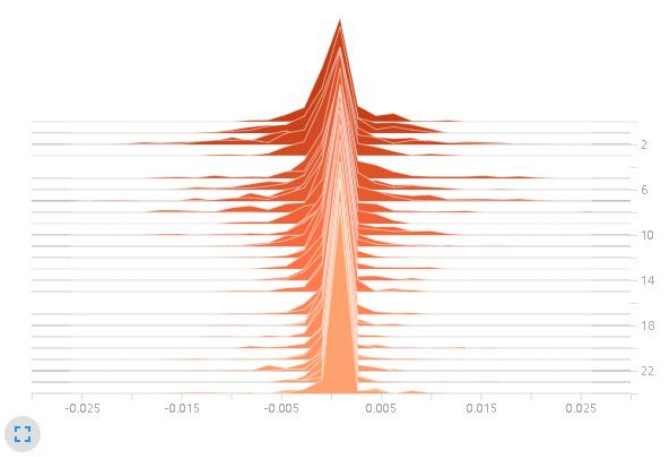
dense_1/bias_0_grad



3.change activation function



(selu)



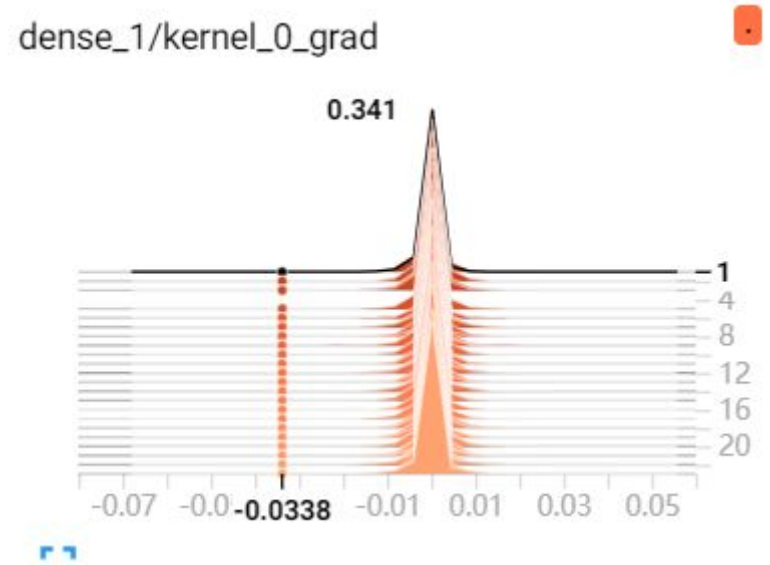
(leaky relu)

case 14(relu-cnn):

1.cnn with 2conv:

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 28, 28, 16)	160
max_pooling2d_1 (MaxPooling2	(None, 14, 14, 16)	0
conv2d_2 (Conv2D)	(None, 14, 14, 36)	5220
max_pooling2d_2 (MaxPooling2	(None, 7, 7, 36)	0
dropout_1 (Dropout)	(None, 7, 7, 36)	0
flatten_1 (Flatten)	(None, 1764)	0
dense_1 (Dense)	(None, 128)	225920
dropout_2 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 10)	1290

from the above figure, we can find that in this test, the conv2d layer seems not to face the dying relu problem, but the dense layers still have this issue like the tests before.



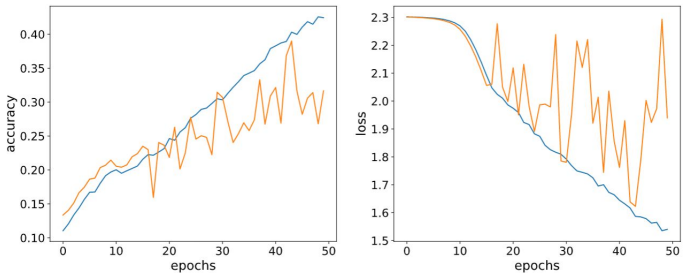
Dense layers has the gradient which is quite close to zero.

using selu:

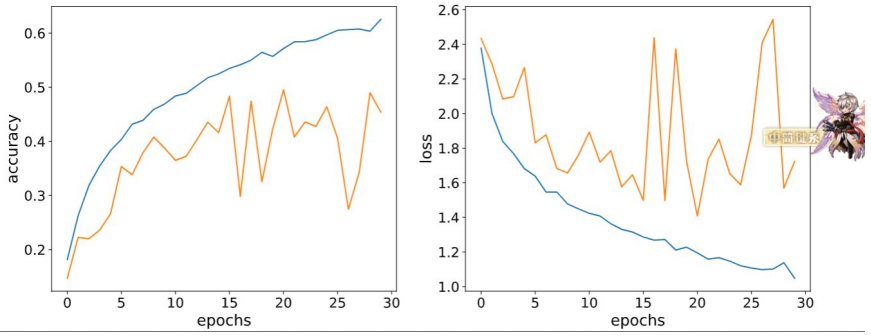
case 14(relu-cnn(SGD))success:

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 32, 32, 16)	448
max_pooling2d_1 (MaxPooling2	(None, 16, 16, 16)	0
conv2d_2 (Conv2D)	(None, 16, 16, 36)	5220
max_pooling2d_2 (MaxPooling2	(None, 8, 8, 36)	0
dropout_1 (Dropout)	(None, 8, 8, 36)	0
flatten_1 (Flatten)	(None, 2304)	0
dense_1 (Dense)	(None, 128)	295040
dense_2 (Dense)	(None, 128)	16512
dense_3 (Dense)	(None, 128)	16512
dense_4 (Dense)	(None, 128)	16512
dense_5 (Dense)	(None, 128)	16512
dense_6 (Dense)	(None, 128)	16512
dense_7 (Dense)	(None, 128)	16512
dense_8 (Dense)	(None, 128)	16512
dense_9 (Dense)	(None, 128)	16512
dense_10 (Dense)	(None, 128)	16512
dense_11 (Dense)	(None, 128)	16512

1. **leaky_relu:**



2. **selu:**



---

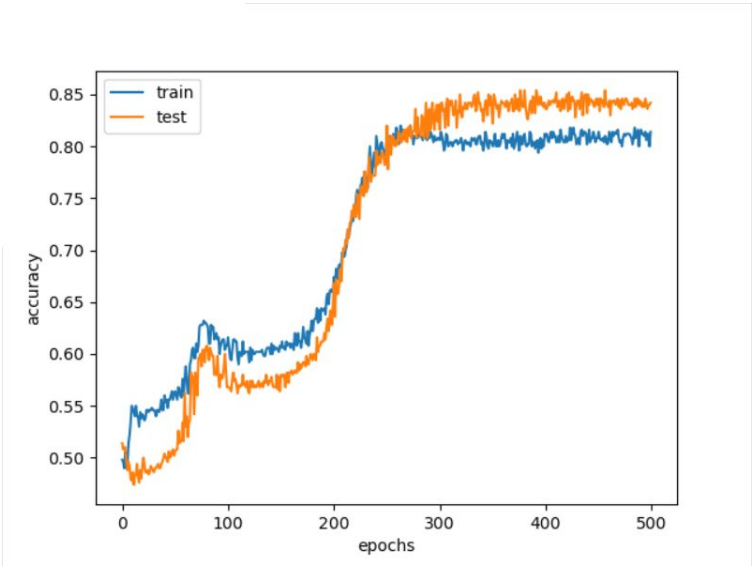
selu works well(overfit.)

3. **initialize(no effect)**

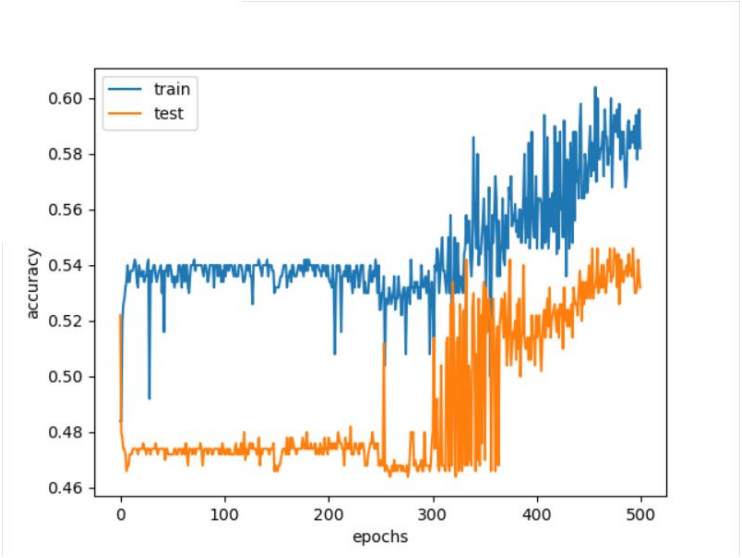
vanish gradient：

1. case16：

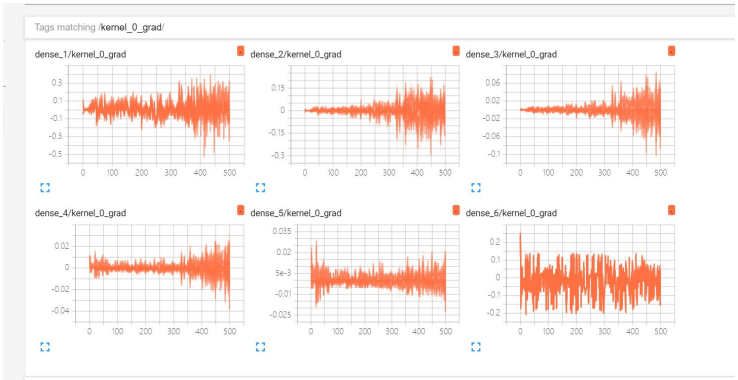
   1.tanh-randomuniform-short model.

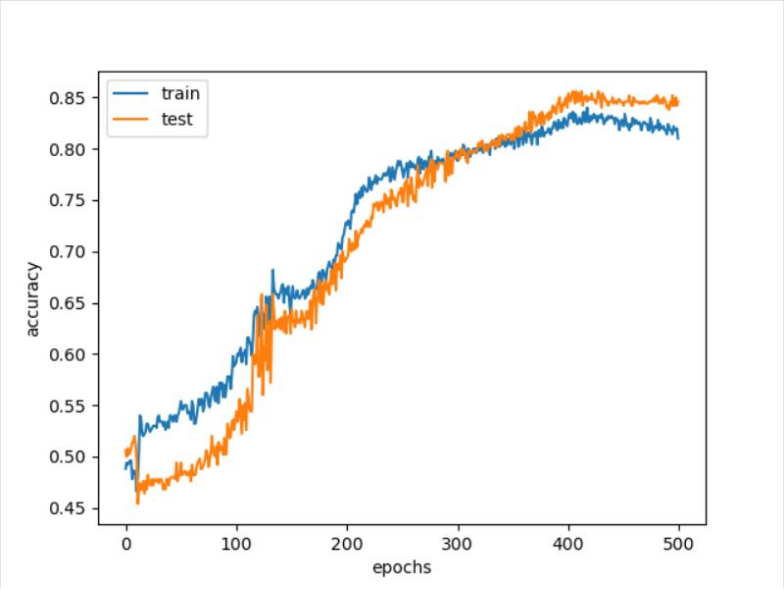


   2.tanh-randomuniform-medium model.

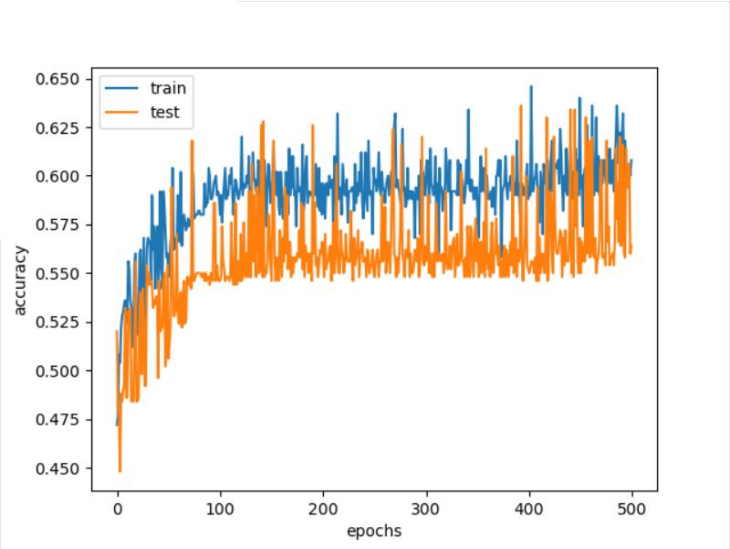


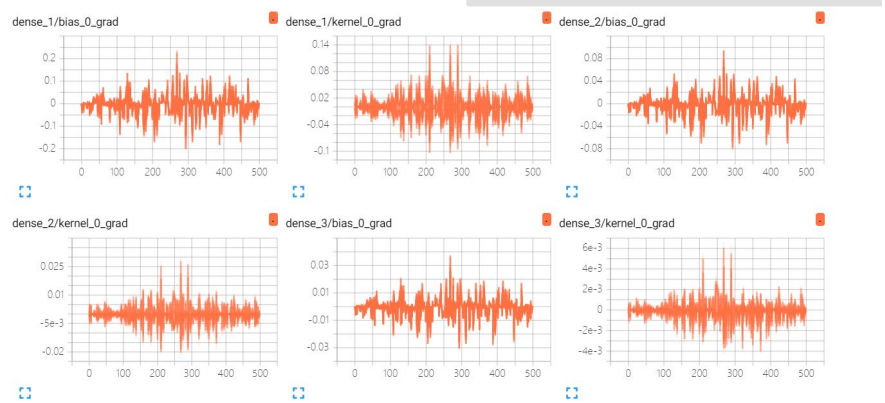
   accuracy didn’t update well in this situation, we can use tensorboard to get more information of gradient.



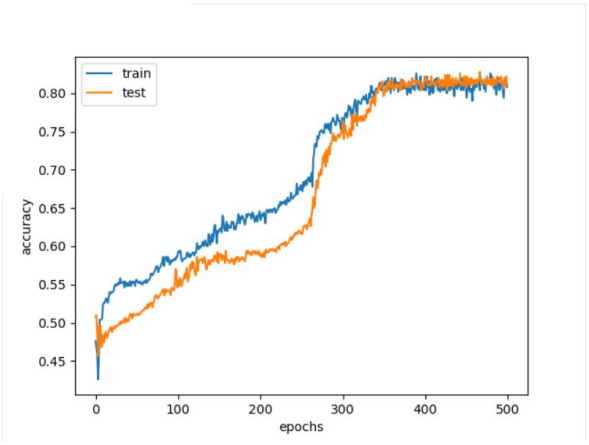


add one dense to model in’1.’

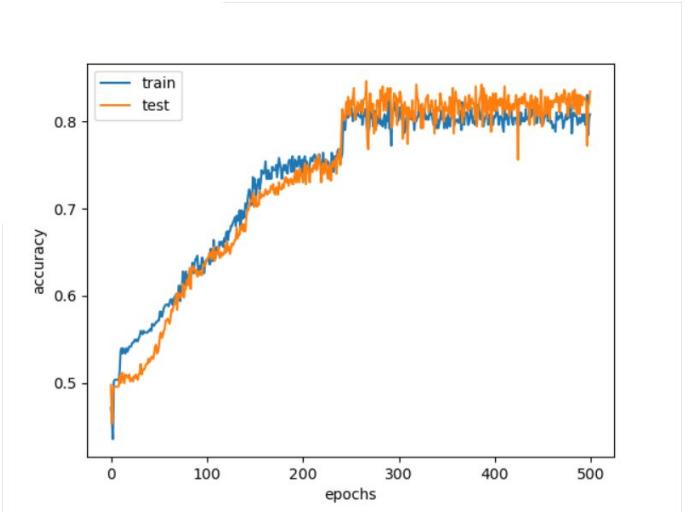




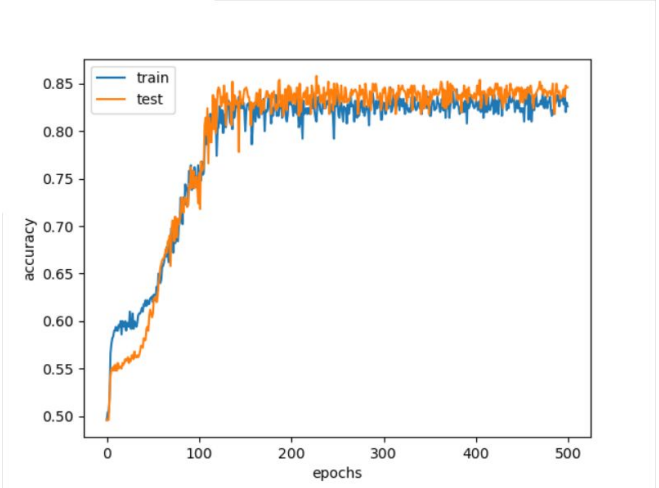
add two dense to 1., It is strange that the layers near input seems have better gradient than the laryers near output, **This is not the same as the disappearance of the gradient in our understanding**



only using relu

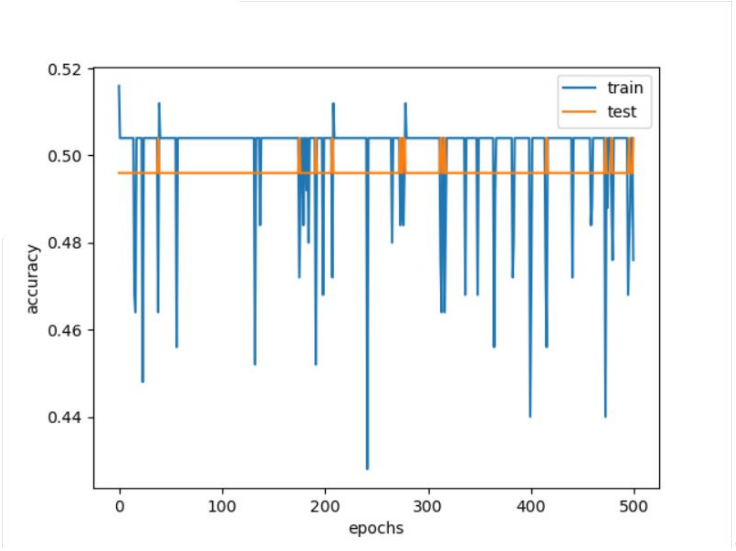


only relu, add one dense

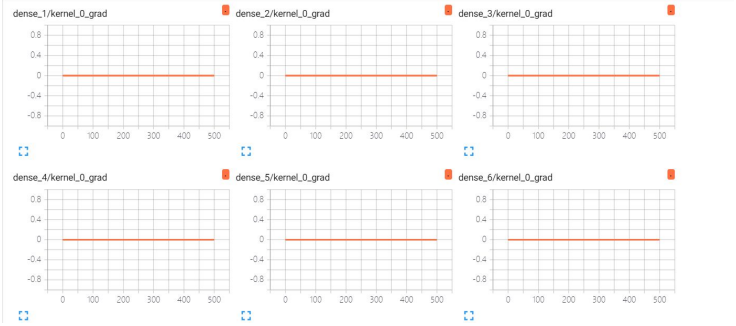


only relu, add two dense----better than ‘tanh’

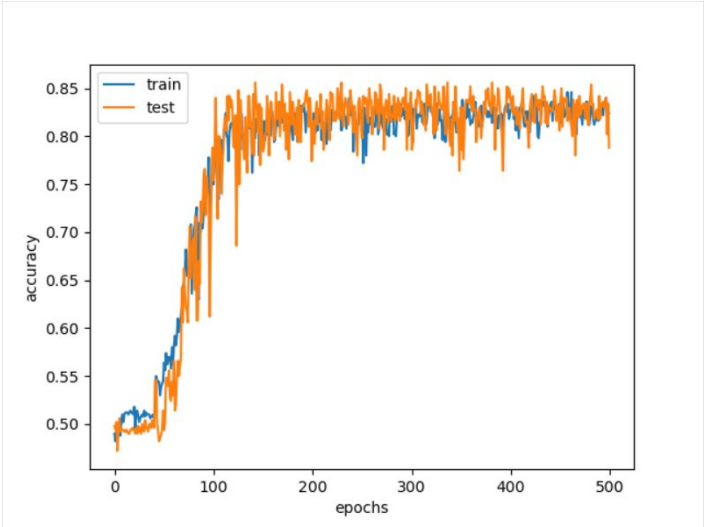
3.relu-randomuniform-medium model.

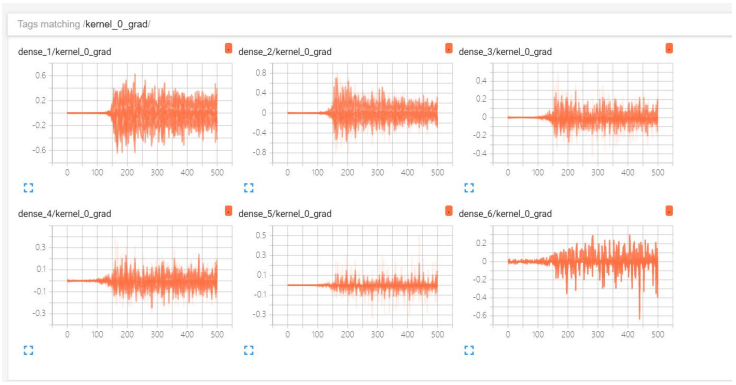


accuracy didn’t perform better.



however when we used the default initialization(glorot_uniform), it will perform better.



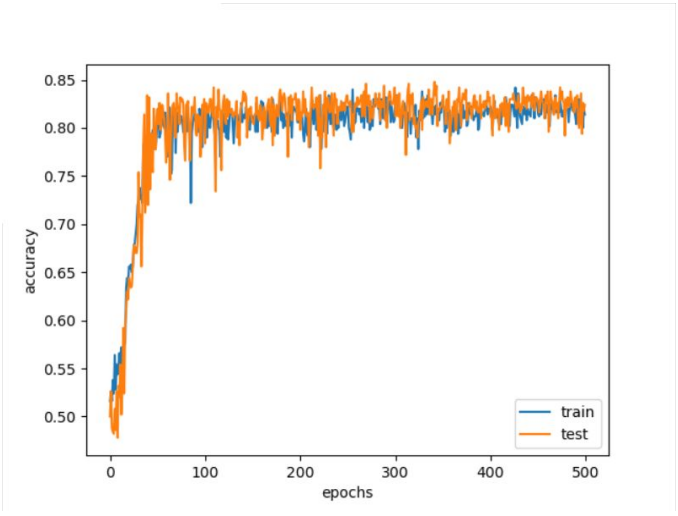


it is obviously that the gradient has been improve.

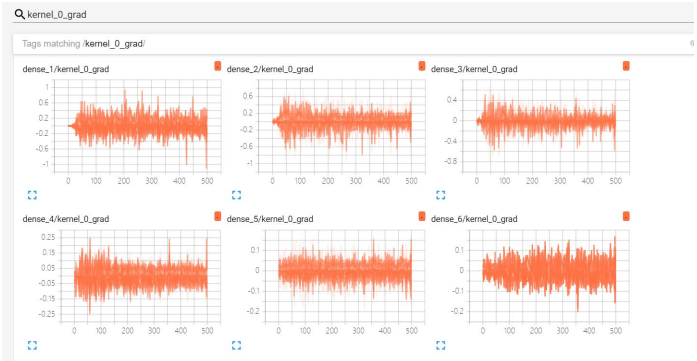
from this part we can't say that Relu can improve the vanish problem. The only thing we can know is that Uniformly distributed initialization weights easily cause the problem of gradient disappearance.

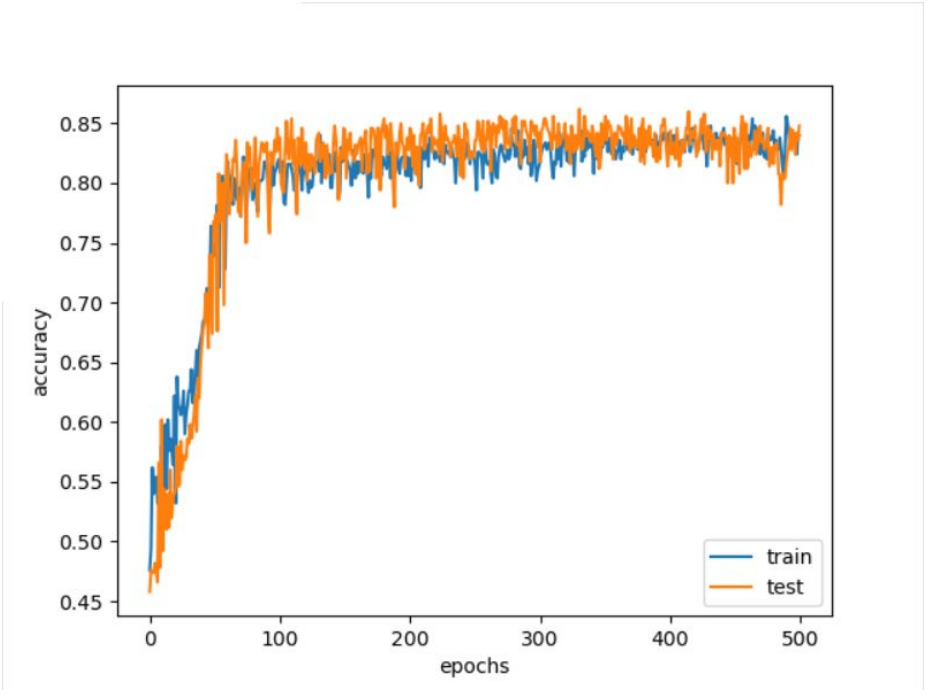
from the tensorboard we can see that the gradient in every layer using relu activation is larger(0.7 times) than using tanh, but we still need an example.

4.tanh-he_initialize-medium model.



only use he_initialize also can improve the result.

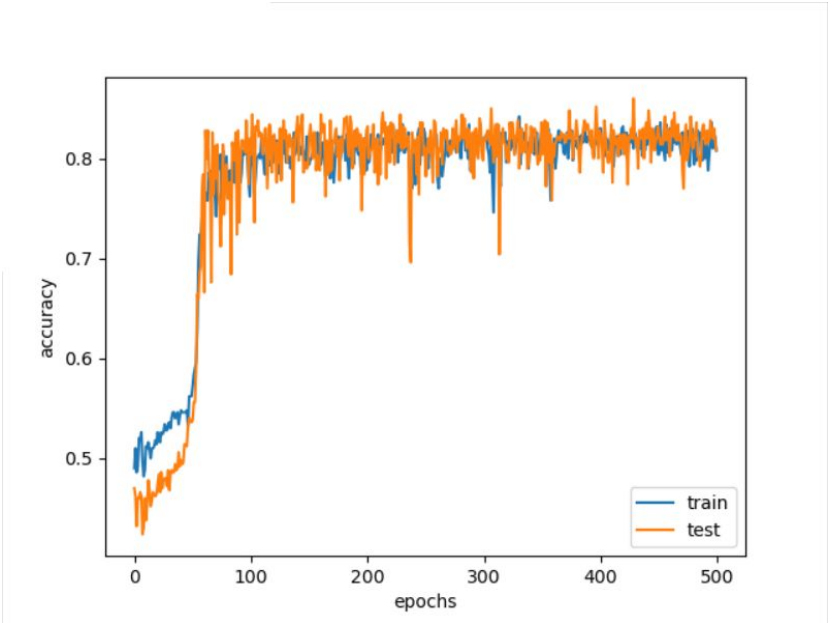


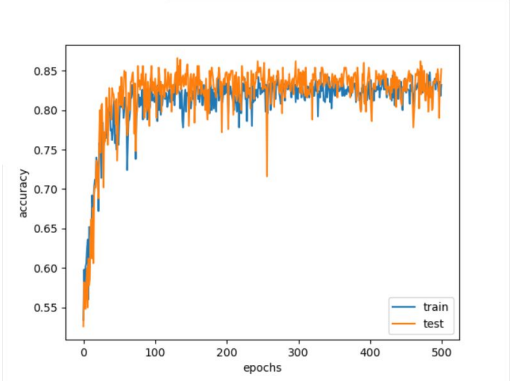


default initialize also can get good result.

5.relu-he_initialize-medium model.

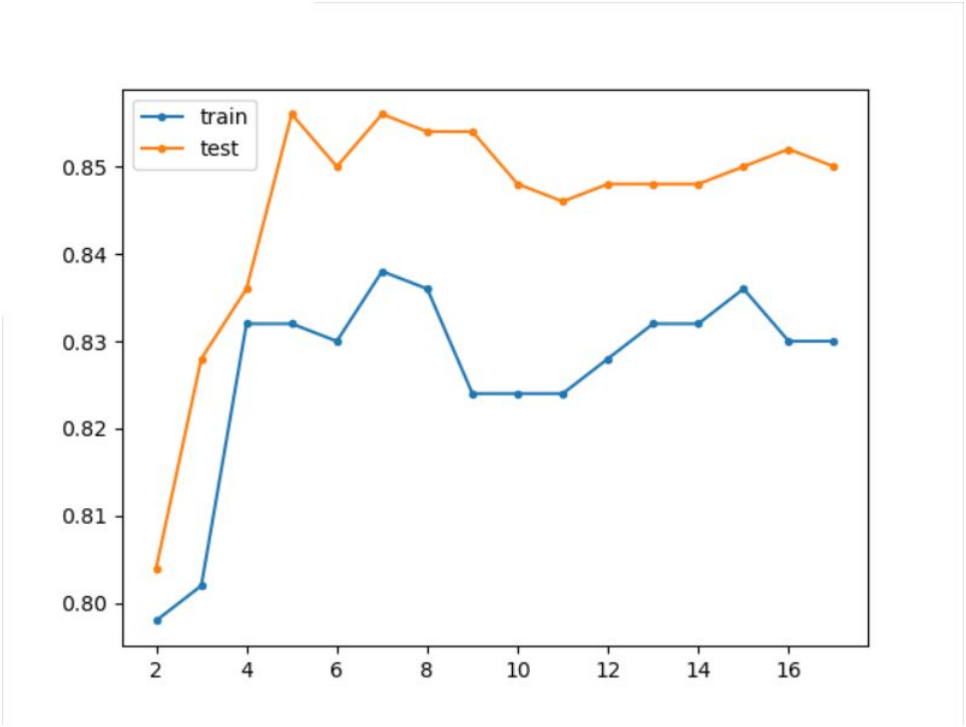
he_nomal



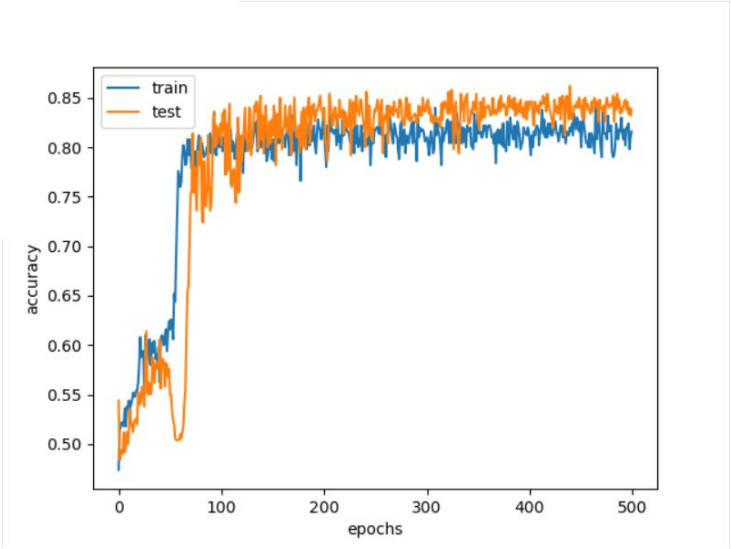


he_uniform

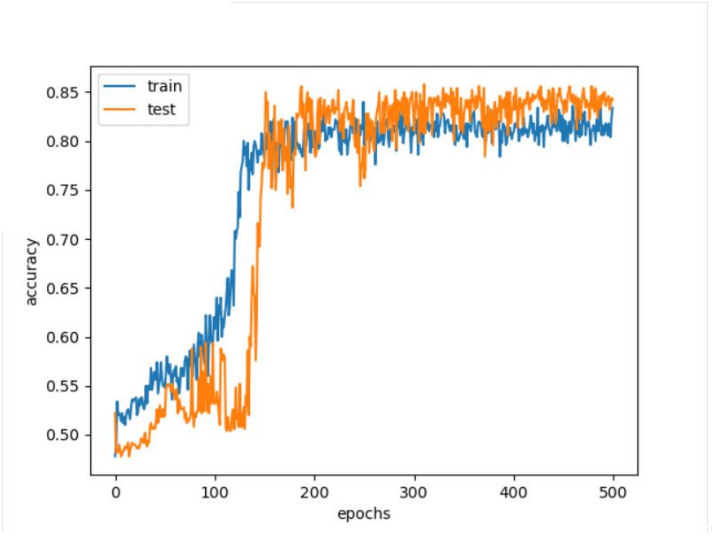
6. tanh-randomuniform-Greedy Layerwise training:(codes in case17)
even have 10 layers, training the model in this way can still get good accuracy.



