فاطمه توکلی_۴۰۰۱۳۱۰۱۶

تمرین شماره ۷ شبکه‌های عصبی

۱. شبکه آموزشی مولد بر اساس دو شبکه discriminator و generator می باشد که بایکدیگر در رقابت می باشند شبکه generator از یک تصویر با ابعاد ثابت نویز شروع میشود و در هربار تلاش و گرفتن نتیجه کار خود تلاش به بهبود نتایج و نزدیک شدن به داده واقعی میکند در حالیکه discriminator تلاش دارد که تمایز بین داده واقعی و جعلی را تشخیص دهد.

تابع هدف دراین بازی به صورت یک بازی minmax تعریف شده است که در آن شبکه discriminator میخواهد دقت خود را max کند تا بتواند عکس های واقعی را از عکس تقلبی تشخیص دهد در حال که شبکه generator سعی میکند با تولید نمونه های غیر واقعی، discriminator را به خطا بیاندازد تا آن ها به عنوان عکس واقعی بشناسد. این کار باید به صورتی انجام شود که پارامتر ها در جهتی تغییر کنند که امید ریاضی خطای discriminator افزایش یابد.

۲. کانولوشن معکوس یک عملیات ریاضی است، هدف استفاده از کانولوشن معکوس این است که دقیقاً عکس عمل کانولوشن را انجام دهیم؛ یعنی با داشتن نقشه‌ی ویژگی بتوانیم عکس ورودی را بازسازی کنیم. به این کار Up-sampling گفته می‌شود.

۳. در قسمت تابع فعالسازی برای شبکه مولد میتوان از نوعی از تابع RelU استفاده کرد اما این تابع برای خروجی نمیباشد که مقادیر زیر صفر را نیز تولید میکند به نام Leaky ReLU استفاده میشود. همچنین در خروجی مولد نیز از تابع tanh استفاده میشود تا مولد و جداکننده در رنج ۱ تا ۱- قرار گیرند.

برای شبکه جداکننده از تابع LeakyReLU و در لایه خروجی آن از تابع sigmoid استفاده می‌کنیم که تعیین کند خروجی جعلی یا واقعی می‌باشد.

در قسمت تابع هزینه از cross entropy استفاده میکنیم که اندازه گیری تفاوت بین دو توزیع احتمال برای یک متغیر تصادفی معین یا مجموعه ای از رویدادها است.

۴.

پیاده سازی FCGAN:

ابتدا دیتاست را دانلود کرده تغییر سایز داده و به تصاویر سیاه سفید تغییر داده سپس بین مقادیر -۱ تا ۱ نرمالایز کردیم وبه batch های ۲۵۶ تایی تغییر دادیم .

سپس مدل generator که لایه های کاملا متصل دارد را به صورت زیر داریم:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 256)               25856

 leaky_re_lu (LeakyReLU)     (None, 256)               0

 dense_1 (Dense)             (None, 512)               131584

 leaky_re_lu_1 (LeakyReLU)   (None, 512)               0

 dense_2 (Dense)             (None, 1024)              525312

 leaky_re_lu_2 (LeakyReLU)   (None, 1024)              0

 dense_3 (Dense)             (None, 3136)              3214400

=================================================================
Total params: 3,897,152
Trainable params: 3,897,152
Non-trainable params: 0
```

و مدل discriminator را به صورت عکس generator تعریف کرده و انرا در زیر داریم:

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_4 (Dense)             (None, 1024)              3212288

 leaky_re_lu_3 (LeakyReLU)   (None, 1024)              0

 dropout (Dropout)           (None, 1024)              0

 dense_5 (Dense)             (None, 512)               524800

 leaky_re_lu_4 (LeakyReLU)   (None, 512)               0

 dropout_1 (Dropout)         (None, 512)               0

 dense_6 (Dense)             (None, 256)               131328

 leaky_re_lu_5 (LeakyReLU)   (None, 256)               0

 dropout_2 (Dropout)         (None, 256)               0

 dense_7 (Dense)             (None, 1)                 257

=================================================================
Total params: 3,868,673
Trainable params: 3,868,673
Non-trainable params: 0
```

