Mini-assignment3

20200045 김건우

Part 1. Implement LSTM model in word-level language modelling

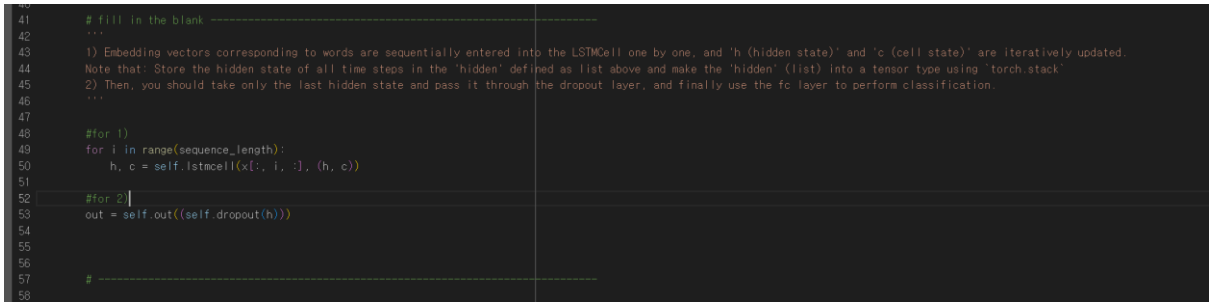
TODO : Fill in the blanks of the codes to perform the above process and write your own description of the source code in your report.

Implement MyLSTMCell class

```
class MyLSTMCell(nn.Module):
    def __init__(self, input_size, hidden_size):
        super(MyLSTMCell, self).__init__()
        # fill in the blank ------------------------------------

        self.input_size = input_size
        self.hidden_size = hidden_size

        self.x2h = nn.Linear(input_size, 4*hidden_size)
        self.h2h = nn.Linear(hidden_size, 4*hidden_size)
        # ------------------------------------------------------
        self.reset_parameters()

    def reset_parameters(self):
        stdv = 1.0 / math.sqrt(self.hidden_size) if self.hidden_size > 0 else 0
        for weight in self.parameters():
            nn.init.uniform_(weight, -stdv, stdv)

    def forward(self, input, hidden_states):
        # fill in the blank ------------------------------------
        h_in, c_in = hidden_states
        x = input.view(-1, input.size(1))

        gates = self.x2h(x) + self.h2h(h_in)

        input_gate, forget_gate, cell_gate, output_gate = gates.chunk(4, 1)

        input_gate = torch.sigmoid(input_gate) # sigmoid for input gate
        forget_gate = torch.sigmoid(forget_gate) # sigmoid for forget gate
        cell_gate = torch.tanh(cell_gate) # tanh for cell gate
        output_gate = torch.sigmoid(output_gate) # sigmoid for output gate
        c_out = torch.mul(c_in, forget_gate) + torch.mul(input_gate, cell_gate)
        h_out = torch.mul(output_gate, torch.tanh(c_out))

        return (h_out, c_out)



        # ------------------------------------------------------
```

LSTM cell의 구조에 맞게 class를 짜주었다. Input 파라미터인 input_size와 hidden_size를 선언해주고 cell로 들어오는 변수를 다뤄주기 위해 nn.Linear함수를 이용하여 x2h, h2h를 정의한다.

