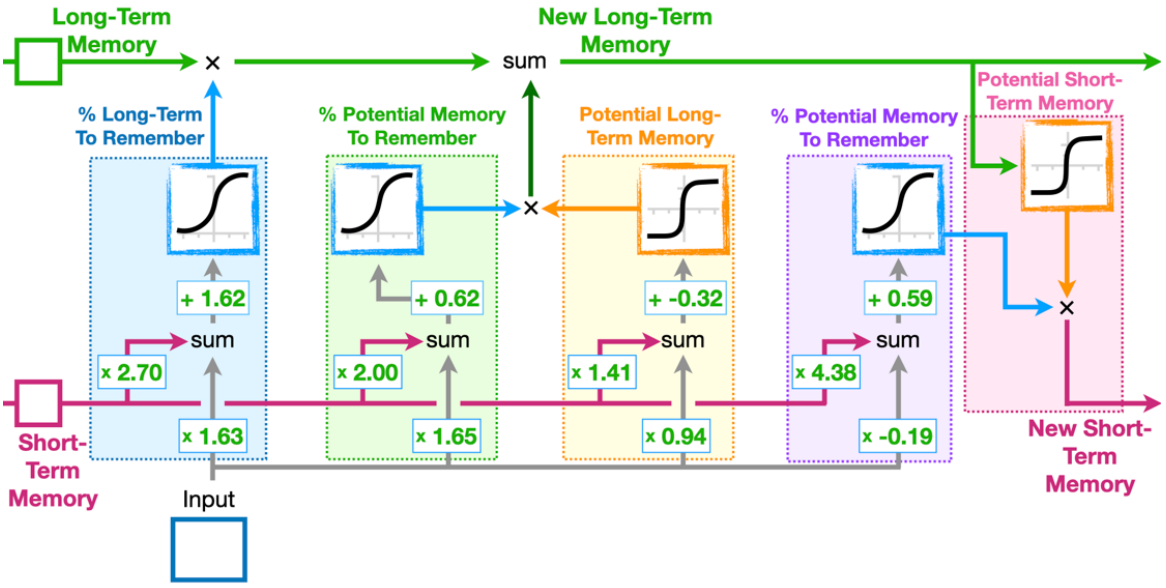
USING LSTM TO FIND THE PATTERN. (from scratch)

(No need to read text on orange)



This structure is made such a way that after a backpropagation or tweaking the weights. It will predict result close to truth.

Short-term memory and long-term memory paths are used.

First block is used to find how much percentage of long-term memory is to remember

From second 2 blocks we find potential long-term memory and add it with the previous long term memory

From last 2 blocks new short-term memory is generated.

USING IT TO FIND THE PATTERN

Making own data

Let’s create a pattern.

RULE

1. *If the graph is falling it will fall and if the graph is rising it will rise.*
2. *If it is constant it will be constant.*

Points will be quantized by 0.25.

i.e 1,0.75,0.5,0.25,0

Can it find this pattern?

Ok let's make inputs

Inputs = [1., 0.75, 0.5, 0.25], [0.5,0.75,1.,0.75], [0.5,0.25,0.5,0.75], [0.,0.25,0.5,0.75], [0.75,0.5,0.5,0.5], [0.5,0.75,1.,1.], [0.5,0.25,0.,0.25], [0.5,0.75,0.75,0.75], [0.5,0.75,0.75,0.50], [0.,0.25,0.25,0.50]

Labels = [0,0.5,1,1,0.5,1,0.5,0.75,0.25,0.75]

Upon 2000 epochs

Now let's compare the observed and predicted values...

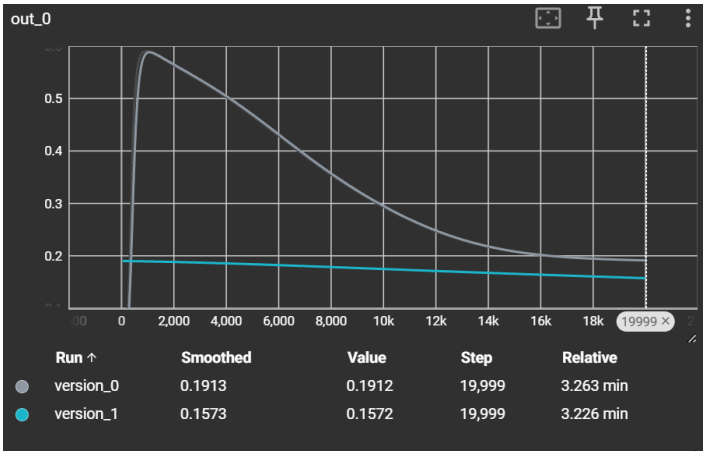
Data A: Observed = 1, Predicted = tensor(0.7698)