داده نویزی ایجاد کرده و آن را ابتدا به generator می دهیم :

```
generator = make_generator_model()

noise = tf.random.normal([1, latent_dim])
generated_image = generator(noise, training=False)
```

و سپس به discriminator میدهیم برای تصمیم گیری:

```
discriminator = make_discriminator_model()
decision = discriminator(generated_image)
print (decision)
```

```
tf.Tensor([[-0.00537442]], shape=(1, 1), dtype=float32)
```

تابع خطا هرکدام جداگانه به صورت زیر تعریف می‌شود خروجی تمام داده های واقعی ۱ و داده های جعلی ۰ میشود و در نهایتcross entropy محاسبه شده و جمع زده مشیود برای خطای کل:

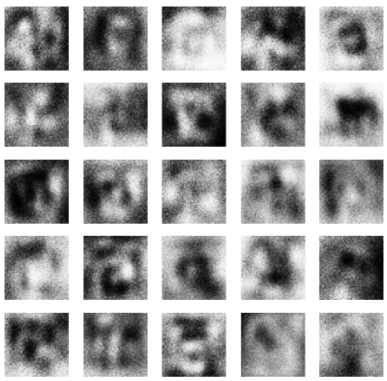
```
def discriminator_loss(real_output, fake_output):
    real_loss = cross_entropy(tf.ones_like(real_output), real_output)
    fake_loss = cross_entropy(tf.zeros_like(fake_output), fake_output)
    total_loss = real_loss + fake_loss
    return total_loss
```

```
def generator_loss(fake_output):
    return cross_entropy(tf.ones_like(fake_output), fake_output)
```

مراحل یادگیری مدل به صورت زیر می‌باشد ابتدا نویز تولید می‌شود و سپس با استفاده از discriminator خروجی تصویر و خروجی نویز گرفته میشود و خطای داده نویزی تولید شده توسط generator_loss گرفته شده همچنین خطای داده واقعی و جعلی discriminator_loss نیز گرفته میشود و از مشتق انها برای اموزش استفاده میشود.

```python
def train_step(images):
    noise = tf.random.normal([BATCH_SIZE, noise_dim])

    with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
      generated_images = generator(noise, training=True)

      real_output = discriminator(images, training=True)
      fake_output = discriminator(generated_images, training=True)

      gen_loss = generator_loss(fake_output)
      disc_loss = discriminator_loss(real_output, fake_output)

    gradients_of_generator = gen_tape.gradient(gen_loss, generator.trainable_variables)
    gradients_of_discriminator = disc_tape.gradient(disc_loss, discriminator.trainable_variables)

    generator_optimizer.apply_gradients(zip(gradients_of_generator, generator.trainable_variables))
    discriminator_optimizer.apply_gradients(zip(gradients_of_discriminator, discriminator.trainable_variables))
```

نتیجه آموزش :



Epoch 182

پیاده سازی DCGAN:

ابتدا داده ها را دانلود کرده و سپس تغییر سایز می دهیم به ابعاد (96,96,3) ان ها را نرمالایز کرده به بازه بین -۱ تا ۱ و سپس ترتیب چینش داده ها را تغییر می دهیم و به دسته های ۱۲۸ تایی در می‌اوریم:

مدل generator که از لایه های Conv2D_transpose تشکیل شده به صورت زیر تعریف شده است :