forward에서는 먼저 input 파라미터를 x로 받고, hidden_states파라미터를 h_in과 c_in으로 unpacking한다. 이후 위에서 정의한 x2h, h2h를 통해 gates를 만들어준다. 이 gates는 바로 아래에서 각각 input_gate, forget_gate, cell_gate, output_gate로 나뉜다. 각각의 gate에 대해 LSTM구조에 맞게 sigmoid 또는 tanh함수를 적용시킨 후, 최종적으로 c_out에는 c_in*gorget_gate + input_gate*cell_gate, h_out에는 outputgate*tanh(c_out)을 인가해준다. 최종적으로 h_out과 c_out를 반환한다.



```
        # fill in the blank ------------------------------------------------------------
        '''
        1) Embedding vectors corresponding to words are sequentially entered into the LSTMCell one by one, and 'h (hidden state)' and 'c (cell state)' are iteratively updated.
        Note that: Store the hidden state of all time steps in the 'hidden' defined as list above and make the 'hidden' (list) into a tensor type using `torch.stack`
        2) Then, you should take only the last hidden state and pass it through the dropout layer, and finally use the fc layer to perform classification.
        '''

        #for 1)
        for i in range(sequence_length):
            h, c = self.lstmcell(x[:, i, :], (h, c))

        #for 2)
        out = self.out((self.dropout(h)))



        # ------------------------------------------------------------------------
```

LSTM class의 forward 부분이다. 문제의 조건대로 for문 안에서 h, c를 업데이트해가며 순차적으로 h, c를 구해주었고 마지막 h에 대해 self.out((self.dropout(h)))으로 out을 정의한 후 return해주었다.

Training and Evaulation

TODO : Attach the training and evaluation results in your report.

TODO : Attach the training and evaluation results in your report.

```
#Model training
best_val_loss = None

for e in tqdm(range(1, epochs+1)):
    train(model, optimizer, train_iter, device, criterion )
    val_loss, val_accuracy = evaluate(model, val_iter, device, criterion)

    print("[Epoch: %d] val accuracy : %5.2f" % (e, val_accuracy))

    # Save best model
    if not best_val_loss or val_loss < best_val_loss:
        if not os.path.isdir("best_lstm_model"):
            os.makedirs("best_lstm_model")
        torch.save(model.state_dict(), './best_lstm_model/best_model.pt')
        best_val_loss = val_loss
```

```
 20%|██        | 1/5 [01:36<06:26, 96.71s/it][Epoch: 1] val accuracy : 51.13
 40%|████      | 2/5 [03:01<04:28, 89.56s/it][Epoch: 2] val accuracy : 53.95
 60%|██████    | 3/5 [04:27<02:55, 87.86s/it][Epoch: 3] val accuracy : 53.58
 80%|████████  | 4/5 [05:51<01:26, 86.52s/it][Epoch: 4] val accuracy : 66.22
100%|██████████| 5/5 [07:14<00:00, 86.92s/it][Epoch: 5] val accuracy : 74.13
```

```
#Model evaluation

model.load_state_dict(torch.load('./best_lstm_model/best_model.pt'))
test_loss, test_acc = evaluate(model, test_iter, device, criterion)
print('test loss: %5.2f | text accuracy: %5.2f' % (test_loss, test_acc))
```

```
test loss:  0.58 | text accuracy: 74.07
```

Task 2. Implement LSTM network with attention mechanism

TODO : Fill in the blanks of the codes to perform the above process and write your own description of the source code in your report.

```
# fill in the blank ------------------------------------------------------------
'''
You should add the following attention mechnism to the model in task 1 (refer to the above figure):
1) The last hidden state is given as a query, and based on this, the similarities with the hidden states in the previous step are m
2) Using a softmax function, the results are normalized and attention scores (α) for each hidden state are obtained.
3) Based on the attention scores (α) of each hidden state, we take a weighted average of hidden states (except for the last) and t
4) The context vector is concatenated with the last hidden state and prediction is performed based on it.
'''

for i in range(sequence_length):
  h, c = self.lstmcell(x[:, i, :], (h, c))
  if i == 0:
    hidden_h = h.unsqueeze(0)
  else:
    if i!=sequence_length:
        hidden_h = torch.cat((hidden_h, h.unsqueeze(0)))


# for 1)
scores = h * hidden_h

# for 2)

alpha = nn.Softmax(dim=-1)(scores)

# for 3)
context_2 = hidden_h*alpha
context = torch.sum(context_2, dim=0)

# for 4)

out = self.out(self.dropout(torch.cat((h, context), dim=1)))


# ------------------------------------------------------------------------------
```

MyLSTM_att class의 forward 부분이다. 먼저 마지막 h와 나머지 각 셀에서의 h 사이의 관계를 구하기 위해 먼저 for 문을 통해 h와 hidden_h를 구해준다. 그 이후 h와 hidden_h 의 dot product를 통해 각각의 scores를 구해주고, softmax 함수를 통해 알파를 구해준다. 이후 각 h에 대해 알파만큼의 가중치를 주어 더하여 context를 구해준다. 마지막으로 out으로는 마지막 h에 context를 합쳐(concatenate) 내보낸다.