Data B: Observed = 0, Predicted = tensor(0.1911)

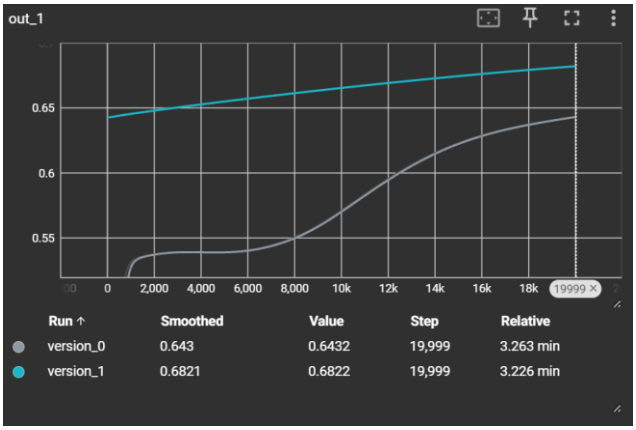
Wow already closed to the truth.

Let's see graph for further inception

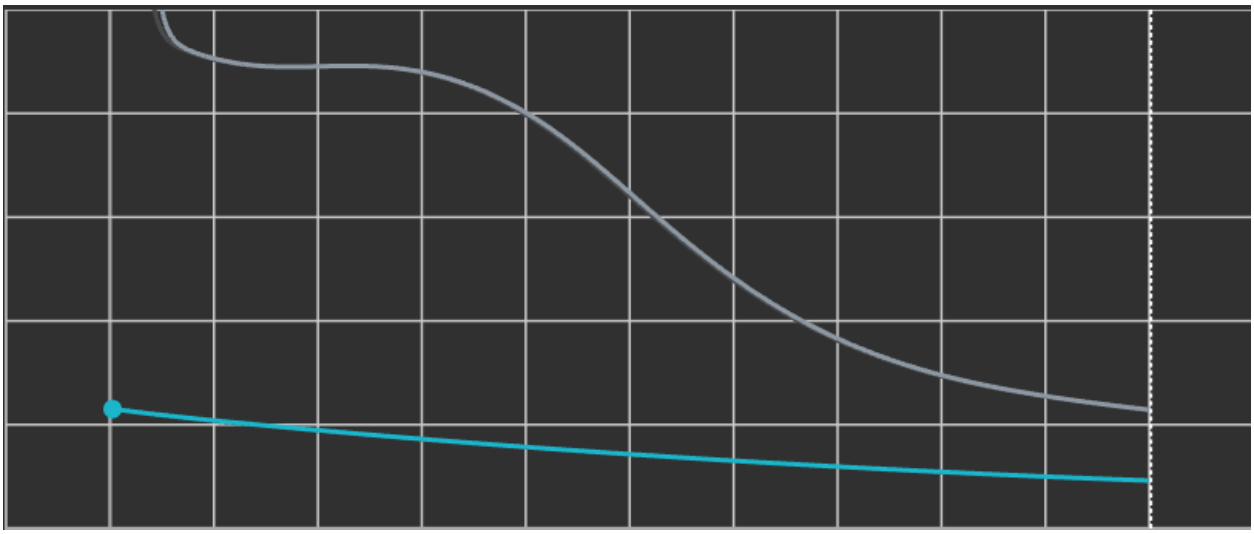
For output 0



It going towards zero, so training more would work.

