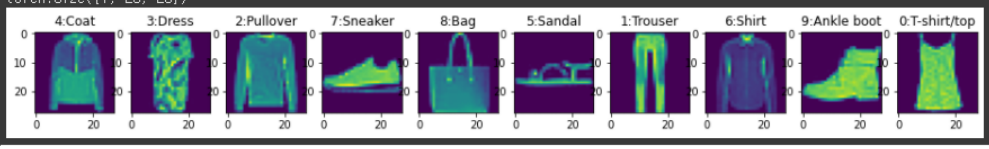
20140269 고혁훈 3차과제

- 데이터 탐색

```
import matplotlib.pyplot as plt
import time
train_dataset = torchvision.datasets.FashionMNIST(root='./datasets',
                                        train=True,
                                        transform=transforms.ToTensor(),
                                        download=True)
tr_loader = torch.utils.data.DataLoader(dataset=train_dataset,
                                          batch_size=2,
                                          shuffle=True)
check = {i:True for i in range(10)}
CLASSES = {
    0: 'T-shirt/top',
    1: 'Trouser',
    2: 'Pullover',
    3: 'Dress',
    4: 'Coat',
    5: 'Sandal',
    6: 'Shirt',
    7: 'Sneaker',
    8: 'Bag',
    9: 'Ankle boot'
}
plt.figure(figsize=(16,16))
plt.subplot(1, 10, 1)               # nrows=2, ncols=1, index=1
num = 0
for i in tr_loader:
    image = i[0][0]
    label = i[1][0].item()
    if check[label] == True:
      num += 1
      plt.subplot(1, 10, num)
      plt.imshow(i[0][0].view(28,28))
      print(i[0][0].shape)
      plt.title(str(label) + ':' + CLASSES[label])
      check[label] = False
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:31: MatplotlibDeprecationWarning: Adding an axes using the same arguments as a previou
torch.Size([1, 28, 28])
torch.Size([1, 28, 28])
torch.Size([1, 28, 28])
torch.Size([1, 28, 28])
torch.Size([1, 28, 28])
torch.Size([1, 28, 28])
torch.Size([1, 28, 28])
torch.Size([1, 28, 28])
torch.Size([1, 28, 28])
torch.Size([1, 28, 28])
```



- valid, train, test data 개수(valid 0.1)

```
trtransform = transforms.Compose([
                    transforms.ToTensor(),
                    transforms.Normalize((0.5), (0.497))])
transform = transforms.Compose([
                    transforms.ToTensor(),
                    transforms.Normalize((0.5), (0.497))])
train_dataset = torchvision.datasets.FashionMNIST(root='./datasets',
                                        train=True,
                                        transform=trtransform,
                                        download=True)
trdata, valdata = torch.utils.data.random_split(train_dataset, (54000,6000))

ttdata = torchvision.datasets.FashionMNIST(root='./datasets',
                                        train=False,
                                        transform=transform,
                                        download=True)

print(len(trdata),len(valdata),len(ttdata))
```

```
54000 6000 10000
```

- 모델 설명

1. normalize 적용

```
trtransform = transforms.Compose([
                    transforms.ToTensor(),
                    transforms.Normalize((0.5), (0.497))])
transform = transforms.Compose([
                    transforms.ToTensor(),
                    transforms.Normalize((0.5), (0.497))])
train_dataset = torchvision.datasets.FashionMNIST(root='./datasets',
                                        train=True,
                                        transform=trtransform,
                                        download=True)
trdata, valdata = torch.utils.data.random_split(train_dataset, (54000,6000))

ttdata = torchvision.datasets.FashionMNIST(root='./datasets',
                                        train=False,
                                        transform=transform,
                                        download=True)
```