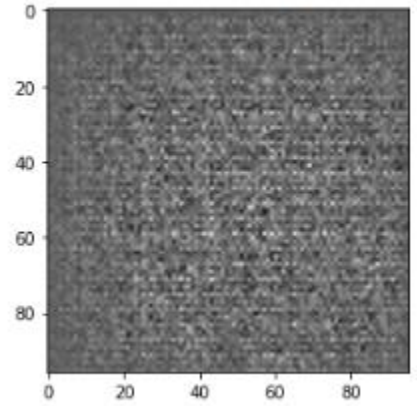
Layer (type)	Output Shape	Param #
dense (Dense)	(None, 2304)	297216
reshape (Reshape)	(None, 6, 6, 64)	0
conv2d_transpose (Conv2DTranspose)	(None, 6, 6, 512)	819200
batch_normalization (BatchNormalization)	(None, 6, 6, 512)	2048
re_lu (ReLU)	(None, 6, 6, 512)	0
dropout (Dropout)	(None, 6, 6, 512)	0
conv2d_transpose_1 (Conv2DTranspose)	(None, 12, 12, 256)	3276800
batch_normalization_1 (BatchNormalization)	(None, 12, 12, 256)	1024
re_lu_1 (ReLU)	(None, 12, 12, 256)	0
dropout_1 (Dropout)	(None, 12, 12, 256)	0
conv2d_transpose_2 (Conv2DTranspose)	(None, 24, 24, 128)	819200
batch_normalization_2 (BatchNormalization)	(None, 24, 24, 128)	512
re_lu_2 (ReLU)	(None, 24, 24, 128)	0
conv2d_transpose_3 (Conv2DTranspose)	(None, 48, 48, 64)	204800
batch_normalization_3 (BatchNormalization)	(None, 48, 48, 64)	256
re_lu_3 (ReLU)	(None, 48, 48, 64)	0
conv2d_transpose_4 (Conv2DTranspose)	(None, 96, 96, 32)	51200
batch_normalization_4 (BatchNormalization)	(None, 96, 96, 32)	128
re_lu_4 (ReLU)	(None, 96, 96, 32)	0
dense_1 (Dense)	(None, 96, 96, 3)	99

مدل discriminator را داریم:

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 48, 48, 64)        3072

 leaky_re_lu (LeakyReLU)     (None, 48, 48, 64)        0

 conv2d_1 (Conv2D)           (None, 24, 24, 64)        65536

 batch_normalization_5 (Batc  (None, 24, 24, 64)       256
 hNormalization)

 leaky_re_lu_1 (LeakyReLU)   (None, 24, 24, 64)        0

 conv2d_2 (Conv2D)           (None, 12, 12, 128)       131072

 batch_normalization_6 (Batc  (None, 12, 12, 128)      512
 hNormalization)

 leaky_re_lu_2 (LeakyReLU)   (None, 12, 12, 128)       0

 conv2d_3 (Conv2D)           (None, 6, 6, 256)         524288

 batch_normalization_7 (Batc  (None, 6, 6, 256)        1024
 hNormalization)

 leaky_re_lu_3 (LeakyReLU)   (None, 6, 6, 256)         0

 flatten (Flatten)           (None, 9216)              0

 dense_2 (Dense)             (None, 1)                 9217

=================================================================
Total params: 734,977
Trainable params: 734,081
Non-trainable params: 896
```

داده نویزی تولید کرده و ان را به generator میدهیم:

```
(1, 96, 96, 3)
(1, 128) -0.043539733 1.0046048
```



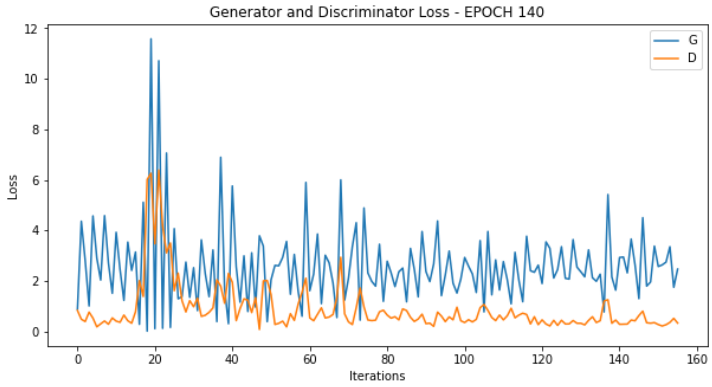
سپس به discriminator برای تصمیم گیری میدهیم:

```
tf.Tensor([[0.49999946]], shape=(1, 1), dtype=float32)
```

تابع هزینه و عملیات آموزش مانند FCGAN می‌باشد:

نتیجه کار به صورت زیر می‌باشد :

تابع هزینه مدل generator و discriminator به صورت زیر می‌باشد:



یکی از مشکلات رایج در آموزش شبکه های مولد تقابلی همگرایی سریع discriminator نسبت به generator می باشد. اگر discriminator را ثابت فرض کنیم؛ در صورت کاهش تعداد لایه های generator، پیچیدگی آن نسبت به discriminator کمتر شده و سریع می تواند فرآیند یادگیری را طی کند و فرصت کافی برای آموزش دارد چرا که discriminator پیچیده بوده و آموزش آن نیازمند زمان می باشد و generator از این زمان استفاده کرده و خطای خود را کاهش می دهد اما در مقابل مسلما میتوان انتظار داشت کیفیت تصاویر تولیدی نیز دچار کاهش خواهد شد.

۵.

۱. میتوان در هر تکرار مولد را آموزش بیشتری داد تا وزن ها چندین بار اصلاح شده و مقادیر بهتری بگیرند. یا میتوان لایه های مولد را کاهش داد تا زودتر به آموزشش همگرا شود و یا برعکس میتوان لایه های متمایز کننده را افزایش داد تا سرعت بخش discriminator کاهش یابد

۲. دسته جدا از تصاویر واقعی و جعلی، مدل تفکیک‌کننده با استفاده از نزول گرادیان تصادفی با مینی‌بچ‌ها آموزش داده می‌شود. بهترین روش این است که به‌جای ترکیب تصاویر واقعی و جعلی در یک دسته واحد، تشخیص‌دهنده را با دسته‌های جداگانه تصاویر واقعی و جعلی به‌روزرسانی کرد.

۳. از Label Smoothing استفاده کرد،استفاده از برچسب کلاس ۱ برای نمایش تصاویر واقعی و برچسب کلاس ۰ برای نمایش تصاویر جعلی هنگام آموزش مدل تفکیک کننده معمول است.اینها برچسب های سخت نامیده می شوند، زیرا مقادیر برچسب دقیق یا واضح هستند.استفاده از برچسب های نرم، مانند مقادیر کمی بیشتر یا کمتر از ۱.۰ یا کمی بیشتر از ۰.۰ به ترتیب برای تصاویر واقعی و جعلی، که در آن تغییرات برای هر تصویر تصادفی است.

۴. از برچسب های نویزی استفاده کرد برچسب‌هایی که هنگام آموزش مدل تمایز استفاده می‌شوند، همیشه صحیح هستند.این بدان معناست که تصاویر جعلی همیشه با کلاس ۰ و تصاویر واقعی همیشه با کلاس ۱ برچسب گذاری می شوند.توصیه می شود در این برچسب ها خطاهایی وارد شود که برخی از تصاویر جعلی به عنوان واقعی و برخی از تصاویر واقعی به عنوان جعلی مشخص شده اند.اگر از دسته‌های جداگانه برای به‌روزرسانی تشخیص‌دهنده تصاویر واقعی و جعلی استفاده می‌کنید، این ممکن است به معنای افزودن تصادفی برخی از تصاویر جعلی به دسته‌ای از تصاویر واقعی یا افزودن تصادفی برخی از تصاویر واقعی به دسته‌ای از تصاویر جعلی باشد.اگر تمایزکننده را با مجموعه‌ای از تصاویر واقعی و جعلی به‌روزرسانی می‌کنید، ممکن است به‌طور تصادفی برچسب‌ها را روی برخی از تصاویر تغییر دهید.

۶.

- در صورتی که نویز بسیار زیاد باشد مدل نمیتواند به خوبی یادیگری خود را انجام دهد و باعث از بین رفتن تعادل بین generator و discriminator میشود و در صورتی که نویز خیلی کم باشد نیز مدل به سرعت همگرا و شبکه فرصت یادگیری نخواهد داشت.
- با افزایش نویز، تصاویر تولید شده نیز دارای کمی نویز می‌باشند درحالیکه کم بودن نویز نیز باعث میشود تصاویر به خوبی یادگرفته نشوند وکیفیت پایین باشد.

۷. همانطور که از نتایج مشخص است خروجی شبکه DCGAN بهتر میباشد.تصاویر تولید شده با استفاده از معماری مدل DCGAN به طور قابل توجهی بهتر نسبت به تصاویر تولید شده با استفاده از شبکه تماما متصل چندلایه GAN بودند. این را می توان به صورت زیر درک کرد: شبکه های عصبی کانولوشن، به طور کلی، مناطق همبستگی را در یک تصویر پیدا می کنند، یعنی به دنبال همبستگی های فضایی می گردند. این بدان معناست که یک DCGAN احتمالاً برای داده های تصویر/ویدئو مناسب تر است.