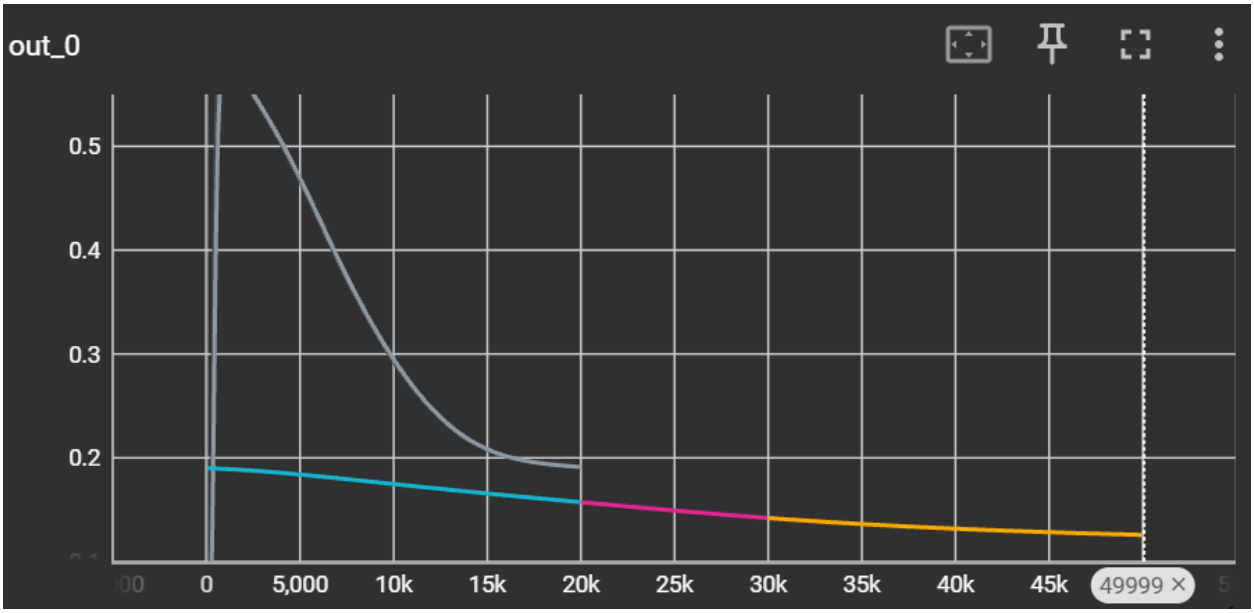


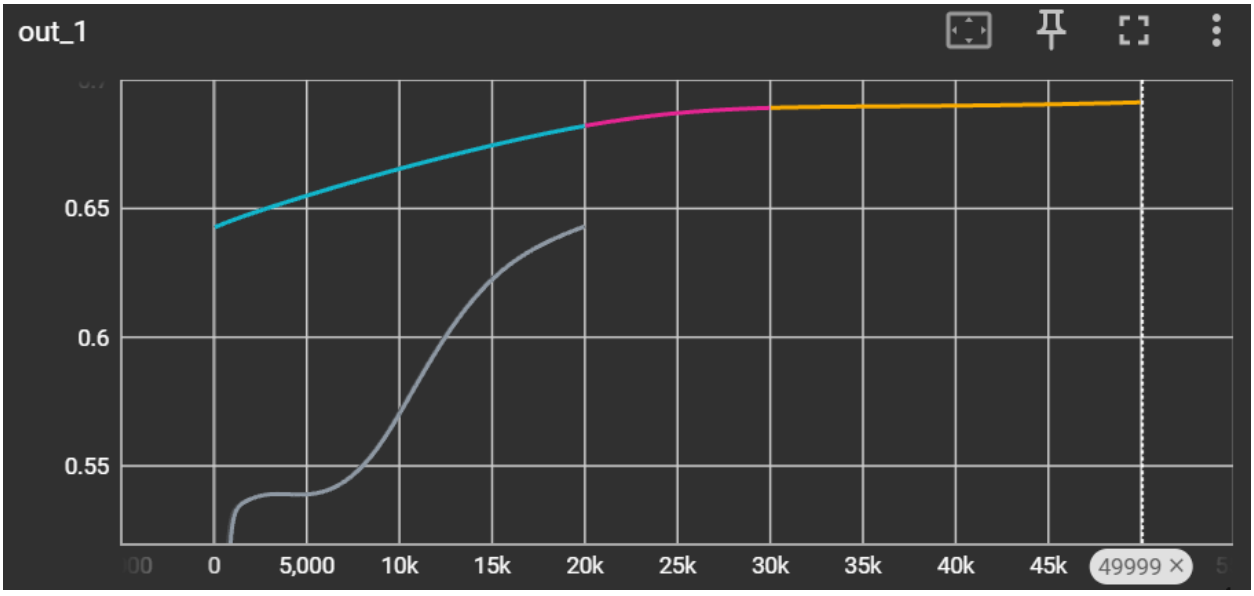
Same for out_1.