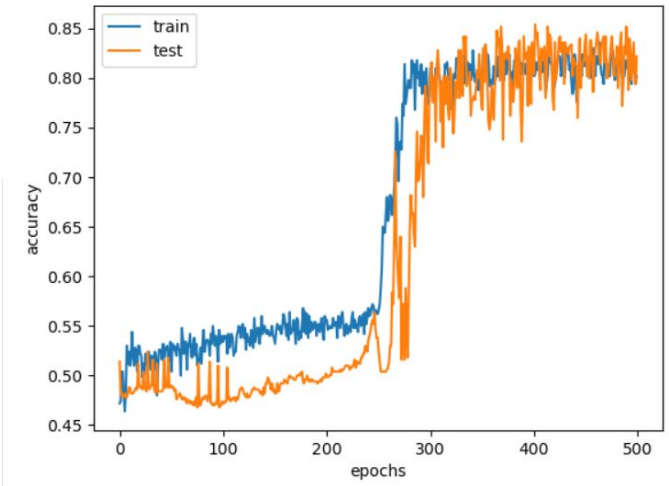
7. batch normalization:



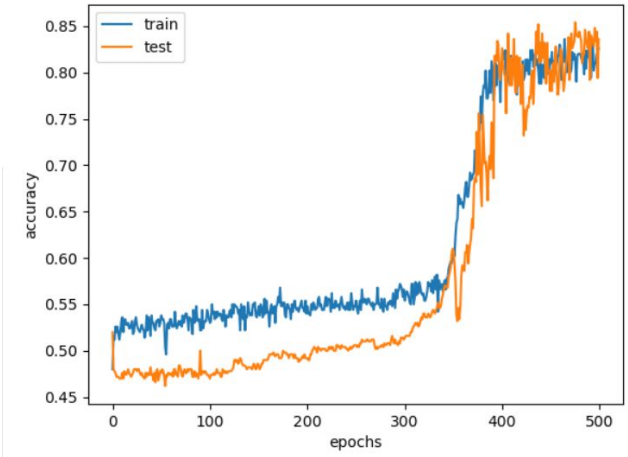
add one dense with BN



add two dense with BN



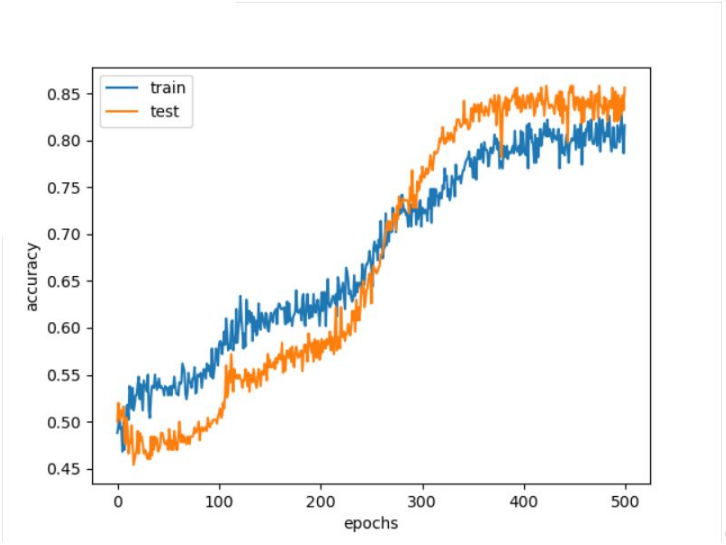
3



4

Batch Normalization can improve the vanish gradient problem, only spend more time.

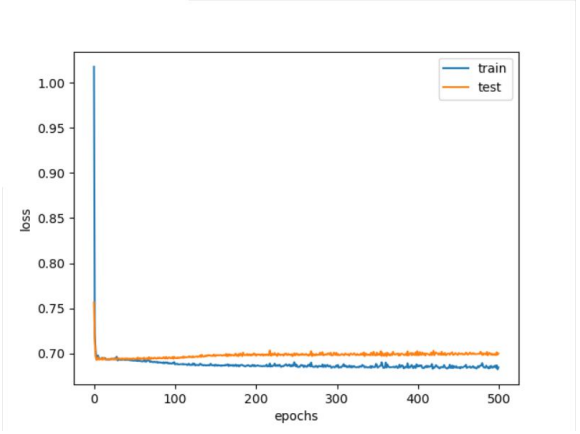
8.dropout(need more units in the layers #)

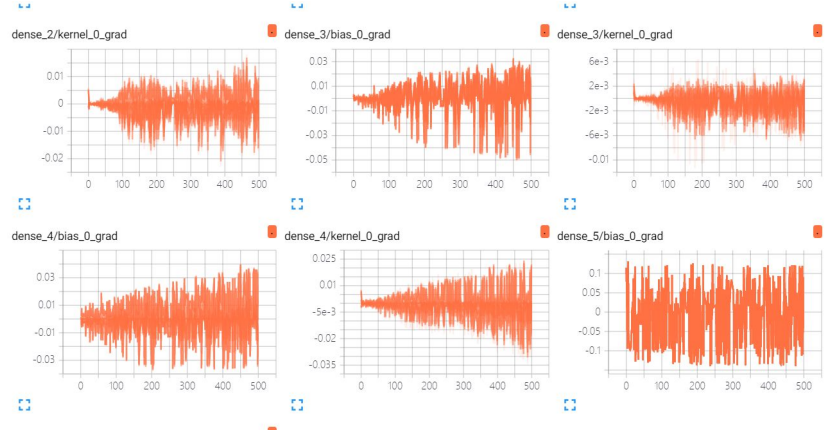


one dense without dropout=0.05, however when the layers number comes to 4, its result didn't well. so do dropout=0.2,0.1. (may because of the amount of unit, we need to set mort units in one dense to test again)

9.unsupervised pre-training(model still need to debug #)

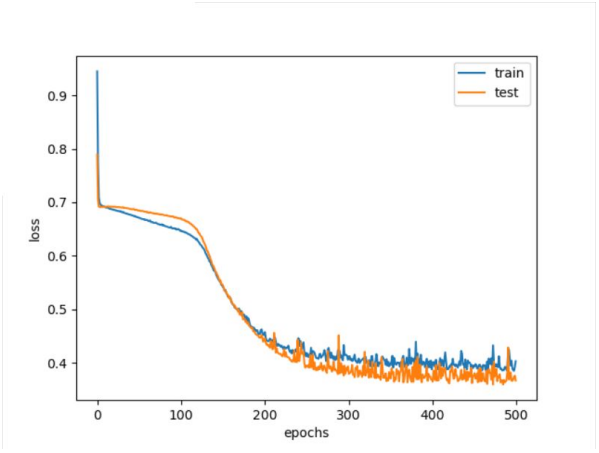
10 comparision：
tanh-RandomUniform 2layers（converge） &5layers（not converge）
4layers:

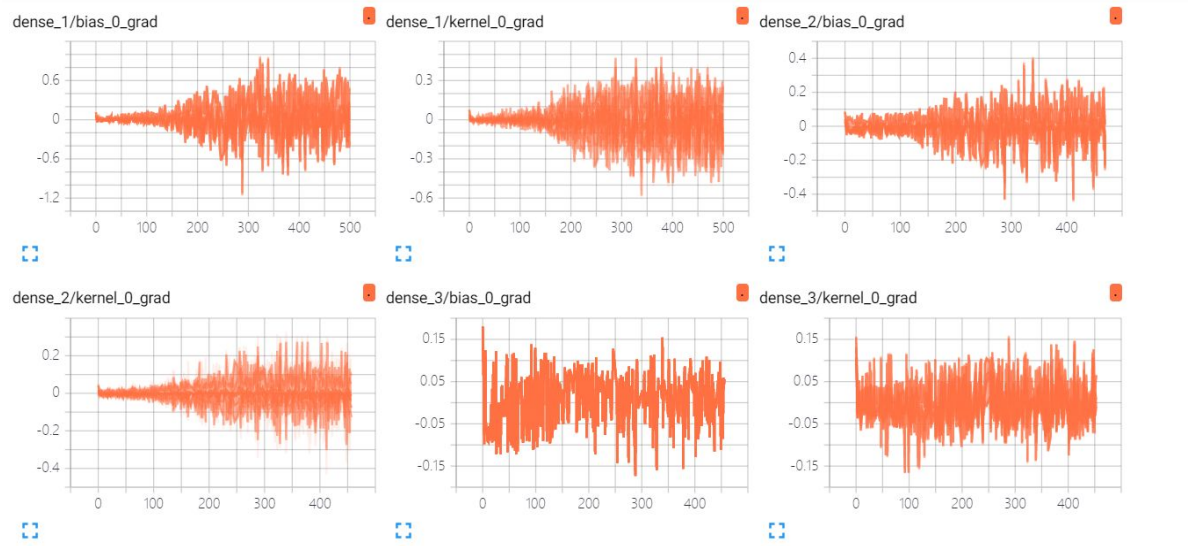




gradient tends to 0 in this situation.

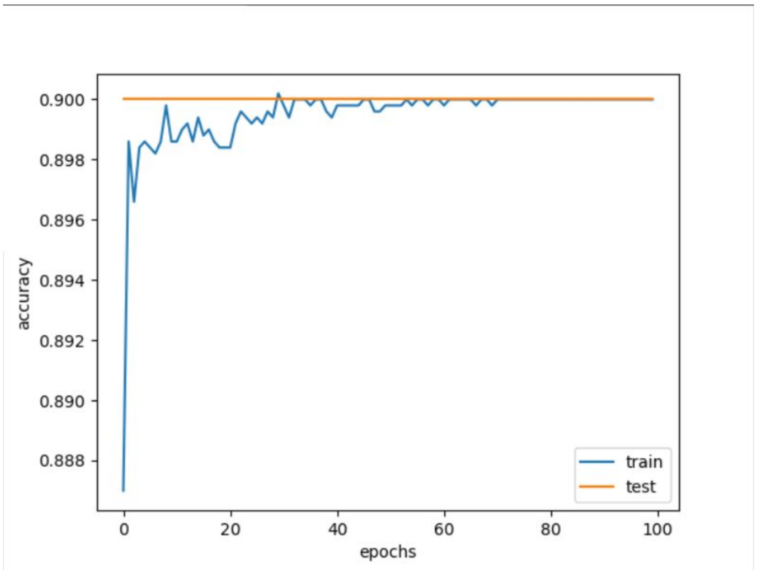
2 layers.

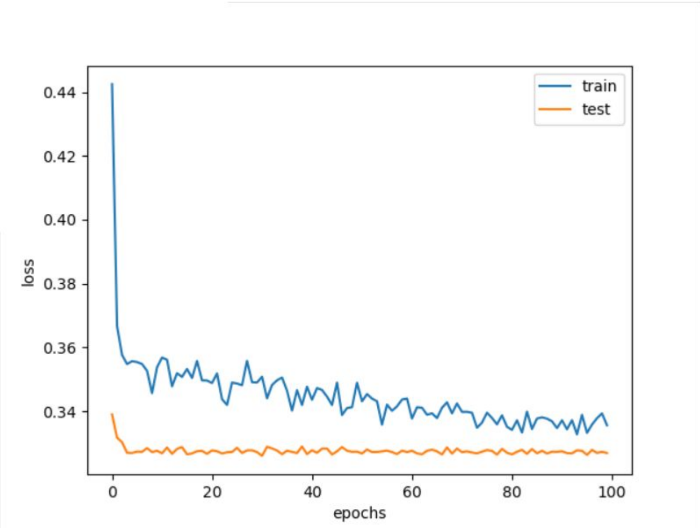


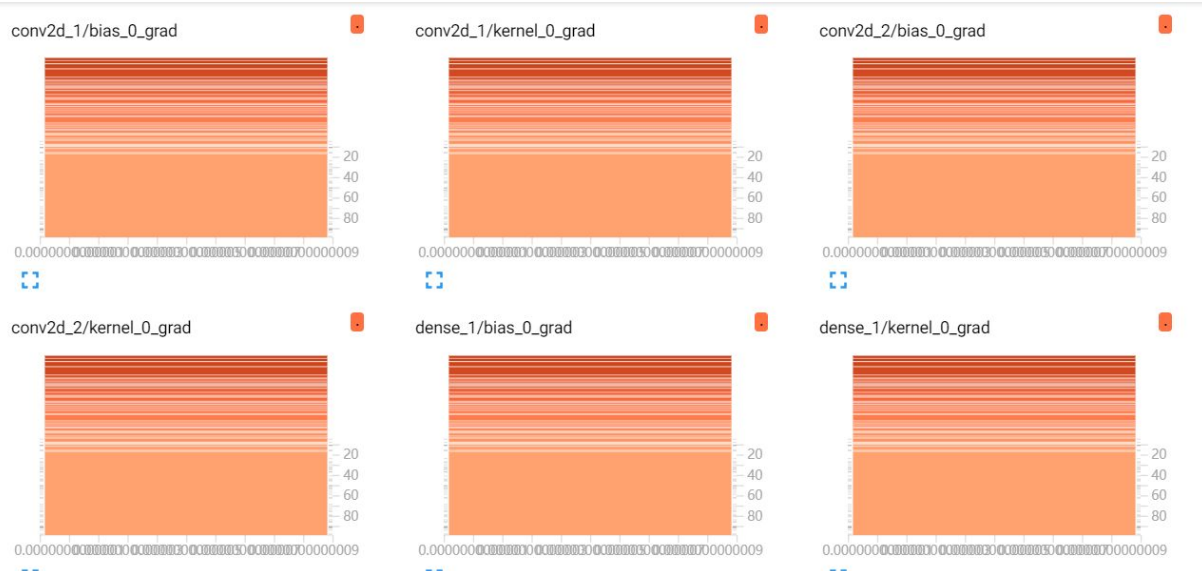


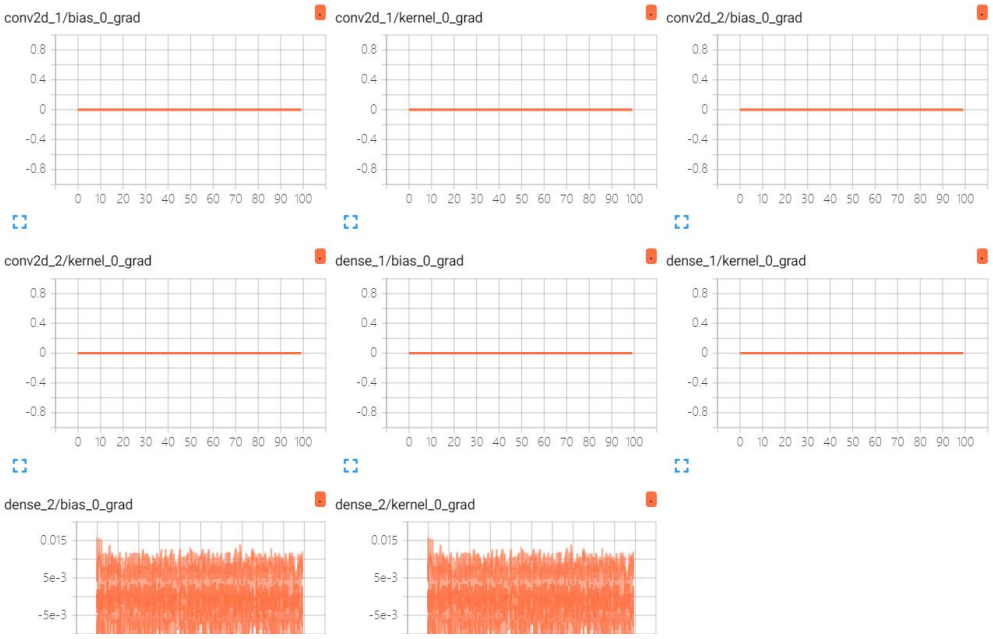
conclusion:
relu, initialization, batch norm, pretraining each layer, can effectively solve this case;
drop out and unsupervised pre-training need time to resign and test.

2. case16-CNN：
   use Minst as dataset, 500 train and 500 test(same as case16's amount)
   1. tanh-RandomUniform:

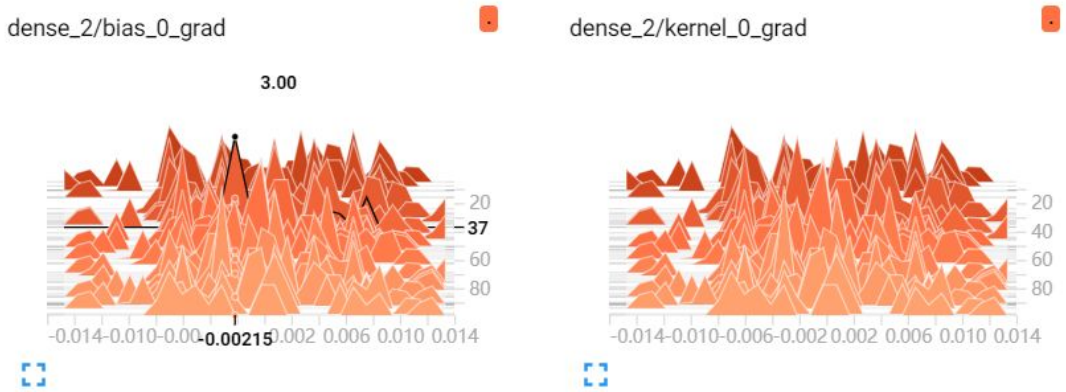

accuracy
0.900
0.898
0.896
0.894
0.892
0.890
0.888
train
test
0
20
40
60
80
100
epochs


loss
0.44
0.42
0.40
0.38
0.36
0.34
train
test
0
20
40
60
80
100
epochs


conv2d_1/bias_0_grad
conv2d_1/kernel_0_grad
conv2d_2/bias_0_grad
conv2d_2/kernel_0_grad
dense_1/bias_0_grad
dense_1/kernel_0_grad
20
40
60
80

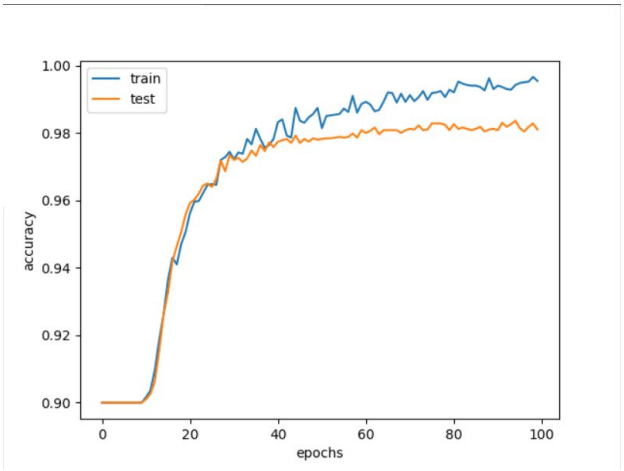


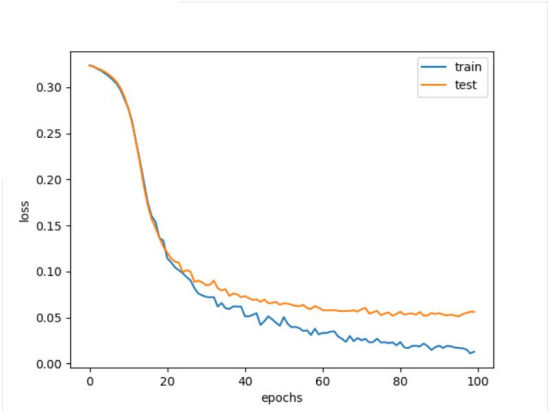
It seems like using tanh-init will lead to gradient 0 in Conv.



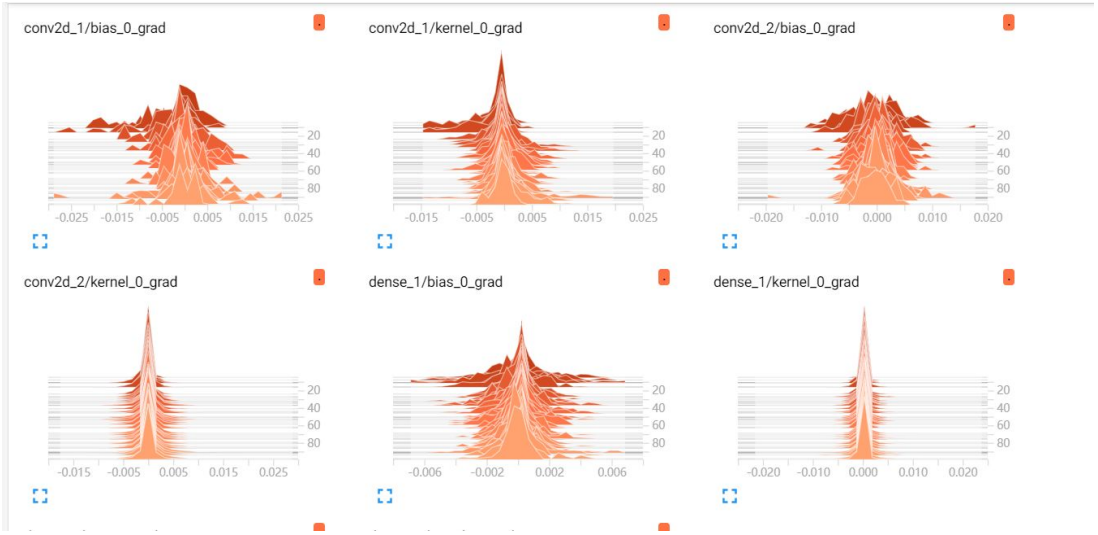
From the above figures, We can see that In this ‘tanh-Randuniform’ situaion,The conv layer grad is stucked and the grad of dense is stay around zero but not zero,
Why this stuck happens? Is it gradient vanish?

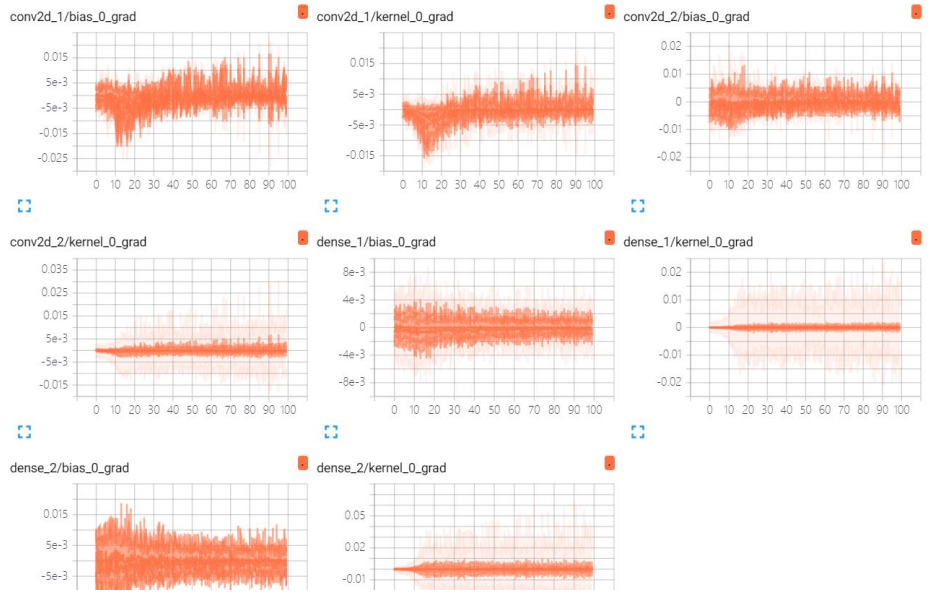
2. relu-'glorot_uniform'(default initializer):

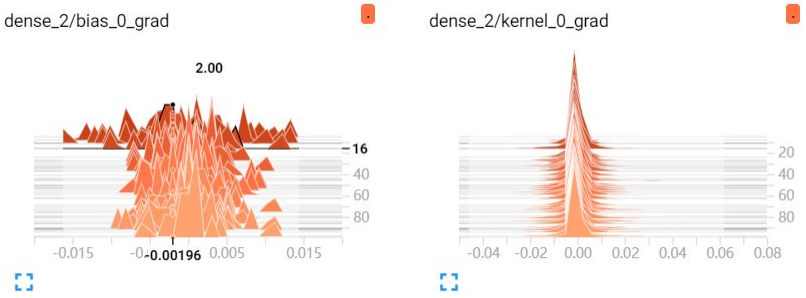




loss and acc have been improved in this setting.



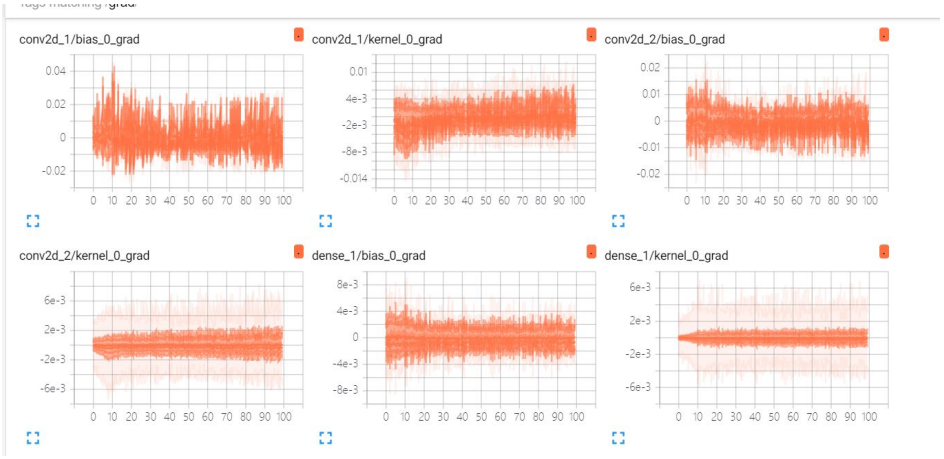




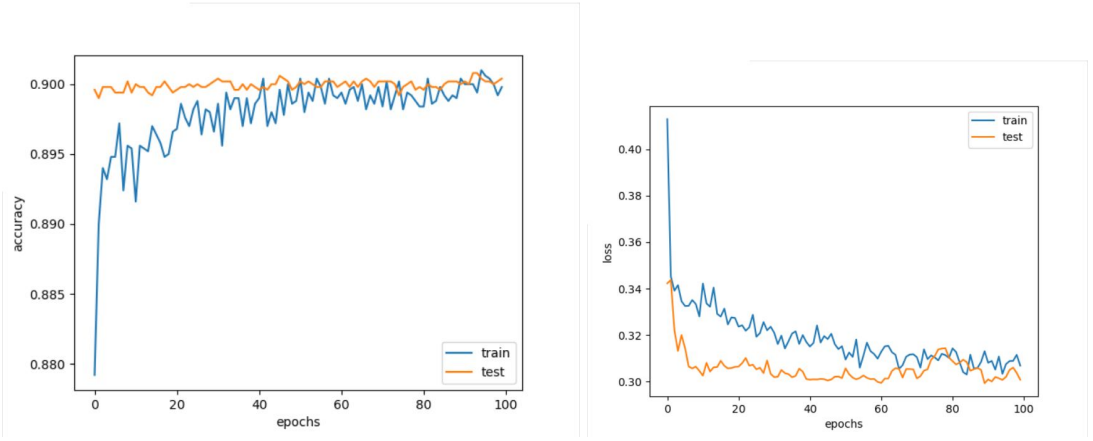
Conv layers have normal gradients now. Using relu-glorot_uniform will improve this situation.

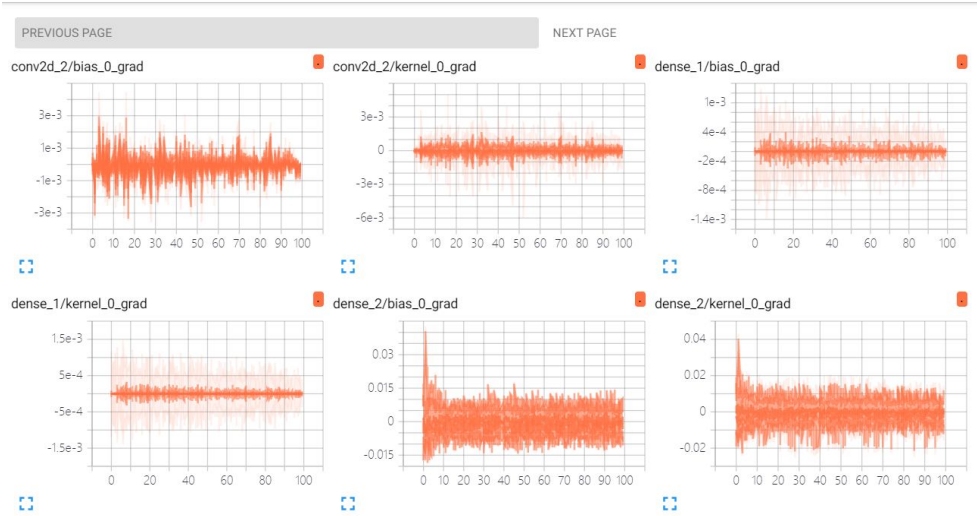
Also I have tested the tanh-glorot and relu -gobleuniform, from these tests We can see that the globaluniform initialize leads to the conv grandient 0. And even using tanh-glorot will perform well:

For CNN gradient 0(vanish), Suggest to use relu-glorot setting.



3. Add batchnormalize(tanh-globaluniform)





The conv2d gradient 0 problem has been solved but the loss convergence is still not well.

.

4. Add more conv&pooling layers

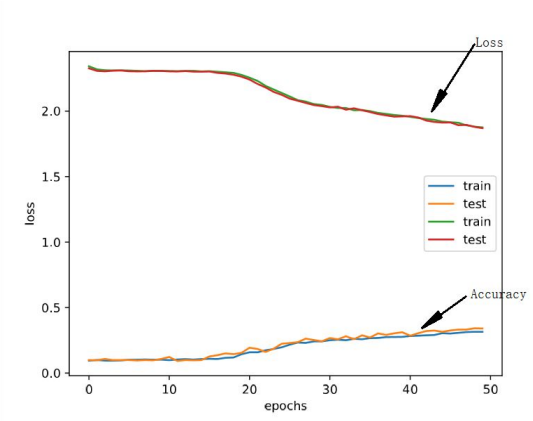
when we add layers (tanh-globaluniform), it still have same problem about 0 gradient in Conv layer. But when we add layers(tanh-glorot), even I add 4 conv&pooling layers, the gradient is still existing and the loss can converge well.

The performance of the above experiments seems to point the cause of the vanishing gradient problem of the convolutional layer to the use of a poor initializer. However, if the default initializer is used, the increase in the number of layers will hardly affect the state of convergence.

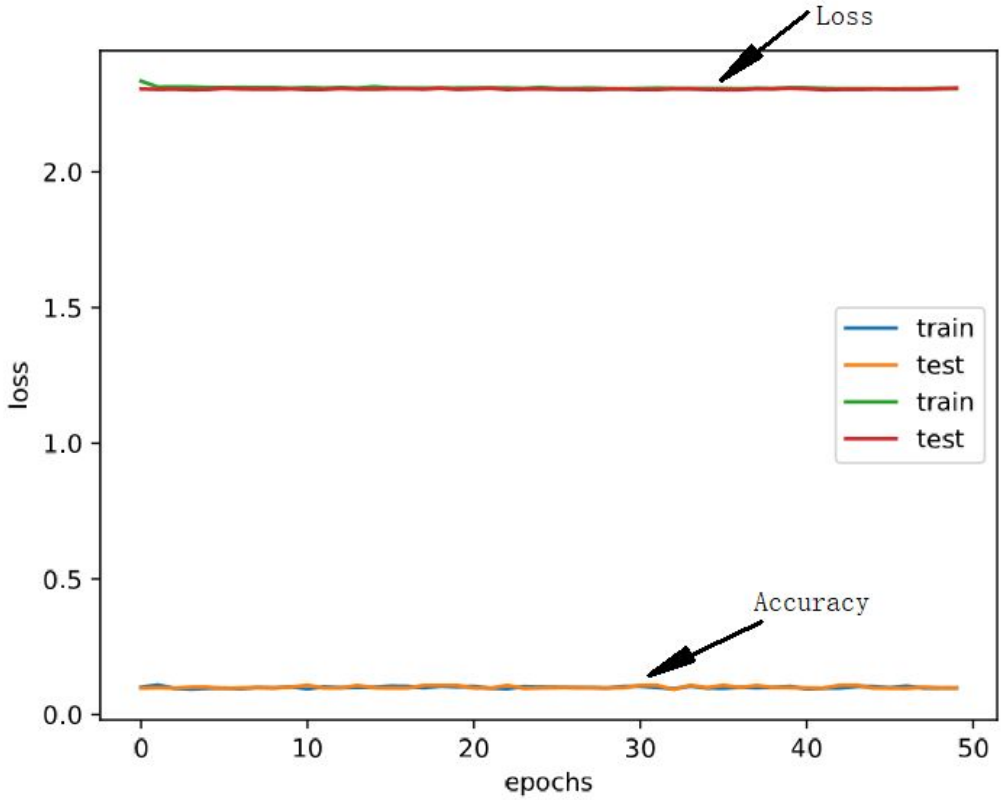
Consider to use VGG19 or deeper to verify the above guess

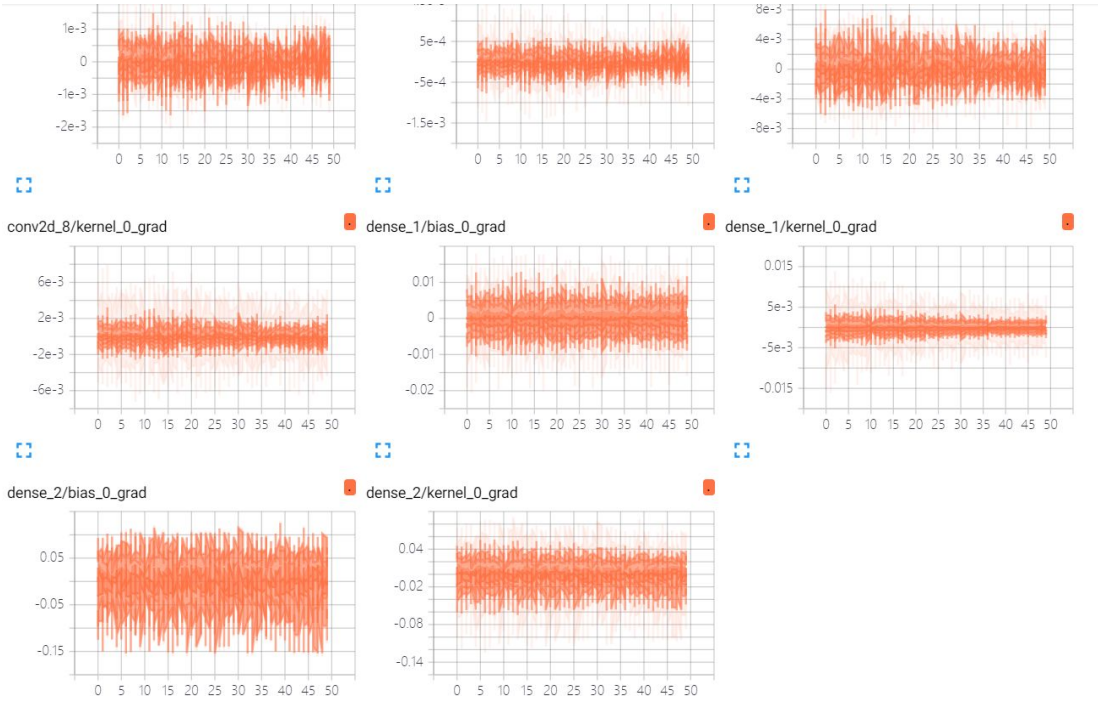
3.case16-CNN（cifar dataset）：

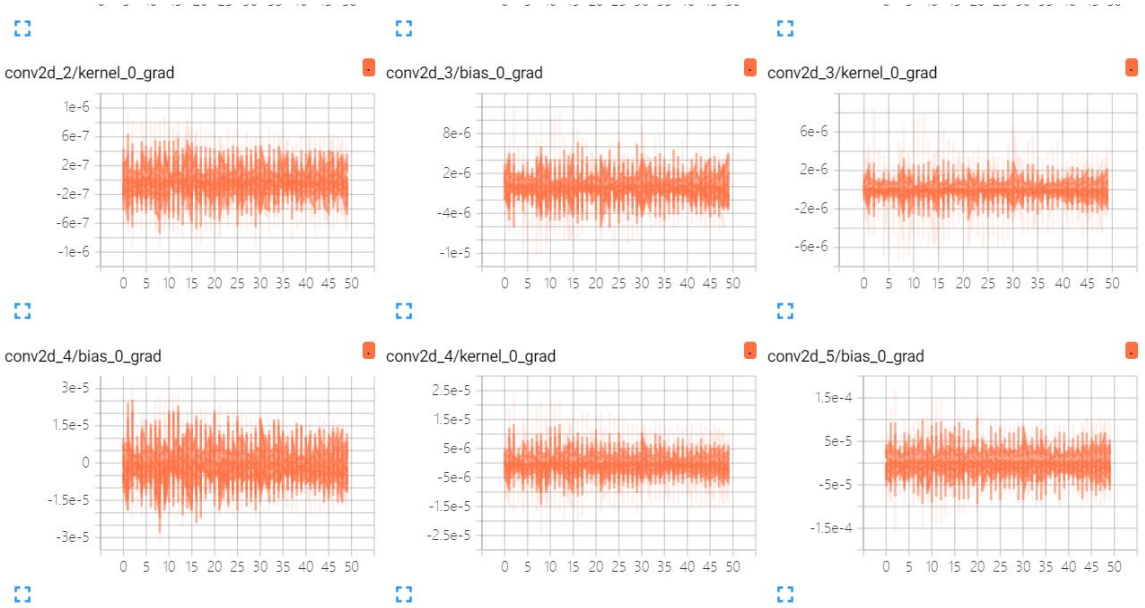
1. 2 Conv2d layer with sigmoid activation function.



