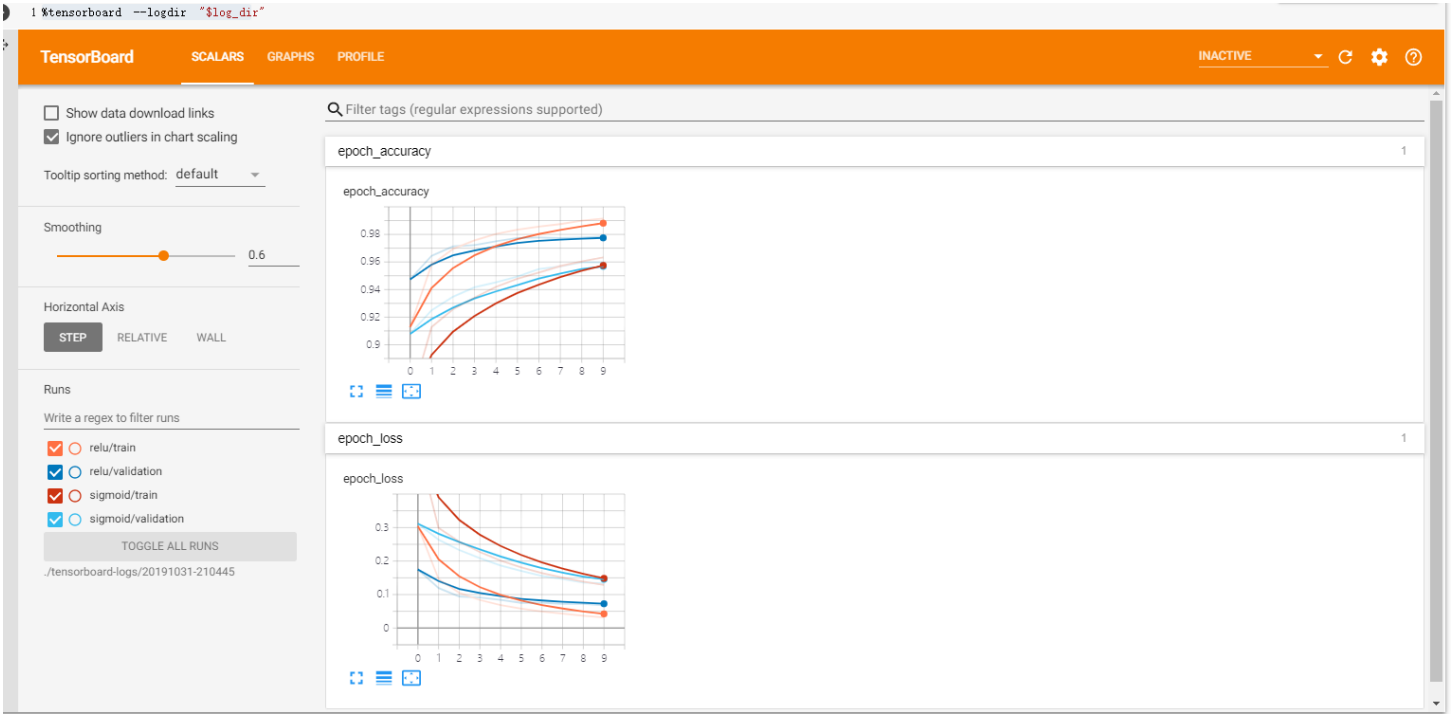
目录

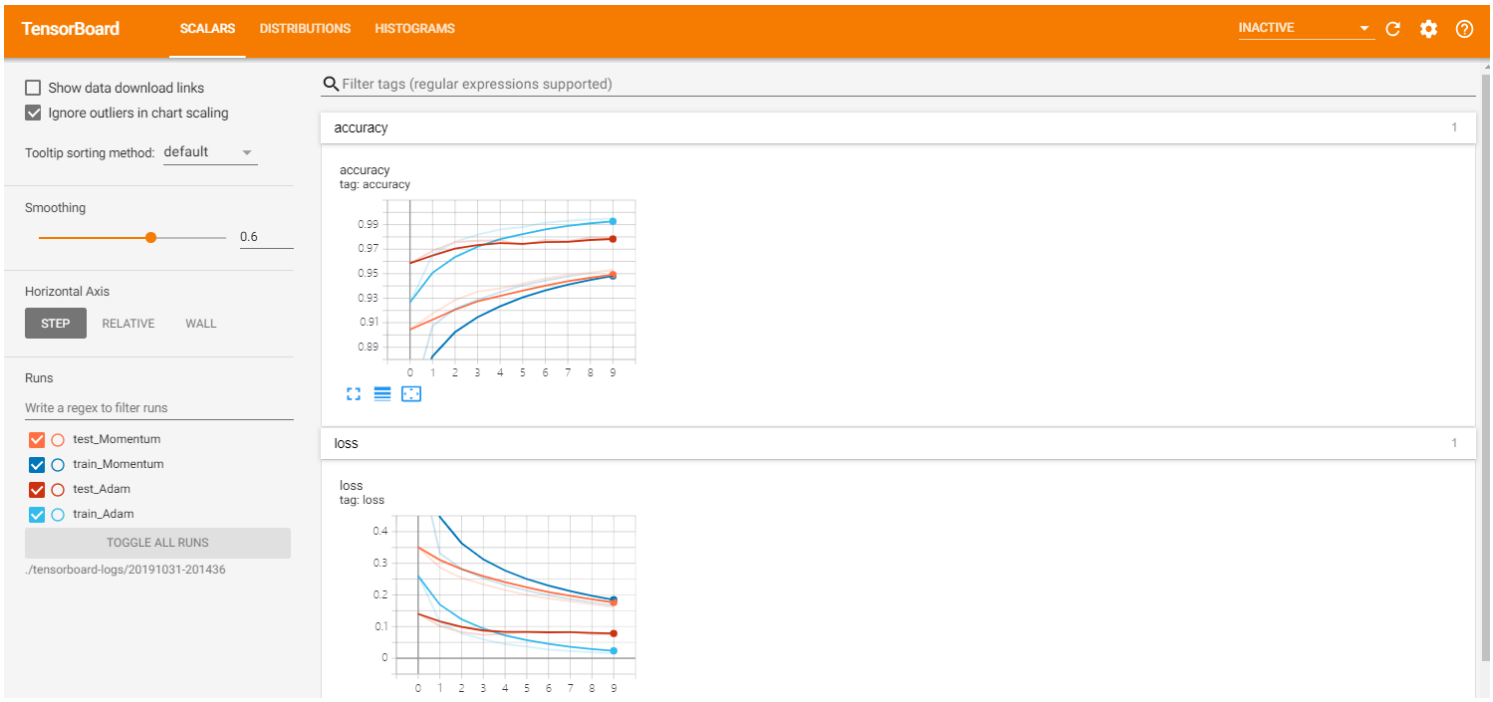
1. ReLU and Sigmoid



We can notice that the Relu is more useful than the Sigmoid. The accuracy of the Relu is higher than the Sigmoid. Because the Sigmoid function is monotonic but function's derivative is not. The Sigmoid function can cause a neural network to get stuck at the training time.

Although the Relu function and its derivative both are monotonic. But the issue is that all the negative values become zero immediately which decreases the ability of the model to fit or train from the data properly. That means any negative input given to the ReLU activation function turns the value into zero immediately in the graph, which in turn affects the resulting graph by not mapping the negative values appropriately.