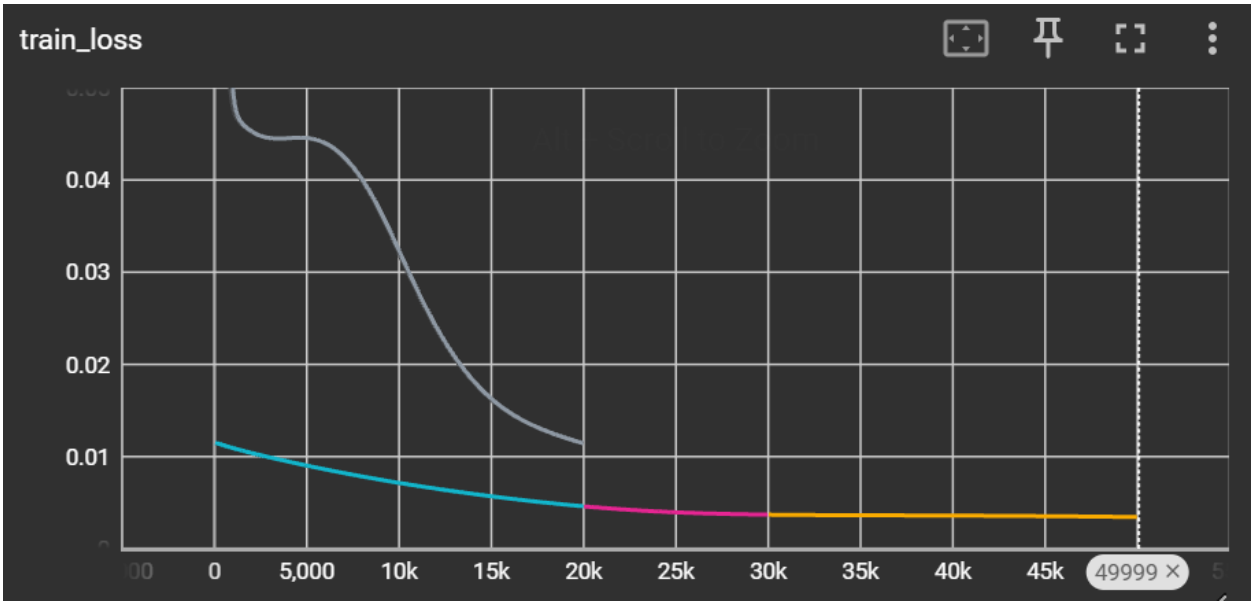


Looking loss also we can conclude we need to train more because loss is decreasing.

After additional 3000 epoch







OK graphs looks promising, let's check on test set.

TEST

Let's check on completely new 3 data (atleast 3).

1 for increasing, 1 for decreasing, 1 for constant.

Inputs.

```
print("Comparing the observed and predicted values...")
print("Data A: Truth = 0.75, Predicted =", model(torch.tensor([0.25, 0.50,
0.25, 0.5])).detach())
print("Data B: Truth = 0, Predicted =", model(torch.tensor([1., 0.75,
0.50, 0.25])).detach())
print("Data C: Truth = 0.5, Predicted =", model(torch.tensor([1., 0.75,
0.50, 0.5])).detach())
print("Data D: Truth = 0.25, Predicted =", model(torch.tensor([0.25, 0.25,
0.25, 0.25])).detach())
print("Data D: Truth = 1, Predicted =", model(torch.tensor([0.25, 0.50,
0.50, 0.75])).detach())
```

Comparing the observed and predicted values...

Data A: Truth = 0.75, Predicted = tensor(0.4464)

Data B: Truth = 0, Predicted = tensor(0.1258)

Data C: Truth = 0.5, Predicted = tensor(0.6710)

Data D: Truth = 0.25, Predicted = tensor(-0.0618)

Data D: Truth = 1, Predicted = tensor(0.8026)

OK it is acceptable. But I was hoping for the best.

Why it wasn’t not perfect?