2. 8 Conv2d layer with sigmoid activation function.

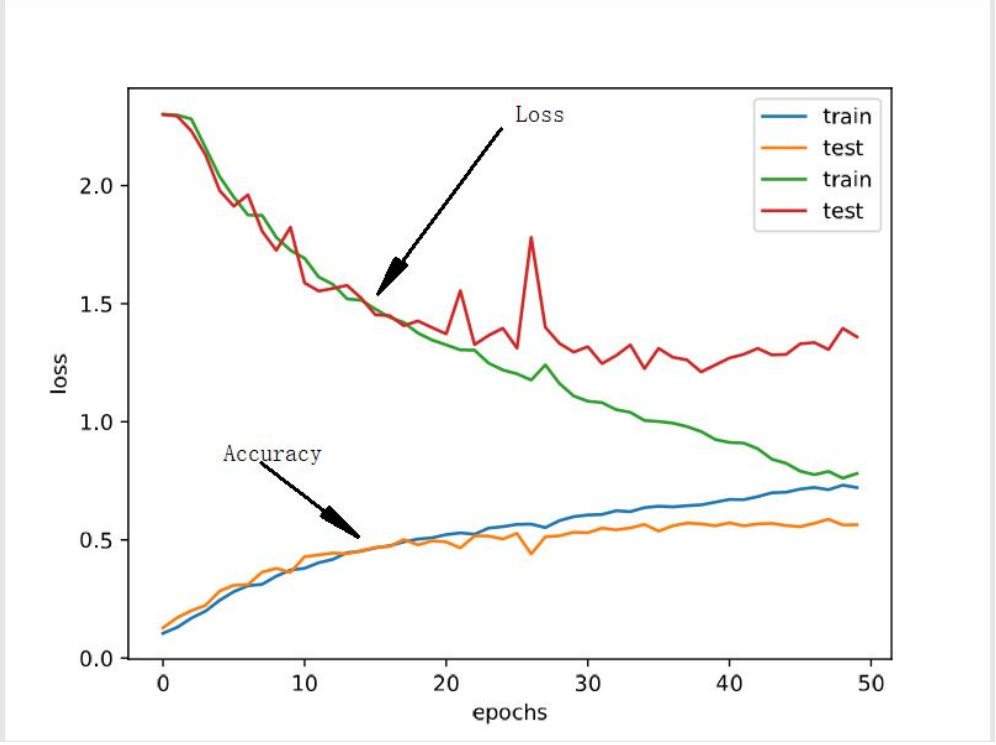

Loss
Accuracy
loss
epochs
train
test
train
test
2.0
1.5
1.0
0.5
0.0
0
10
20
30
40
50


conv2d_8/kernel_0_grad
dense_1/bias_0_grad
dense_1/kernel_0_grad
dense_2/bias_0_grad
dense_2/kernel_0_grad

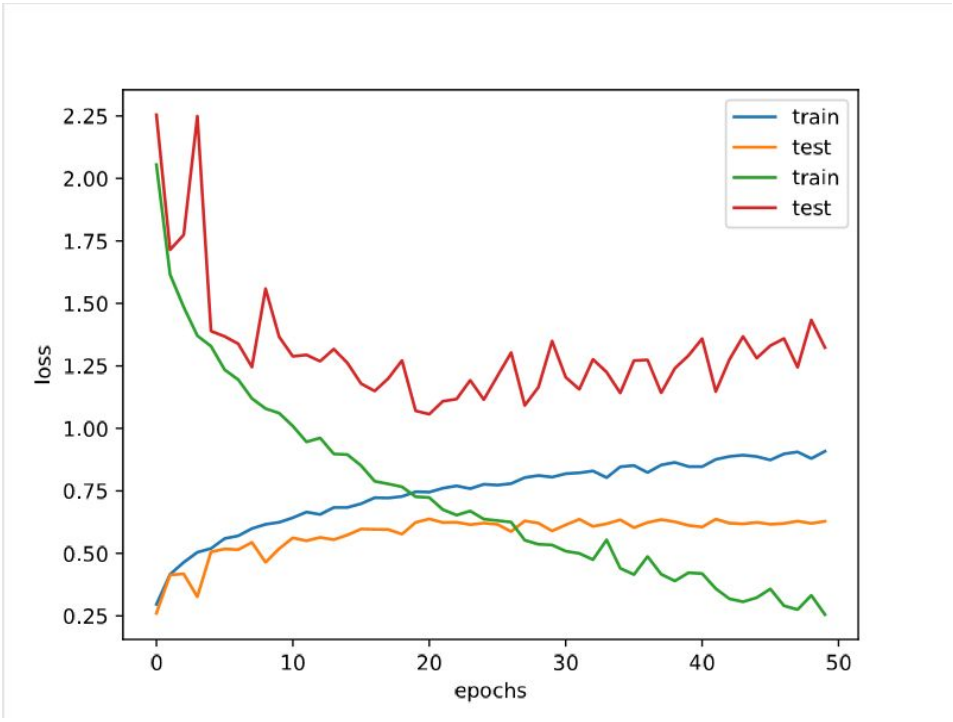


Gradient Decreasing Layer by Layer in Back Propagation——Gradient Vanished

3. 8 Conv2d layer with relu activation function.



4. 8 Conv2d layer with sigmoid activation function add batchnormalization between layers

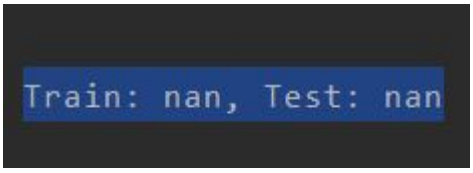


5. greedly pretraining

explode gradient：

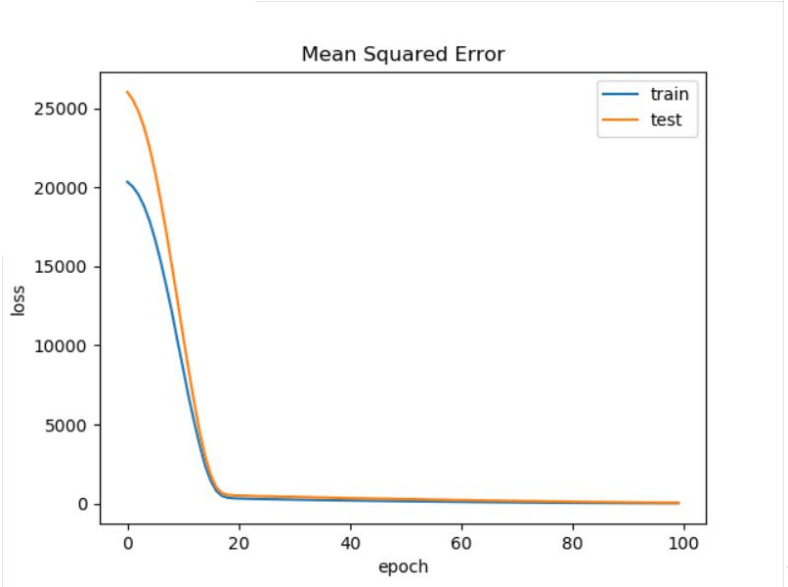
case15:

1.without clipnorm or clip value



return nan, gradient explode lead to nan loss

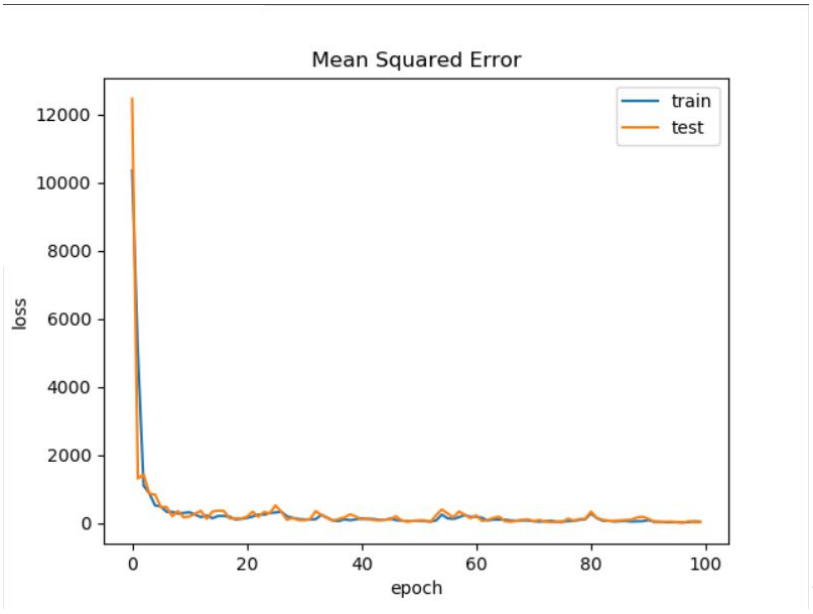
2.clipnorm



Train: 12.159, Test: 48.433;

this solution shows that the clipnorm in ‘optimizer’ can improve this problem.

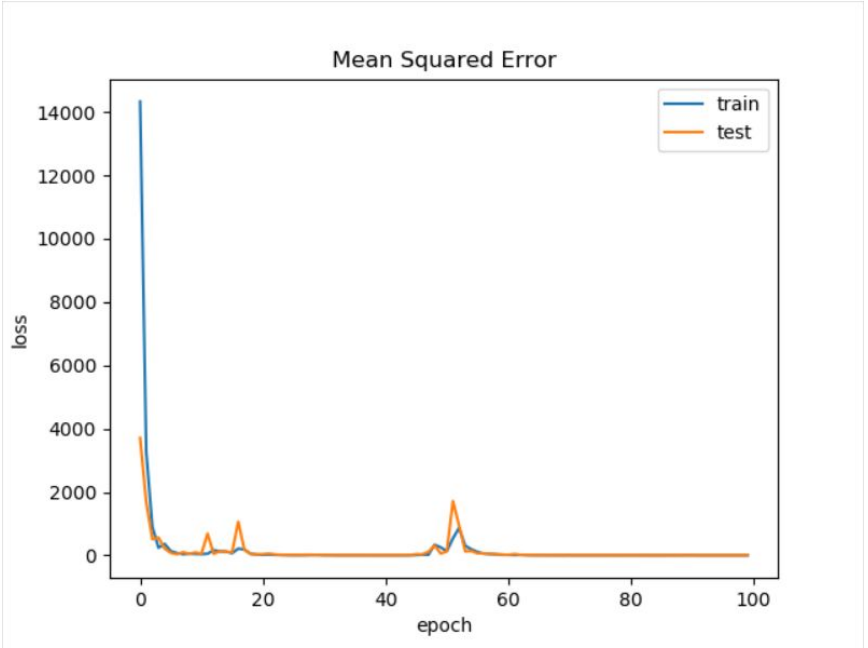
3.clipvalue:



Train: 35.506, Test: 29.890

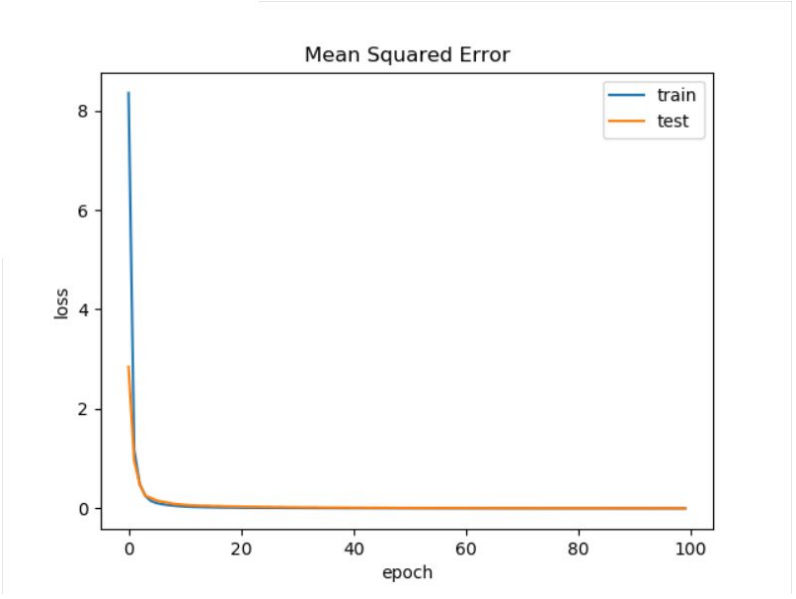
this solution shows that the clipvalue in ‘optimizer’ also works.
Compared with ‘clipnorm’, the loss line didn’t smooth, but it affected quickly.

3.changing parameters:

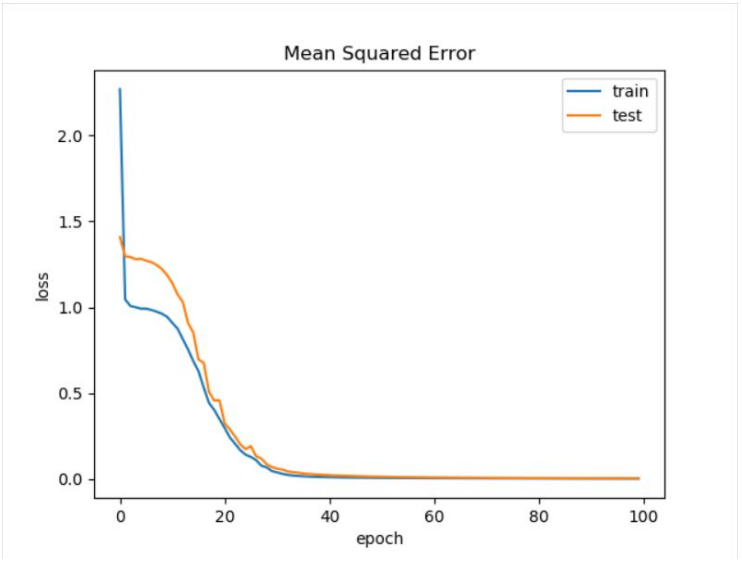


learning rate 0.01-> 0.001

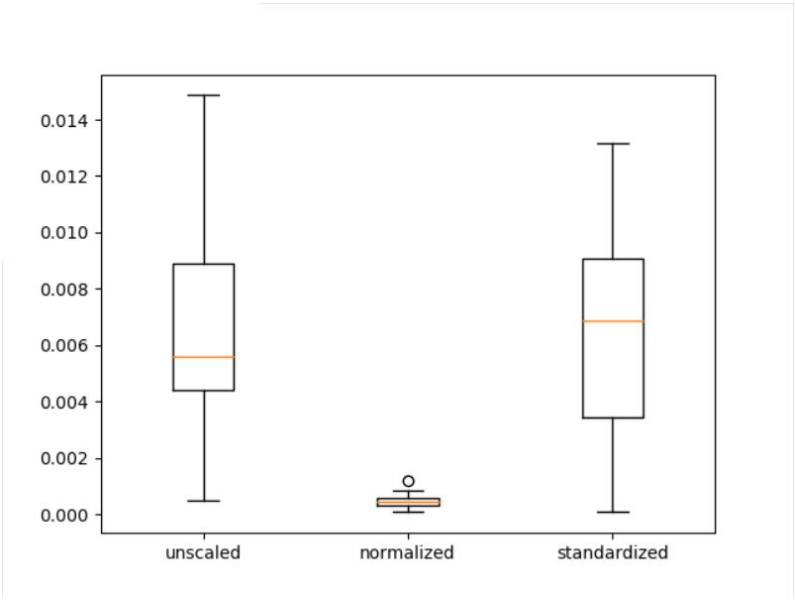
4. rescale the dataset:



use sklearn.scaler StandardScaler as inputscaler and outputscaler to rescale the dataset.



using MaxminSacler as inputscaler and standard as outputscaler.



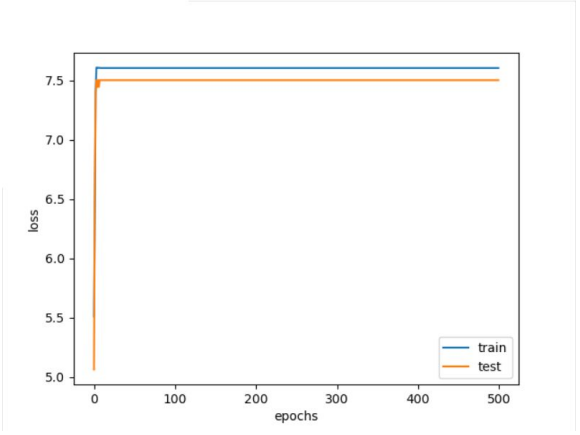
using different scaler to the dataset,

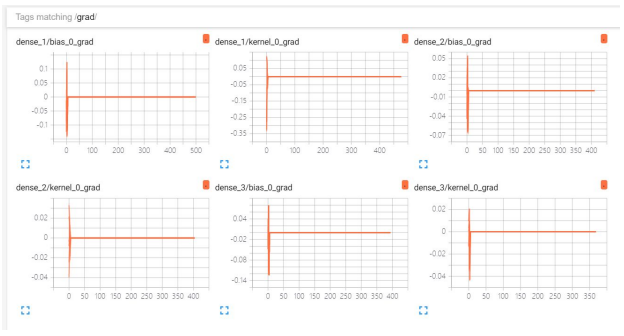
This solution shows that normalize/standard the dataset can sovle the explode gradient.

case26(from case16):

1.

(1)using linear only in the last layer(3layer dense model which should converge):



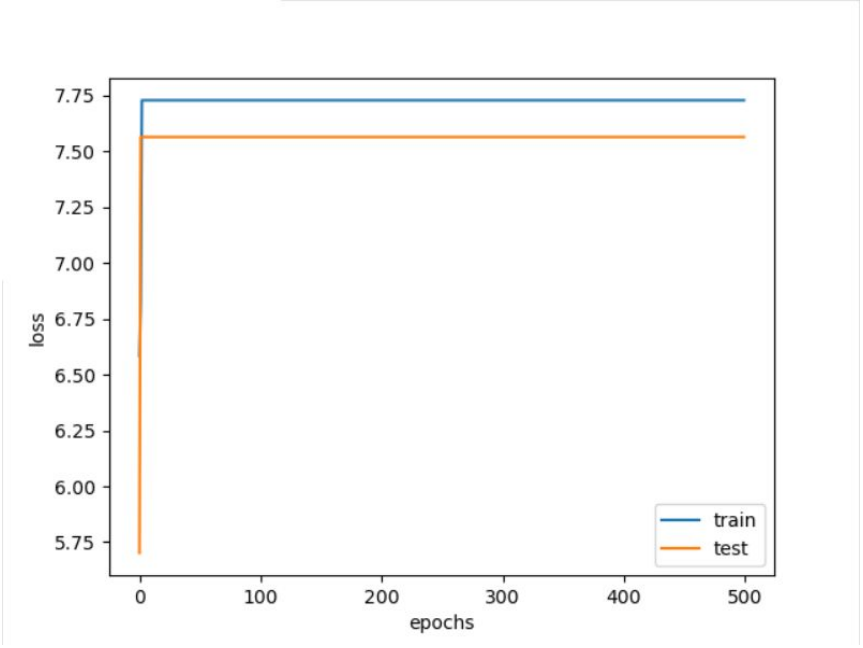


```
 32/500 [>.............................] - ETA: 0s - loss: 6.2944 - accuracy: 0.0625
500/500 [==============================] - 0s 56us/step - loss: 6.6407 - accuracy: 0.1080 - val_loss: 5.8294 - val_accuracy: 0.1820
Epoch 3/500

 32/500 [>.............................] - ETA: 0s - loss: 9.5865 - accuracy: 0.1250
500/500 [==============================] - 0s 78us/step - loss: 7.3877 - accuracy: 0.0720 - val_loss: 7.3953 - val_accuracy: 0.0600
Epoch 4/500
```

loss **increase rapidly and then the val-acc is blocking in 0.**

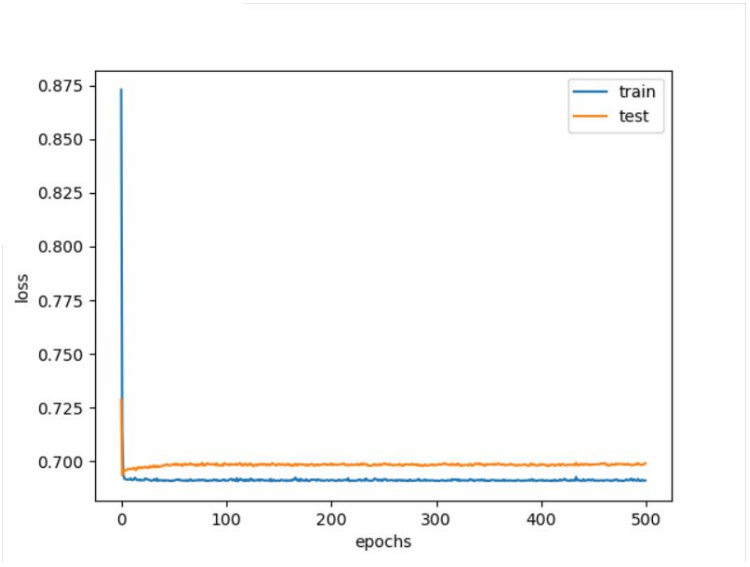
(2)using linear only in all layers
similar to (1), the loss increases quickly and the acc decrease to 0.



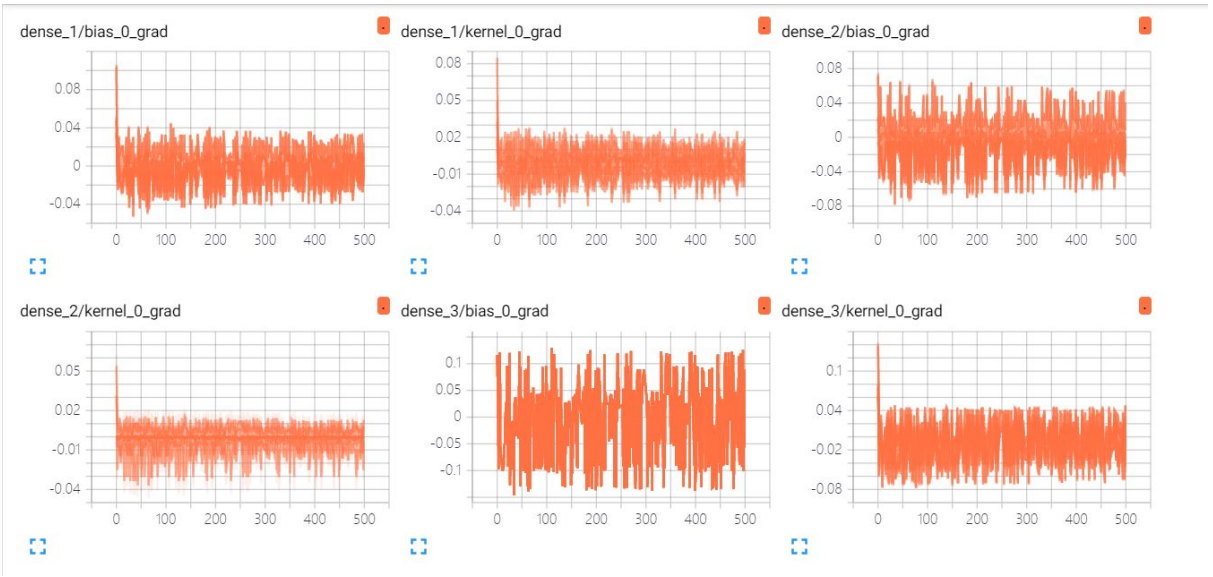

```
 32/500 [>.............................] - ETA: 0s - loss: 6.7083 - accuracy: 0.0000e+00
500/500 [==============================] - 0s 58us/step - loss: 7.7280 - accuracy: 0.0000e+00 - val_loss: 7.5636 - val_accuracy: 0.0000e+00
Epoch 4/500

 32/500 [>.............................] - ETA: 0s - loss: 8.6249 - accuracy: 0.0000e+00
500/500 [==============================] - 0s 52us/step - loss: 7.7280 - accuracy: 0.0000e+00 - val_loss: 7.5636 - val_accuracy: 0.0000e+00
Epoch 5/500
```

(3)using linear only except last layers(last use sigmoid.)

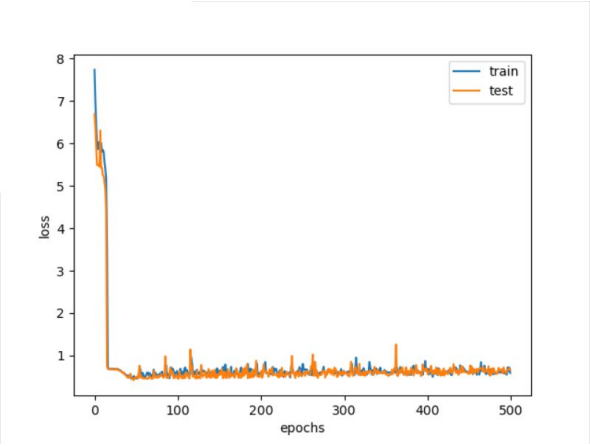


and acc is block around 0.5.



gradient exists but no contribution to the convergency.

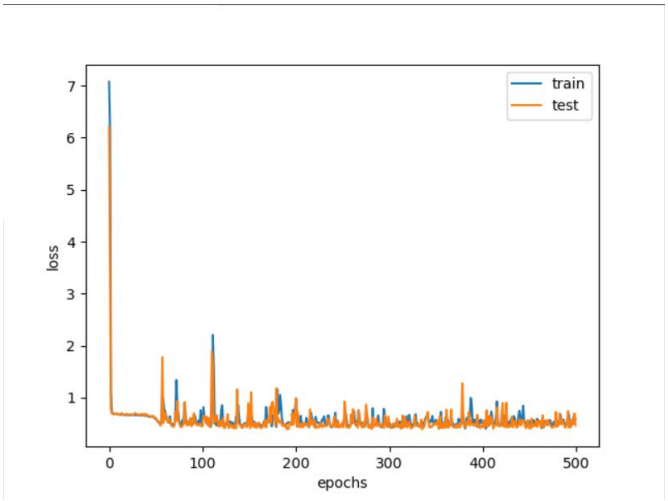
2, using graident clip/gradient norm in the optimizer(SGD).
(1)using linear only in the last layer(3layer dense model）
clipnorm=1.0:




```
 32/500 [>.............................] - ETA: 0s - loss: 0.3588 - accuracy: 0.6250
500/500 [==============================] - 0s 58us/step - loss: 0.5879 - accuracy: 0.6860 - val_loss: 0.6672 - val_accuracy: 0.7080
Train: 0.684, Test: 0.708
```

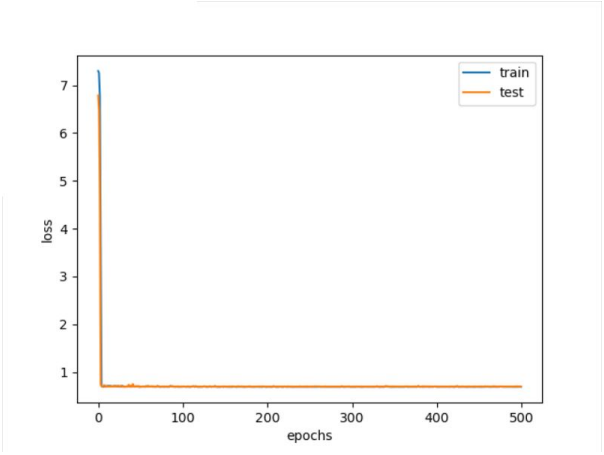
It converges better than before.

clipvalue=2.0:




```
 32/500 [>.............................] - ETA: 0s - loss: 0.4101 - accuracy: 0.8438
500/500 [==============================] - 0s 52us/step - loss: 0.5886 - accuracy: 0.7620 - val_loss: 0.4697 - val_accuracy: 0.7940
Train: 0.798, Test: 0.794
```

(2)using linear only in all layers
clipnorm=1.0




```
 32/500 [>.............................] - ETA: 0s - loss: 0.6705 - accuracy: 0.6250
500/500 [==============================] - 0s 90us/step - loss: 0.6923 - accuracy: 0.5360 - val_loss: 0.6997 - val_accuracy: 0.4900
Train: 0.556, Test: 0.490
```

This picture shows the improvement brought by cip, although it doesn't converge to the global optimal point.

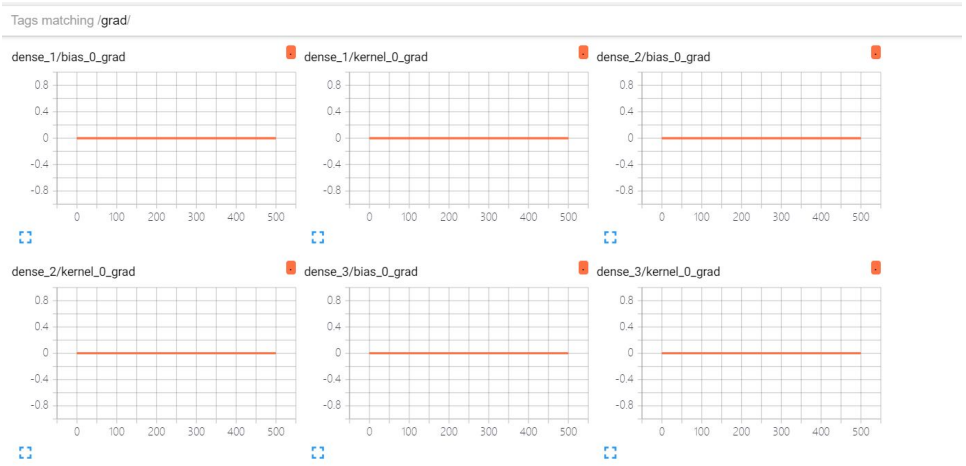
clipvalue=2.0

```
 32/500 [>.............................] - ETA: 0s - loss: 0.6812 - accuracy: 0.5312
500/500 [==============================] - 0s 54us/step - loss: 0.6920 - accuracy: 0.5340 - val_loss: 0.6989 - val_accuracy: 0.4540
Train: 0.526, Test: 0.454
```

same as clipnorm.

(3)using linear only except last layers
clipnorm=1.0&clipvalue=2.0: all block in 0.504 acc and didn't improve any more.

```
 32/500 [>.............................] - ETA: 0s - loss: 7.6666 - accuracy: 0.5000
500/500 [==============================] - 0s 62us/step - loss: 7.7280 - accuracy: 0.4960 - val_loss: 7.5636 - val_accuracy: 0.5040
Train: 0.496, Test: 0.504
```



the gradient is 0 in this situation.

conclusion:

Clipnorm and clipvalue did improve the gradient explode problem in most of cases.

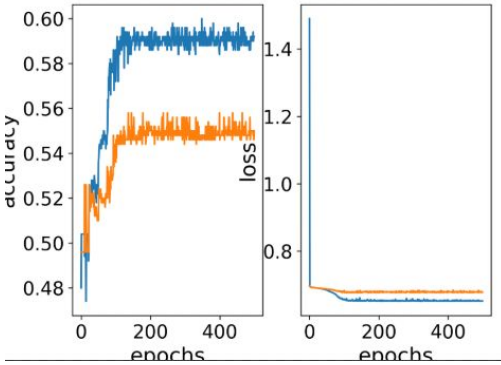
The best way to solve this explode issue is to use 'tanh' or 'relu' to avoid gradient>1 in propagation.