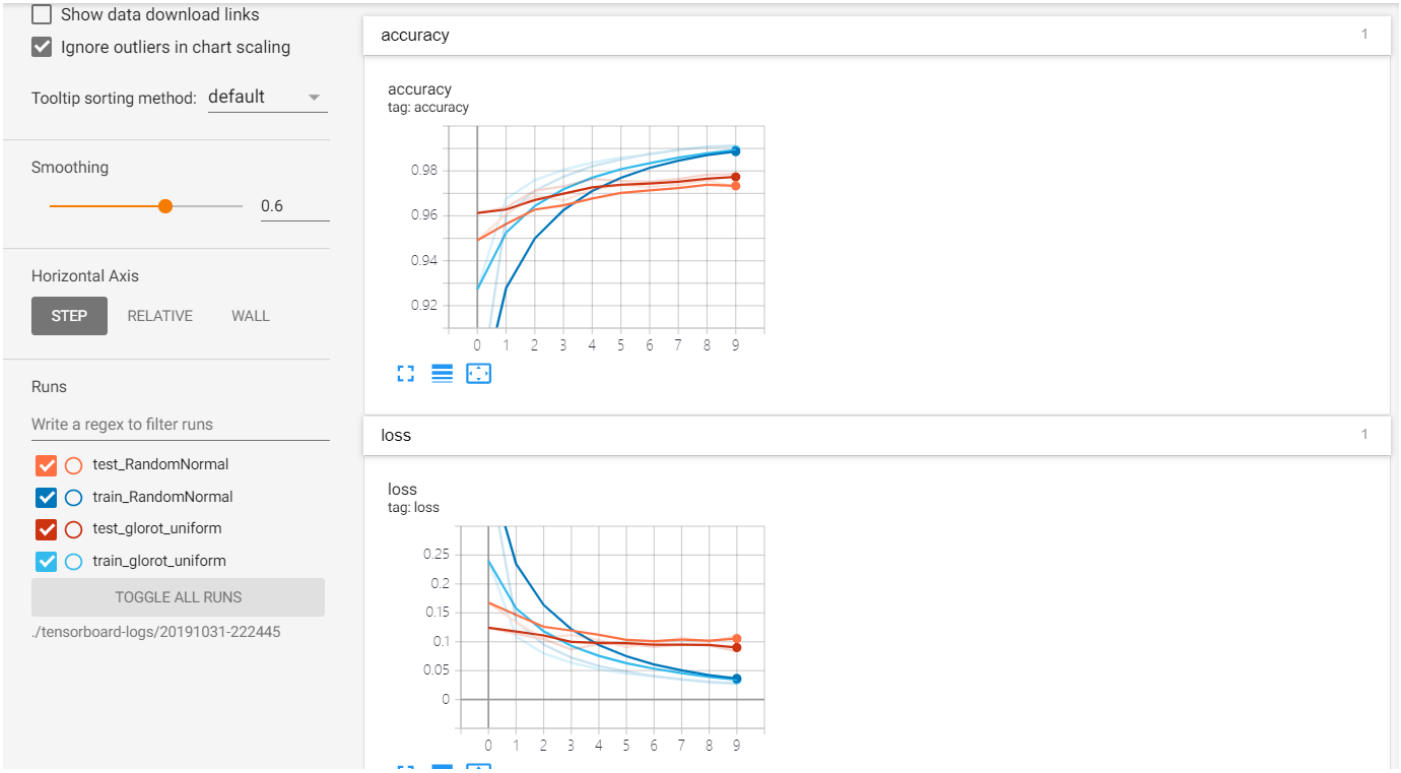
2. Optimizer: Monmentum and Adam



It shows empirically that Adam works well in practice and compares favorably to other adaptive learning-method algorithms.

Because in addition to storing an exponentially decaying average of past squared gradients Vt like Adadelta and RMSprop, Adam also keeps an exponentially decaying average of past gradients mt, similar to momentum. Whereas momentum can be seen as a ball running down a slope, Adam behaves like a heavy ball with friction, which thus prefers flat minima in the error surface.

3. Initializer: random normal and glorot_uniform



From the picture above, I found that Glorot is more useful than Random normal. Why shouldn't initialize the weights with zeroes or randomly (without knowing the distribution)? Because, if the weights in a network start too small, then the signal shrinks as it passes through each layer until it's too tiny to be useful. If the weights in a network start too large, then the signal grows as it passes through each layer until it's too massive to be useful.

