




#27 - Implementing Vanilla RNN with Pytorch

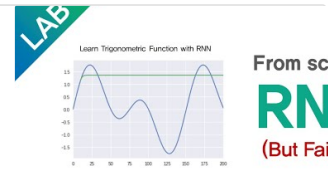
주제 : Implementing Vanilla RNN with Pytorch

링크 :

[#27.Lab] Implementing Vanilla RNN with Pytorch (Failed to learn) - 딥러닝 홀로서기

이번 라이브 코딩은 안타깝게도 좋지 못한 결과로 끝났습니다!! 다음 수업 앞부분에서 제대로 작동하는 코드를 가지고 돌아오겠습니다! 실습자료 링크 : <https://github.com/heartcored98/Standalone-DeepLearning/blob/master/Lec7/L...>

 <https://youtu.be/tlyzfYvMWE>



1. Implementing RNN

- Generating Dataset
 - Sin, Cos과 같은 Trigonometric을 RNN으로 학습이 가능한지를 구현
 - Lab.2에서 사용했던 Non-Linear 함수와 (=Sin, Cos, Log 함수) Toy Data Set(=연습용 데이터)을 활용할 예정

```
%matplotlib inline

import numpy as np
import matplotlib.pyplot as plt
import torch.optim as optim

# ===== Generating Dataset ===== #
num_data = 2400
t = np.linspace(0.0, 100.0, num_data)
y = np.sin(t) + np.sin(2*t) #+ np.sin(0.4*t)
e = np.random.normal(0, 0.1, num_data)
# y = y + e
```

- Model Define
 - Many to One의 모델
 - 파라미터 Linear Layer 3개 필요 (U, W, V)
 - 차원결정
 - X 차원 : 1차원 (값이 하나밖에 없으므로)
 - h : hidden vector의 dimension은 10차원정도로
 - U : 1차원 Input을 입력 받아서 10차원으로 변형시켜야 하므로 1 x 10

- W : hidden vector 입력 받아 hidden vector로 보내주는 역할이므로 10 x 10차원
- V : h가 10차원이었으므로, 10차원 → 1차원으로 10 x 1 차원
- Hidden State를 초기화
 - h_{t-1} 을 만들어 0이나 랜덤한 숫자로 초기화
 - batch size 갯수만큼 초기화 시켜줘야함
- Forward 함수 작성
 - $h_{t+2} = f(Ux_t + Wh_{t-1})$ 부분을 구현
 - $y_t = f(Vh_t)$ 부분도 구현
 - hidden dimension 업데이트는 모델 외부에서 수행

```
import torch
import torch.nn as nn

class RNN(nn.Module):
    def __init__(self, input_dim, output_dim, hid_dim, batch_size):
        super(RNN, self).__init__()

        self.input_dim = input_dim
        self.output_dim = output_dim
        self.hid_dim = hid_dim
        self.batch_size = batch_size

        self.u = nn.Linear(self.input_dim, self.hid_dim, bias=False)
        self.w = nn.Linear(self.hid_dim, self.hid_dim, bias=False)
        self.v = nn.Linear(self.hid_dim, self.output_dim, bias=False)
        self.act = nn.Tanh()

        self.hidden = self.init_hidden()

    def init_hidden(self, batch_size=None):
        if batch_size is None:
            batch_size = self.batch_size
        return torch.zeros(batch_size, self.hid_dim)

    def forward(self, x):
        h = self.act(self.u(x) + self.w(self.hidden))
        y = self.v(h)
        return y, h
```

- Data Set 이어서...
 - 10개를 보여주고 그 다음 값을 예측하려고 한다면, 11개의 연속된 Sequence를 샘플링해야함
 - LSTM이나 RNN을 사용할 때 차원은 보통 [Sequence Length, Batch_size, Input dimension]을 Input으로 입력받아 맞춰준다
 - 이유 : Sequence Length를 제일 앞에 두면 각 Step별로 for문을 돌려서, 각각의 time step에 대해서 batch_size와 input dimension으로 element를 가져오기 쉬워진다
 - x의 차원을 변경
 - [10, 2390, 1]로 변경하기 위하여 numpy의 swapaxes함수를 사용
 - Numpy Swapaxes함수로 첫 번째와 두 번째 Axis를 변경 → np.swapaxes(x, 0, 1)
 - Input Dimension을 위한 1을 하나 더 붙여주자

