13주차 DCGAN

모두의 딥러닝 참고

https://thebook.io/080324/part05/ch19/03-02/

DCGAN_MNIST

한쪽으로 쏠리지 않도록 표준화

discriminator 학습 시킬 때는 참/거짓 데이터를 주고 가중치를 업데이트를 해야되지만, 이 discriminator가 gan 안에서 generator와 같이 학습할 때는 가중치가 고정되어 있어야 합니다. 그래서 gan 모델을 생성하기 전에 discriminator.trainable = False으로 설정하긴 했는데… 여기서 헷갈리기 시작했습니다.

- discriminator.trainable = False 으로 하면 gan에서는 고정되겠지만 discriminator을 학습할 때도 가중치가 고정되는 것이 아니야?
- 앞에서 생성한 discriminator과 gan에 삽입할 때의 discriminator는 다른 객체인가?

등등으로 생각을 했었는데, 알고보니 compile() 함수가 호출될 때 trainable 속성이 모델에 적용되더라구요. 즉 다음과 같습니다.

- discriminator을 생성한 뒤 compile() 하면 trainable = True로 컴파일 됨
- discriminator.trainable = False으로 적용하면 일단 trainable 속성만 비활성화된 상태임
- gan 모델에 discriminator가 삽입됨
- gan.compile() 하면 gan 모델 안에서 discriminator의 가중치가 업데이트 되지 않음
- gan.compile()과 discriminator.compile()은 별개이고, discriminator.compile()가 다시 호출 되지 않았으므로, discriminator 모델에서의 trainable 속성은 True임
- 여기서 하나 알 수 있는 것은 discriminator이라는 네트워크는 discriminator 모델과 gan 모델에 둘 다 사용되고 가중치도 공유되나 discriminator 모델에서는 가중치 갱신이 일어나고, gan 모델에서는 가중치 갱신이 일어나지 않음
- gan 모델에서의 discriminator 네트워크는 단순 가중치를 가진 네트워크로만 받아들이고 discriminator 모델에 적용된 compile()은 아무 영향을 주지 않음. 즉 gan 모델은 따로 complie()을 해야 함

배성호교수님의 '간은 로스일 뿐이야'라는 말씀을 이제야 이해한 듯 합니다.

Generator

```
generator=Sequential()
generator.add(Dense(128*7*7,input_dim=100,activation=LeakyReLu(0.2))) #flatten #128은 임의로 정한 노드 숫자. 100은 100차원 랜덤 벡터 의미.
#7*7의 이미지의 최초의 크기, 이미지의 크기를 점점 늘린다음에 conv를 지나치게 하는 게 DCGAN 의 특징
generator.add(BatchNormalization())
generator.add(Reshape(7,7,128))
```

```
generator.add(UpSampling2D())

generator.add(Conv2D(64,kernel_size=5,padding='same'))
generator.add(BatchNormalizaion())
generator.add(Activation(LeakyReLU(0.2)))
generator.add(UpSmapling2D())
generator.add(Conv2D(1,kernel_size=5,padding='same',activation='tanh'))
#tanh : 지금 불러올 이 데이터의 픽셀 값을 -1~1 사이의 값으로 지정
```

Model: "sequential_1"

```
Layer (type) Output Shape Param #
dense_1 (Dense)              (None, 6272)              633472
batch_normalization_2 (Batc  (None, 6272)              25088
hNormalization)
activation_1 (Activation)    (None, 6272)              0
reshape_1 (Reshape)          (None, 7, 7, 128)         0
up_sampling2d_2 (UpSampling  (None, 14, 14, 128)       0
2D)
conv2d_2 (Conv2D)            (None, 14, 14, 64)        204864
batch_normalization_3 (Batc  (None, 14, 14, 64)        256
hNormalization)
activation_2 (Activation)    (None, 14, 14, 64)        0
up_sampling2d_3 (UpSampling  (None, 28, 28, 64)        0
2D)
conv2d_3 (Conv2D)            (None, 28, 28, 1)         1601
=================================================================
Total params: 865,281
Trainable params: 852,609
Non-trainable params: 12,672
```