TODO : Attach the training and evaluation results in your report.

```
#Model training
best_val_loss = None



for e in tqdm(range(1, epochs+1)):
    train(model, optimizer, train_iter, device, criterion )
    val_loss, val_accuracy = evaluate(model, val_iter, device, criterion)

    print("[Epoch: %d] val accuracy : %5.2f" % (e, val_accuracy))

    if not best_val_loss or val_loss < best_val_loss:
        if not os.path.isdir("best_lstm_model_with_attention"):
            os.makedirs("best_lstm_model_with_attention")
        torch.save(model.state_dict(), './best_lstm_model_with_attention/best_model.pt')
        best_val_loss = val_loss
```

```
 20%|██        | 1/5 [01:33<06:12, 93.24s/it][Epoch: 1] val accuracy : 78.52
 40%|████      | 2/5 [03:06<04:39, 93.00s/it][Epoch: 2] val accuracy : 83.92
 60%|██████    | 3/5 [04:37<03:04, 92.22s/it][Epoch: 3] val accuracy : 85.23
 80%|████████  | 4/5 [06:08<01:31, 91.95s/it][Epoch: 4] val accuracy : 85.73
100%|██████████| 5/5 [07:38<00:00, 91.61s/it][Epoch: 5] val accuracy : 85.60
```

```
#Model evaluation
model.load_state_dict(torch.load('./best_lstm_model_with_attention/best_model.pt'))
test_loss, test_acc = evaluate(model, test_iter, device, criterion)
print('test loss: %5.2f | text accuracy: %5.2f' % (test_loss, test_acc))
```

```
test loss:  0.01 | text accuracy: 84.50
```

Check the result with a test sample

```
import numpy as np

# Bring a sample from test dataset
test_example = vars(test_data[16])
print('Test example: ', test_example['text'])
```

```
Test example:  ['the', 'movie', 'is', 'okay.', 'meaning', 'that', 'i', "don't", 'regret', 'watching', 'it!', 'i', 'found', 'the', 'acting', 'purely', 'and', 'the', 'most', 'of', 'the', 'dialog', 'stupid', '("oh',
```

TODO : Attach the results in your report.

```
# Check the result with a test sample using our trained model

index_vector = []

for text in test_example['text']:
    index_vector.append(TEXT.vocab.stoi[text])

index_vector_numpy = np.array(index_vector)
index_vector_tensor = torch.from_numpy(index_vector_numpy)
index_vector_tensor = index_vector_tensor.view(1,index_vector_tensor.size(0))

prediction = model(index_vector_tensor.to(device)).max(1)[1].item() # make a prediction through our trained model
prediction = 'Positive review' if prediction==1 else 'Negative review'
Ground_truth = 'Positive review' if test_example['label']==1 else 'Negative review'

# Compare the result with groundtruth
print('The prediction of our trained model on the test example:', prediction)
print('The true label of the test example:', Ground_truth)
```