3.using exponential activation in 3 dense model.
kernel_initializer:RandomUniform(minval=0, maxval=1)

```
Epoch 162/500
500/500 [==============================] - 0s 69us/step - loss: nan - accuracy: 0.0000e+00 - val_loss: nan - val_accuracy: 0.0000e+00
Epoch 163/500
500/500 [==============================] - 0s 69us/step - loss: nan - accuracy: 0.0000e+00 - val_loss: nan - val_accuracy: 0.0000e+00
Epoch 164/500
500/500 [==============================] - 0s 58us/step - loss: nan - accuracy: 0.0000e+00 - val_loss: nan - val_accuracy: 0.0000e+00
Epoch 165/500
500/500 [==============================] - 0s 62us/step - loss: nan - accuracy: 0.0000e+00 - val_loss: nan - val_accuracy: 0.0000e+00
Epoch 166/500
500/500 [==============================] - 0s 59us/step - loss: nan - accuracy: 0.0000e+00 - val_loss: nan - val_accuracy: 0.0000e+00
Epoch 167/500
500/500 [==============================] - 0s 60us/step - loss: nan - accuracy: 0.0000e+00 - val_loss: nan - val_accuracy: 0.0000e+00
Epoch 168/500
500/500 [==============================] - 0s 66us/step - loss: nan - accuracy: 0.0000e+00 - val_loss: nan - val_accuracy: 0.0000e+00
Epoch 169/500
500/500 [==============================] - 0s 68us/step - loss: nan - accuracy: 0.0000e+00 - val_loss: nan - val_accuracy: 0.0000e+00
Epoch 170/500
500/500 [==============================] - 0s 68us/step - loss: nan - accuracy: 0.0000e+00 - val_loss: nan - val_accuracy: 0.0000e+00
Epoch 171/500
500/500 [==============================] - 0s 64us/step - loss: nan - accuracy: 0.0000e+00 - val_loss: nan - val_accuracy: 0.0000e+00
Epoch 172/500
500/500 [==============================] - 0s 75us/step - loss: nan - accuracy: 0.0000e+00 - val_loss: nan - val_accuracy: 0.0000e+00
Epoch 173/500
500/500 [==============================] - 0s 71us/step - loss: nan - accuracy: 0.0000e+00 - val_loss: nan - val_accuracy: 0.0000e+00
Epoch 174/500
500/500 [==============================] - 0s 63us/step - loss: nan - accuracy: 0.0000e+00 - val_loss: nan - val_accuracy: 0.0000e+00
Epoch 175/500
500/500 [==============================] - 0s 62us/step - loss: nan - accuracy: 0.0000e+00 - val_loss: nan - val_accuracy: 0.0000e+00
Epoch 176/500
500/500 [==============================] - 0s 63us/step - loss: nan - accuracy: 0.0000e+00 - val_loss: nan - val_accuracy: 0.0000e+00
```

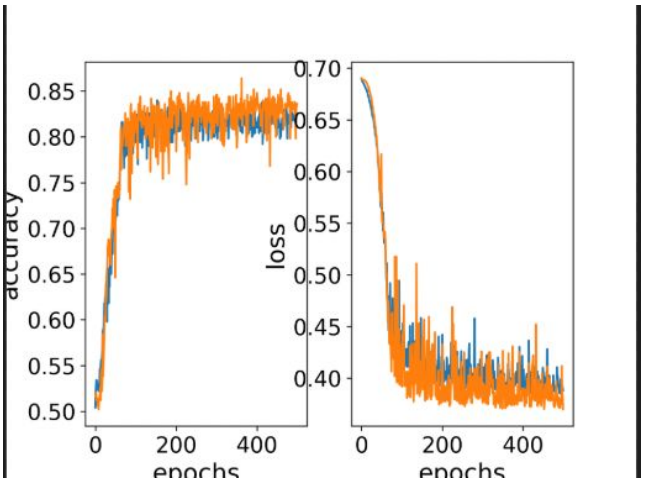
get NaN loss and 0 acc. ----Gradient explode

1. **Change activation:# can improve this issue,but for the longer model, some activation may vanish**
    a. **exp: NaN loss----explode**
    b. **relu:**
        i. **RandomUniform(minval=0, maxval=1)**



improved ,no nan loss.

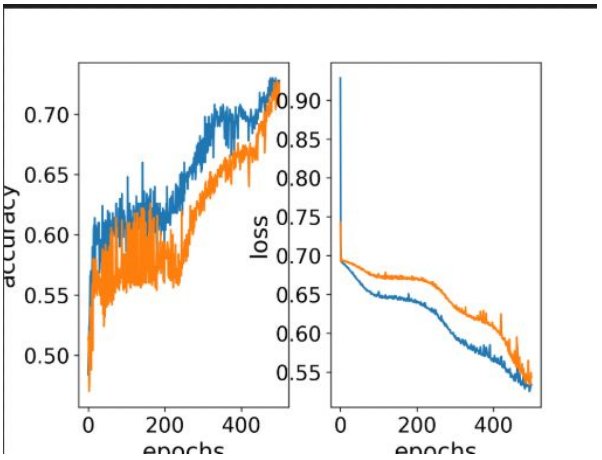
        ii. **default:**



improved well.

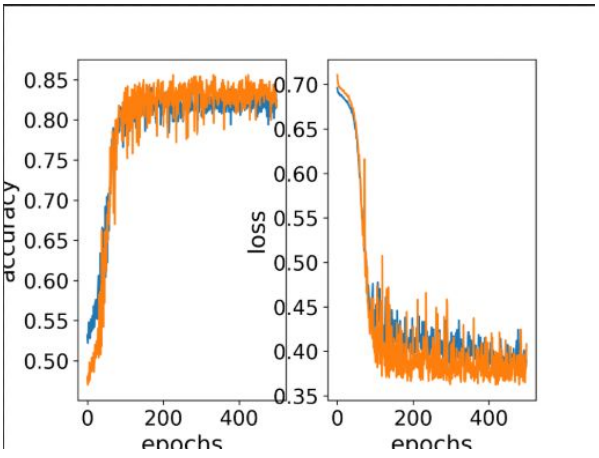
c. tanh

i. RandomUniform(minval=0, maxval=1)



improved well

ii. default



perform better.

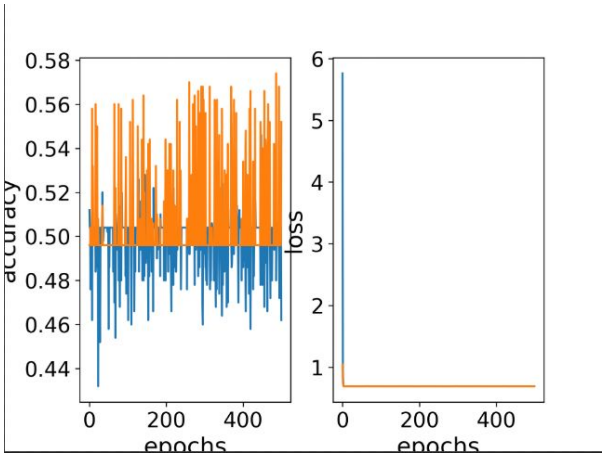
d. sigmoid

i. RandomUniform(minval=0, maxval=1)
no nan but not converge

ii. default

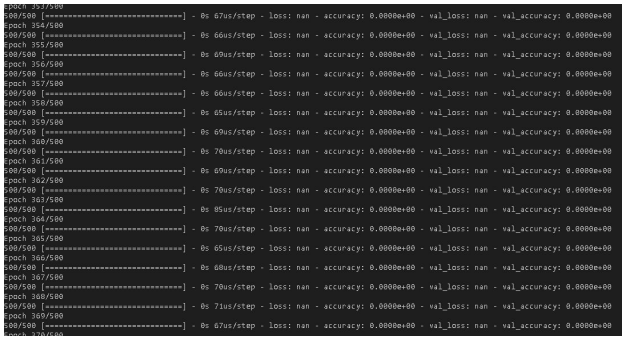
no nan but not converge

2. **Change initializer:# can't improve too much.**
   a. **RandomUniform(minval=0, maxval=1): NaN loss----explode**
   b. **RandomUniform(minval=0, maxval=0.5):**



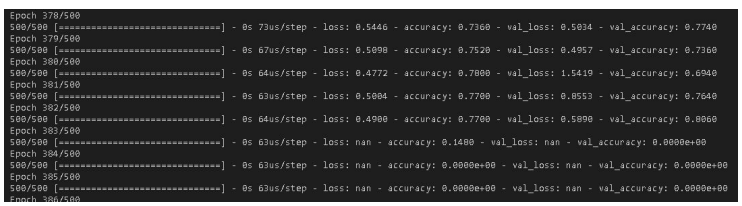
no nan loss but not converge.

   c. **RandomUniform(minval=-1, maxval=0.5):**



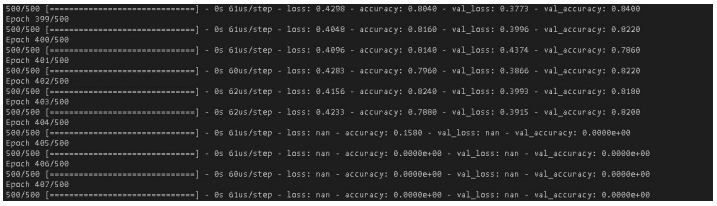
still nan

   d. **RandomUniform(minval=-1, maxval=1):**

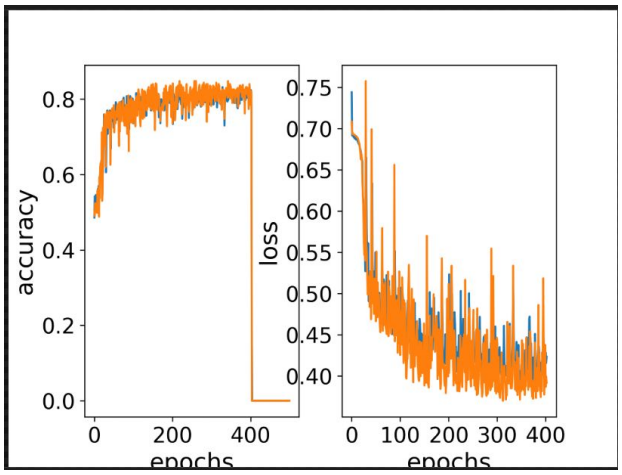


still nan.

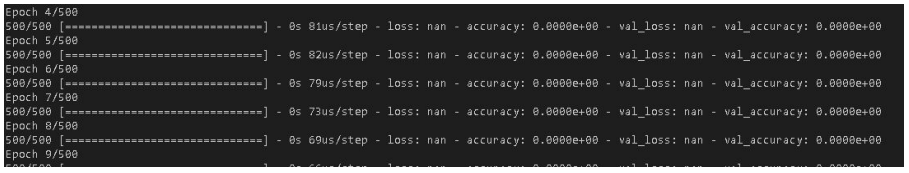
   e. **'glorot_uniform'(default initializer.)**



converge in the start and then turn to nan.

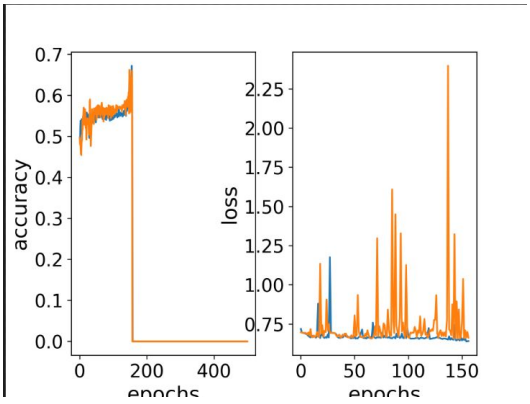


f. 'he_uniform'



still nan.

3. using clipnorm/clipvalue# cannot improve in this situation.
   a. clipnorm1
      i. RandomUniform(minval=0, maxval=1)
      still nan
      ii. default

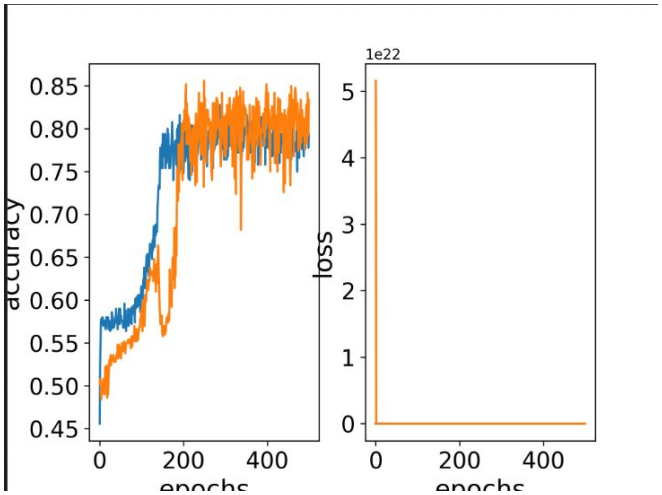


still nan

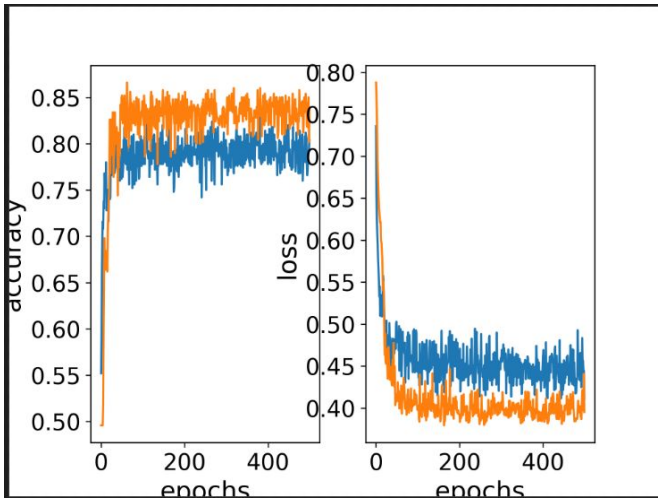
   b. clipvalue2
      i. RandomUniform(minval=0, maxval=1)
      still nan
      ii. default

```
Epoch 88/500
500/500 [==============================] - 0s 69us/step - loss: 0.5028 - accuracy: 0.7560 - val_loss: 0.5990 - val_accuracy: 0.6720
Epoch 89/500
500/500 [==============================] - 0s 71us/step - loss: 0.5259 - accuracy: 0.7520 - val_loss: 0.5565 - val_accuracy: 0.7300
Epoch 90/500
500/500 [==============================] - 0s 78us/step - loss: 0.5297 - accuracy: 0.7280 - val_loss: 0.4424 - val_accuracy: 0.8140
Epoch 91/500
500/500 [==============================] - 0s 69us/step - loss: 0.5087 - accuracy: 0.7660 - val_loss: 0.5380 - val_accuracy: 0.7440
Epoch 92/500
500/500 [==============================] - 0s 71us/step - loss: nan - accuracy: 0.5180 - val_loss: nan - val_accuracy: 0.0000e+00
Epoch 93/500
500/500 [==============================] - 0s 70us/step - loss: nan - accuracy: 0.0000e+00 - val_loss: nan - val_accuracy: 0.0000e+00
Epoch 94/500
500/500 [==============================] - 0s 75us/step - loss: nan - accuracy: 0.0000e+00 - val_loss: nan - val_accuracy: 0.0000e+00
Epoch 95/500
500/500 [==============================] - 0s 68us/step - loss: nan - accuracy: 0.0000e+00 - val_loss: nan - val_accuracy: 0.0000e+00
Epoch 96/500
500/500 [==============================] - 0s 66us/step - loss: nan - accuracy: 0.0000e+00 - val_loss: nan - val_accuracy: 0.0000e+00
```

4. **batchnormlization.# can solve the explode well.**
    a. **RandomUniform(minval=0, maxval=1)**



    b. **default**



improved well

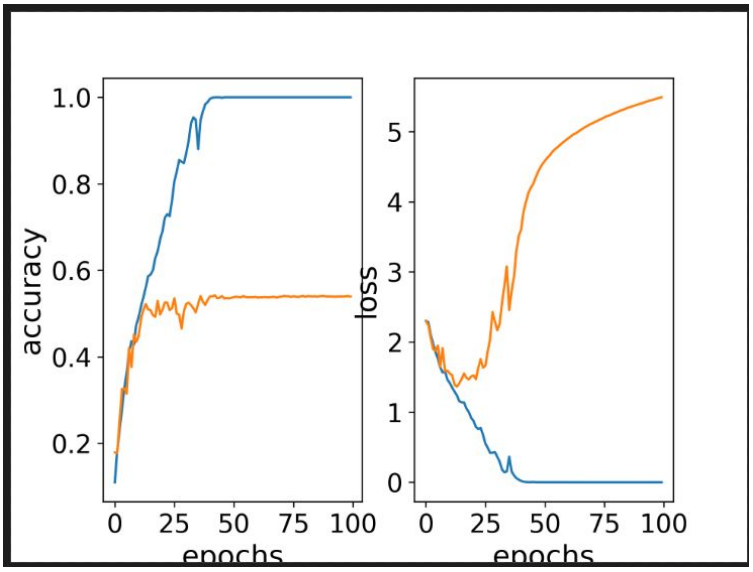
case26(CNN)(from case 16 cnn):

1. **exp activation 4conv2maxpool model, sgd+default initializer.**

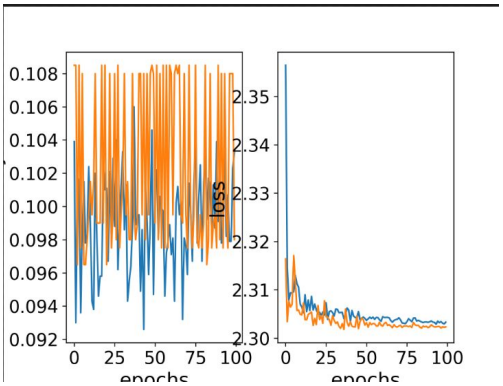
```
Train on 10000 samples, validate on 2000 samples
Epoch 1/100
10000/10000 [==============================] - 4s 352us/step - loss: nan - accuracy: 0.1005 - val_loss: nan - val_accuracy: 0.0980
Epoch 2/100
10000/10000 [==============================] - 4s 371us/step - loss: nan - accuracy: 0.1005 - val_loss: nan - val_accuracy: 0.0980
Epoch 3/100
10000/10000 [==============================] - 3s 326us/step - loss: nan - accuracy: 0.1005 - val_loss: nan - val_accuracy: 0.0980
Epoch 4/100
10000/10000 [==============================] - 3s 323us/step - loss: nan - accuracy: 0.1005 - val_loss: nan - val_accuracy: 0.0980
```

nan happens in the start.

2. **Change activation# can solve this explode well.**
   a. **relu**

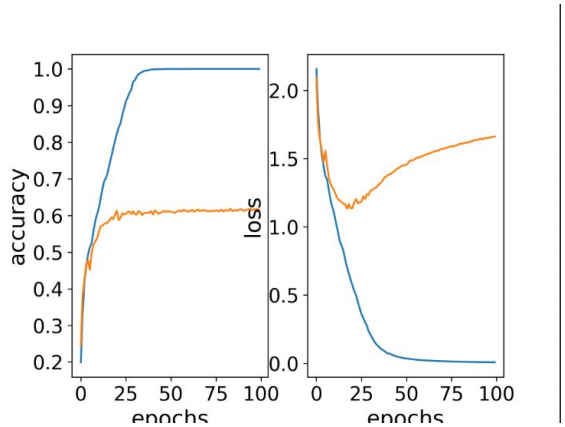


   b. **sigmoid**



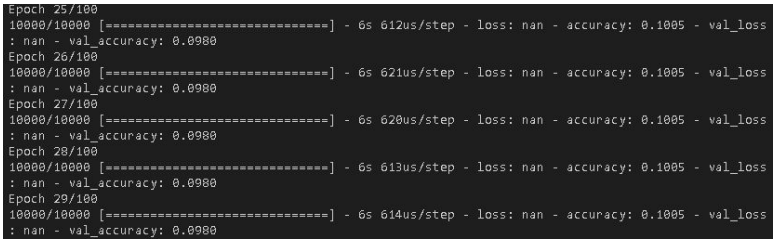
seems like gradient vanish in sigmoid

   c. **tanh**



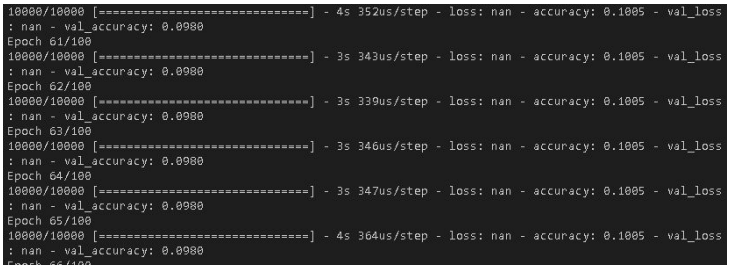
3. **clipnorm/clipvalue**

a. **cilpnorm**



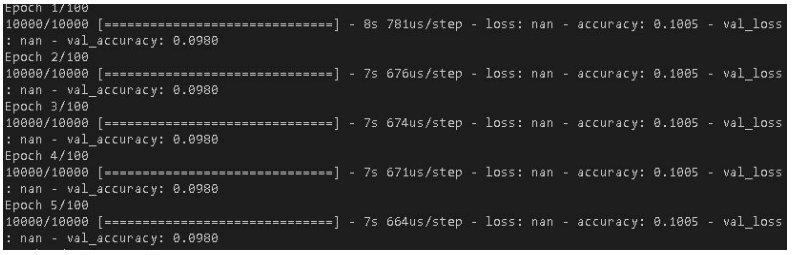
still NaN loss

b. **clipvalue**



still NaN

4. **BN**



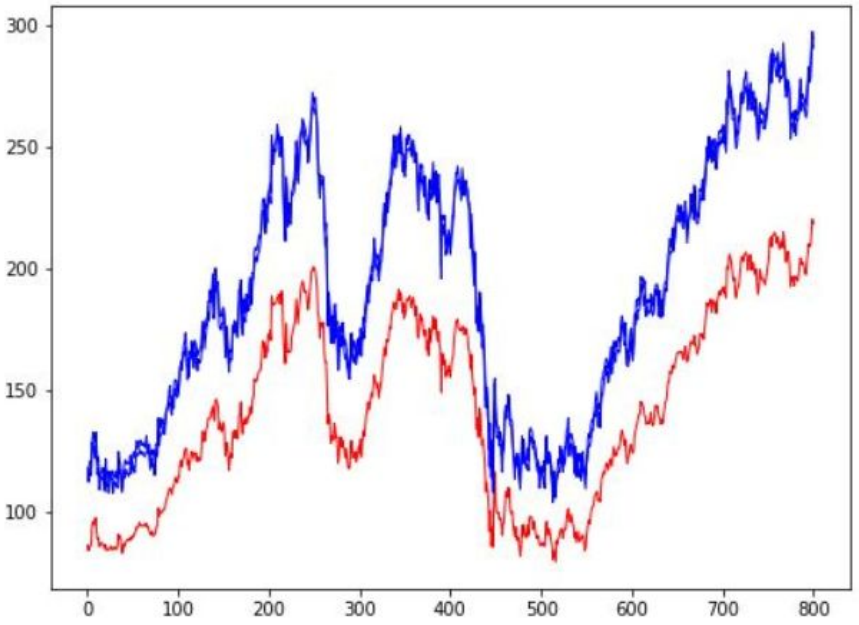
still NaN.

Unstable training:

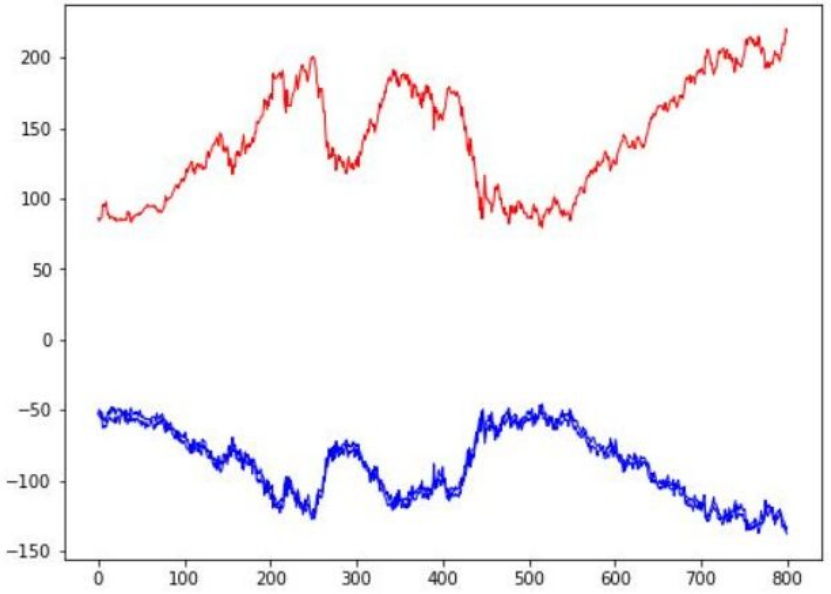
case7:

In fact, this problem of this case is not the unstable training, but the unstable predict by tensorflow, and the key to solve this problem is to run the `tf.global_variables_initializer` to initialize the variable before using in the model.

before initialize at first, there may be large gap between real and predict.



red is real,blue is predict

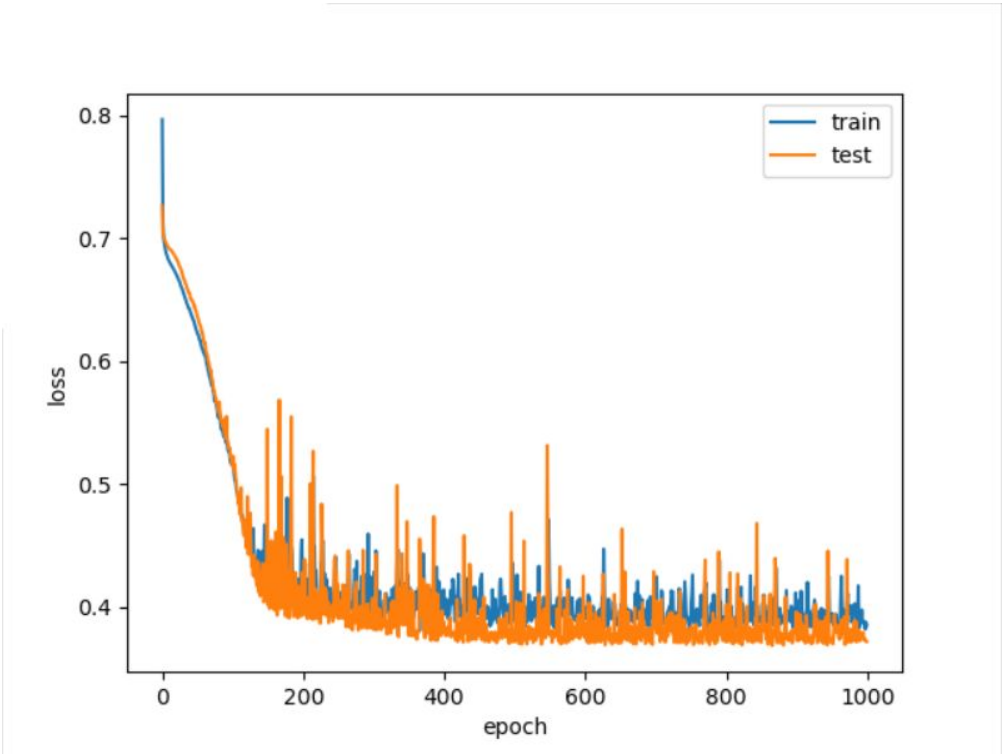


after using initializer, the large diff disapear.

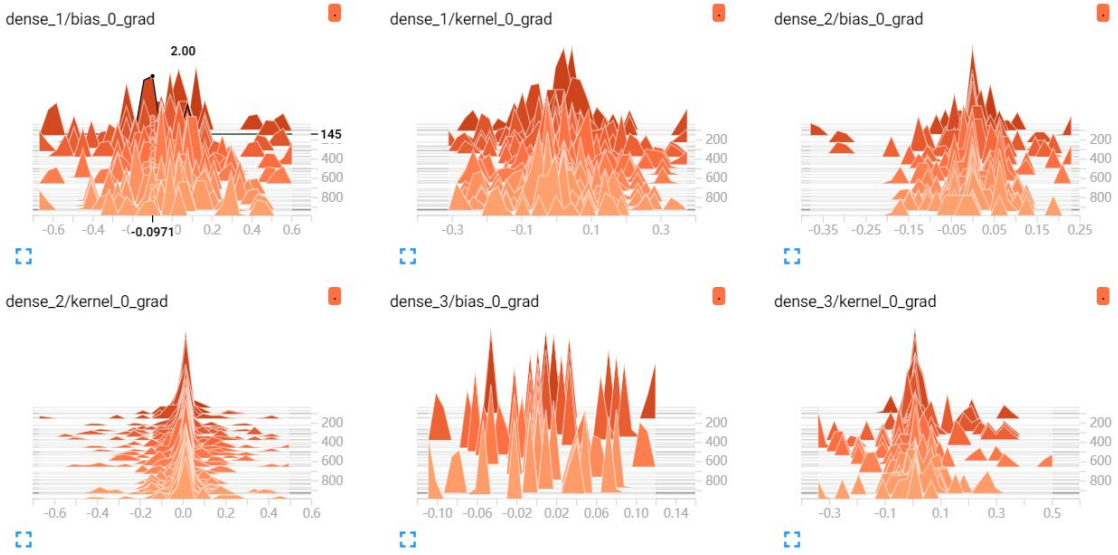
case 20:

based on case 16.

1. when loss comes to 0.35-0.4,epoch comes to 200, loss comes up and down.(opt: SGD(lr=0.01,momentum=0.9))

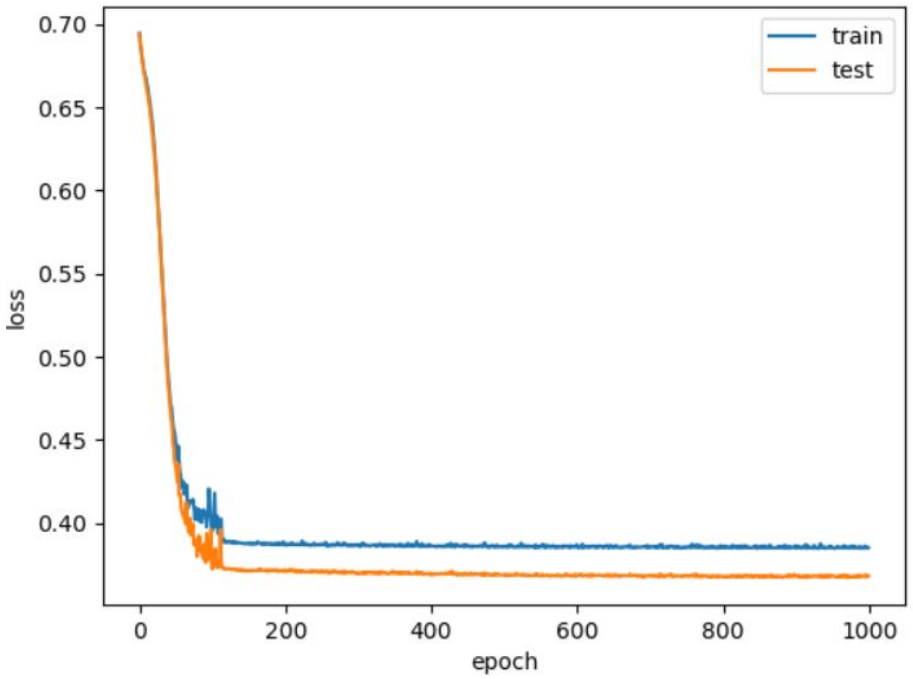


but it gradient didn't vainish in fact:

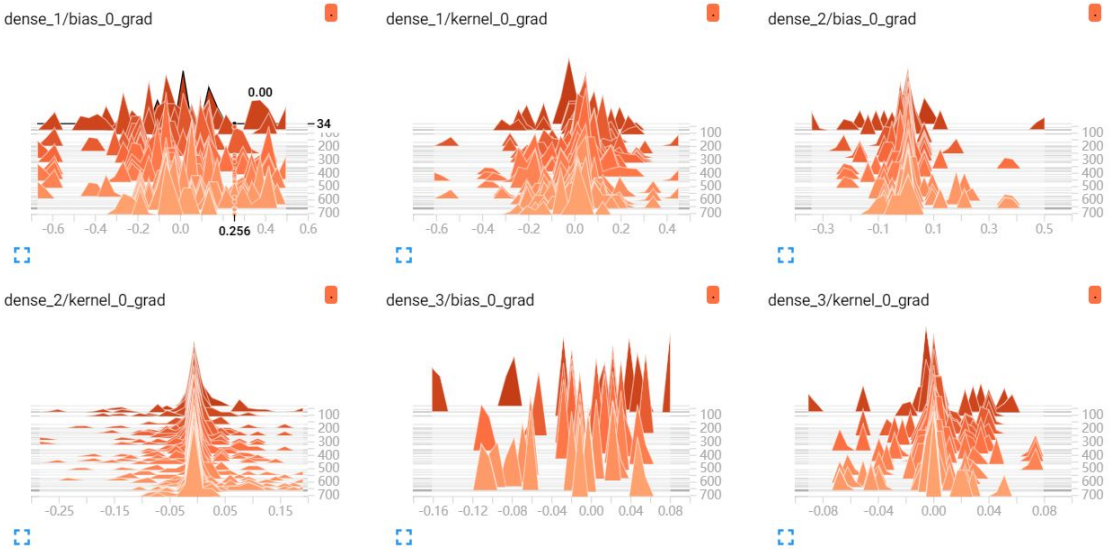


we can judge the cause may be unstable training due to **too large learning rate**

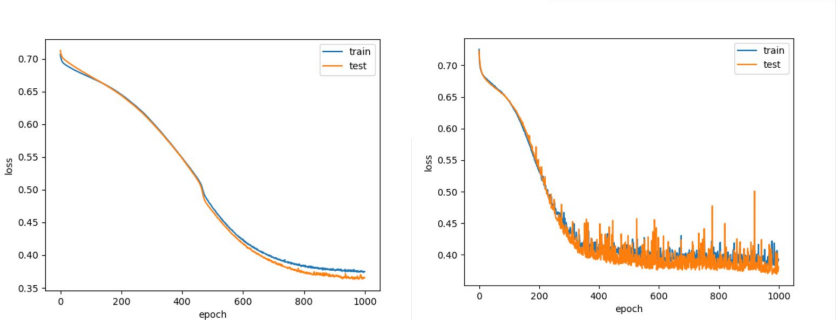
2. after using ReduceLROnPlateau(keras api, when loss didn't get better, the learning rate will be lower):



the loss performs well after epoch 200, and it comes down slowly.