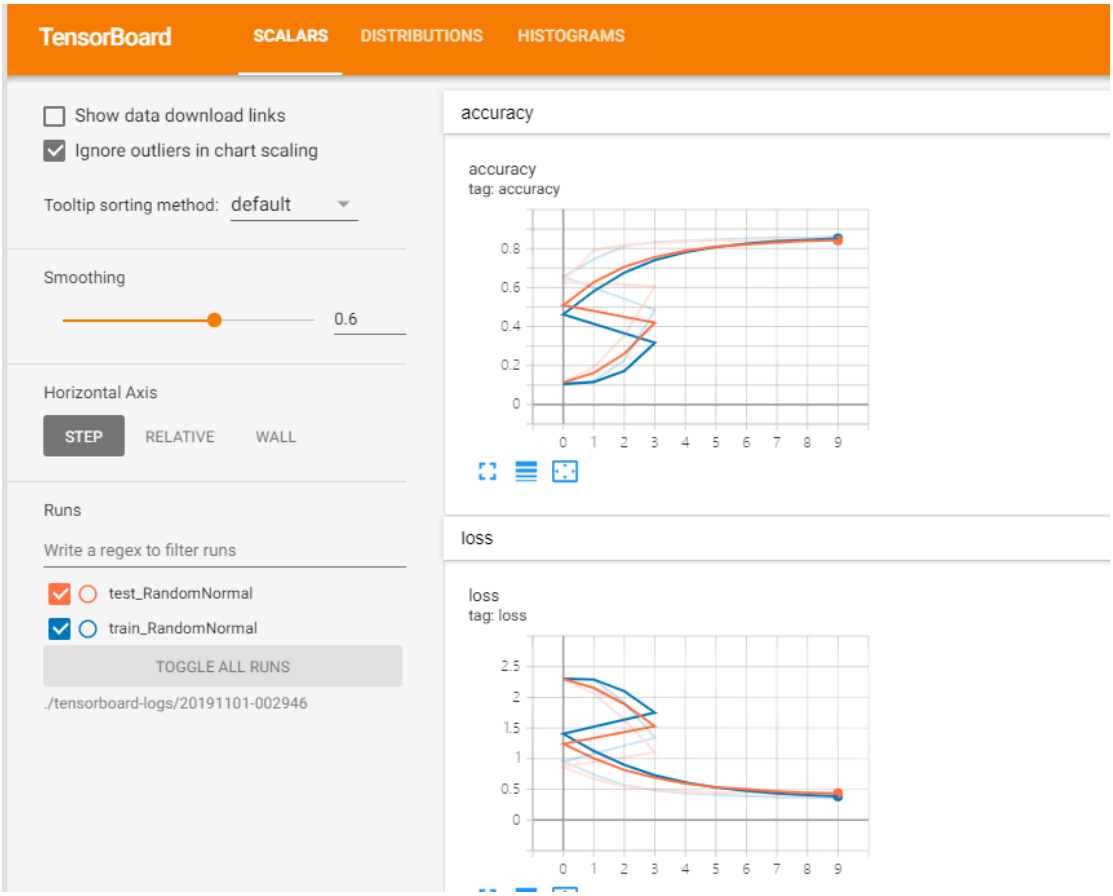
4. Extra points

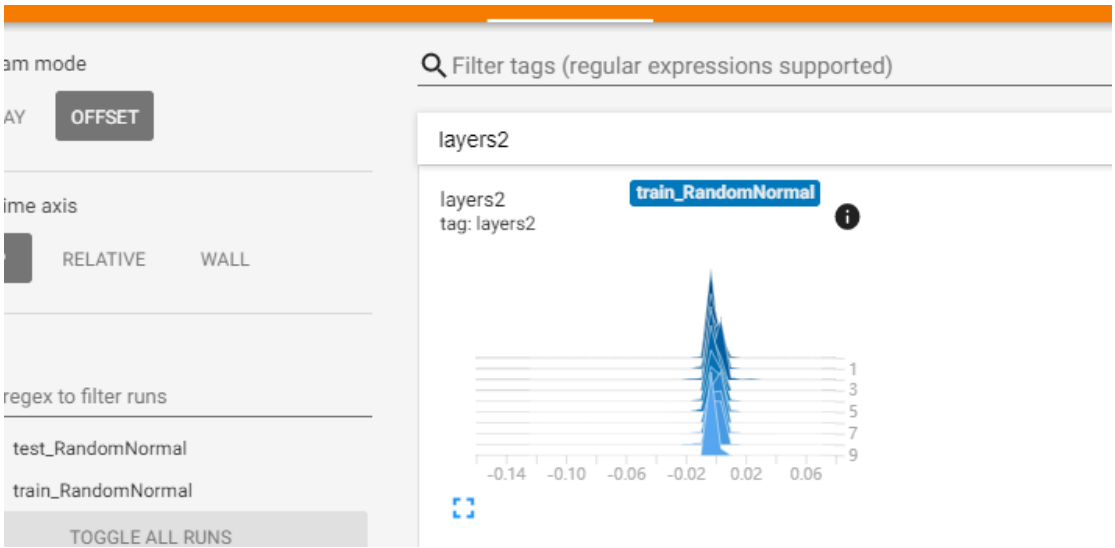
The worse module use

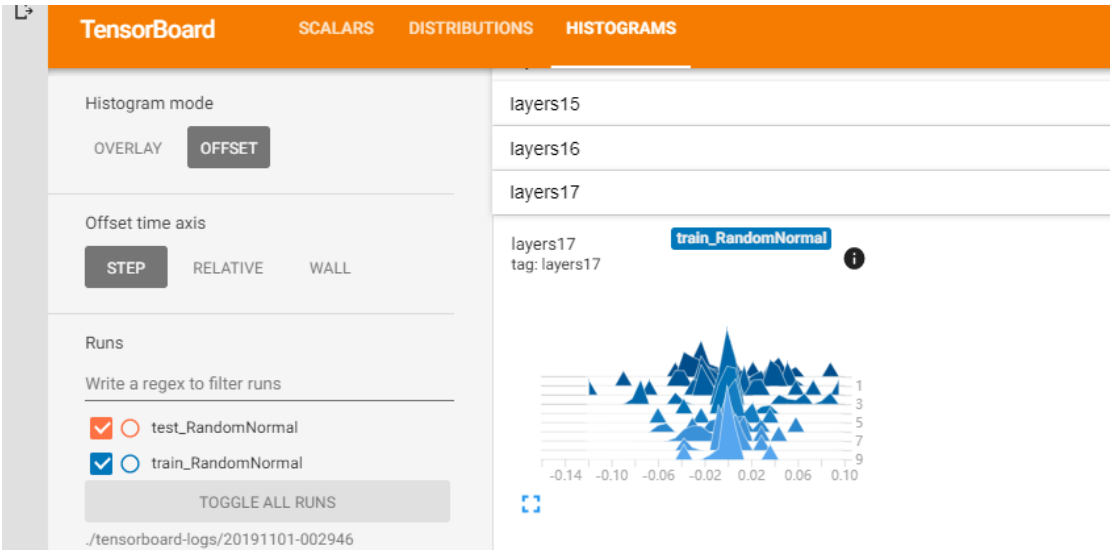
```
activation='sigmoid', kernel_initializer='RandomNormal'
```

output is

```
Epoch 1, Loss: 0.9546007513999939, Accuracy: 65.45893859863281,
Test Loss: 0.8699145913124084, Test Accuracy: 62.769996643066406
Epoch 2, Loss: 0.7407181262969971, Accuracy: 74.5233383178711,
Test Loss: 0.6722531318664551, Test Accuracy: 78.98999786376953
Epoch 3, Loss: 0.5697672963142395, Accuracy: 81.25167083740234,
Test Loss: 0.5472183227539062, Test Accuracy: 81.97000122070312
Epoch 4, Loss: 0.4818856120109558, Accuracy: 83.35499572753906,
Test Loss: 0.5042455792427063, Test Accuracy: 82.66000366210938
Epoch 5, Loss: 0.4387161135673523, Accuracy: 83.98332977294922,
Test Loss: 0.45968160033226013, Test Accuracy: 83.63999938964844
Epoch 6, Loss: 0.41364315152168274, Accuracy: 84.66166687011719,
Test Loss: 0.442256897687912, Test Accuracy: 84.09000396728516
Epoch 7, Loss: 0.39138153195381165, Accuracy: 85.19999694824219,
Test Loss: 0.45372965931892395, Test Accuracy: 83.60000610351562
Epoch 8, Loss: 0.36889100074768066, Accuracy: 85.67333221435547,
Test Loss: 0.4170512855052948, Test Accuracy: 84.75999450683594
Epoch 9, Loss: 0.36696454882621765, Accuracy: 85.70333099365234,
Test Loss: 0.4131409227848053, Test Accuracy: 84.8699951171875
Epoch 10, Loss: 0.3516475260257721, Accuracy: 86.16832733154297,
Test Loss: 0.4209926128387451, Test Accuracy: 84.6500015258789
```