- numpy의 `expand_dims`를 사용하여 차원을 하나 추가
- 장점
 - Step별로 Model에 Forward 시킬 때, `x.shape`이 batch size, input dimension별로 잘 보이게 되고, 따라서 차원을 맞추기가 쉬워진다

```
seq_len = 10
X = []
y_true = []
for i in range(len(t)-seq_len):
    X.append(y[i:i+seq_len])
    y_true.append(y[i+seq_len])

X = np.array(X)
y_true = np.array(y_true)

X = np.swapaxes(X, 0, 1)
X = np.expand_dims(X, axis=2)

plt.plot(t, y)
import torch
import torch.nn as nn

class RNN(nn.Module):
    def __init__(self, input_dim, output_dim, hid_dim, batch_size):
        super(RNN, self).__init__()

        self.input_dim = input_dim
        self.output_dim = output_dim
        self.hid_dim = hid_dim
        self.batch_size = batch_size

        self.u = nn.Linear(self.input_dim, self.hid_dim, bias=False)
        self.w = nn.Linear(self.hid_dim, self.hid_dim, bias=False)
        self.v = nn.Linear(self.hid_dim, self.output_dim, bias=False)
        self.act = nn.Tanh()

        self.hidden = self.init_hidden()

    def init_hidden(self, batch_size=None):
        if batch_size is None:
            batch_size = self.batch_size
        return torch.zeros(batch_size, self.hid_dim)

    def forward(self, x):
        h = self.act(self.u(x) + self.w(self.hidden))
        y = self.v(h)
        return y, h
```

- Model Define2
 - RNN 모델 생성
 - hidden dimension이 잘 초기화 되었는지 확인 → 0벡터로 초기화
 - Loss함수 수행
 - Many to One의 모델에서 One은 Continuous Space의 값을 예측하는 것이기에 Regression task이며 따라서, Loss 함수는 MSE를 사용
 - Optimizer SGD 수행
 - Training
 - Train 함수 수행

- Model과 Optimizer를 Zero Gradient로 초기화
- (10, 2390, 1)차원인 X를, 각 Sequence마다(0~9) forward하여 y와 h를 리턴
- h를 업데이트
- 마지막 Sequence(Time Step)에서의 Y_Pred가 그 다음값이어야 하므로 loss 계산
 - loss 계산 시 차원 오류가 나올 경우, y_pred와 y_true를 출력해보면 된다
- backward 함수로 학습 수행
- optimizer도 Step 수행
- Epoch수만큼 for문 생성 : 2390개의 Sequence를 모두 학습하면 1 Epoch
 - 주의점 : Epoch을 돌때마다 항상 hidden state를 초기화해 주어야 한다
 - LSTM과 같이 Recurrent 모델을 수행할 때는 초기화를 매번 수행해야 함

```
model = RNN(1, 1, 50, 2390)
loss_fn = nn.MSELoss()
optimizer = optim.SGD(model.parameters(), lr=0.005)
epoch = 100

for i in range(epoch):
    model.train()
    model.zero_grad()
    optimizer.zero_grad()

    model.hidden = model.init_hidden()

    for x in X:
        x = torch.Tensor(x).float()
        y_true = torch.Tensor(y_true).float()

        y_pred, hidden = model(x)
        model.hidden = hidden

    loss = loss_fn(y_pred.view(-1), y_true.view(-1))
    loss.backward()
    optimizer.step()
    print(loss.item())
```