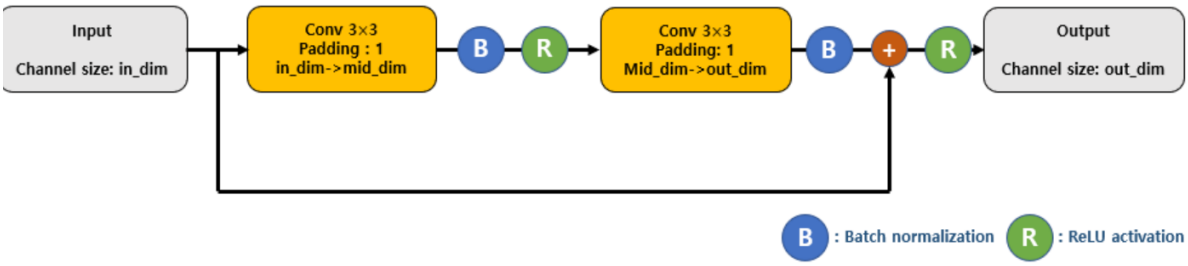
```
The prediction of our trained model on the test example: Negative review
The true label of the test example: Negative review
```

TODO : Describe the advantages of using the attention mechanism in the above task.

Task2에서 attention mechanism(마지막 h(쿼리)와 각 cell에서의 h 간의 관계성 계산)을 통해 각각의 값을 가중치로 하여 이전 h들에 대한 정보를 마지막h와 함께 전달하였다. 이는 vanishing gradient 문제를 해결할 수 있고, fc layer에 의미있는 정보들을 선택하여 전달함으로써 모델의 성능을 더 높일 수 있었다. 실제로 코드를 실행해본 결과 정확도는 약10% 증가하였다.

Part 2. CycleGAN implementation using PyTorch

TODO: Fill in the blanks of the codes below and write your own description of the source code in your report.




```python
class Residual_Block(nn.Module):
    def __init__(self, in_dim, mid_dim, out_dim):
        super(Residual_Block, self).__init__()
        ### Fill in the blank ------------------------------------


        self.conv1 = nn.Conv2d(in_dim, mid_dim, kernel_size=3, padding = 1)
        self.bn1 = nn.BatchNorm2d(mid_dim)
        self.conv2 = nn.Conv2d(mid_dim, out_dim, kernel_size=3, padding = 1)
        self.bn2 = nn.BatchNorm2d(out_dim)


        ### ------------------------------------------------------


    def forward(self, x):
        ### Fill in the blank ------------------------------------

        x2 = F.relu(self.bn1(self.conv1(x)))
        x3 = x + self.bn2(self.conv2(x2))

        out = F.relu(x3)

        ### ------------------------------------------------------
        return out
```

Class Residual_Block은 위 사진에 나와있는 구조를 토대로 구성해주었다. 2개의 convolution과정과 batch normalization과정을 각각 선언해주었고, forward에서는 input으로 받은 x에 대해 위에서 정의한 conv과 bn, 그리고 relu함수를 통해 output을 구해주었다. 이 때 마지막 relu이전에 처음 x를 더해주어야 하는 점에 유의하자.

3. Train and evaluate CycleGAN

TODO: Fill in the blanks of the codes below and write your own description of the source code in your report.

```python
def Train_D_ms(Dx, Dy, G, m_data, m_label):
    '''
    # parameter:
    Dx, Dy: Discriminator for source domain(X, MNIST) and target domain(Y, SVHN), respectively.
    G: Generator which transfers X to Y
    m_data: source domain data (i.e., MNIST real images)
    m_label: source domain label (from 0 to 9)

    Train two discriminators(Dx, Dy) in mnist-svhn cycle.
    Please refer to the above figure that represents the path from X to Y and equation (1),(3).
    Unlike the conventional GAN, here, the discriminator classifies 11 classes. You have to use 1) the original labels(from 0 to 9) when training the discriminator with real images, and 2) the fake label(10) when
    So, "torch.nn.functional.cross_entropy()" will be used for calculating losses.


    Return:
    D_ms_loss.item(): item of the loss
    '''

    # train mnist-svhn cycle

    # 1) Compute the loss of Dx (Dx_loss) with real images
    ### Fill in the blank ------------------------------------
    Dx_loss = nn.functional.cross_entropy(Dx(m_data), m_label)

    ### ------------------------------------------------------

    # 2) Compute the loss of Dy (Dy_loss) with fake images
    ### Fill in the blank ------------------------------------
    fake_label = m_label + 10
    fake_label = fake_label - m_label

    fake_image = G(m_data)

    Dy_loss = nn.functional.cross_entropy(Dy(fake_image), fake_label)



    ### ------------------------------------------------------
```