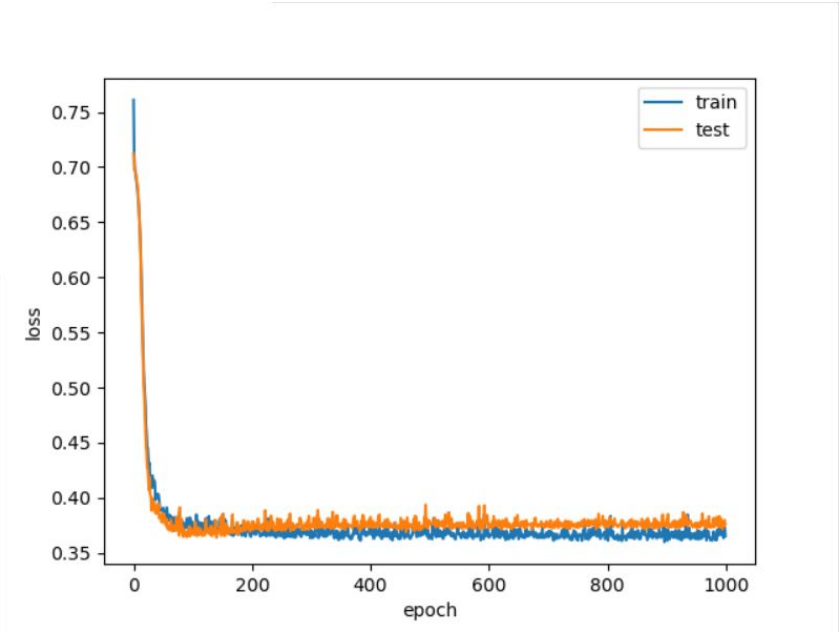
3. lr =0.01 and compare with SGD different momentum:



left: momentum=0; right: momentum=0.7

SGD with low momentum can alleviate the problem in som

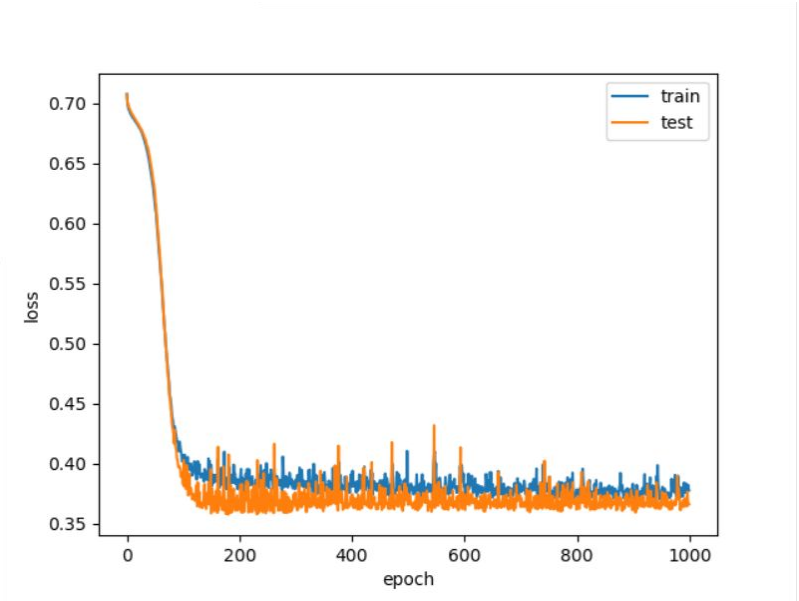
4. using Adam(lr=0.01):



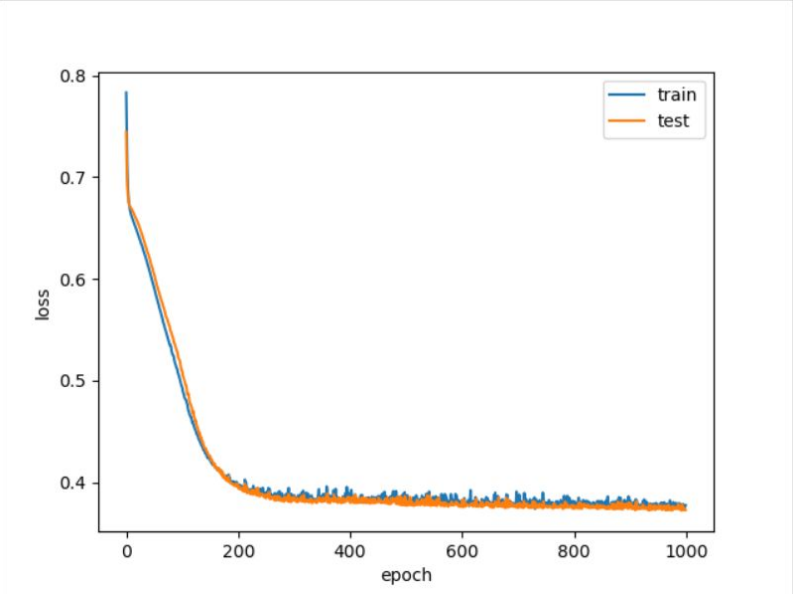
this case shows us that the Adam preforms better than SGD, which is different from solutions in the community.

using Adam(lr=0.001)--default:

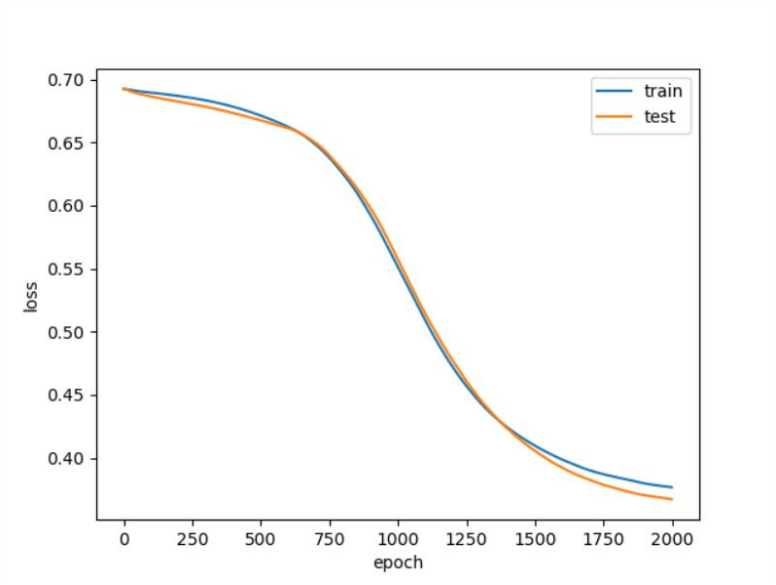
5. using bigger batch size:



batch size=32(default)↑



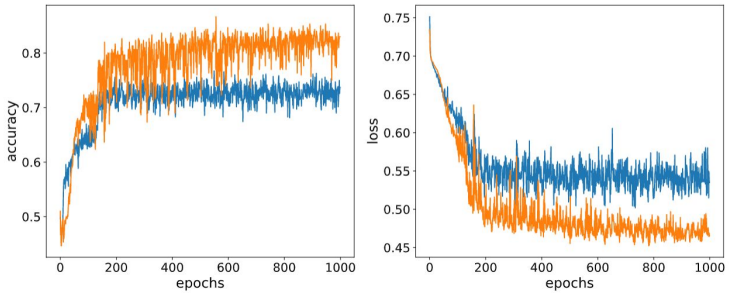
batch size=64, the zigzag loss has been improved.



batch_size=len(Xtrain), it converge slowly but smoothly.

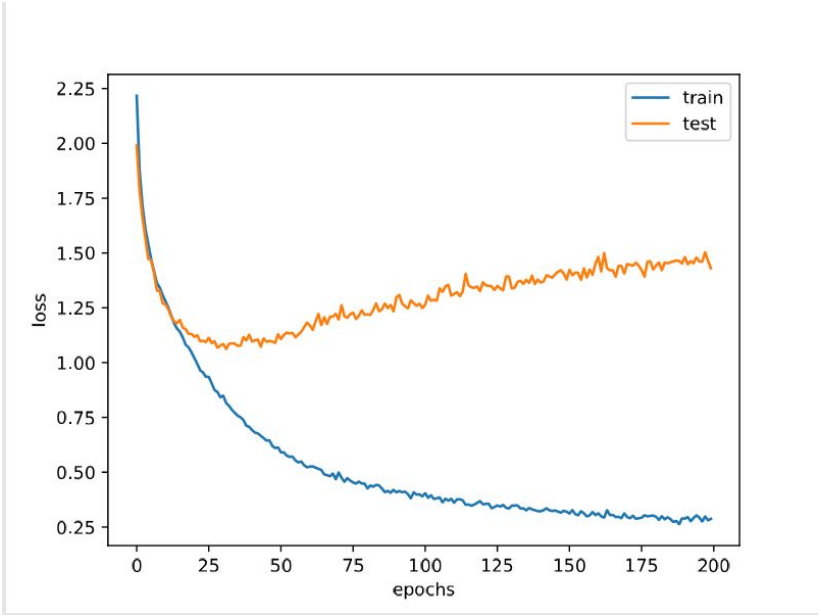
it seems that bigger batch size leads to slower and smoother converge.

6. GaussionNoise:



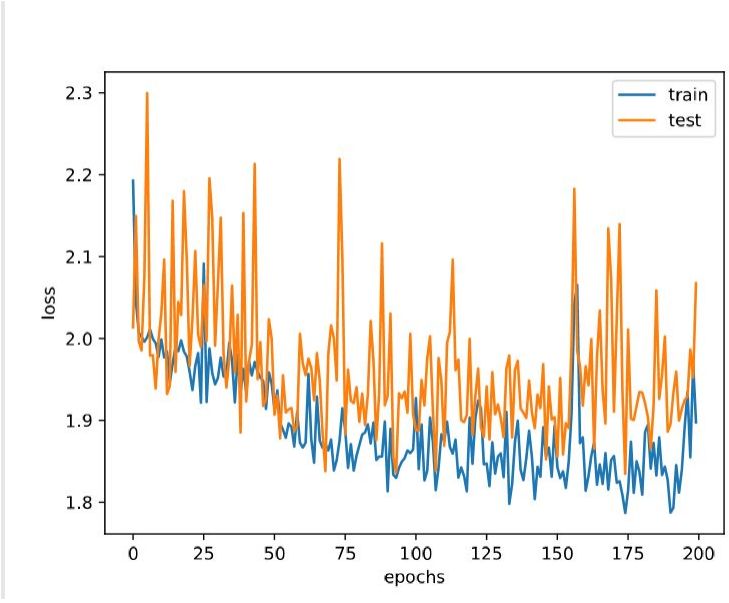
case20-cnn(cifar10,2 conv2d layer)

1. 2 conv2d layer with adam



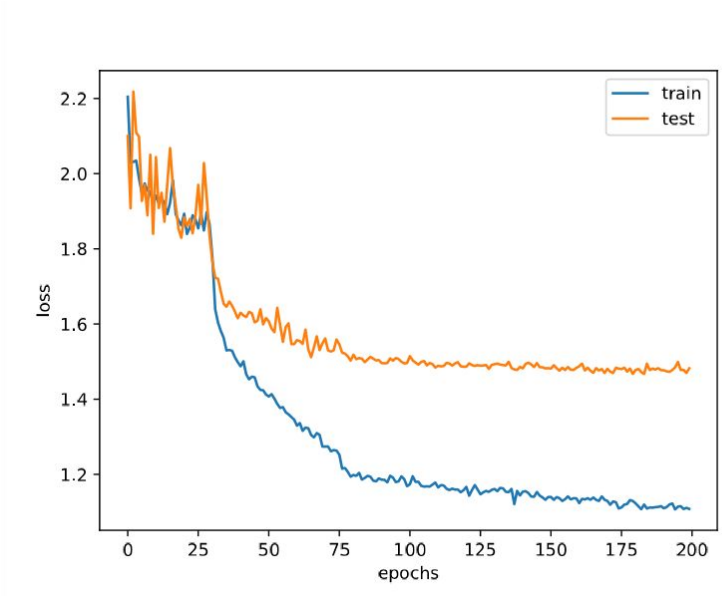
The convergence process can be optimized but the whole is relatively stable

2. 2 conv2d layer with sgd(0.1)



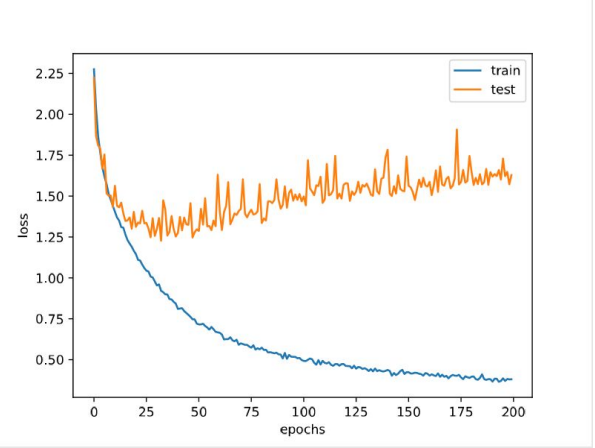
loss converge unstable, has large zig zag.

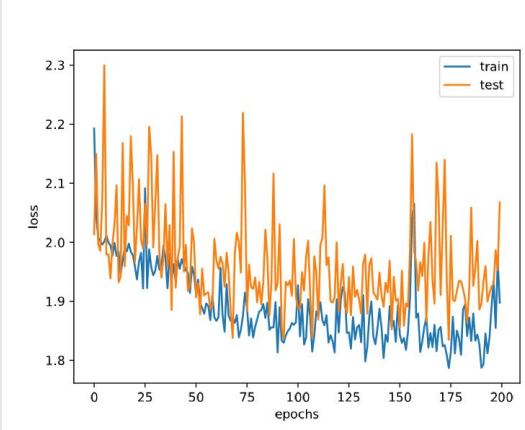
3. 2 conv2d layer with sgd(0.1),ReduceLROnPlateau



Adjusting the gradient can effectively reduce the unstable problem

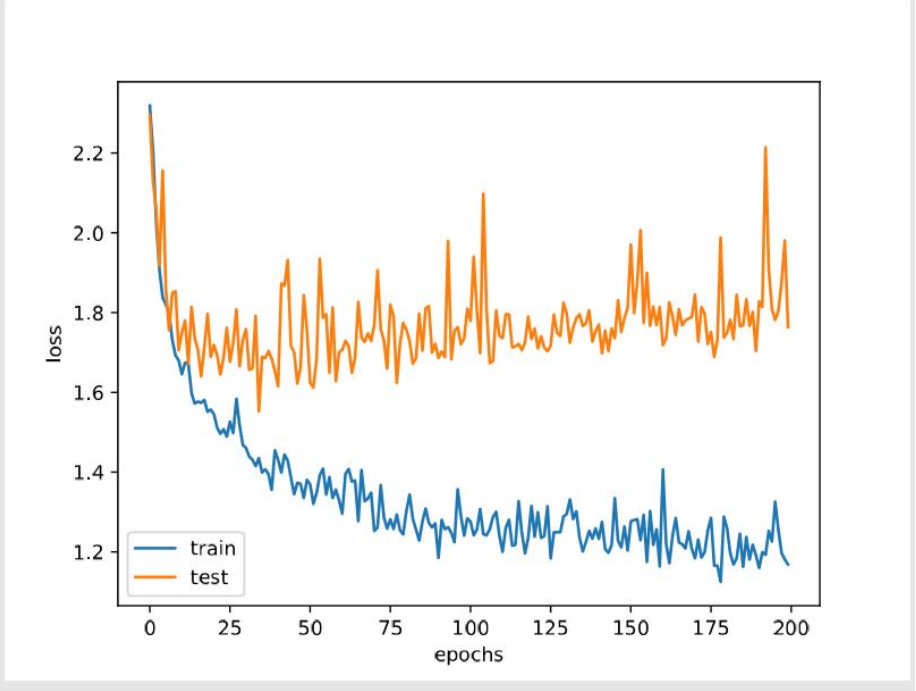
4. 2 conv2d layer with sgd(0.1),momentum=0.1(0.9 in previous.)



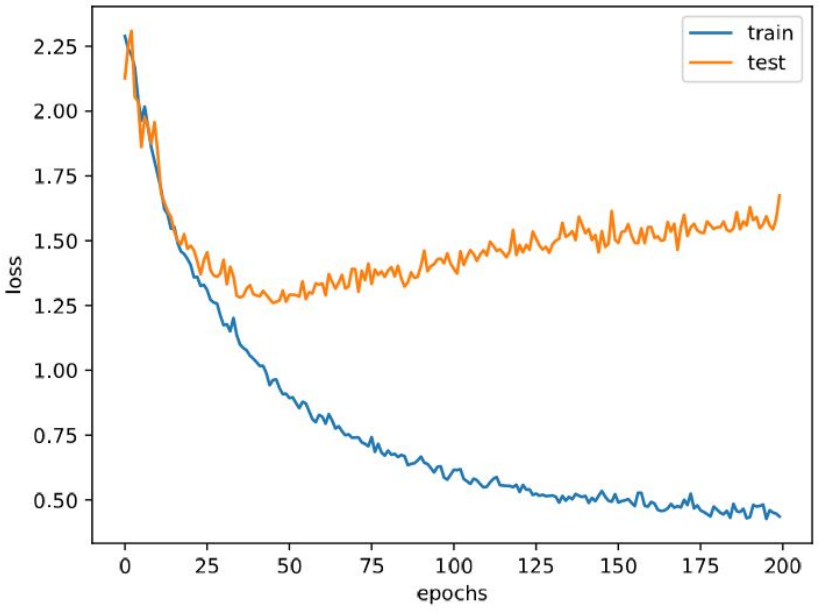


(momentum=0.1 VS momentum=0.9), the unstable issue can be improved well by decreasing the momentum.

5. 2 conv2d layer with sgd(0.1) batch=128(64 in previous)

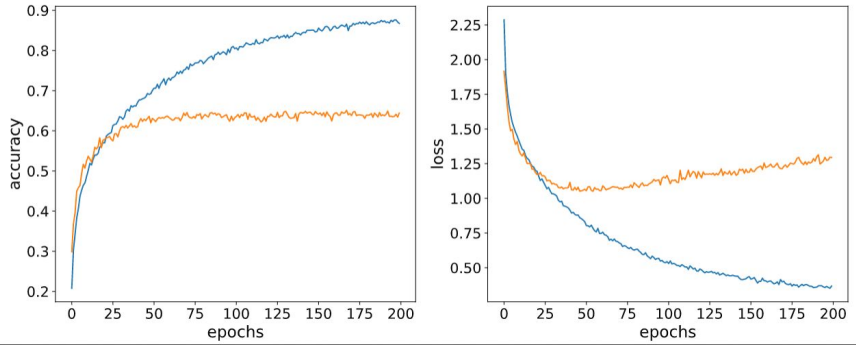


although the unstable issue has been improved, but it still exists,



when batch size=512, the unstable issue wil be nearly solved.

6. GausssionNoise



no effect

not converge:

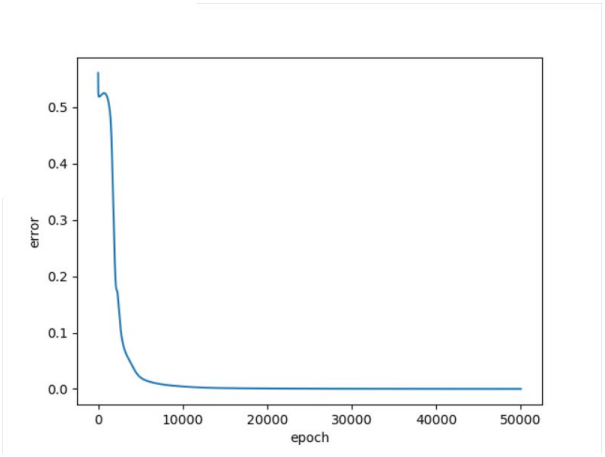
case9:

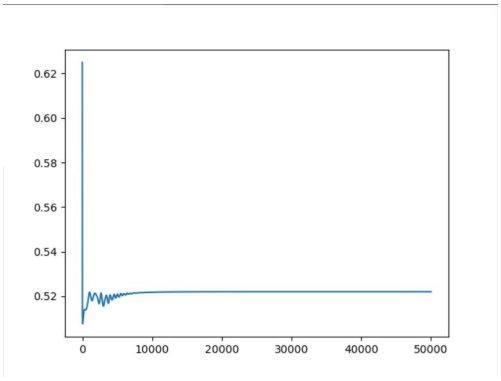
Case 9 use “accuracy” to evaluate the result of a regression problem, so accuracy always return 0, which made it look like a not converge problem.

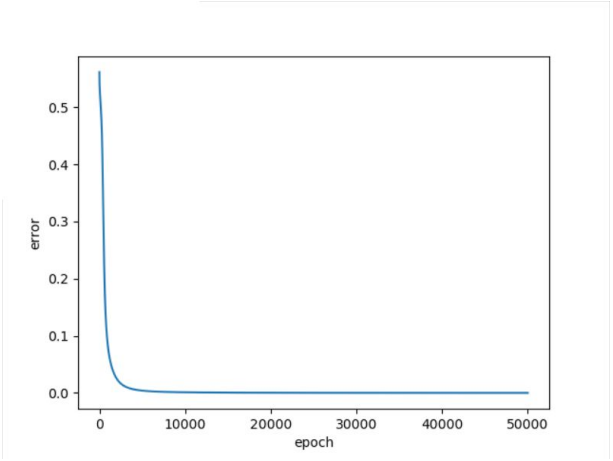
```
  32/17000 [..............................] - ETA: 1s - loss: 12361126912.0000 - accuracy: 0.0000e+00
1344/17000 [=>.............................] - ETA: 0s - loss: 8608725857.5238 - accuracy: 0.0000e+00
2784/17000 [===>...........................] - ETA: 0s - loss: 8637277645.9770 - accuracy: 0.0000e+00
4160/17000 [======>........................] - ETA: 0s - loss: 8411895246.7692 - accuracy: 0.0000e+00
5568/17000 [========>......................] - ETA: 0s - loss: 8356102517.7011 - accuracy: 0.0000e+00
Process finished with exit code -1
```

Changing accuracy to mse and others can help us observe the converge of model in iteration.

case13:





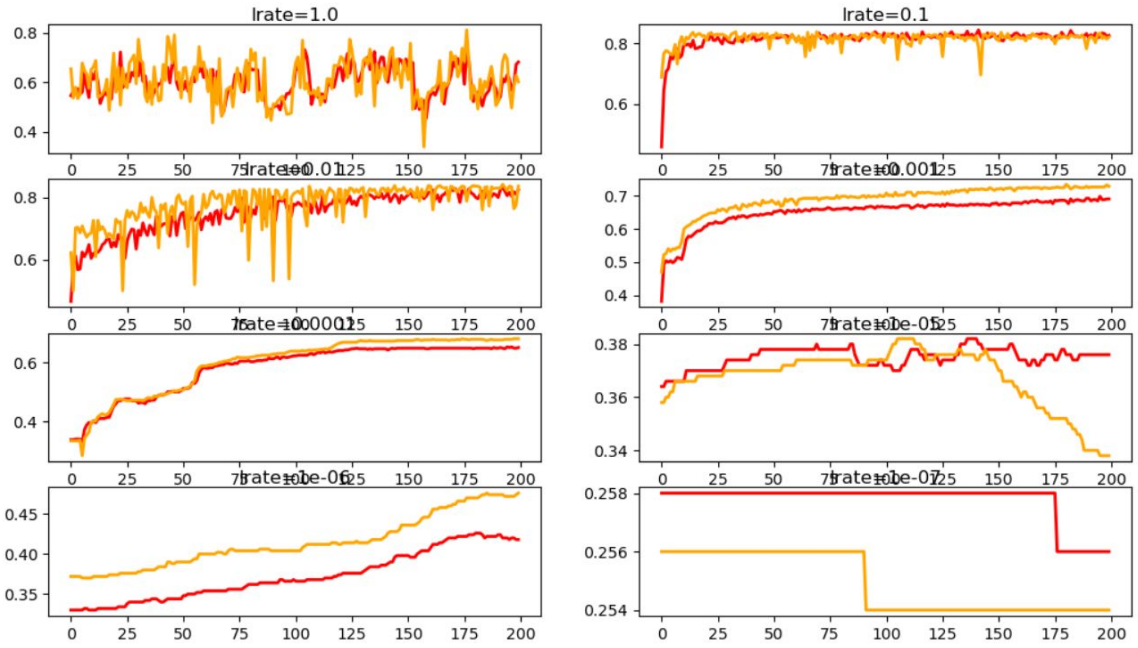


get multi result from multi running ---- **not converge issue or gradient vanish**?

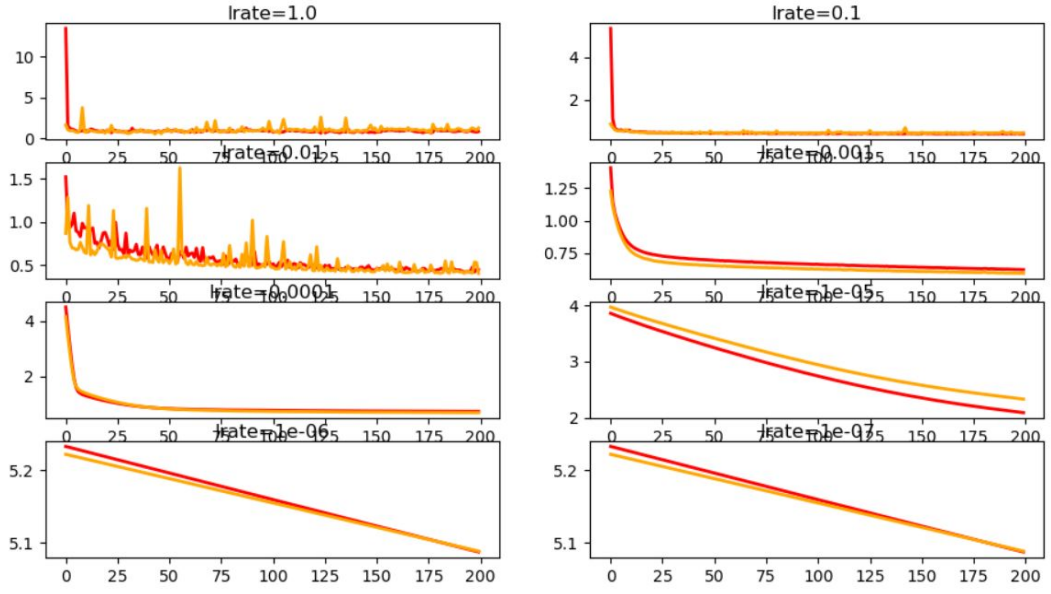
case24:

1.chage

learning rate for SGD optimizer:

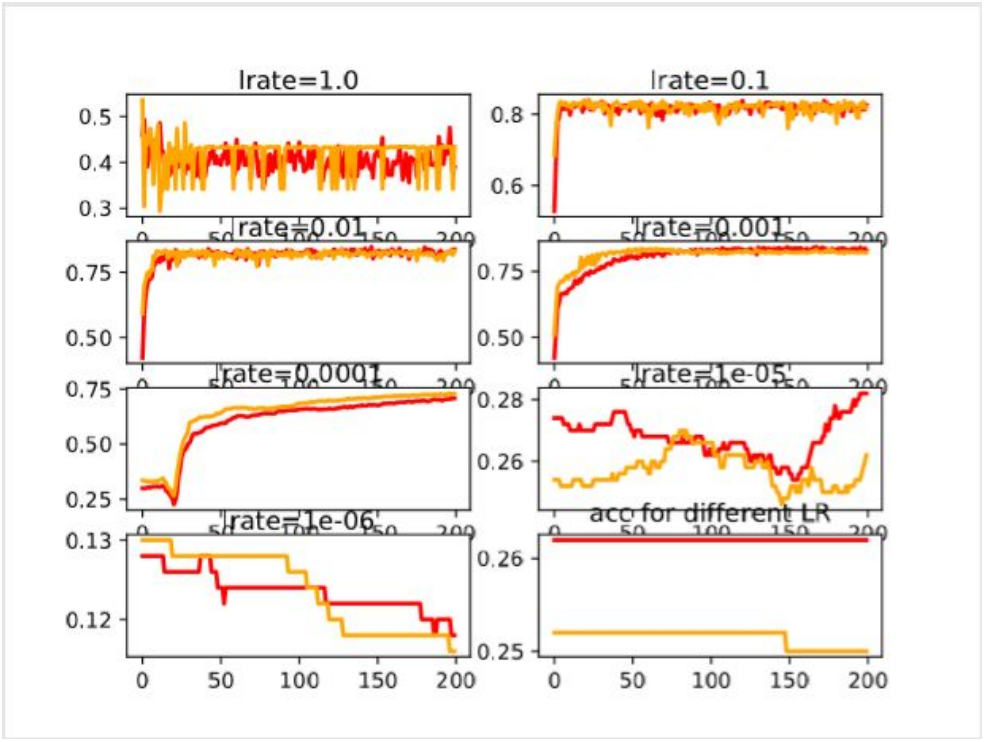


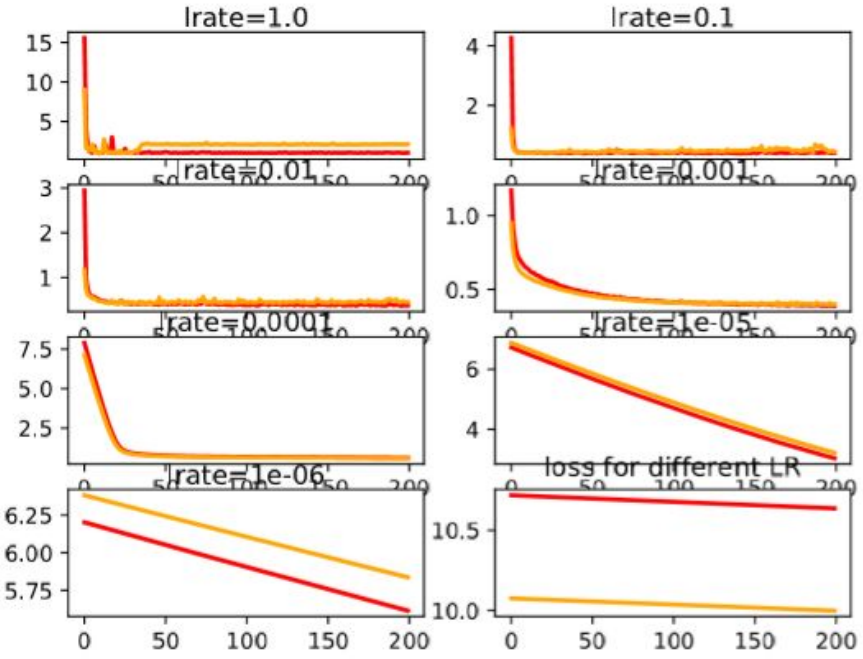
yellow means val_acc(test),red means acc(train)
also, we can see the loss line. Similarly, the yellow means val_loss(test), red means loss(train)



from the pictures above, we can find that, when the **lr is too low, the loss will be easily stuck in the local minimum**, which will bring trouble to the convergence.

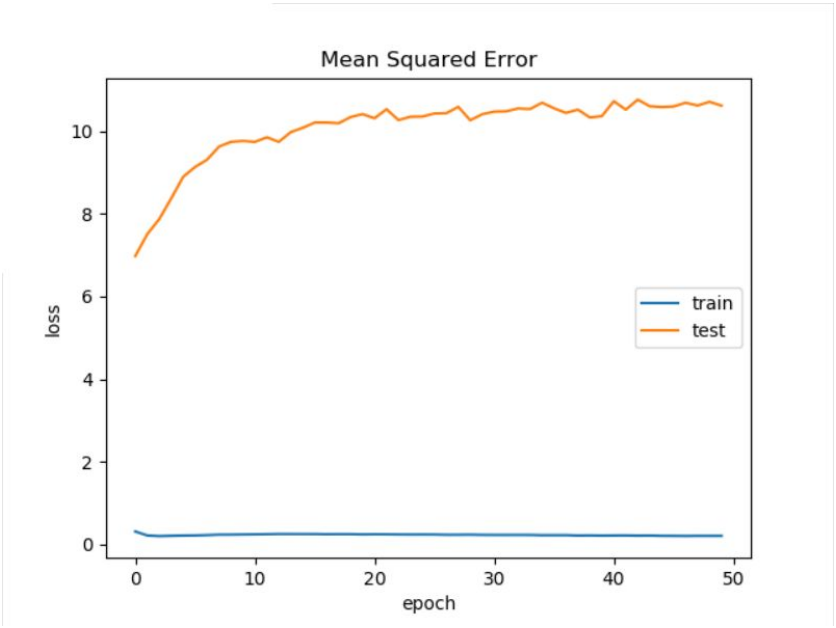
2.adam:

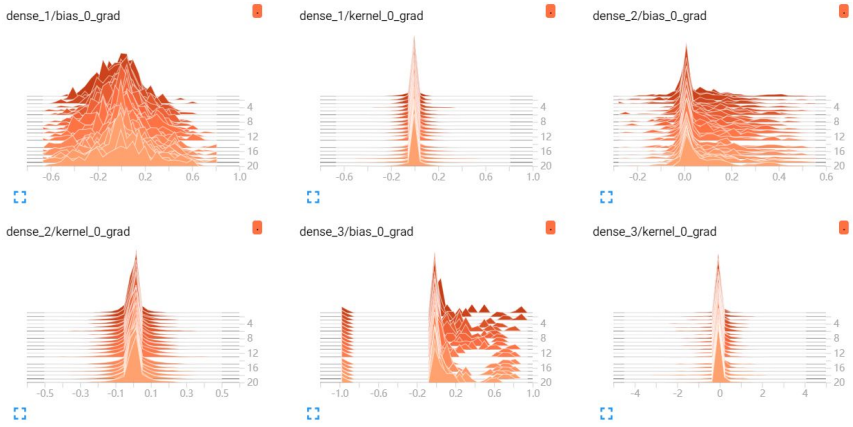




case25:(based on case23):
By shuffling your data, you ensure that each data point creates an "independent" change on the model, without being biased by the same points before them.