- Validation & Visualization
 - RNN이 Sequential한 정보를 학습할 수 있는지를 검증
 - 일반적으로 하는 정량적인 검증이 아닌 정성적인 검증 방식을 활용
 - Input X를 처음에 10개만 입력한 후 그 이후 과정을 스스로 예측할 수 있는지, 그리고 예측한 결과 값을 사용하여 계속해서 새롭게 예측이 가능한지를 검증
 - 첫 10개 Input 값을 활용하여 예측한 모형과, 실제 데이터를 비교하여 오차가 발생하면 얼마나 편중이 있는지를 검증하는 방식
 - 첫번째 Sequence가 맨 첫번째 값이기에 X[:, 0, :]을 통해 첫 번째 Sequence 값 10개를 가져옴
→ test_X
 - Dimension을 맞춰줘야 하기에, batch_size=1로 설정해주는 expand_dims 함수 수행
 - with torch.no_grad()문을 활용하여 모든 Gradient option을 False값으로 세팅하여 Gradient가 없는 상황이라는 것을 명시

- model.hidden은 계속 accumulated 될 수 있으므로, 항상 초기화하는 문장 삽입
- Sequence별로 생성된 10개 Input X값을 model을 돌려 y_pred, hidden값을 생성하여 hidden을 업데이트 하는 for문 생성
 - y_pred 값을 저장할 리스트 list_y_pred를 생성하여 for문을 돌며 생성된 y_pred의 마지막 값을 저장
- 옛날 X 9개와(test_X) 새로운 pred_y를 저장할 temp_x 리스트를 생성
 - temp_X += list(np.squeeze(test_X))[1:] 이미 첫번째 원소를 활용하여 Y값을 예측하였으므로 첫 원소를 제거하는 코드
- 마지막으로 y_pred를 Input값을 사용하여 마치 X처럼 사용해 주어야 하므로, 이미 만들어진 10개를 제외한 2390개 (여기서 마지막 Y를 뺀 2389개)의 Sequence를 돌리는 for문 생성
 - hidden을 초기화
 - pred_y를 도출하기 위해 temp_x값에서 하나의 값을 추출하여 temp2_X로 할당
 - temp2_x에서 Input x값을 꺼내어 model을 돌려 y_pred와 hidden을 업데이트하는 for문 생성
 - 주의할 점
 - y_pred를 이제 Input값으로 사용해야 하므로, X와 차원을 동일하게 맞춰야 함
 - print(y_pred.shape, x.shape)을 활용하여 두 변수의 차원 확인
 - 두 변수의 shape이 다른 것은(y_pred : [2390, 1], x : [1, 1]) Batch Size가 다르기 때문이므로 batch_size를 맞추기 위해 hidden dimension을 initialize할 때 batch size를 argument로 함께 넣어주는 코드를 Class안의 init_hidden 함수에 삽입
 - batch size가 None일땐 batch size는 원래 갖고 있던 batch size로 설정하는 코드
 - 아닐 경우 새로 받은 batch size로 return
 - 최종 산출된 y_pred는 list_y_pred 리스트에 저장
 - temp_X에 y_pred값을 저장하고, 이미 사용한 첫번째 원소를 제거
 - model에 y_pred를 Input값으로 넣어 그 다음 step의 y_pred를 예측하기 위함
- Visualization
 - 실제 y값을 actual value라고 설정하고, list_y_pred를 dot형태로 표현하는 predicted 그래프로 설정
 - 실제 값과 list_y_pred값을 그래프에 표현

```
test_X = np.expand_dims(X[:, 0, :], 1)

list_y_pred = []

model.eval()
```

```

with torch.no_grad():
    model.hidden = model.init_hidden(batch_size=1)

    for x in test_X:
        x = torch.Tensor(x).float()
        y_pred, hidden = model(x)
        model.hidden = hidden
        list_y_pred.append(y_pred.view(-1).item())

    temp_X = list()
    temp_X += list(np.squeeze(test_X)[1:])
    temp_X.append(y_pred.view(-1).item())

    for i in range(2389):
        model.hidden = model.init_hidden(batch_size=1)

        temp2_X = torch.unsqueeze(torch.unsqueeze(torch.Tensor(temp_X), 1), 1)

        for x in temp2_X:
            y_pred, hidden = model(x)
            model.hidden = hidden
            list_y_pred.append(y_pred.view(-1).item())

        temp_X.append(y_pred.view(-1).item())
        temp_X.pop(0)

plt.plot(y, label='actual value')
plt.plot(list(range(10, 2400)), list_y_pred, '*', label='predicted')
plt.xlim(0, 40)
plt.legend()

```