Dx_loss는 실제 label이 있는 실제 이미지에 대한 교차 엔트로피 손실이며, 이는 discriminator Dx의 실제 이미지에 대한 예측과 실제 label 사이의 차이이다. Dy_loss는 가짜 label(10)과 G에 의해 생성된 가짜 이미지에 대한 교차 엔트로피 손실로 discriminator Dy의 가짜 이미지에 대한 예측과 가짜 label 사이의 차이이다

```python
def Train_D_sm(Dx, Dy, F, s_data, s_label):
    '''
    # parameter:
    Dx, Dy: Discriminator for source domain(X, MNIST) and target domain(Y, SVHN), respectively.
    F: Generator which transfers Y to X
    s_data: target domain data (i.e., SVHN real images)
    s_label: target domain label (from 0 to 9)

    Train discriminators in svhn-mnist cycle.
    Please refer to the above figure that represents the path from Y to X and equation (1),(3).
    Unlike the conventional GAN, here, the discriminator classifies 11 classes. You have to use 1) the original labels(from 0 to 9) when training the discriminator with real images, and 2) the fake label(10) when
    So, "torch.nn.functional.cross_entropy()" will be used for calculating losses.

    Return:
    D_sm_loss.item(): item of the loss
    '''

    # train svhn-mnist cycle

    # 1) Compute the loss of Dy (Dy_loss) with real images
    ### Fill in the blank ------------------------------------

    Dy_loss = nn.functional.cross_entropy(Dy(s_data), s_label)

    ### ------------------------------------------------------

    # 2) Compute the loss of Dx (Dx_loss) with fake images
    ### Fill in the blank ------------------------------------

    fake_label = s_label + 10
    fake_label = fake_label - s_label

    fake_image = F(s_data)

    Dx_loss = nn.functional.cross_entropy(Dx(fake_image), fake_label)

    ### ------------------------------------------------------
```

이전에 구한 Train_D_ms와 거의 비슷한 방식으로 Train_D_sm을 구현하였다.

Dy_loss는 실제 label이 있는 실제 이미지에 대한 교차 엔트로피 손실이며, 이는 discriminator Dy의 실제 이미지에 대한 예측과 실제 label 사이의 차이이다. Dx_loss는 가짜 label(10)과 G에 의해

생성된 가짜 이미지에 대한 교차 엔트로피 손실로 discriminator Dx의 가짜 이미지에 대한 예측과 가짜 label 사이의 차이이다.

```python
# train mnist-svhn-mnist cycle
def Train_G_msm(Dy, G, F, m_data, m_label):
    '''
    # parameter:
    Dy: Discriminator for target domain(Y, SVHN).
    G, F: Generator for X to Y and Y to X.
    m_data: source domain data (i.e., MNIST real images)
    m_label: source domain label (from 0 to 9)

    Train mnist-svhn-mnist cycle.
    Please refer to the above figure that represents the path from X to Y to X and equation (1),(5).
    Note that you have to use original labels(from 0 to 9) when training the generator.
    So, "torch.nn.functional.cross_entropy()" will be used for calculating losses.
    Also, this function has to include cycle consistency loss(i.e. Lcyc) with mean squared error loss.

    Return:
    G_msm_loss.item(): item of the loss
    '''

    # Compute the generator G loss (G_loss) with fake images
    ### Fill in the blank ------------------------------------

    fake_image = G(m_data)

    G_loss = nn.functional.cross_entropy(Dy(fake_image), m_label)



    ### ------------------------------------------------------

    # Compute the cycle consistency loss (Lcyc)
    ### Fill in the blank ------------------------------------

    Lcyc = nn.functional.mse_loss(F(fake_image), m_data)

    ### ------------------------------------------------------
```

