AI502: Deep Learning (Spring 2023)

Programming Assignment 2

Hyogon Ryu, 20233477, KAIST AI

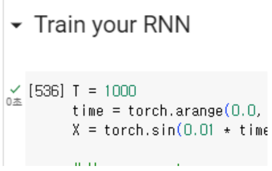
1. What I tried
   1) Change the hyperparameter
   2) Change the Architecture (small change)
      A. Attach one more fc layer
      B. Change the kind of rnn (to LSTM, GRU)
      C. etc.
   3) Remove the perturbation
   4) Change the Training Strategy(1)
      A. Learn the multi-step prediction
      B. Add the noise to the prediction

```
            optim.zero_grad()
            _batch = x.shape[0]
            inp = torch.zeros(_batch, tau+predict_num)
            inp[:,:tau] = x.reshape(_batch, -1)
            for i in range(predict_num):
                inp[:,i+tau:i+tau+1] = net(
                    inp[:,i:i+tau].reshape(_batch, -1, 1)
                )
                inp[:,i+tau:i+tau+1] += torch.randn(inp[:,i+tau:i+tau+1].shape)*0.2

            loss = criterion(inp[:,tau:], y)
            loss.backward()
```

   5) Change the Architecture (big change)
      A. Add the noise to input value
      B. Add the noise to output value
      C. Make a residual connection

```
    def forward(self, inputs):
        #######################
        # Implement here
        #inputs += torch.randn(inputs.shape)*0.1
        outputs, hidden = self.rnn(inputs)
        outputs = outputs[:, -1, :]
        outputs = self.final(outputs)

        #outputs += inputs[:,-1,:]
        #outputs += torch.randn(outputs.shape)*0.05
        #######################
        return outputs
```

   6) Etc.



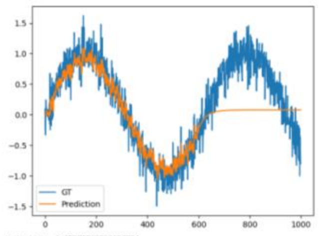
▾ Train your RNN

```
✓ [536] T = 1000
0초        time = torch.arange(0.0,
           X = torch.sin(0.01 * time
```

As you can see through above figure, I tried a lot of times to solve this problem (Even I restart the kernel several times, so maybe I tried more than 2000 times.). However, I couldn't find the solution…

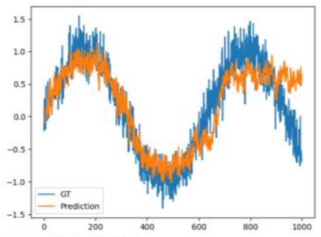
2. Outcome,

Fig1.



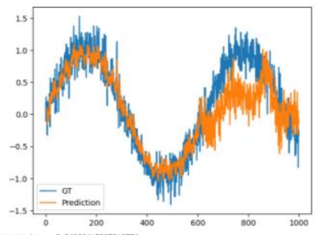
```
train loss: 0.05076783522963524
test loss: 0.3988398313522339
```

Fig2.



```
train loss: 0.06817246973514557
test loss: 0.25337496399879456
```

Fig3.



```
train loss: 0.04833417385816574
test loss: 0.2550044059753418
```

All of outputs are similar with above three figures. I couldn't find or failed to improve algorithm which fit the sin wave well.

3. Analytic Analysis

1) With many failures, I wonder why RNN cannot sin wave. First of all, I checked the reason why 1-step prediction works well. It was because 1-step shifted value is almost same with input values. So even RNN output the mean of 4 inputs, loss will go to a tiny value. However, in the case of multi-step prediction, if there are little error on prediction, then it became bigger and bigger through the predictions.

2) To control this problem, I've give the loss until 4-step(or more) prediction like below code. I thought that by this method, RNN will learn the more detailed data structure and will generate the sin wave well. But, it failed. (Actually, there was almost no difference with 1)'s result.)

```
_batch = x.shape[0]
inp = torch.zeros(_batch, tau+predict_num)
inp[:,:tau] = x.reshape(_batch, -1)
for i in range(predict_num):
    inp[:,i+tau:i+tau+1] = net(
        inp[:,i:i+tau].reshape(_batch, -1, 1)
    )
```

3) Therefore, I thought the gradient vanishing is the cause of no difference between 1) and 2). So I made the residual connection between input and output of net. But, it failed…

```
    def forward(self, inputs):
        ######################
        # Implement here
        #inputs += torch.randn(inputs.shape)*0.1
        outputs, hidden = self.rnn(inputs)
        outputs = outputs[:, -1, :]
        outputs = self.final(outputs)

        #outputs += inputs[:,-1,:]
```

4) I also tried to improve this algorithm with random functions. So I've add the random vectors to many variables (to input of network, to output of network, to prediction output, label value, etc.), but all of them failed. Fig2. and Fig3. are the results of these parts.

5) I've search the multi-step prediction with RNN on GitHub, and found an interesting repository(https://github.com/chickenbestlover/RNN-Time-series-Anomaly-Detection). This project success to train RNN wave signals, so I've followed this project's instructions but failed. The one I've not tried is the simplified professor forcing loss, which is the method that add the hidden state difference to loss. (there is some similar points with concept of knowledge distillation) But, in our project, since net's output format is fixed so couldn't tried that. But I'm not sure even this method works well.

6) To train RNN to learn sine wave, it is necessary that strong loss function to train wave not the mean value. But based on my knowledge, I don't know the way to train that…