2. 모델 detail - rnn

```
class MY_RNN(nn.Module):
  def __init__(self, input_size, hidden_size, num_layers, num_classes=10,bi=True):
    super(MY_RNN, self).__init__()
    self.hidden_size = hidden_size
    self.num_layers = num_layers
    self.bi = bi
    self.rnn = nn.RNN(input_size, hidden_size, num_layers, batch_first=True,bidirectional=True) #1,28,28,50을 원하지 않음
    if bi == True:
      self.fc = nn.Linear(2*sequence_length*hidden_size, 512)
    else:
      self.fc = nn.Linear(sequence_length*hidden_size, 512)
    self.drops = nn.Dropout(0.2)
    self.fc2 = nn.Linear(512,num_classes)
  def forward(self, x):
    out, _ = self.rnn(x) # output: tensor [batch_size, seq_length, hidden_size]
    if self.bi:
      out = out.reshape(-1,2*sequence_length*self.hidden_size)
    else:
      out = out.reshape(-1,sequence_length*self.hidden_size)
    out = F.relu(self.drops(self.fc(out)))
    out = self.fc2(out)
    return out

input_size = 28
num_layers = 1
hidden_dim = 128

model = MY_RNN(input_size, hidden_dim, num_layers, 10).to(device)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
```

: rnn layer 1개, hidden_dim 128, lr =0.001, sequence_length = 28, classes = 10, bidirectional add

: 2 fully connected layer with 1 dropout

: input -> bidirectional rnn -> fc -> dropout -> Relu -> fc2 -> 결과

3. 모델 detail - gru

```
    class MY_GRU(nn.Module):
      def __init__(self, input_size, hidden_size, num_layers, num_classes=10):
        super(MY_GRU, self).__init__()
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.gru = nn.GRU(input_size, hidden_size, num_layers, batch_first=True) #1,28,28,50을 원하지 않음
        # self.fc = nn.Linear(2*sequence_length*hidden_size, 512)
        self.fc = nn.Linear(hidden_size, 10)
        # self.drops = nn.Dropout(0.1)
        # self.fc2 = nn.Linear(512, 10)
      def forward(self, x):
        out, _  = self.gru(x) # output: tensor [batch_size, seq_length, hidden_size]
        out = out[:,-1,:]
        # out = out.reshape(-1,2*sequence_length*self.hidden_size)

        # out = F.relu(self.drops(self.fc(out)))
        # out = self.fc2(out)
        # out = self.fc(out)

        return self.fc(out)

      def init_weights(self, init_gain=0.02):
        def init_func(m):
            classname = m.__class__.__name__
            if hasattr(m, 'weight') and (classname.find('Conv') != -1 or classname.find('Linear') != -1):
                nn.init.xavier_normal_(m.weight.data, gain=init_gain)
        self.apply(init_func)


    num_layers = 2
    hidden_dim = 128
    model2 = MY_GRU(input_size, hidden_dim, num_layers, 10).to(device)
    criterion2 = nn.CrossEntropyLoss()
    optimizer2 = torch.optim.Adam(model2.parameters(), lr=0.001)
```

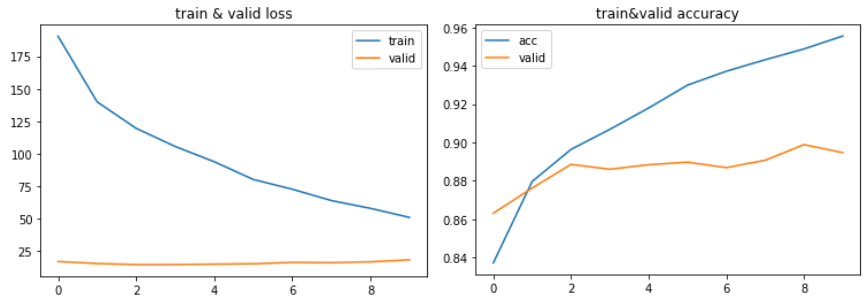
: layer 2개, hidden_dim 128, lr=0.001, sequence_length = 28, classes = 10