G_loss는 실제 label로 식별된 가짜 이미지에 대한 교차 엔트로피 손실이다. 이는 G에 의해 생성된 이미지가 discriminator Dy를 속일 수 있다는 걸 의미한다. Lcyc는 실제 데이터와 G,F를 거치고 온 데이터 사이의 mean sqared error loss이다. (F와 G^{-1}(G의 역함수)가 얼마나 비슷한지를 나타낸다.)

```python
# train svhn-mnist-svhn cycle
def Train_G_sms(Dx, G, F, s_data, s_label):
    '''
    # parameter:
    Dx: Discriminator for source domain(X, MNIST).
    G, F: Generator for X to Y and Y to X.
    s_data: target domain data (i.e., SVHN real images)
    s_label: target domain label (from 0 to 9)

    Train svhn-mnist-svhn cycle.
    Please refer to the above figure that represents the path from Y to X to Y and equation (3),(5).
    Note that you have to use original labels(from 0 to 9) when training the generator.
    So, "torch.nn.functional.cross_entropy()" will be used for calculating losses.
    Also, this function has to include cycle consistency loss(i.e. Lcyc) with mean squared error loss.

    Return:
    G_sms_loss.item(): item of the loss
    '''

    # Compute the generator F loss (F_loss) with fake images
    ### Fill in the blank ------------------------------------

    fake_image = F(s_data)

    F_loss = nn.functional.cross_entropy(Dx(fake_image), m_label)


    ### ------------------------------------------------------

    # Compute the cycle consistency loss (Lcyc)
    ### Fill in the blank ------------------------------------

    Lcyc = nn.functional.mse_loss(G(fake_image), s_data)

    ### ------------------------------------------------------

```

Trian_G_sms 역시 앞에서 구한 Train_G_msm과 거의 비슷한 구조로 구현하였다. F_loss는 실제 label로 식별된 가짜 이미지에 대한 교차 엔트로피 손실이다. 이는 F에 의해 생성된 이미지가 discriminator Dx를 속일 수 있다는 걸 의미한다. Lcyc는 실제 데이터와 F,G를 거치고 온 데이터 사이의 mean sqared error loss이다. (G와 F^{-1}(F의 역함수)가 얼마나 비슷한지를 나타낸다.)

