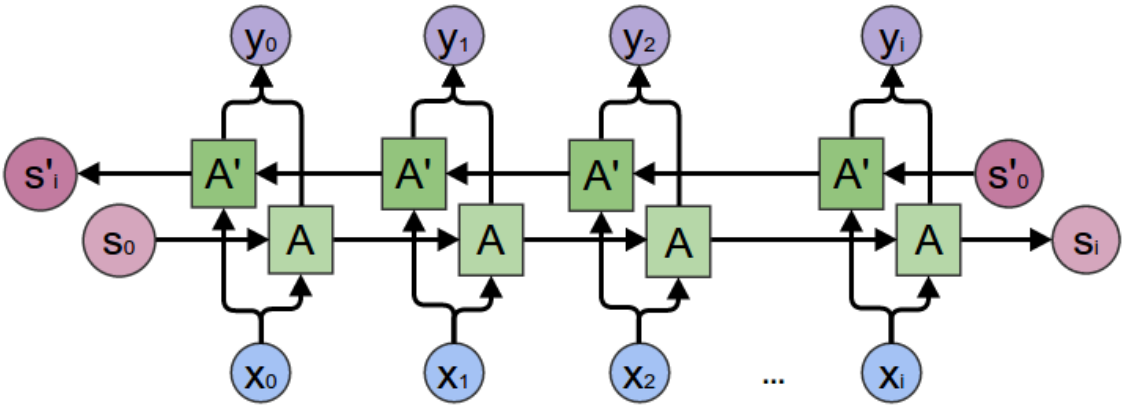
Bi-directional Recurrent Neural Network Example

Xây dựng mạng RNN 2 chiều với PyTorch

Tổng quan về BiRNN



Tổng quan về bộ dữ liệu MNIST

Ví dụ này sử dụng bộ dữ liệu về chữ số viết tay MNIST. Bộ dữ liệu chữa 60k mẫu cho huấn luyện và 10k mẫu cho kiểm thử.

Để phân loại hình ảnh sử dụng RNN, chúng ta sẽ coi mỗi hàng là 1 chuỗi pixels. Bởi vì kích thước ảnh là 28*28px, ta sẽ sử lý 28 chuỗi của 28 timesteps cho tất cả các sample.

In [ ]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
from torch.autograd import Variable
import numpy as np
```

In [ ]:
```
# Tham số của MNIST dataset
num_classes = 10 # tổng số class (0-9 digits).

# Tham số huấn luyện
learning_rate = 0.001
training_steps = 1000
batch_size = 32
display_step = 100

# Tham số của mạng
# Kích thước của ảnh là 28*28px, ta sẽ sử lý 28 chuỗi của 28 timesteps cho
num_input = 28 # số lượng chuỗi.
timesteps = 28 # timesteps.
num_units = 32 # số lượng neurons cho 1 layer LSTM.
```

In [ ]:
```
# Chuẩn bị dữ liệu
from tensorflow.keras.datasets import mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
# Chuyển đổi sang định dạng float32.
x_train, x_test = np.array(x_train, np.float32), np.array(x_test, np.float3
x_train, x_test = x_train.reshape([-1, 28, 28]), x_test.reshape([-1, 28, 28
# Chuẩn hóa ảnh từ from [0, 255] to [0, 1].
x_train, x_test = x_train / 255., x_test / 255.
x_train, x_test, y_train, y_test = torch.from_numpy(x_train), torch.from_nu
```

In [ ]:
```
trainloader = []
for (i,j) in zip(x_train, y_train):
    trainloader.append([i,j])
trainloader = torch.utils.data.DataLoader(trainloader, shuffle=True, batch_

testloader = []
for (i,j) in zip(x_test, y_test):
    testloader.append([i,j])
testloader = torch.utils.data.DataLoader(testloader, shuffle=True, batch_si
```

In [ ]:
```
# Khởi tạo mô hình BiRNN
class BiRNNModel(nn.Module):
    def __init__(self, input_dim, hidden_dim, layer_dim, output_dim):
        super(BiRNNModel, self).__init__()

        # Thiết lập số chiều của tầng ẩn
        self.hidden_dim = hidden_dim

        # Thiết lập số layers
        self.layer_dim = layer_dim

        # RNN
        self.rnn = nn.RNN(input_dim, hidden_dim, layer_dim, batch_first=Tru

        # Readout layer
        self.fc = nn.Linear(2*hidden_dim, output_dim)

    def forward(self, x):

        # Khởi tạo hidden state
        h0 = Variable(torch.zeros(2*self.layer_dim, x.size(0), self.hidden_
        # print(1)
        # One time step
        out, hn = self.rnn(x, h0)
        # print(2)
        # print(out.shape)
        out = self.fc(out[:, -1, :])
        # print(3)
        return out
```

In [ ]:
```
# Create RNN
input_dim = 28    # chiều của input
hidden_dim = 100  # chiều của hidden state
layer_dim = 1     # số tầng ẩn
output_dim = 10   # chiều của vector output

model = BiRNNModel(input_dim, hidden_dim, layer_dim, output_dim)

# Cross Entropy Loss
import torch.optim as optim
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
```

In [ ]:
```
for epoch in range(2):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data
        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()
        if i % 1000 == 999:    # print every 2000 mini-batches
            print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 2000))
            running_loss = 0.0

print('Finished Training')
```

```
[1,  1000] loss: 1.149
[1,  2000] loss: 1.125
[1,  3000] loss: 0.736
[2,  1000] loss: 0.361
[2,  2000] loss: 0.287
[2,  3000] loss: 0.230
Finished Training
```

In [ ]:
```
correct = 0
total = 0
# quá trình kiểm thử ko cần thiết phải tính gradients cho output
with torch.no_grad():
    for data in testloader:
        images, labels = data
        # calculate outputs by running images through the network
        outputs = model(images)
        # the class with the highest energy is what we choose as prediction
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print('Accuracy of the network on the 10000 test images: %d %%' % (
    100 * correct / total))
```

```
Accuracy of the network on the 10000 test images: 89 %
```

In [ ]:
```

```