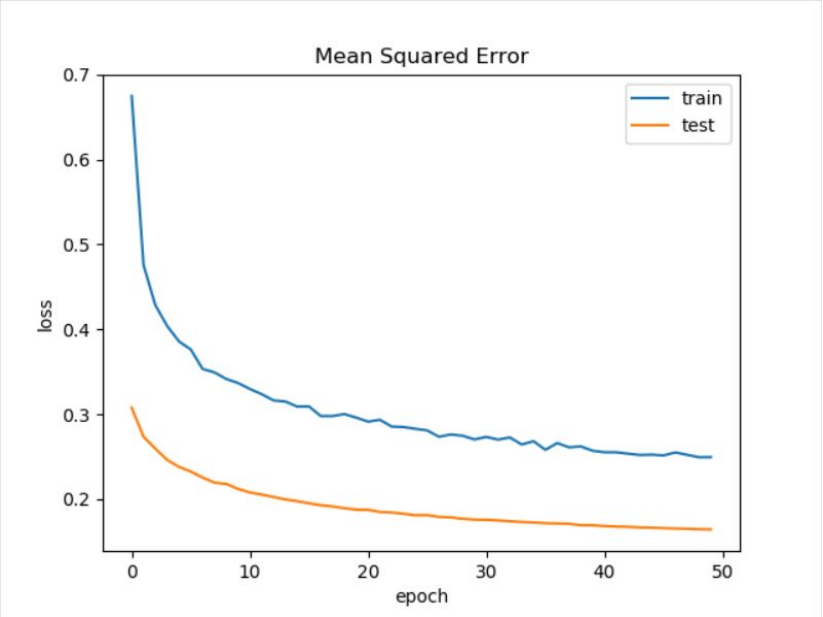
1) without any shuffle(the training set is sorted by label):

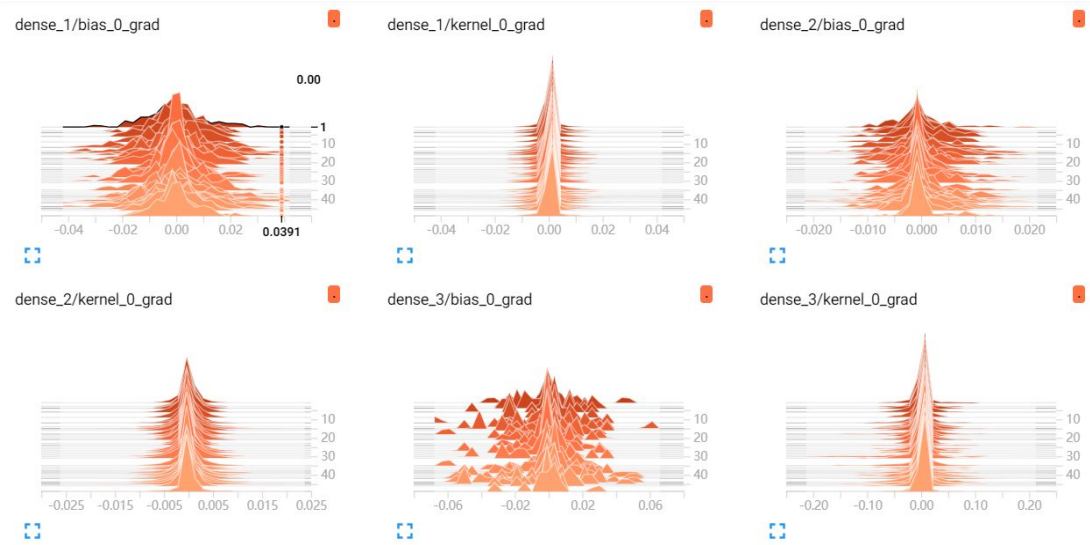




test loss didn’t converge, and the gradient changed little between different epochs.

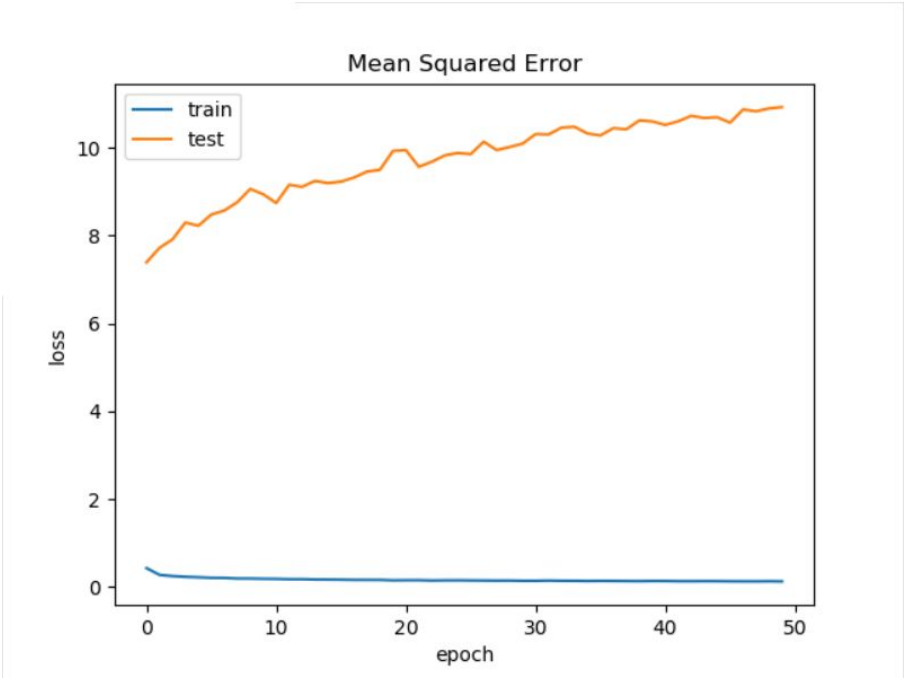
2)using shuffle before training:





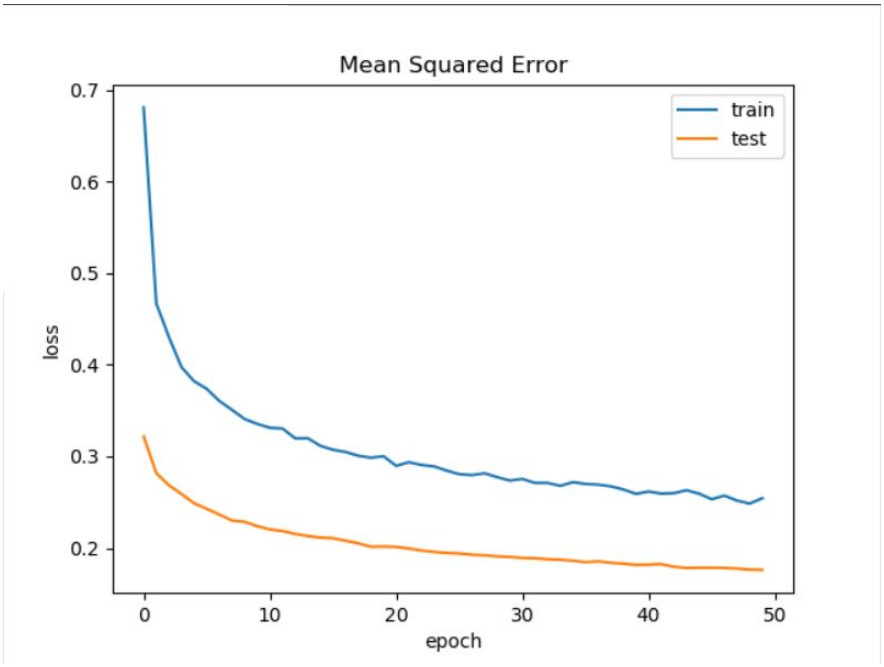
the loss curve of training and testing have converged well, gradient also changed little.

3)only shuffle in everty epoch(model.fit(shuffle=true)):



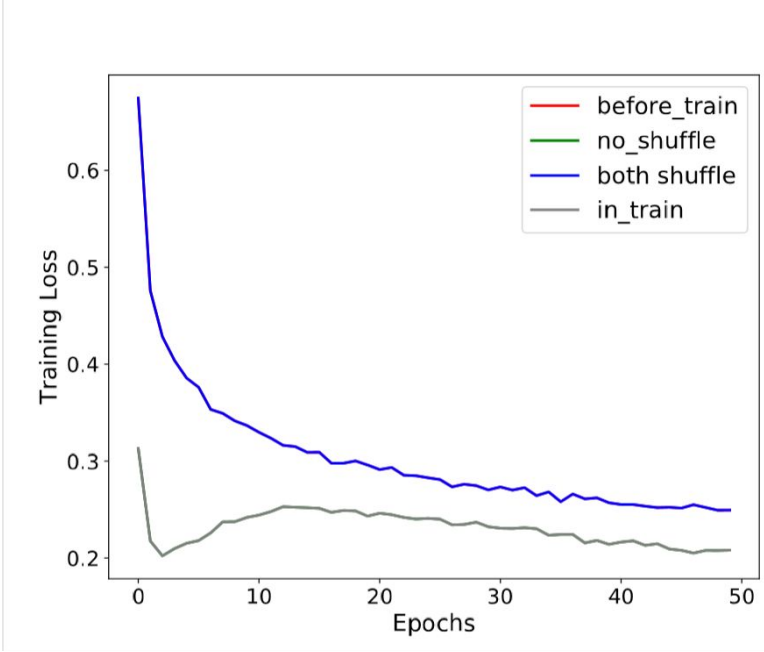
only shuffle in training epoch can't solve the problem. *Using shuffle=True carefully.*

4)shuffle before training & in every training epoch:

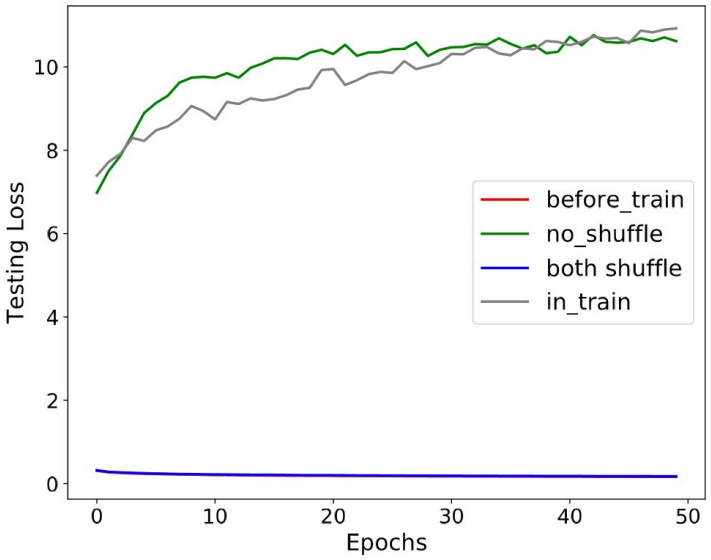


also can converge well.

we can see the comparison chart between different settings.



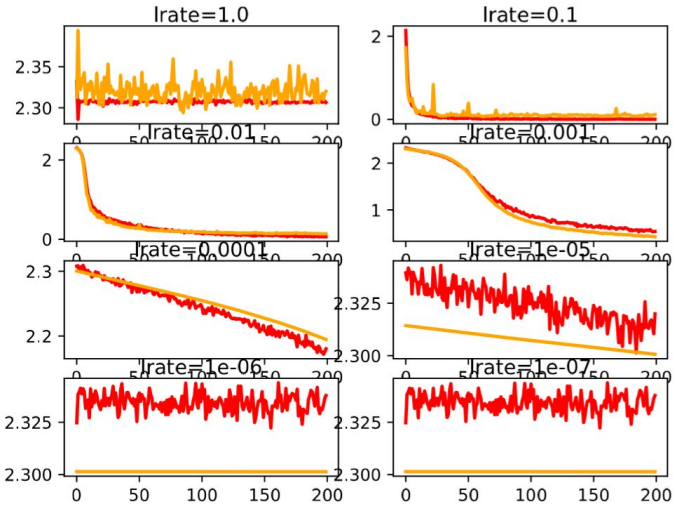
In this picture, Red and blue are similiar and green and gray are similar.



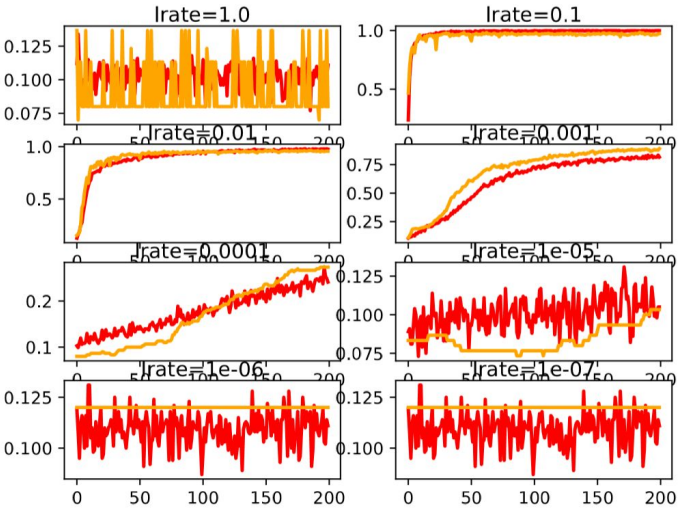
In this one, the blue line is similar to the red line.

Though the above comparison, we can see that, a regularly sorted data set will bring good train loss but horrible test loss . Also, the shuffle in training may not solve this problem. And the shuffle before training can slove this kind of problem well.
This kind of not converge can be detected by : good training loss come with no converge test loss.

case24-cnn:(Mnist)



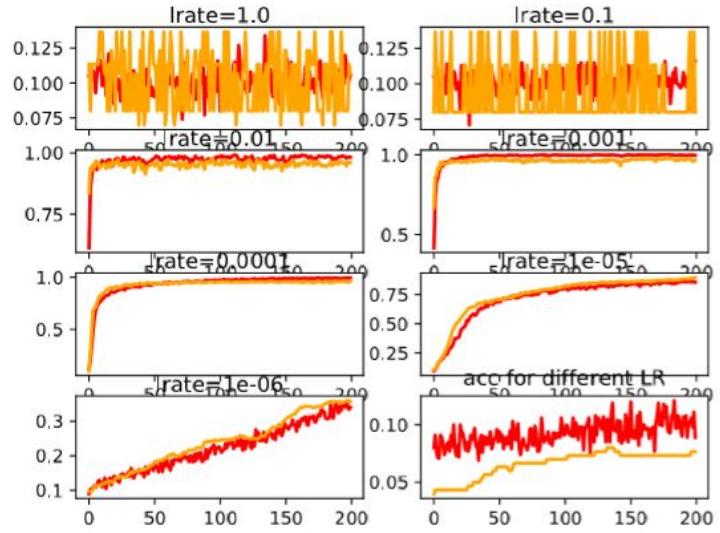
Loss

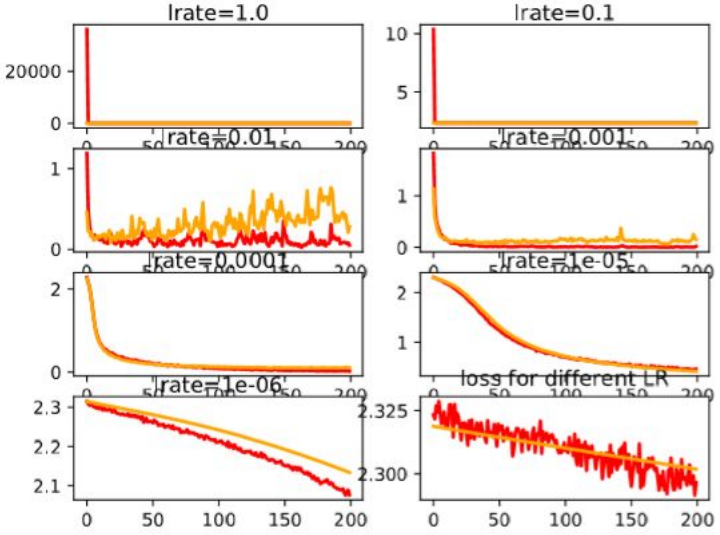


Accuracy

Similar conclusion with case24, when the **lr is too low, the loss will be easily stuck in the local minimum**, which will bring trouble to the convergence.

adam:

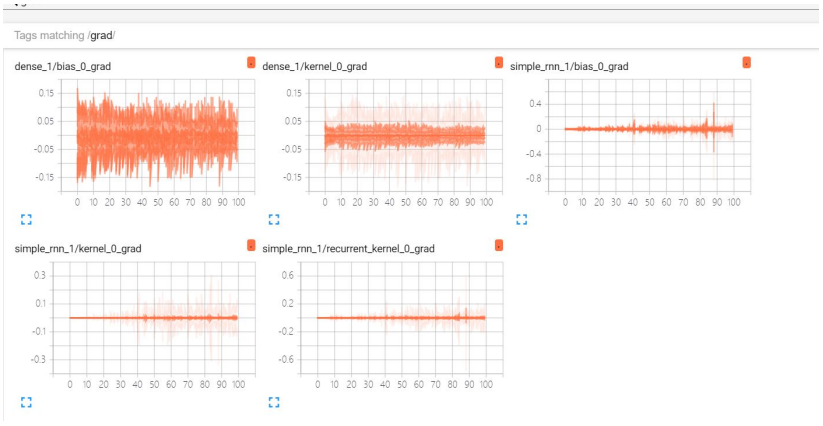

lrate=1.0
lrate=0.1
lrate=0.01
lrate=0.001
lrate=0.0001
lrate=1e-05
lrate=1e-06
acc for different LR

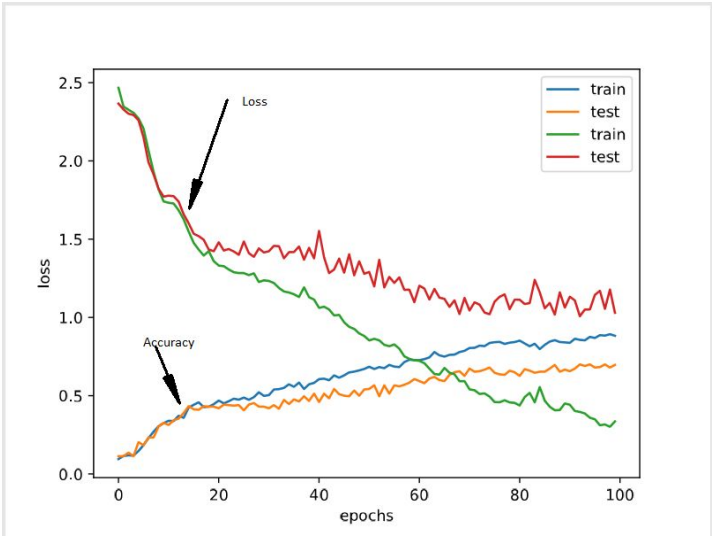

lrate=1.0
lrate=0.1
lrate=0.01
lrate=0.001
lrate=0.0001
lrate=1e-05
lrate=1e-06
loss for different LR

RNN Structure:

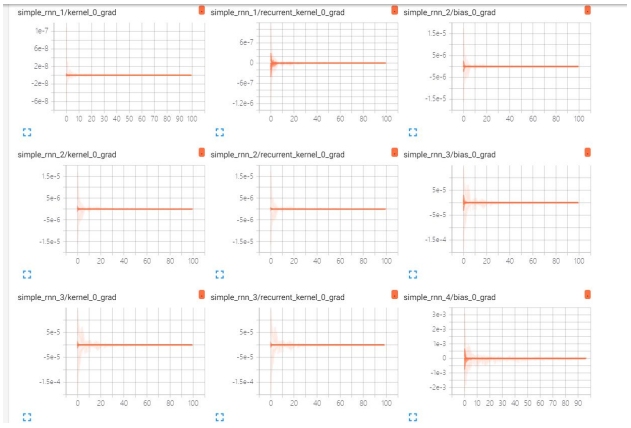
1. case16-RNN(Gradient Vanish)
   [Reference]

   a. 1rnn-mnist

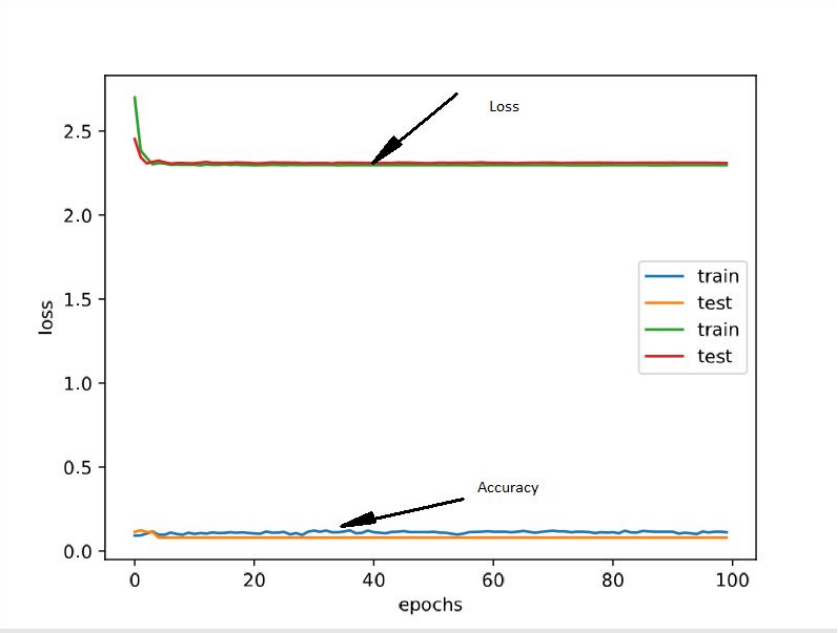




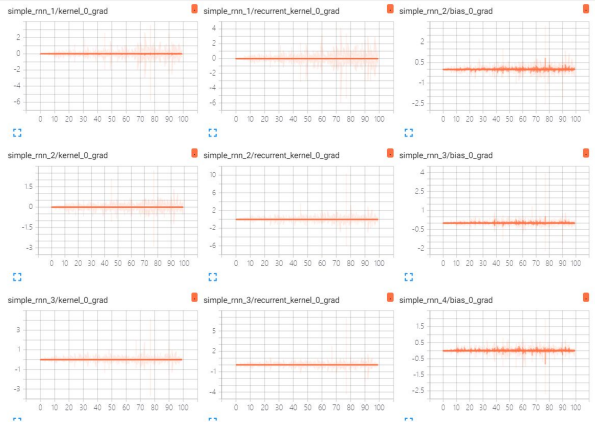
   b. 5rnn-mnist



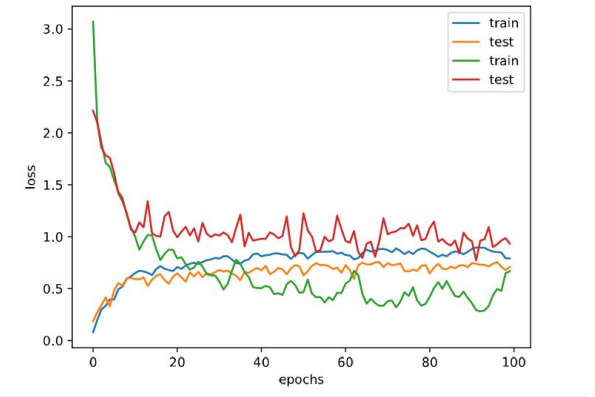
Gradient decrease when spread forward.



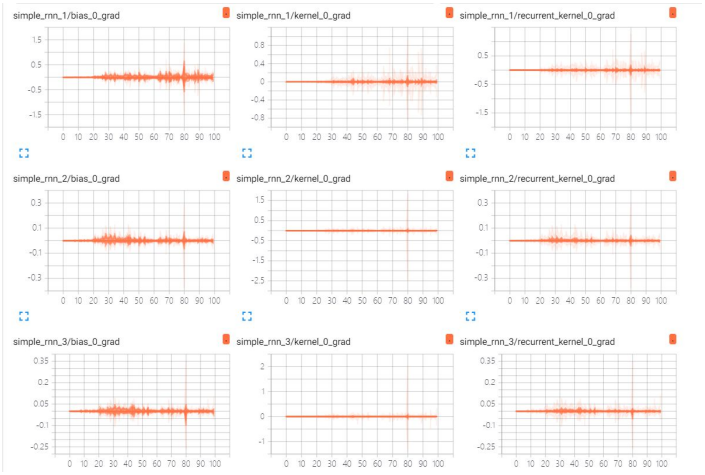
c. 5rnn-mnist-relu



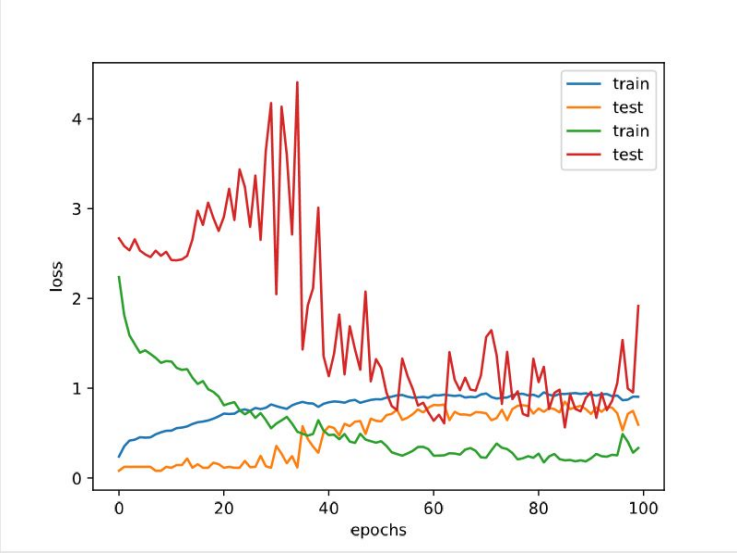
gradient has been improved obviously.



d. 5rnn-mnist-BatchNorm



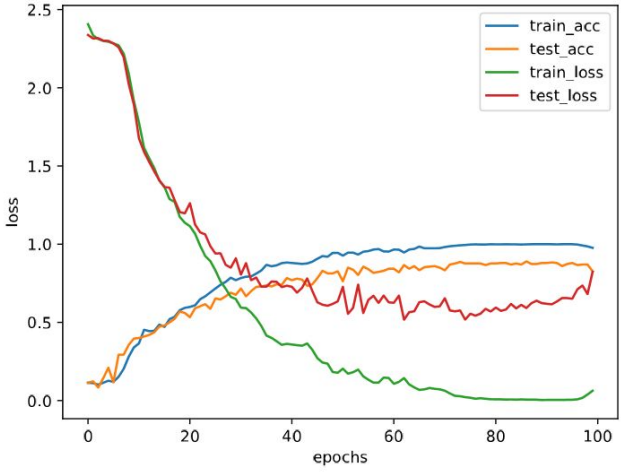
can be improved by BatchNormalization

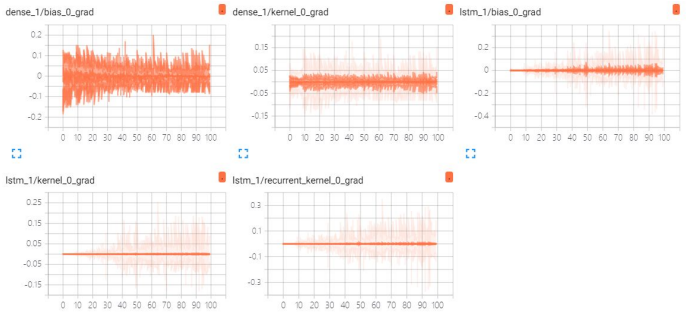


e. Layerwise Pretraining

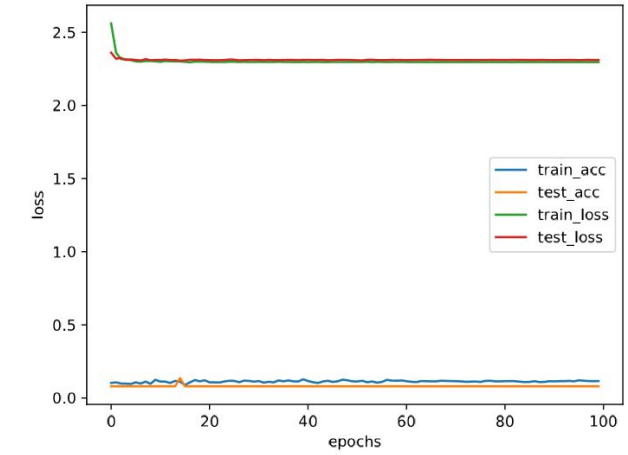
f. Multi: Lstm-mnist

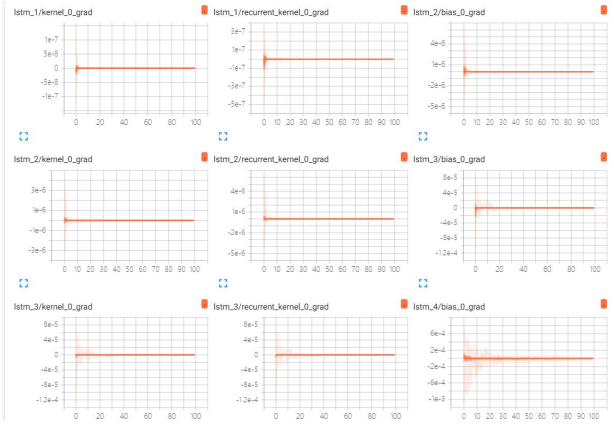
1LSTM:



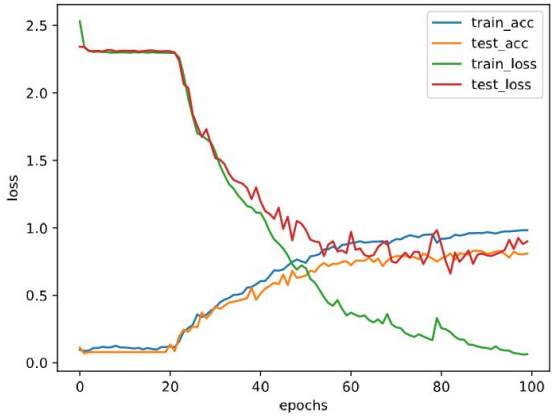

dense_1/bias_0_grad
dense_1/kernel_0_grad
lstm_1/bias_0_grad
lstm_1/kernel_0_grad
lstm_1/recurrent_kernel_0_grad

5LSTM:

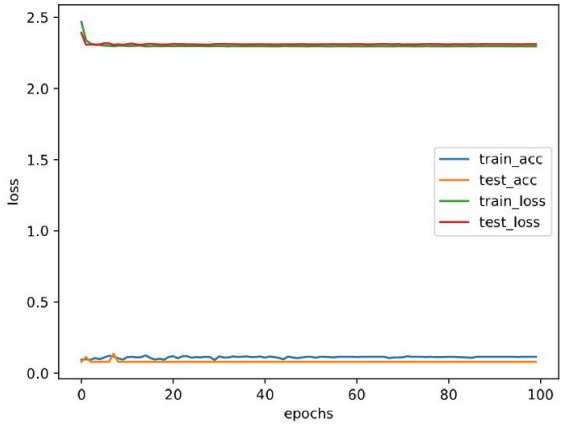

loss
epochs
train_acc
test_acc
train_loss
test_loss


lstm_1/kernel_0_grad
lstm_1/recurrent_kernel_0_grad
lstm_2/bias_0_grad
lstm_2/kernel_0_grad
lstm_2/recurrent_kernel_0_grad
lstm_3/bias_0_grad
lstm_3/kernel_0_grad
lstm_3/recurrent_kernel_0_grad
lstm_4/bias_0_grad

3LSTM:

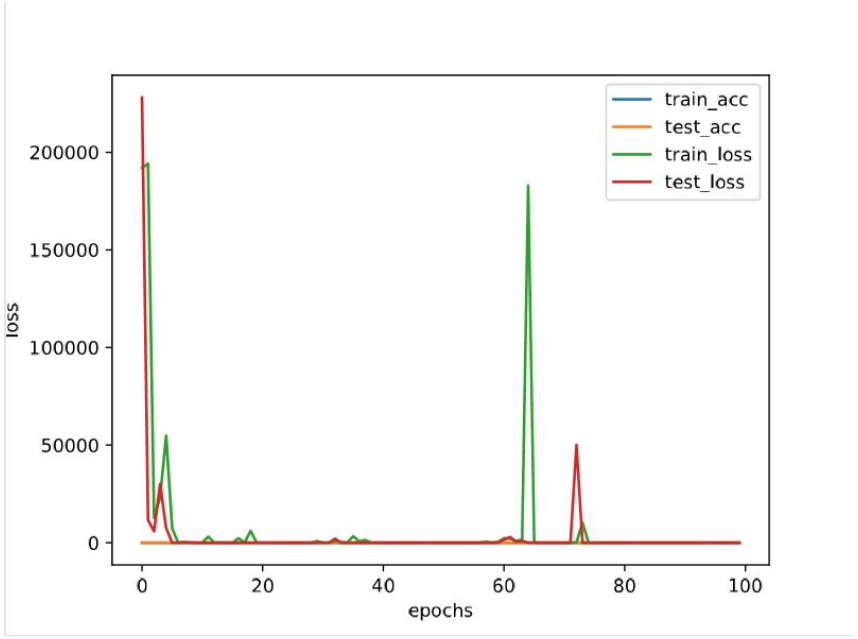


4LSTM:



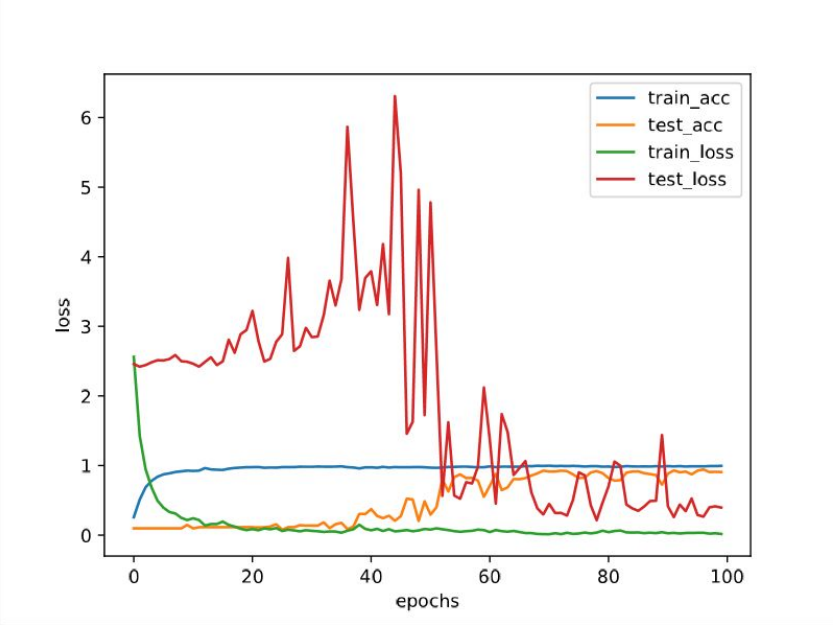
As the number of layers increases, the LSTM does produce a gradient disappearance

g. Relu for 5LSTM



using Relu has no improvement for this situation

h. BatchNorm or 5LSTM

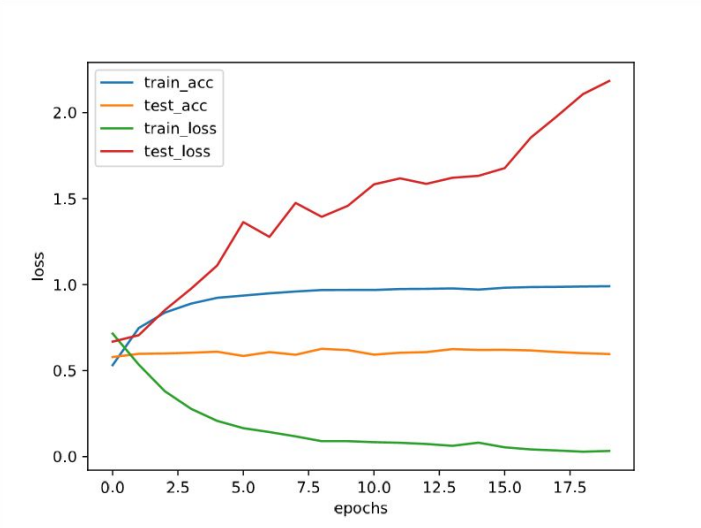


Batchnorm can improve the LSTM gradient issue.