Discriminator

```
discriminator=Sequential()
discriminator.add(Conv2D(64,kernel_size=5,strides=2,input_shape=(28,28,1),padding='sam
e'))
discriminator.add(Acitvation(LeakyReLU(0.2)))
discriminator.Dropout(0.3)) #or BatchNormalization
discriminator.add(Conv2D(128,kernel_size=5,strides=2,padding='same'))
discriminator.add(Activation(LeakyReLU(0.2)))
discriminator.add(Dropout(0.3))

#일치하는지 체크해야하니까 Dense(1,activation='sigmoid'(binary))
discriminator.add(Flatten())
discriminator.add(Dense(1,activation='sigmoid'))
discriminator.compile(loss='binary_crossentropy',optimizer='adam')
```

```
discriminator.trainable =False #가중치를 공유하는데 GAN 학습할때.... 업데이트 되지않도록 !!gan.tra
in_on_batch(noise, true)전에도 바꿔줘야함
```

Model: "sequential_2"

Layer (type) Output Shape Param

conv2d_4 (Conv2D) (None, 14, 14, 64) 1664

activation_3 (Activation) (None, 14, 14, 64) 0

dropout (Dropout) (None, 14, 14, 64) 0

conv2d_5 (Conv2D) (None, 7, 7, 128) 204928

activation_4 (Activation) (None, 7, 7, 128) 0

dropout_1 (Dropout) (None, 7, 7, 128) 0

flatten (Flatten) (None, 6272) 0

dense_2 (Dense) (None, 1) 6273

===

Total params: 212,865
Trainable params: 0
Non-trainable params: 212,865

생성자와 판별자의 모델을 연결 DCGAN 모델 using MNIST

```
ginput=Input(shape=(100,))
dis_output=discriminator(generator(ginput))
gan= Model(ginput,dis_output)
gan.compile(loss='binary_crossentropy',optimizer='adam')
gan.summary()
```

Model: "model"

Layer (type) Output Shape Param

input_1 (InputLayer) [(None, 100)] 0

sequential_1 (Sequential) (None, 28, 28, 1) 865281

sequential_2 (Sequential) (None, 1) 212865

==========

Total params: 1,078,146
Trainable params: 852,609
Non-trainable params: 225,537

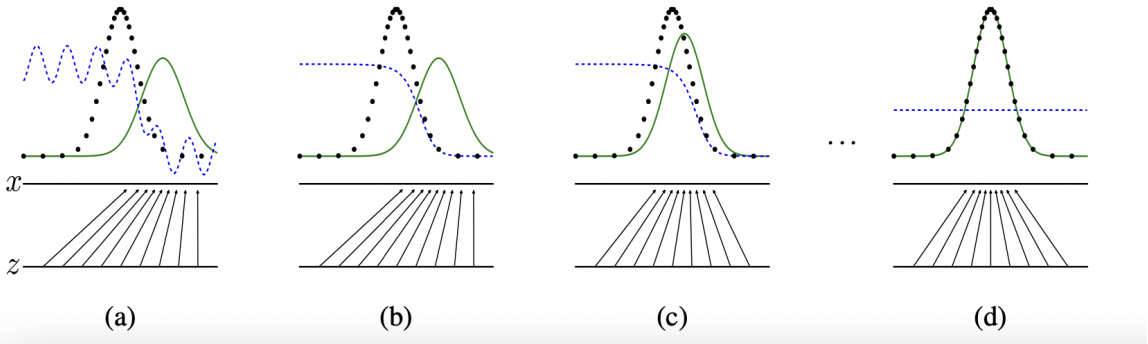
신경망을 실행시키는 함수 구축

train_on_batch ⇒ 고정된 batch_size를 쓰지 않고 weights를 명시적으로 업데이트

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]. \quad (1)$$

In the next section, we present a theoretical analysis of adversarial nets, essentially showing that the training criterion allows one to recover the data generating distribution as G and D are given enough capacity, i.e., in the non-parametric limit. See Figure 1 for a less formal, more pedagogical explanation of the approach. In practice, we must implement the game using an iterative, numerical approach. Optimizing D to completion in the inner loop of training is computationally prohibitive, and on finite datasets would result in overfitting. Instead, we alternate between k steps of optimizing D and one step of optimizing G. This results in D being maintained near its optimal solution, so long as G changes slowly enough. This strategy is analogous to the way that SML/PCD [31, 29] training maintains samples from a Markov chain from one learning step to the next in order to avoid burning in a Markov chain as part of the inner loop of learning. The procedure is formally presented in Algorithm 1.