TensorBoard
SCALARS
DISTRIBUTIONS
HISTOGRAMS
Show data download links
Ignore outliers in chart scaling
Tooltip sorting method: default
Smoothing
0.6
Horizontal Axis
STEP
RELATIVE
WALL
Runs
Write a regex to filter runs
test_RandomNormal
train_RandomNormal
TOGGLE ALL RUNS
./tensorboard-logs/20191101-002946
accuracy
accuracy
tag: accuracy
0.8
0.6
0.4
0.2
0
0 1 2 3 4 5 6 7 8 9
loss
loss
tag: loss
2.5
2
1.5
1
0.5
0
0 1 2 3 4 5 6 7 8 9


am mode
AY
OFFSET
ime axis
RELATIVE
WALL
regex to filter runs
test_RandomNormal
train_RandomNormal
TOGGLE ALL RUNS
Filter tags (regular expressions supported)
layers2
layers2
tag: layers2
train_RandomNormal
1
3
5
7
9
-0.14 -0.10 -0.06 -0.02 0.02 0.06



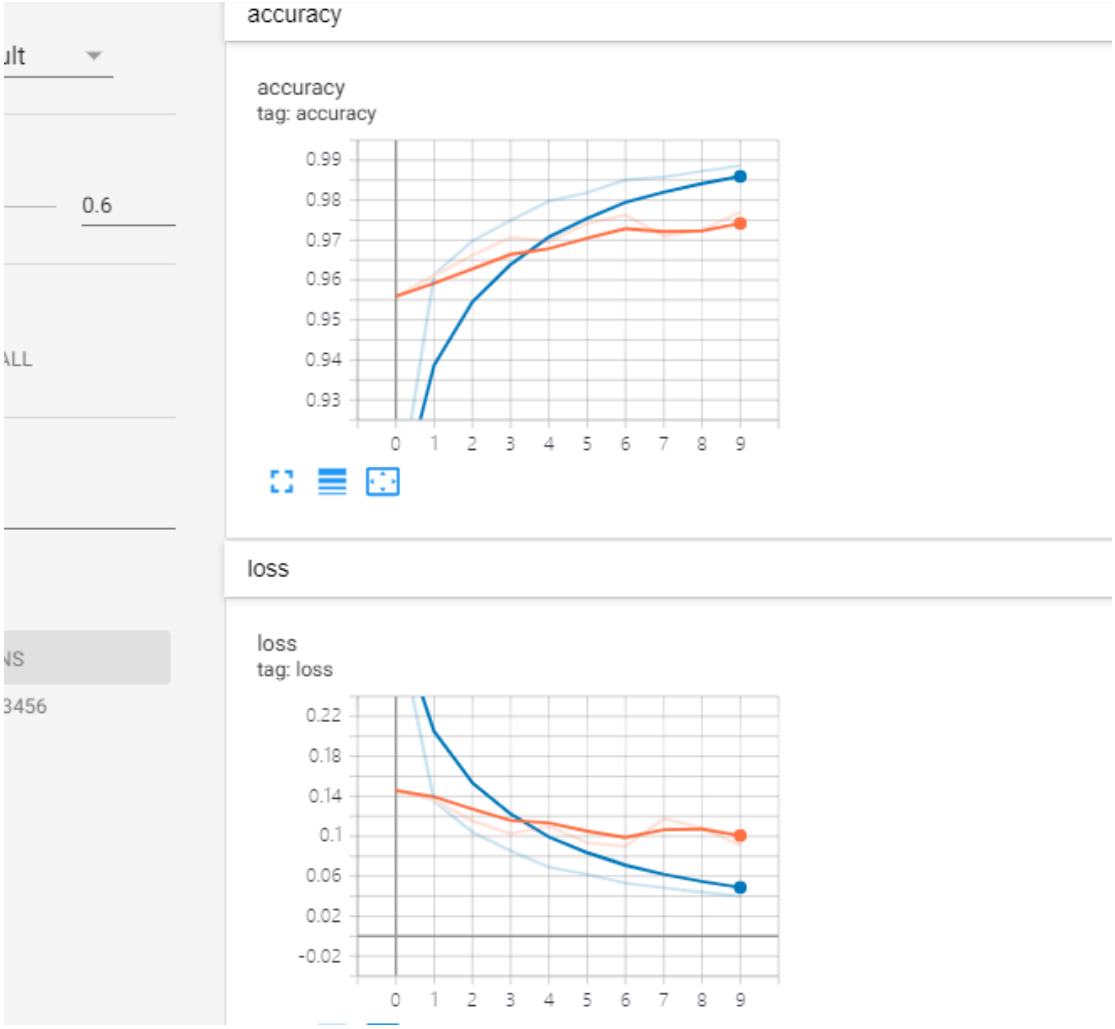
We can notice that the output is so bad and the accuracy is very low.