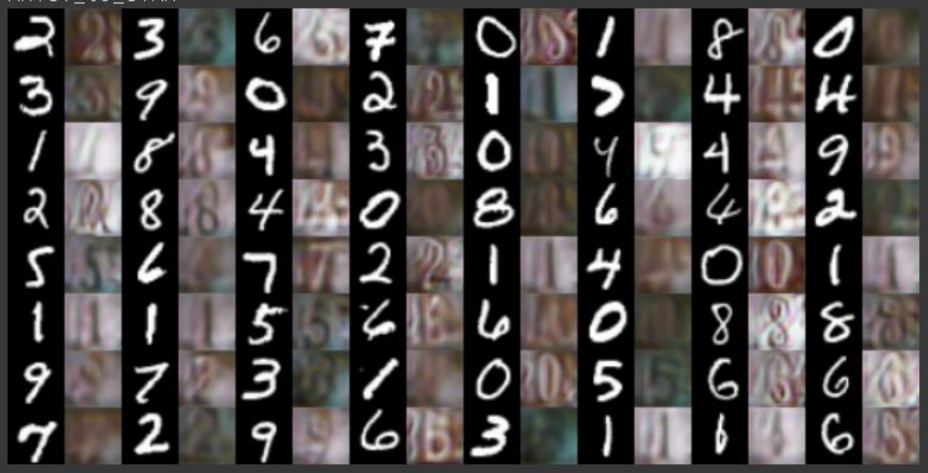
TODO : Attach the training and evaluation results in your report.

```python
D_ms_losses, D_sm_losses, G_msm_losses, G_sms_losses = [], [], [], []

for step in range(train_iters + 1):
    # reset data_iter for each epoch
    if (step + 1) % iter_per_epoch == 0:
        mnist_iter = iter(mnist_loader)
        svhn_iter = iter(svhn_loader)

    # load svhn and mnist dataset
    s_data, s_label = next(svhn_iter)
    s_data, s_label = s_data.cuda(), s_label.cuda().long().squeeze()
    m_data, m_label = next(mnist_iter)
    m_data, m_label = m_data.cuda(), m_label.cuda()

    D_ms_losses.append(Train_D_ms(Dx, Dy, G, m_data, m_label))
    D_sm_losses.append(Train_D_sm(Dx, Dy, F, s_data, s_label))
    G_msm_losses.append(Train_G_msm(Dy, G, F, m_data, m_label))
    G_sms_losses.append(Train_G_sms(Dx, G, F, s_data, s_label))

    # print the log info
    if (step + 1) % 100 == 0:
        show_log(step, train_iters, D_ms_losses, D_sm_losses, G_msm_losses, G_sms_losses)

    # save the sampled images
    if (step + 1) == 1000:
        save_img(step, fixed_mnist, fixed_svhn, G, F, m_data, s_data, sample_path)
```

```
Step [100/1000], D_ms_loss: 0.6116, D_sm_loss: 2.1786, G_msm_loss: 16.6341, G_sms_loss: 12.3640
Step [200/1000], D_ms_loss: 0.4466, D_sm_loss: 1.7314, G_msm_loss: 13.2845, G_sms_loss: 12.3778
Step [300/1000], D_ms_loss: 0.4009, D_sm_loss: 1.5231, G_msm_loss: 11.7384, G_sms_loss: 12.7736
Step [400/1000], D_ms_loss: 0.3797, D_sm_loss: 1.3927, G_msm_loss: 10.8698, G_sms_loss: 13.1157
Step [500/1000], D_ms_loss: 0.3823, D_sm_loss: 1.3042, G_msm_loss: 10.3866, G_sms_loss: 13.2154
Step [600/1000], D_ms_loss: 0.3630, D_sm_loss: 1.2172, G_msm_loss: 10.0949, G_sms_loss: 13.3480
Step [700/1000], D_ms_loss: 0.3547, D_sm_loss: 1.1567, G_msm_loss: 9.9307, G_sms_loss: 13.3693
Step [800/1000], D_ms_loss: 0.3525, D_sm_loss: 1.1121, G_msm_loss: 9.8428, G_sms_loss: 13.4349
Step [900/1000], D_ms_loss: 0.3366, D_sm_loss: 1.0641, G_msm_loss: 9.7645, G_sms_loss: 13.6881
Step [1000/1000], D_ms_loss: 0.3229, D_sm_loss: 1.0229, G_msm_loss: 9.7766, G_sms_loss: 13.9650
saved ./samples/sample-1000-m-s.png
saved ./samples/sample-1000-s-m.png
```

```
1 print('MNIST_to_SVHN')
2 cv2_imshow(result_m_s)
```

```
MNIST_to_SVHN
```



```
[ ]  1 print('SVHN_to_MNIST')
     2 cv2_imshow(result_s_m)
```

```
SVHN_to_MNIST
```

TODO : If the discriminator classifies only two classes (whether real or fake) as in the conventional GAN, what problem will arise?

Discriminator가 2개의 class에 대해서만 분류한다면, complexity가 커지고 따라서 training time도 길어질 것으로 예상된다. 10개의 class가 있는 경우 판별자는 숫자 사진을 보고 어느 숫자와 가장 비슷한 지 판별하고 그리고 이후 해당 사진이 real인지 fake인지 판별한다. 그러나 주어진 문제 상황에서는 숫자에 대한 labeling 없이 이미지만 보고 해당 사진이 real인지 fake인지 판별해야 하는데 이 과정은 label에 대한 정보가 없는 셈이므로 더 복잡할 것이다.