In practice, equation 1 may not provide sufficient gradient for G to learn well. Early in learning, when G is poor, D can reject samples with high confidence because they are clearly different from the training data. In this case, $\log(1 - D(G(\boldsymbol{z})))$ saturates. Rather than training G to minimize $\log(1 - D(G(\boldsymbol{z})))$ we can train G to maximize $\log D(G(\boldsymbol{z}))$. This objective function results in the same fixed point of the dynamics of G and D but provides much stronger gradients early in learning.



```
def gan_train(epoch,batch_size,saving_interval):
  (X_train,_),(_,_) = mnist.load_data()
  X_train =X_train.reshape(X_train.shape[0],28,28,1).astype('float32')
  X_train=(X_train-127.5)/127.5
  #0~255의 값으로 되어 있는 픽셀 값을 -1~1 사이의 값으로 바꾸려면 현재의 픽셀 값에서 127.5를 뺀 후 127.
5

  true=np.ones((batch_size,1))
  fake=np.zeros((batch_size,0))

  for i in range(epoch):

      discriminator.trainable=True
      #실제 데이터를 판별자에 입력
      #batch_size 단위로 실제 데이터를 판별자에 학습
      (0부터 X_train 개수 사이의 숫자를 랜덤하게 선택해 batch_size만큼 반복해서 가져옴 )
      idx=np.random.randint(0,X_train.shape[0],batch_size)
      imgs=X_train[idx] #고정되어있는 batch_size를 사용하지 않기때문에 ->  idx random으로 추출
      d_loss_real=discriminator.train_on_batch(imgs,true)

      noise=np.random.normal(0,1,(batch_size,100)) #가상의 이미지 생성 / batch_size만큼 100
열 뽑기
      gen_imgs=generator.predict(noise)
      d_loss_fake=discrimnator.train_on_batch(gen_imgs,fake)

      #판별자의 오차 => 진위 판단후 오차 갱신
      d_loss = 0.5 * np.add(d_loss_real,d_loss_fake)
      print('epoch:%d' % i, ' d_loss_real:%.4f' % d_loss_real, ' d_loss_fake:%.4f' % d_
loss_fake)
      discriminator.trainable=False
      #GAN 학습
      g_loss=gan.train_on_batch(noise,true)

      print('epoch:%d' % i, ' d_loss_real:%.4f' % d_loss_real, ' d_loss_fake:%.4f' % d_
loss_fake)

      if i % saving_interval==0 :
          noise = np.random.normal(0,1,(25,100))
          gen_imgs=generator.predict(noise)
          # Rescale images 0 - 1
          gen_imgs= 0.5* gen_imgs +0.5

          fix,axs=plt.subplots(5,5)
          count =0
          for j in range(5):
              for k in range(5):
                  axs[j,k].imshow(gen_imgs[count,:,:,0],axs[j,k].axis['off'])
                  count+=1
          fig.savefig("./gan_mnist_%d.png" % i)
```

```
#def gan_train(epoch,batch_size,saving_interval): 200번마다 저장 epoch 2001 / batch_size
 32
gen_train(2001,32,200)
```

실행 결과

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 100)]             0
_________________________________________________________________
sequential_2 (Sequential)    (None, 28, 28, 1)         865281
_________________________________________________________________
sequential_3 (Sequential)    (None, 1)                 212865
=================================================================
Total params: 1,078,146
Trainable params: 852,609
Non-trainable params: 225,537
_________________________________________________________________
epoch:0  d_loss:0.7238  g_loss:0.5319
... (중략) ...
epoch:2000 d_loss:0.4667 g_loss:2.1844
```