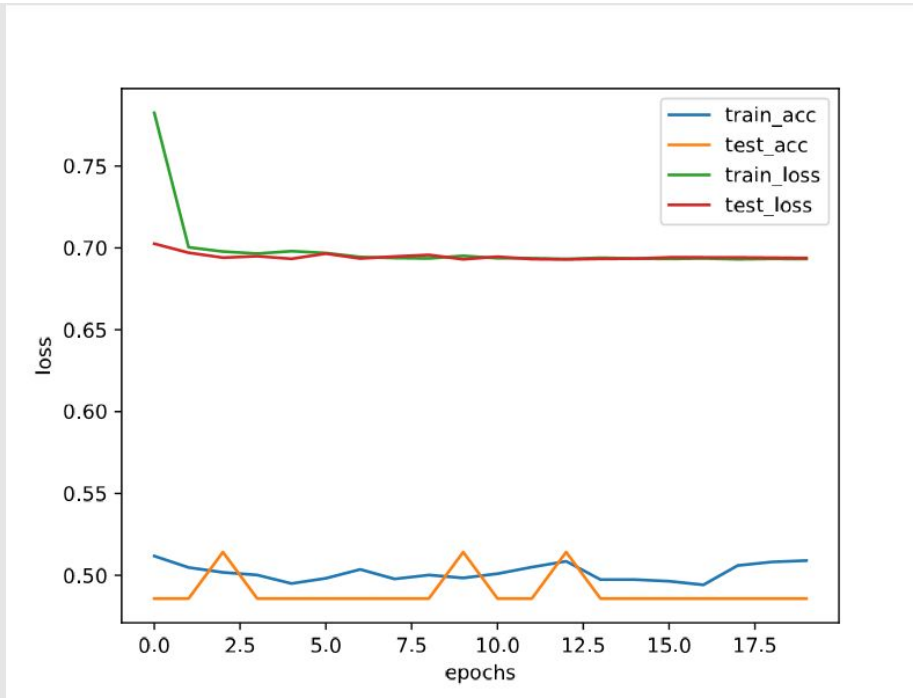
i. Pretrain for LSTM

2. case16-RNN(Gradient Vanish) IMDB

a. 1rnn sigmoid



b. 8rnn sigmoid



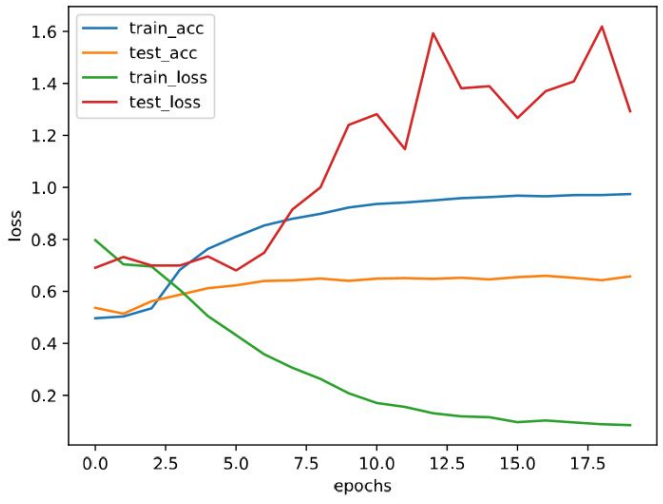
c. 8rnn relu

Gradient explode!!!

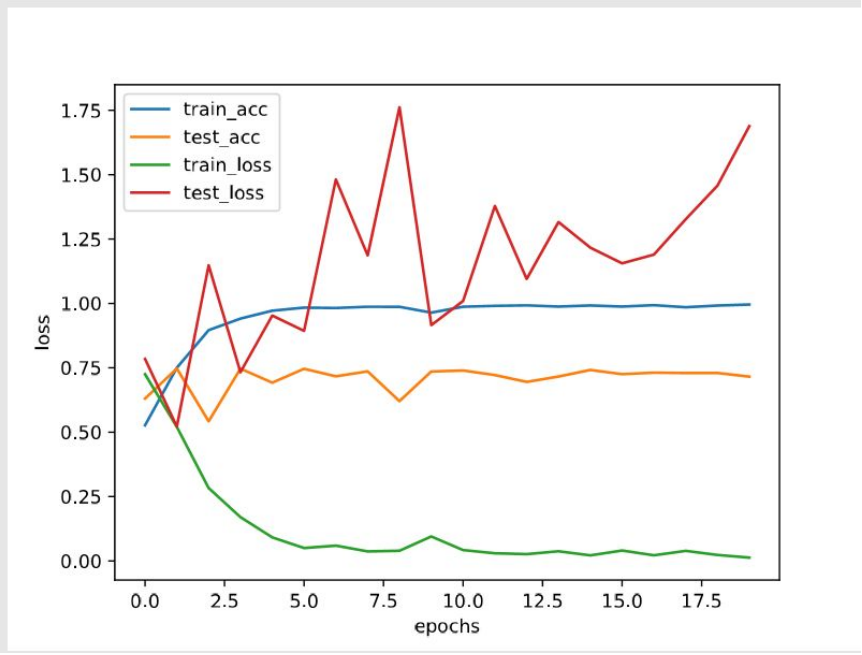
```
 384/5000 [=>............................] - ETA: 13s - loss: 6.2588 -
 512/5000 [==>...........................] - ETA: 13s - loss: 6.9363 -
 640/5000 [==>...........................] - ETA: 12s - loss: 6.9202 -
 768/5000 [===>..........................] - ETA: 12s - loss: 6.9508 -
 896/5000 [====>.........................] - ETA: 12s - loss: 6.8832 -
1024/5000 [=====>........................] - ETA: 11s - loss: 6.9574 -
1152/5000 [=====>........................] - ETA: 11s - loss: 7.0985 -
1280/5000 [======>.......................] - ETA: 11s - loss: 7.1495 -
1408/5000 [=======>......................] - ETA: 10s - loss: 7.1681 -
1536/5000 [========>.....................] - ETA: 10s - loss: 7.2354 -
1664/5000 [========>.....................] - ETA: 9s - loss: nan - acc
1792/5000 [=========>....................] - ETA: 9s - loss: nan - acc
1920/5000 [==========>...................] - ETA: 8s - loss: nan - acc
2048/5000 [===========>..................] - ETA: 8s - loss: nan - acc
```

https://segmentfault.com/a/1190000019687826

d. 8rnn batchnorm

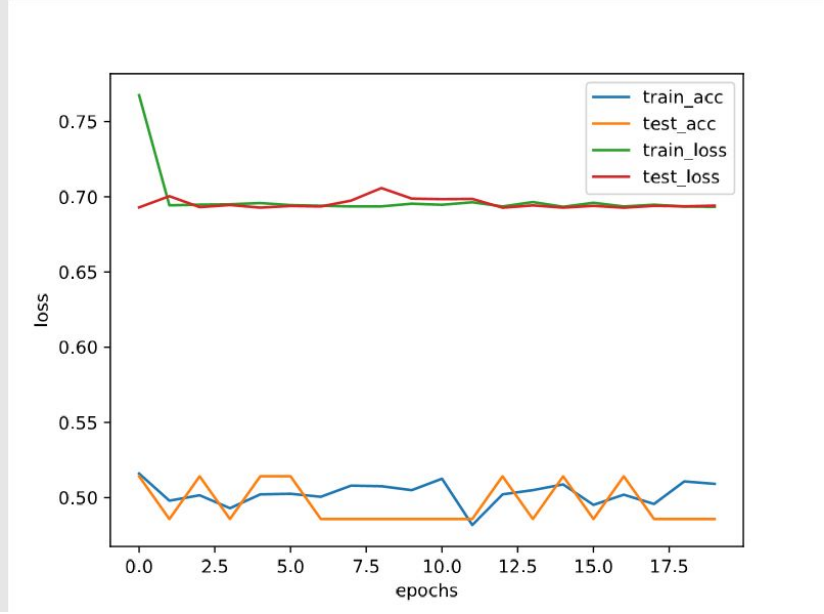


e. 8LSTM sigmoid with batchnorm

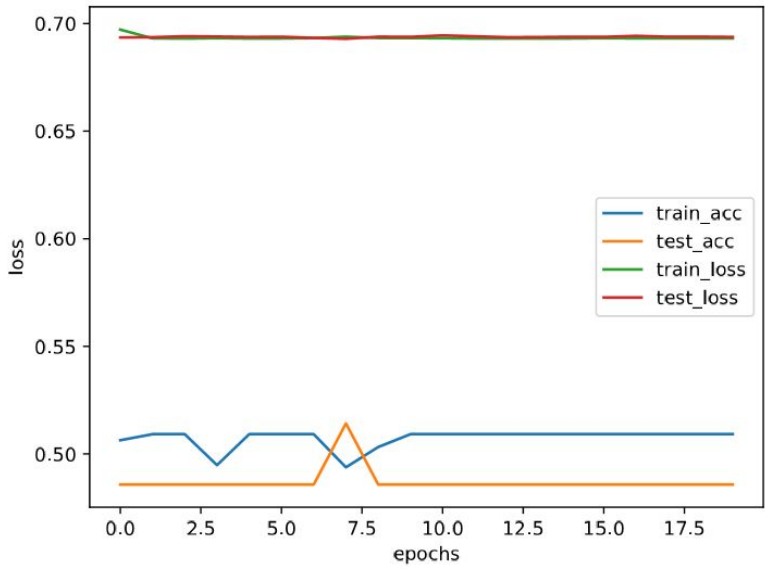


still overfit. but not vanish.

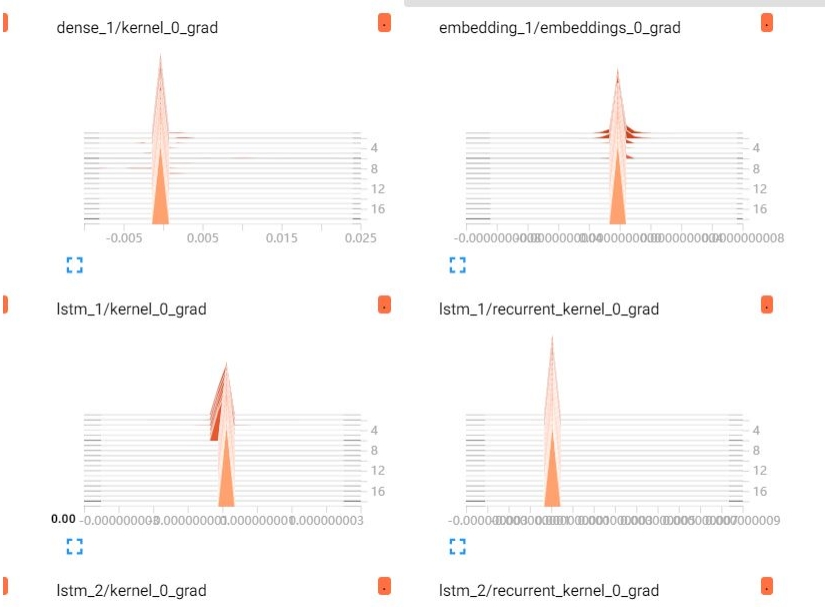
f. 8LSTM sigmoid



g. 8LSTM relu

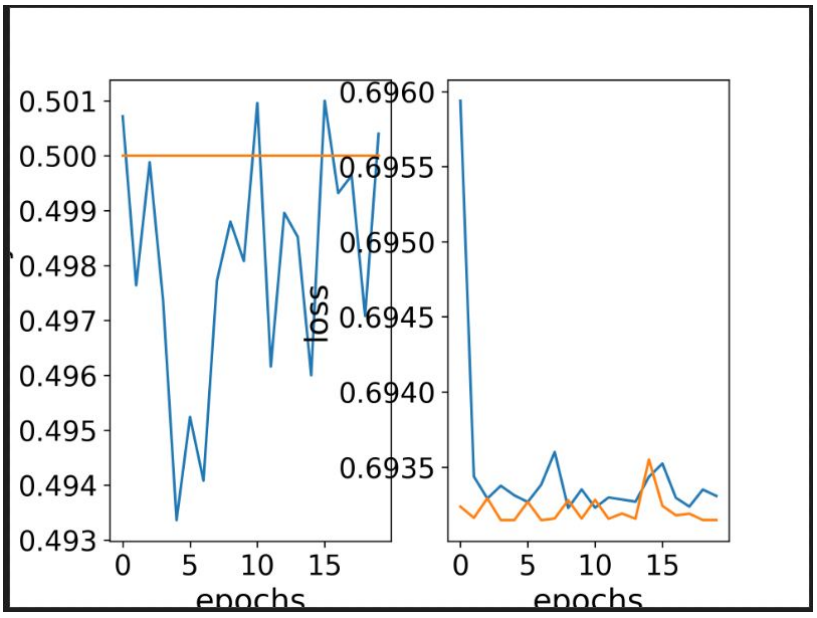


still not converge(Maybe dyingRelu?)



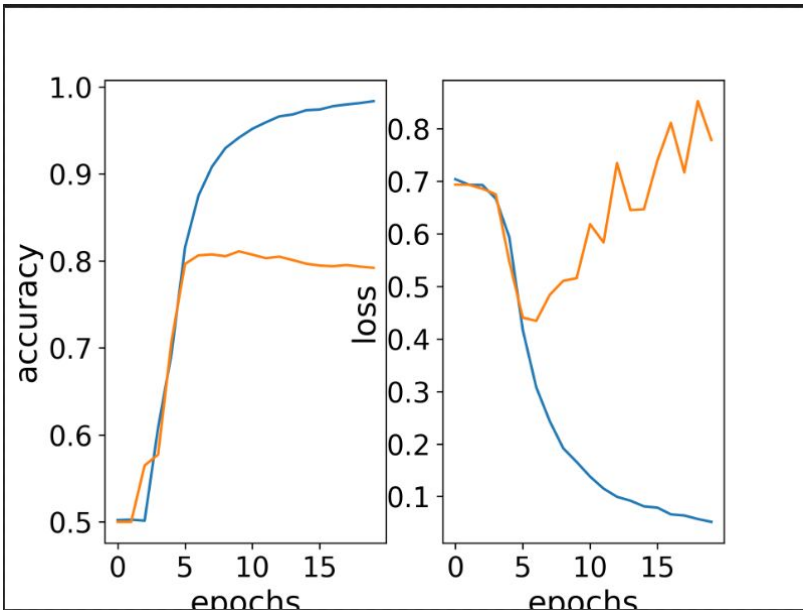
case26 -imdb

1. 3LSTM exp + 1dense relu + sigmoid+adam
   Starting with NaN loss
2. change activation #sovle
   a. Tanh



not converge but no Nan, seems like vanish.

b. sigmoid



solve the problem well.

3. clipvalue、clipnorm# no effect

a. clipvalue2

no effect

```
Epoch 1/20
25000/25000 [==============================] - 139s 6ms/step - loss: nan - accuracy: 0.0000e+00 - val_
loss: nan - val_accuracy: 0.0000e+00
Epoch 2/20
11392/25000 [============>.................] - ETA: 57s - loss: nan - accuracy: 0.0000e+00/data/zxy/an
aconda3/envs/py36_gpu/bin/python /data/zxy/DL_tools/case26-rnn-imdb.py
25000/25000 [==============================] - 129s 5ms/step - loss: nan - accuracy: 0.0000e+00 - val_
loss: nan - val_accuracy: 0.0000e+00
Epoch 3/20
 2048/25000 [=>............................] - ETA: 1:21 - loss: nan - accuracy: 0.0000e+00
```

b. clipnorm1

no effect

```
Epoch 1/20
25000/25000 [==============================] - 139s 6ms/step - loss: nan - accuracy: 0.0000e+00 - val
loss: nan - val_accuracy: 0.0000e+00
Epoch 2/20
 3200/25000 [==>...........................] - ETA: 1:32 - loss: nan - accuracy: 0.0000e+00
```

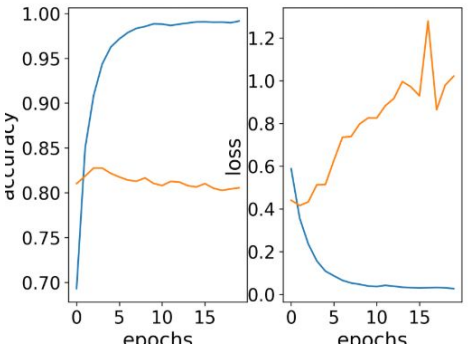
4. BN # no effect.
   no effect

5. 3 Bidirectional LSTM+ 1dense exp+sigmoid+ adam

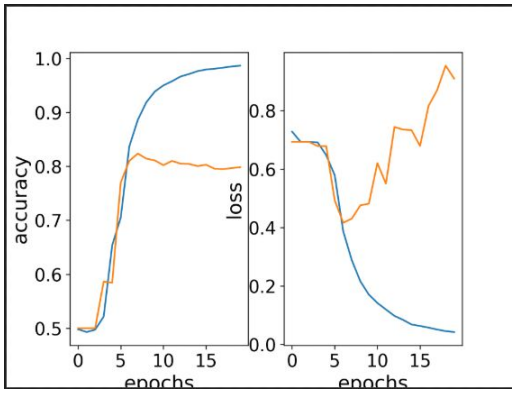
```
3072/25000 [==>...........................] - ETA: 1:44 - loss: nan - accu
3200/25000 [==>...........................] - ETA: 1:42 - loss: nan - accu
3328/25000 [==>...........................] - ETA: 1:41 - loss: nan - accu
3456/25000 [===>..........................] - ETA: 1:40 - loss: nan - accu
3584/25000 [===>..........................] - ETA: 1:39 - loss: nan - accu
3712/25000 [===>..........................] - ETA: 1:38 - loss: nan - accu
3840/25000 [===>..........................] - ETA: 1:37 - loss: nan - accu
3968/25000 [===>..........................] - ETA: 1:36 - loss: nan - accu
5120/25000 [=====>........................] - ETA: 1:28 - loss: nan - accuracy: 0.0000e+00
```

   nan at start.

6. change activation #solve the problem
   a. tanh : no NaN loss and converge



   b. sigmoid: no NaN loss and converge



   c. Additionally, Using BN will help with the tanh and sigmoid activation to converge.

7. clipnorm/clipvalue
   a. clipnorm:

```
Train on 25000 samples, validate on 25000 samples
Epoch 1/20
 1152/25000 [>.............................] - ETA: 3:58 - loss: nan - accuracy: 0.0000e+00
```

no effect

b. cilpvalue:
   also no effect, start with nan loss.

8. BN:

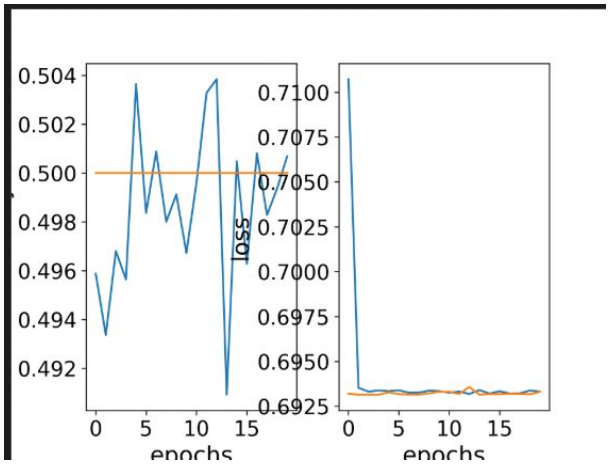
```
Epoch 1/20
25000/25000 [==============================] - 200s 8ms/step - loss: nan - accuracy: 0.0000e+00 - val_
 nan - val_accuracy: 0.0000e+00
Epoch 2/20
19712/25000 [======================>.......] - ETA: 31s - loss: nan - accuracy: 0.0000e+00^CTraceback
```

no effect.

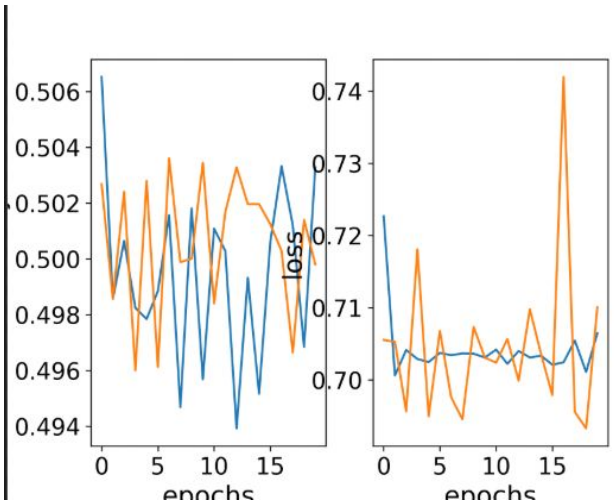
1. SimpleRNN: 3 exp+1dense relu+sigmoid+ adam
   start with NaN loss

```
Train on 25000 samples, validate on 25000 samples
Epoch 1/20
25000/25000 [==============================] - 61s 2ms/step - loss: nan
- accuracy: 0.0000e+00 - val_loss: nan - val_accuracy: 0.0000e+00
Epoch 2/20
25000/25000 [==============================] - 64s 3ms/step - loss: nan
- accuracy: 0.0000e+00 - val_loss: nan - val_accuracy: 0.0000e+00
Epoch 3/20
25000/25000 [==============================] - 61s 2ms/step - loss: nan
- accuracy: 0.0000e+00 - val_loss: nan - val_accuracy: 0.0000e+00
Epoch 4/20
 9600/25000 [==========>...................] - ETA: 26s - loss: nan - ac
curacy: 0.0000e+00
```

2. change activationg
   a. sigmoid



   b. tanh



c. graident vanish in this case, BN will improve this situation.

3. clipnorm/clipvalue
   a. clipnorm1
      still NaN loss
   b. clipvalue2
      still NaN loss
4. BN

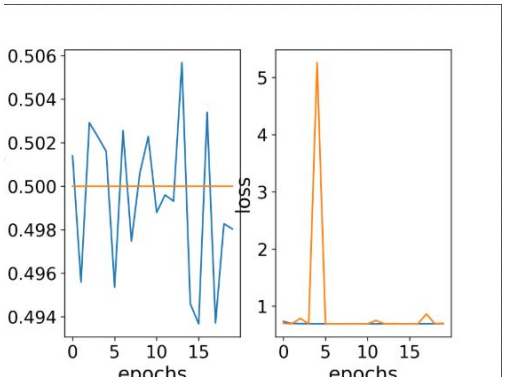
```
Train on 25000 samples, validate on 25000 samples
Epoch 1/20
24960/25000 [============================>.] - ETA: 0s - loss: nan - acc
uracy: 0.0000e+00
```

still NaN loss.

5. Bidirectional SimpleRNN: 3 exp+1dense relu+sigmoid+ adam
   start with NaN loss

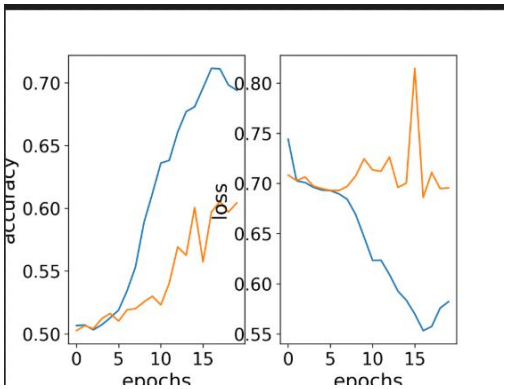
```
Epoch 1/20
 1664/25000 [>.............................] - ETA: 1:42 - loss: nan - a
ccuracy: 0.0000e+00
```

6. change activationg
   a. sigmoid



not explode but seems not to converge

b. tanh



improved well.

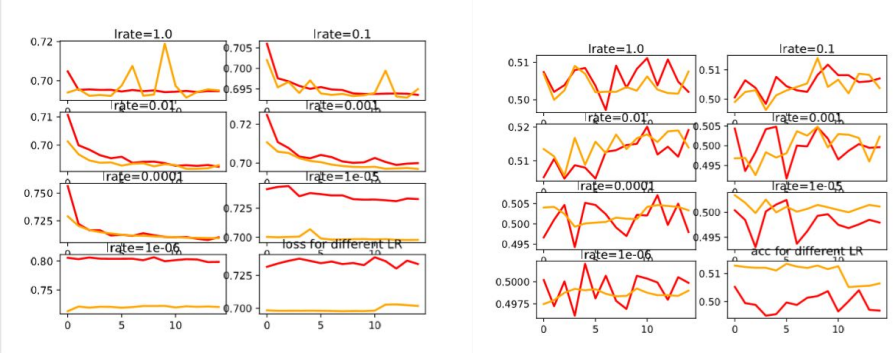
7. clipnorm/clipvalue
   a. clipnorm1
      still NaN loss
   b. clipvalue2
      still NaN loss
8. BN

still NaN loss.

Training Unstable:

case24-rnn (128batchsize)

1. SimpleRNN-sgd:

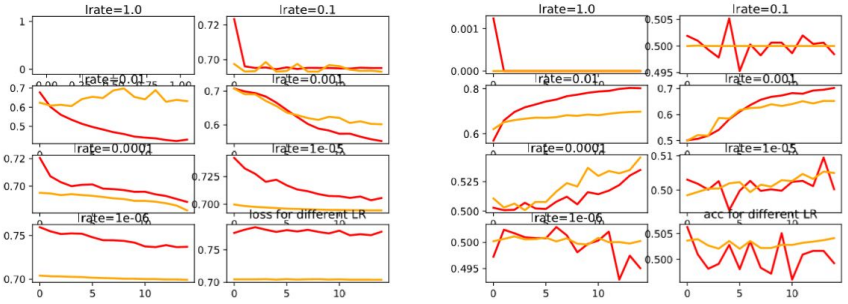


Left is the loss value, right is the accuracy.

SimpleRNN+sgd seems hardly converge at this time.

only lrate=0.01 tends to converge, but its effect is not so good.

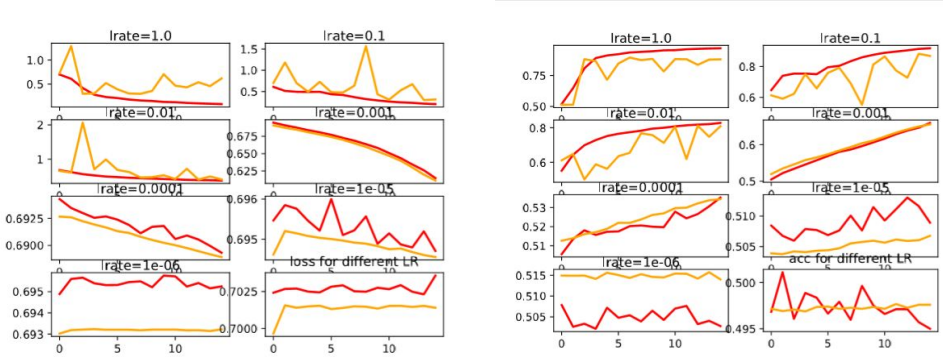
2. SimpleRNN-Adam:



It can be seen that Adam obviously performs much better than SGD in this case, and tends to converge in more lr-value settings.

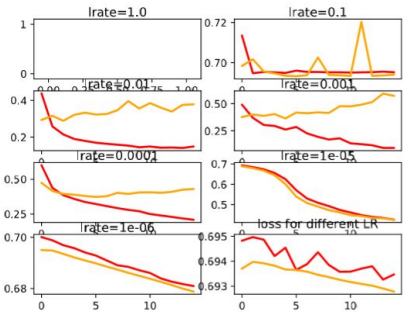
In addition, it is interesting that **Adam will have nan loss at lr = 1, and it seems that a gradient problem has occurred**, so the merits of Adam and sgd still cannot be easily distinguished.

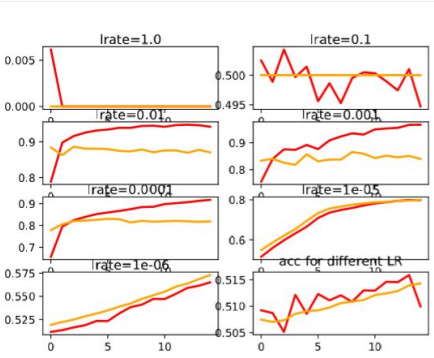
3. LSTM-sgd:



Obviously in the above figure, The LSTM performs better than SimpleRNN in Imdb dataset.LSTM network has better training effect.

4. LSTM-Adam:



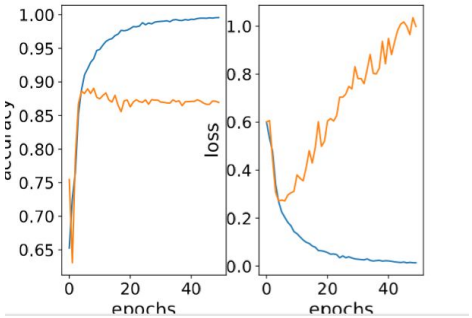


Althought Adam still has the loss issue, but it can help the network be trained well even the lr is only 1e-6.

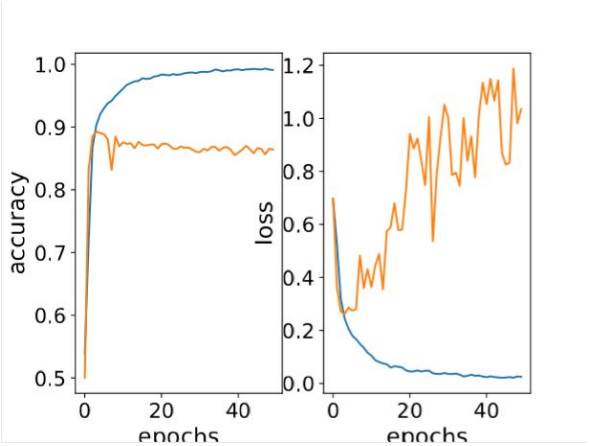
5. Conclusion:
    a. LSTM often works better than RNN in the training procedure.
    b. For sdg and Adam optimizer, Each optimizer has its own advantages.
    If the individual optimizer does not work well during training, you can consider replacing the optimizer

Training Not converge:

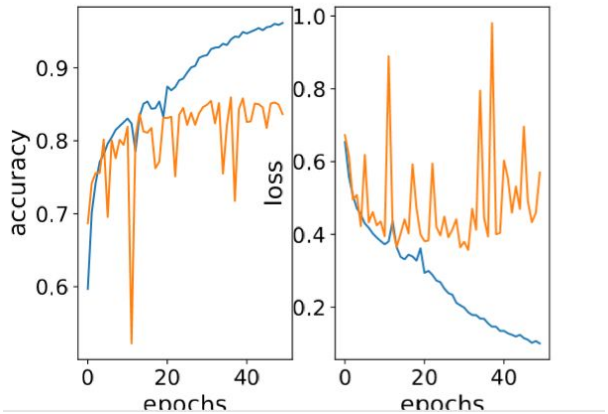
1. case20-rnn (1lstm_tanh_batch128_sgd0.1lr)



2. change lr
    a. 0.6

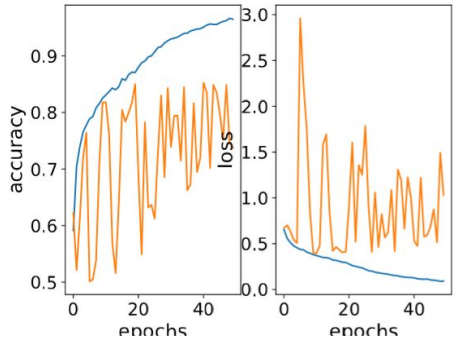


b. 0.01

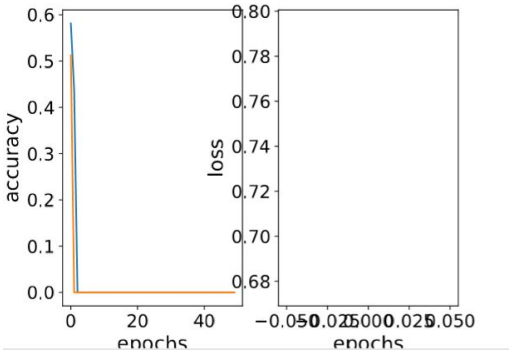


It seems that although the learning rate is reduced at this time, the value of loss function jitters at the local optimum.