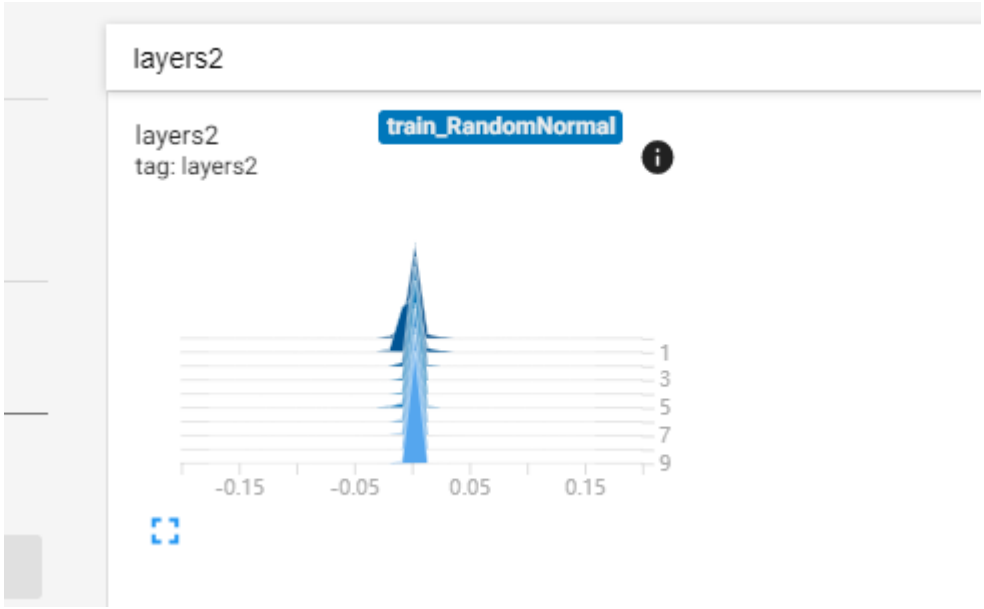
The better one

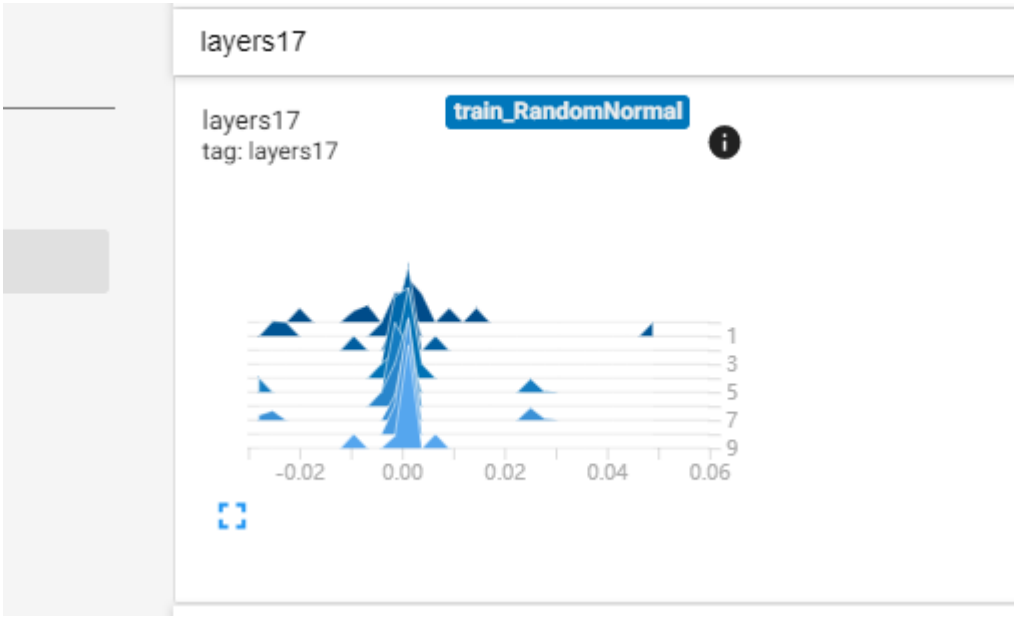
We use :
`activation='relu', kernel_initializer='he_normal'`

`output is`

```
Epoch 1, Loss: 0.32113274931907654, Accuracy: 90.09000396728516,
Test Loss: 0.14567680656909943, Test Accuracy: 95.59000396728516
Epoch 2, Loss: 0.1348998248577118, Accuracy: 96.13666534423828,
Test Loss: 0.13587947189807892, Test Accuracy: 96.12000274658203
Epoch 3, Loss: 0.10406810790300369, Accuracy: 96.97999572753906,
Test Loss: 0.11550696194171906, Test Accuracy: 96.62000274658203
Epoch 4, Loss: 0.08551705628633499, Accuracy: 97.49333190917969,
Test Loss: 0.1026444211602211, Test Accuracy: 97.06999969482422
Epoch 5, Loss: 0.06911714375019073, Accuracy: 97.97333526611328,
Test Loss: 0.11003483086824417, Test Accuracy: 96.97000122070312
Epoch 6, Loss: 0.061961881816387177, Accuracy: 98.18999481201172,
Test Loss: 0.09346961230039597, Test Accuracy: 97.41999816894531
Epoch 7, Loss: 0.052867550402879715, Accuracy: 98.50833129882812,
Test Loss: 0.089903824031353, Test Accuracy: 97.6199951171875
Epoch 8, Loss: 0.048346761614084244, Accuracy: 98.57833099365234,
Test Loss: 0.1175815612077713, Test Accuracy: 97.0999984741211
Epoch 9, Loss: 0.04398675262928009, Accuracy: 98.7249984741211,
Test Loss: 0.10840363800525665, Test Accuracy: 97.25
Epoch 10, Loss: 0.039987437427043915, Accuracy: 98.86500549316406,
Test Loss: 0.09082263708114624, Test Accuracy: 97.69999694824219
```


accuracy
accuracy
tag: accuracy
0.99
0.98
0.97
0.96
0.95
0.94
0.93
0 1 2 3 4 5 6 7 8 9
loss
loss
tag: loss
0.22
0.18
0.14
0.1
0.06
0.02
-0.02
0 1 2 3 4 5 6 7 8 9


layers2
layers2
tag: layers2
train_RandomNormal
1
3
5
7
9
-0.15
-0.05
0.05
0.15



After using a deep layer, the output still keeps well.