: 1 fully connected layer with 1 dropout

: input -> gru -> gru -> fc -> 결과

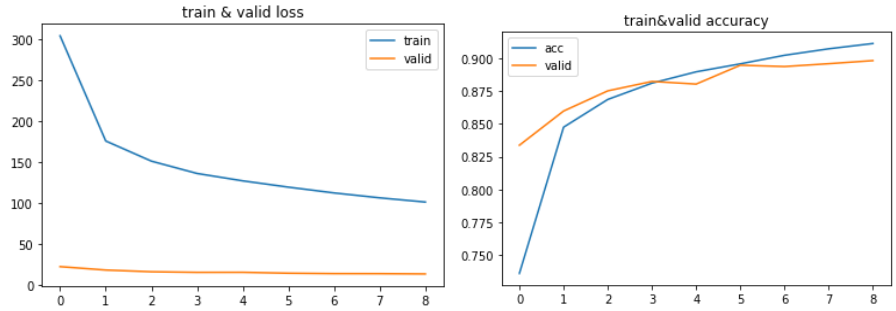
- 2개 모델의 learning curve(결과)

Rnn(with epoch 10)



: rnn은 train loss 및 accuracy는 지속적으로 변하지만, valid loss 및 accuracy는 4 epoch부터 같은 수준에 머무르고 loss는 오히려 증가한다. Acc의 경우 8 epoch에서 잠시 커졌다가 이후에 다시 떨어져서, loss를 기준으로 학습 중단 시점을 채택했다. 따라서 overfitting을 방지하기 위해서 epoch 4에서 early stopping을 했다.

Gru(with epoch 10)



: gru는 rnn에 비해 수렴속도가 느렸으며, epoch가 9 이후로 실험했을 때 loss는 더 줄지 않았고, 성능은 90%를 넘지 않았다. 따라서 overfitting을 방지하기 위해 9까지 학습을 진행하였다.

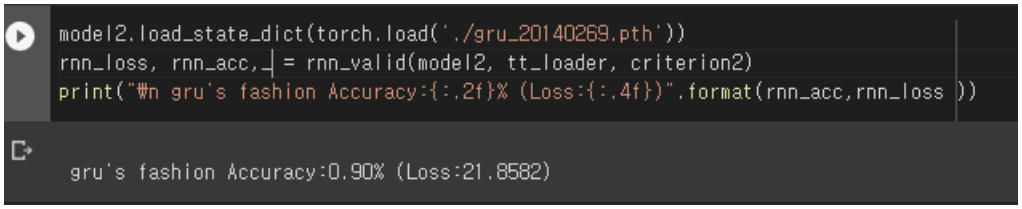
- 결과

1. rnn -> 88% on test dataset

```
model.load_state_dict(torch.load('./rnn_epoch_4.pth'))
rnn_loss, rnn_acc,_ = rnn_valid(model, tt_loader, criterion)
print("\n rnn's fashion Accuracy:{:.2f}% (Loss:{:.4f})".format(rnn_acc,rnn_loss))
```

```
 rnn's fashion Accuracy:0.88% (Loss:27.2451)
```

2. gru -> 90% on test set



3. 교훈

: rnn,gru의 경우 layer를 무작정 늘린다고 성능이 좋아지는 것이 아니었다. 오히려 layer를 줄여야 overfitting이 방지되고, 성능도 개선되는 것을 알 수 있다. 특히 rnn의 경우 layer 및 hidden 노드가 늘면 아무리 weight init을 잘하고 dropout을 넣어도 성능이 매우 안 좋았다. Gru의 경우 비교적 gradient vanishing이 덜했지만, 마찬가지로 layer를 늘릴수록 overfitting 현상만 일어났다. 따라서 rnn,gru는 처음에는 적게 layer로 시작해서 점진적으로 늘리면서 data에 맞는 모델 채택이 중요함을 깨달았다.