3. momentum
    a. 0.1

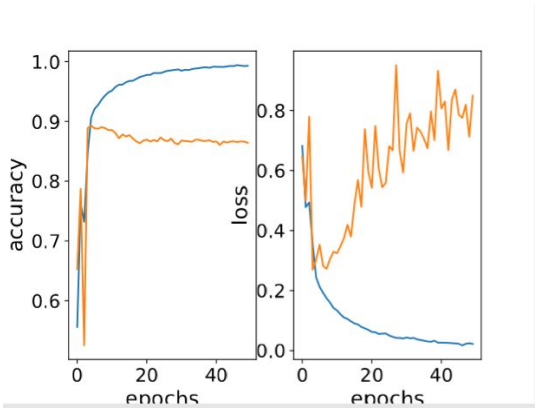


    b. 1.0

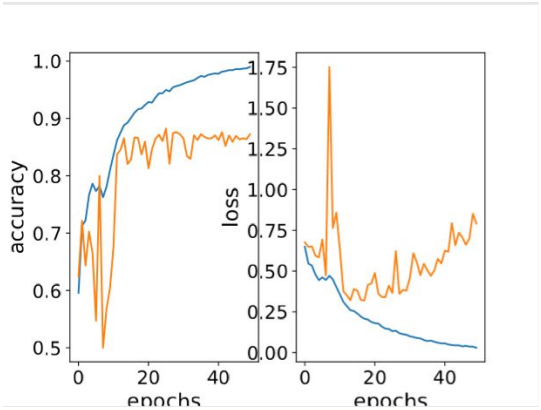


too small momentum will lead to great jitter, and too big momentum will cause nan loss.

4. change batch size
    a. 32

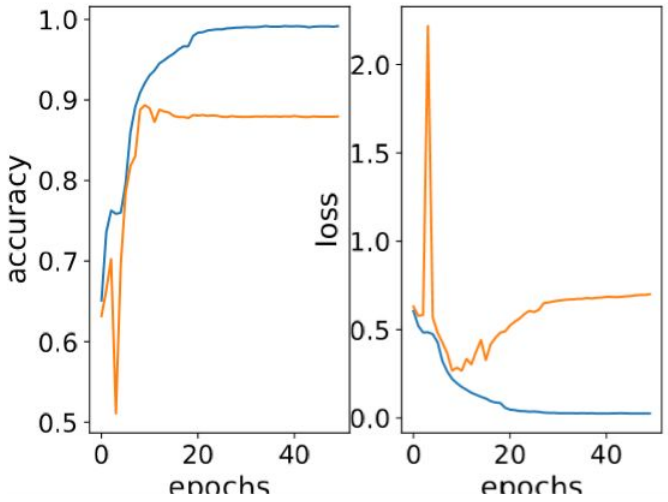


b. 512



Proper adjustment of batchsize can improve the stability of convergence.

5. reducelr

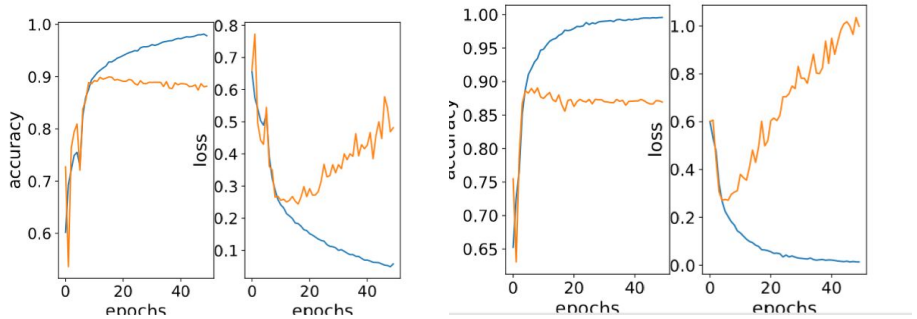


Improved well.

6. change optimizer(see case24-rnn)

Usually, adam performs better than sgd, because adam will adjust the step size adaptively

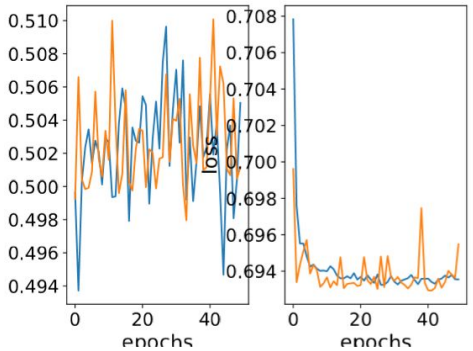
7. Gaussion Noise:



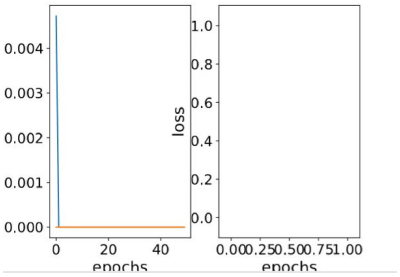
No significant improvement

RNN+sgd doesnot converge well in rnn structure....

1. case20-rnn (1rnn_tanh_batch128_sgd0.1lr)

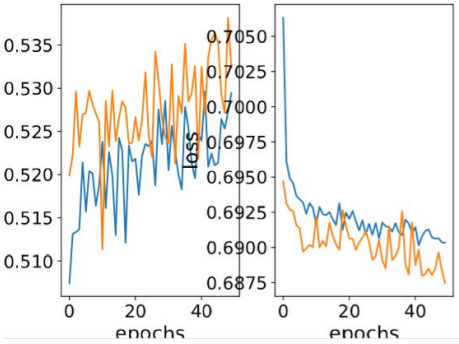


2. change lr
    a. 0.6

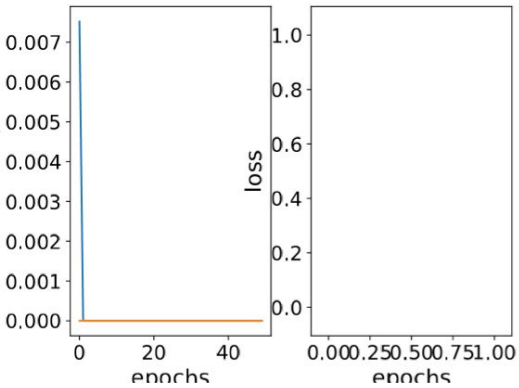


nan loss.

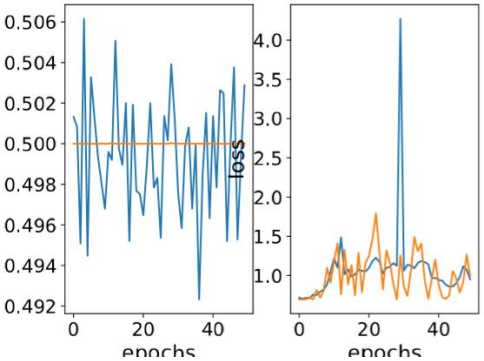
    b. 0.01



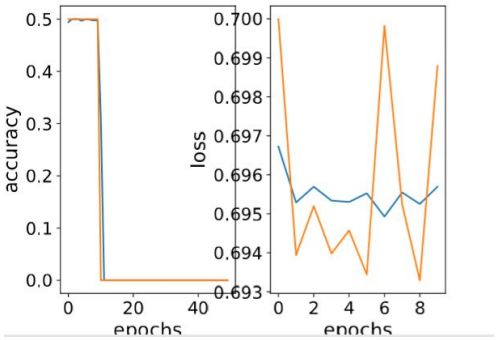
3. momentum
    a. 0.1



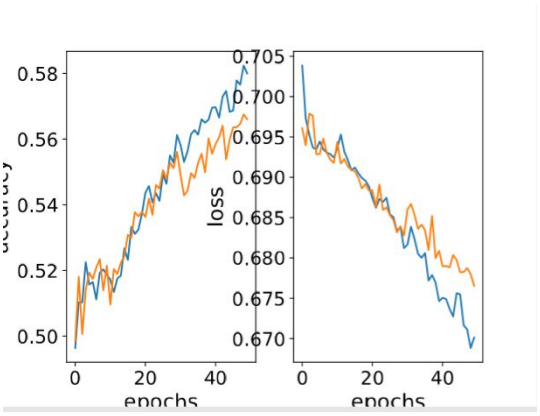
    b. 1.0



4. change batch size
    a. 32

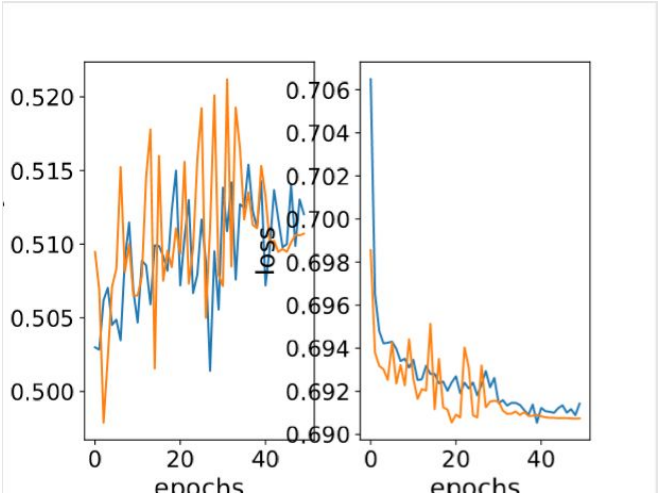


    b. 512



Proper adjustment of batchsize can improve the stability of convergence.

5. reducelr

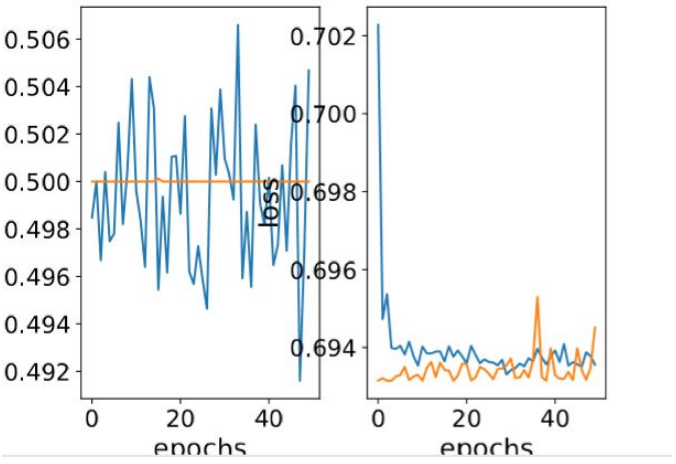


No effect

6. change optimizer(see case24-rnn)

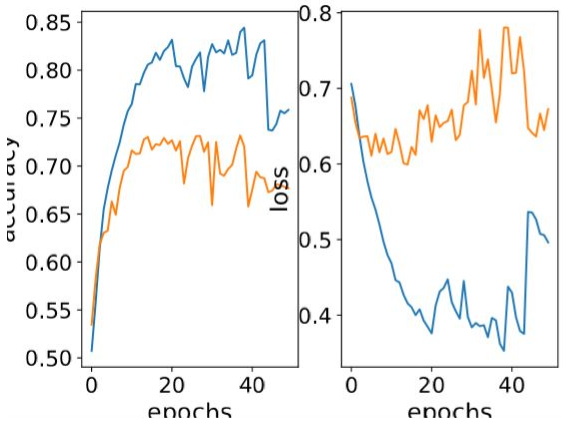
Usually, adam performs better than sgd, because adam will adjust the step size adaptively

7. Gaussion Noise:

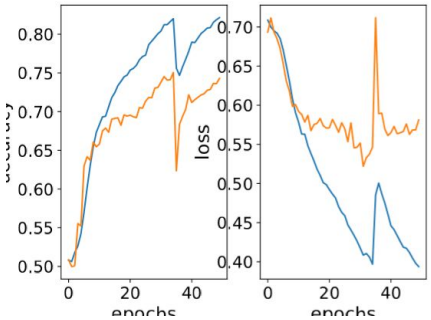


No effect.

1. Case20-rnn(1rnn_tanh_batch128_adam0.01lr)

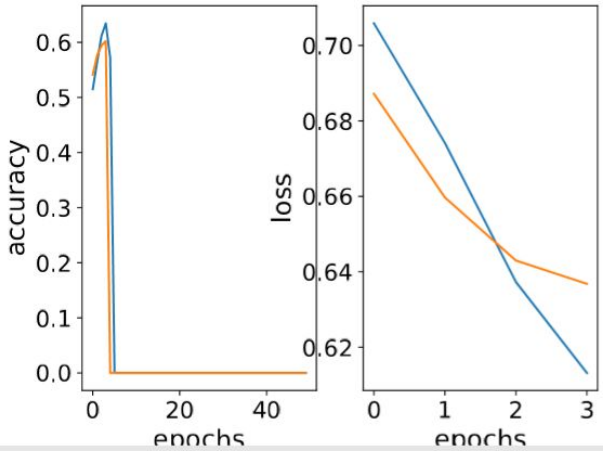


2. change lr
    a. 0.01
        Nan loss
    b. 0.001

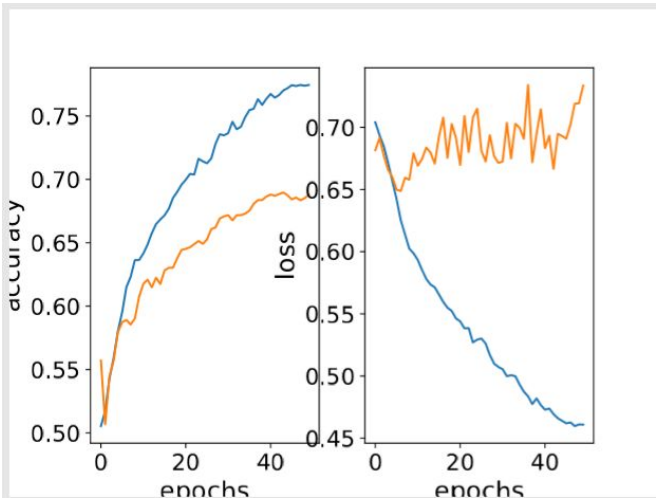


more stable when using lower lr

3. reduce lr
    nan loss----Adam will adjust the lr by itself.

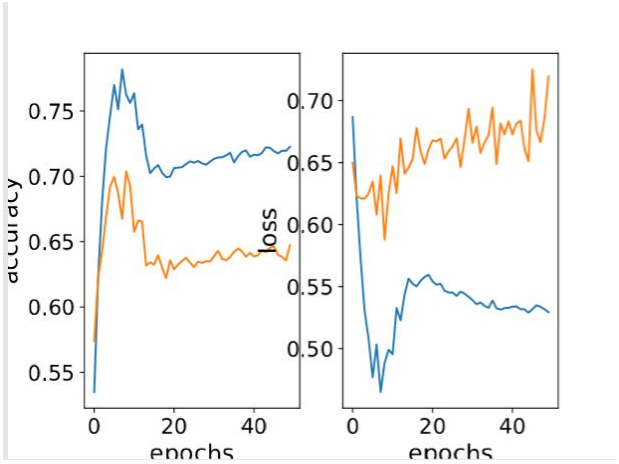


4. GN

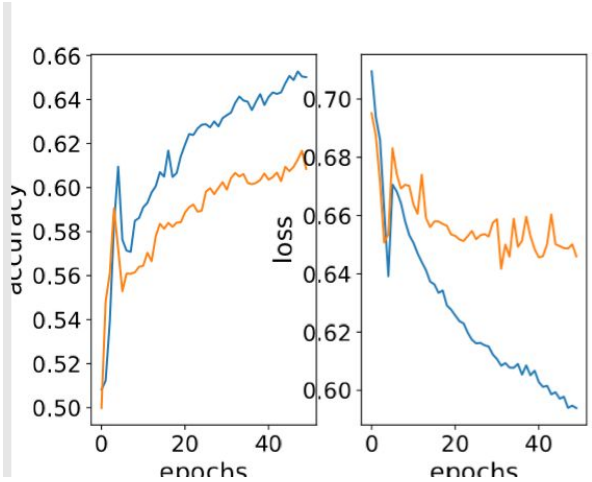


seems work, improved the loss curve

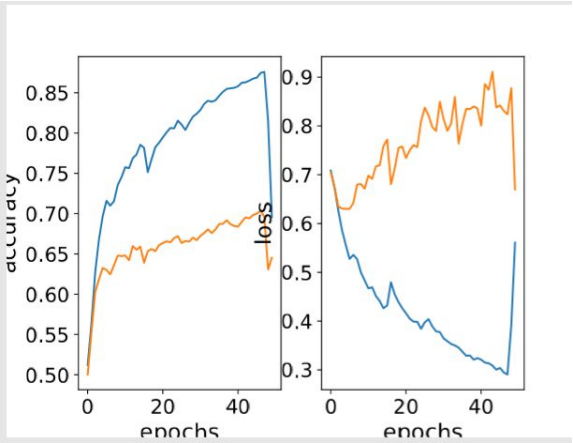
5. batchsize
    a. 32



    b. 256



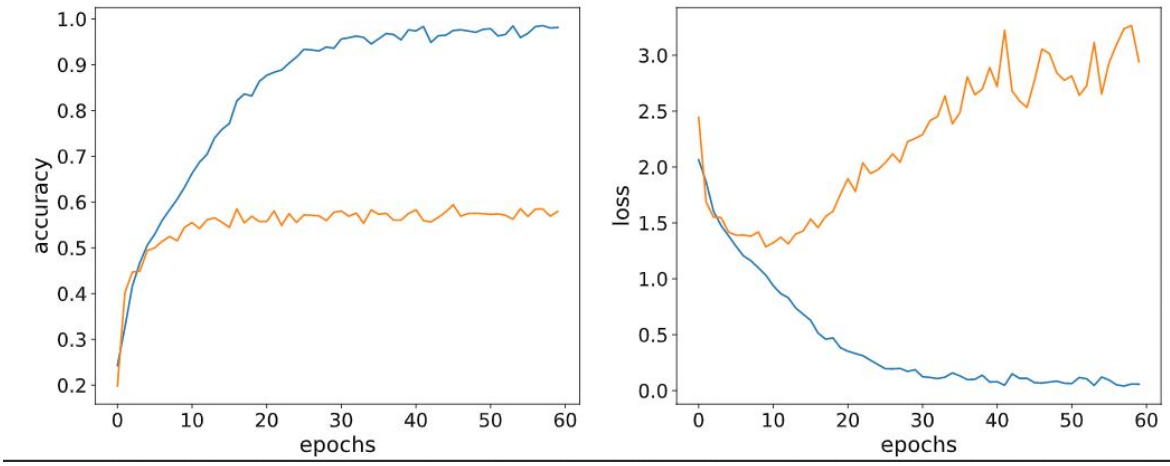
    c. 512



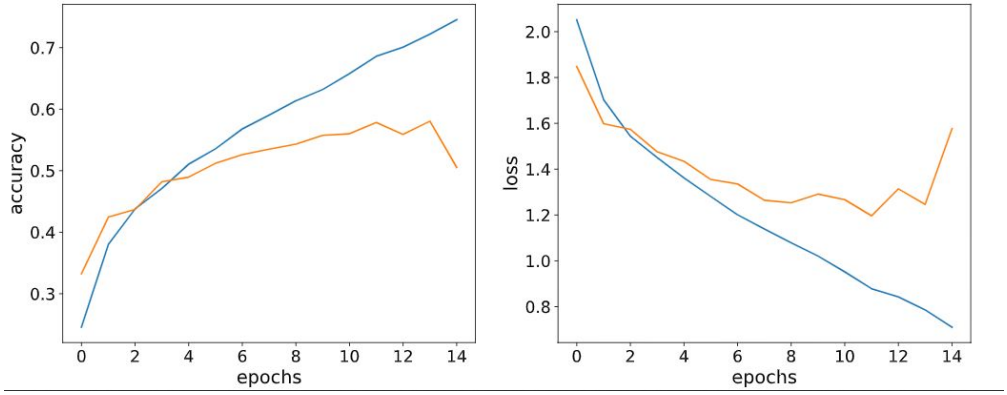
bigger batch size may lead to better stability, also may lead to not converge.

Overfit：

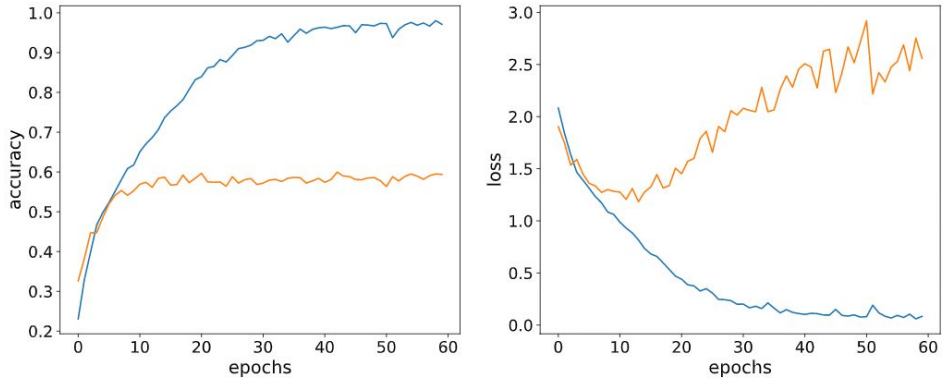
1. case30-cnn（from case16 cnn,relu,adam,256batchsize,cifar10）



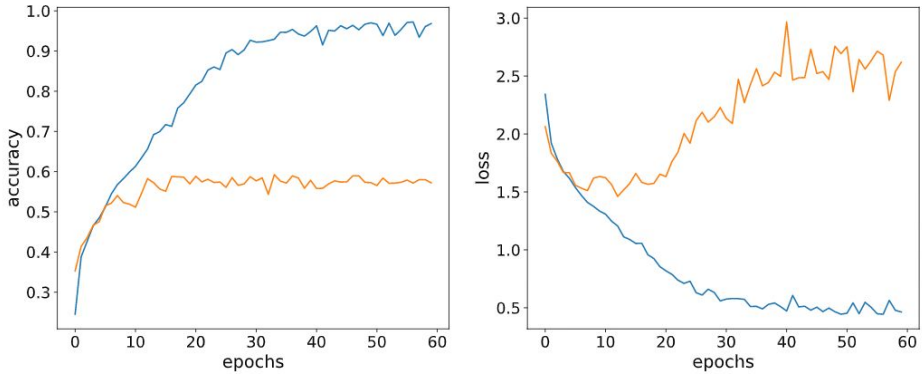
a. earlystop callback：



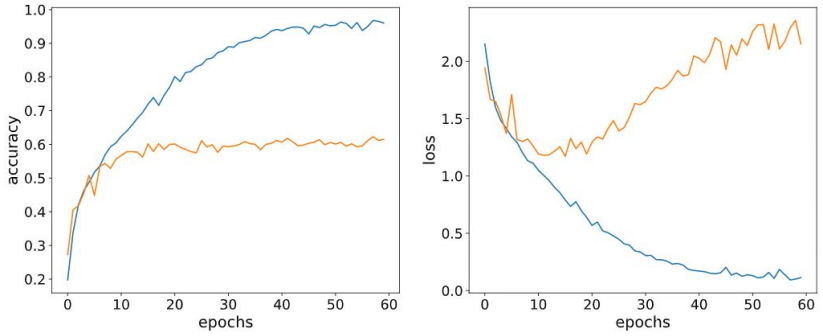
b. Dropout



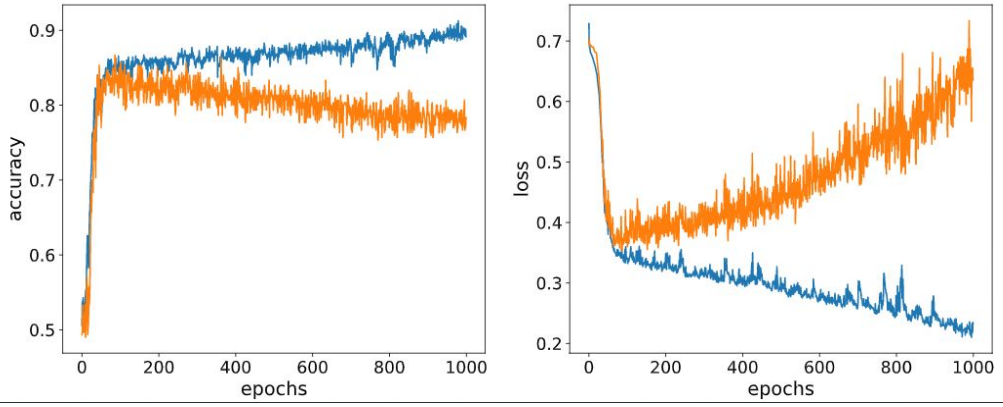
c. l2 regularize



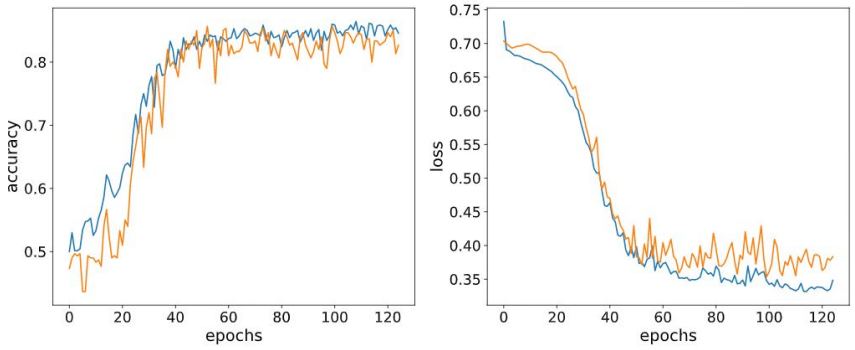
d. GN



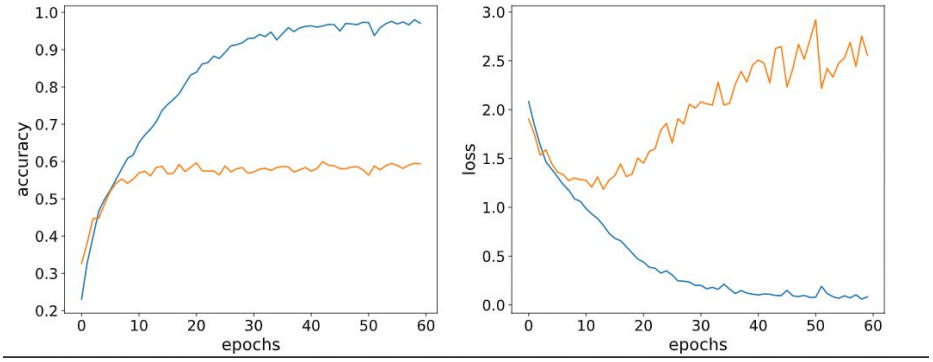
2. case30(from case16.relu,adam,batchsize=512,two circles)



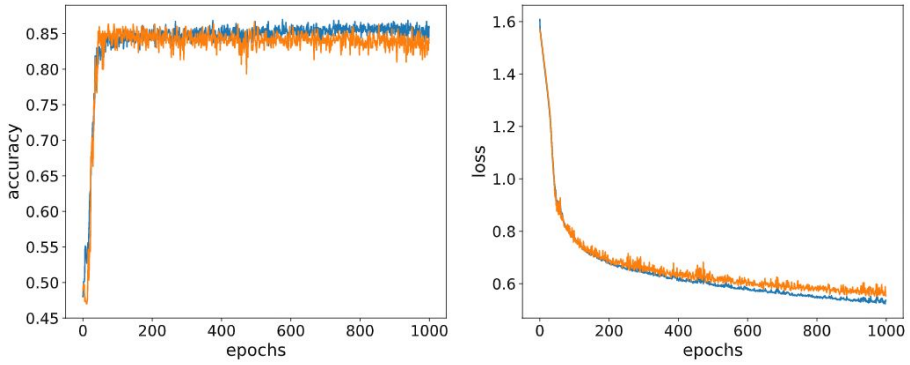
a. earlystop callback：



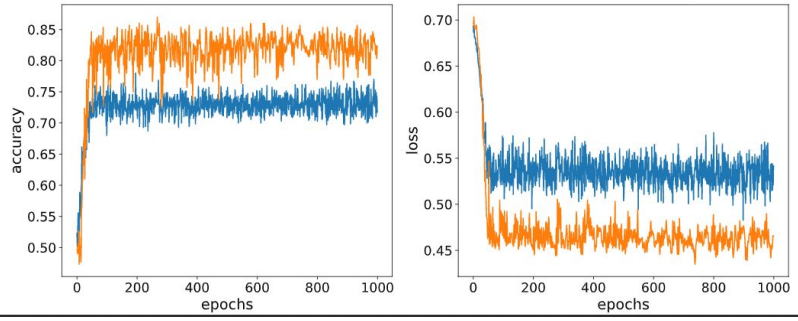
b. Dropout #too much Dropout will also lead to bad test_loss



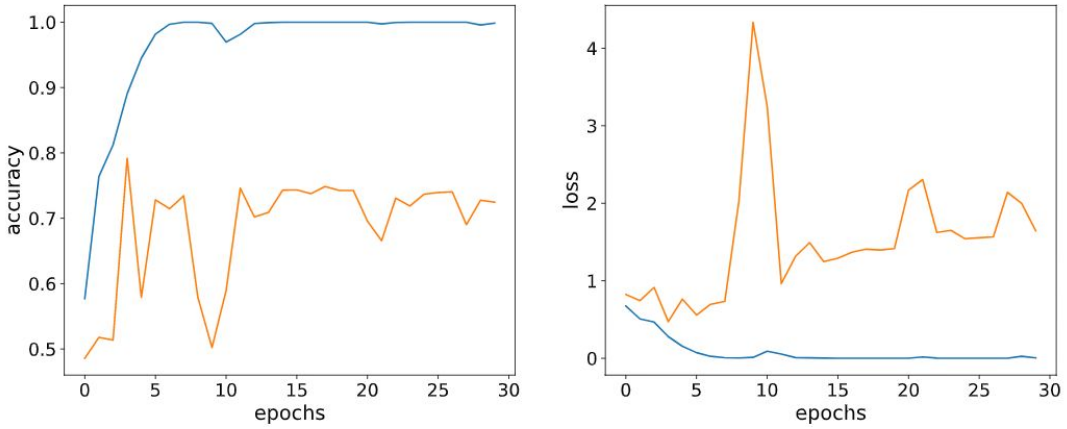
c. l2 regularize



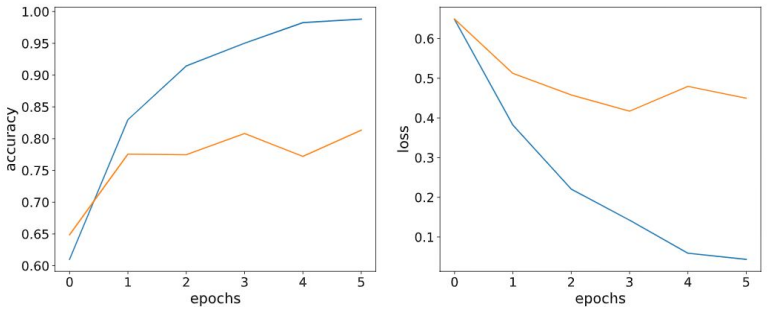
d. GN



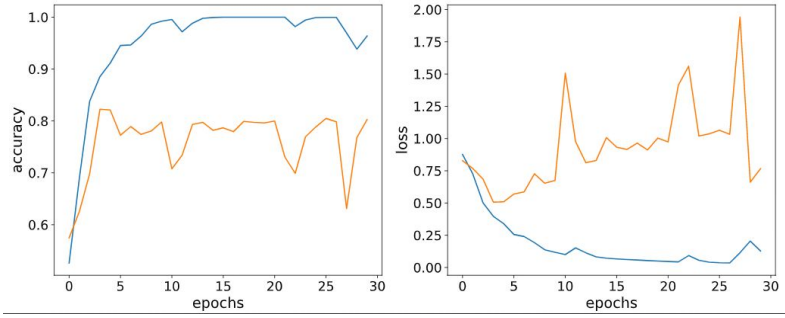
3. case30_rnn(from case20 rnn, 1rnn tanh,batch128_adam0.1lr)



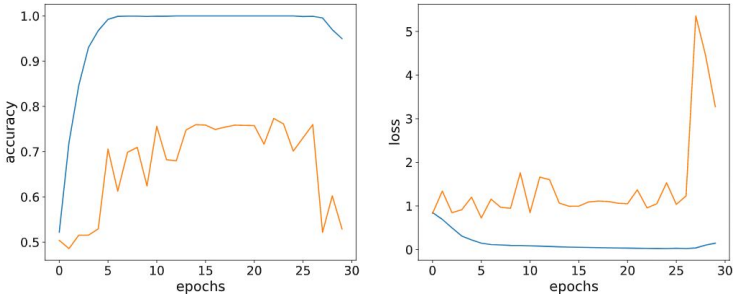
a. earlystop callback :



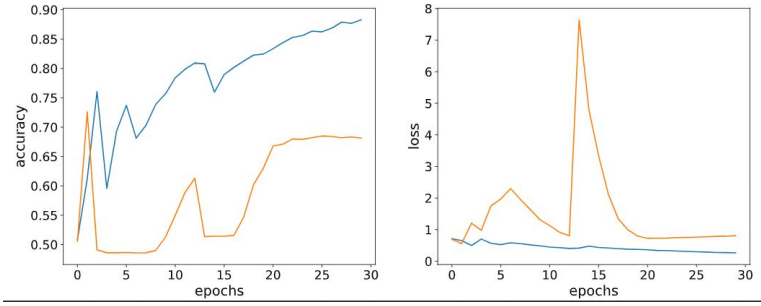
b. Dropout



c. l2 regularize



d. GN



4. case30_lstm(from case20 rnn, 1rnn tanh,batch128_adam0.1lr)

a. earlystop callback：

b. Dropout

c. l2 regularize

d. GN