یادگیری عمیق

مینی پروژه چهارم

نام استاد : دکتر ستوده

نام دانشجو : حمزه قائدی

شماره دانشجویی: 9831419

مقدمه

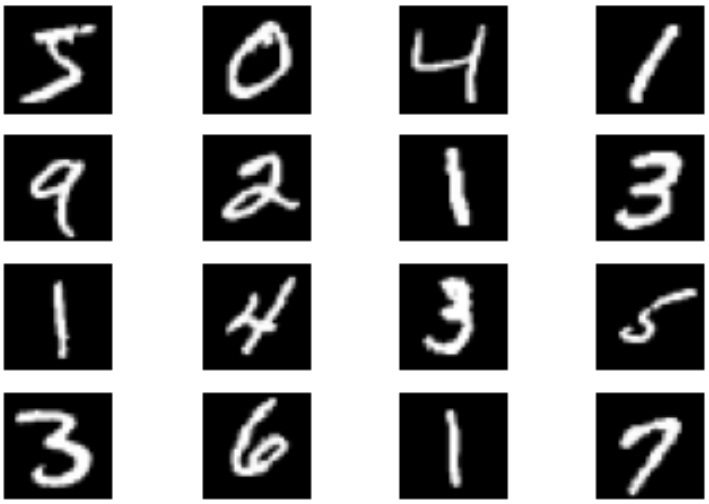
شبکه های GAN یکی از زیباترین ایده های مطرح شده در حوزه یادگیری ماشین در ده سال اخیر میباشد

این شـــبکه ها ـبا یادگیری توزیع احتمال داده های ورودی، میتوانند از توزیعی که یادگرفته لند نمونه بســـازند (حرف G در GAN اشـــاره به generative بودن این شبکه ها دارد) ایده کلی آنها بدین صورت است که یک شبکه عصبی به نام Generator با دریافت نویز، نمونه هایی از جنس ورودی میسازد و این نمونه های ساخته شده را به شبکه discriminator میدهد که این شبکه نیز میتواند نمونه واقعی و جعلی را از هم تفکیک کند. در ابتدای فرایند یادگیری، نمونه های Generator به راحتی توسط discriminator رد میشود اما رفته رفته شبکه Generator قادر به ساخت نمونه هایی بسیار شبیه به توزیع داده هاست این فرایند تا آنجا ادامه میابد که شبکه discriminator قادر به تشخیص نمونه های ساخته شده توسط generator نباشد در این مرحله شبکه Generator توزیع احتمال داده های ورودی را یادگرفته است. از منظری دیگر Generatorرا میتوان بدیل Encoder در شبکه های Autoencoder دانست همانگونه که encoder ها داده های ورودی را به یک فضای معمولا با بعد کمتر نگاشت میکنند (که به این فضا latent space میگویند) Generator ها نیز داده های ورودی را به یک فضا با بعد کمتر(که هم بعد با نویز دریافتی Generator است) نگاشت میکنند. در نهایت و پس از فرایند یادگیری (یا همان پیدا کردن نگاشت بهینه) میتوان با ارائه یک بردار از این فضا به Generator نمونه ای شبیه به داده های اولیه استفاده شده برای آموزش، دریافت کرد.

هدف

در این پروژه یک شـــبکه GAN طراحی میشـــود که با یادگرفتن توزیع داده های MNIST میتواند عکســـهایی قلابی شـــبیه به اعداد دستنویس موجود در MNIST بسازد.چون در ساختار این شبکه از لایه های کانولوشنی استفاده شده است به این ساختار DCGAN میگویند.

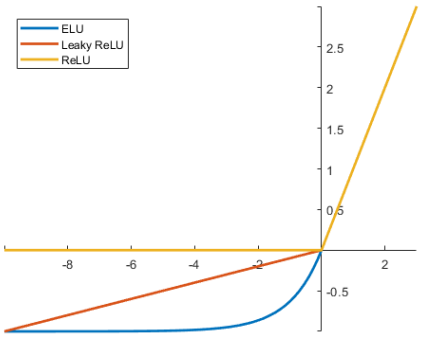
در شکل زیر تعدادی از نمونه های دیتابیس MNIST قابل مشاهده است:



نکات کلی:

در این پروژه از تابع LeakyReLU به عنوان فعالساز استفاده شده است.ضابطه این تابع به صورت زیر است:

$$y = \max(0.1x, x)$$



بهینه ساز مورد استفاده در این پروژه Adam است که رابطه آن به صورت زیر است. این بهینه ساز با ترکیب ایده های بهنیه سازهای momentum و RMSprop هم مشکل سقوط در نقاط پست[1] و هم مسئله نوسان را کاهش داده است

[1] Saddle point

$$For\ each\ Parameter\ w^j$$
$$(j\ subscript\ dropped\ for\ clarity)$$

$$\nu_t = \beta_1 * \nu_{t-1} - (1-\beta_1) * g_t$$
$$s_t = \beta_2 * s_{t-1} - (1-\beta_2) * g_t^2$$
$$\Delta\omega_t = -\eta \frac{\nu_t}{\sqrt{s_t + \epsilon}} * g_t$$
$$\omega_{t+1} = \omega_t + \Delta\omega_t$$

$\eta : Initial\ Learning\ rate$
$g_t : Gradient\ at\ time\ t\ along\ \omega^j$
$\nu_t : Exponential\ Average\ of\ gradients\ along\ \omega_j$
$s_t : Exponential\ Average\ of\ squares\ of\ gradients\ along\ \omega_j$
$\beta_1, \beta_2 : Hyperparameters$

BatchNormalization: این لایه ورودی خود را به صورت z-score (یعنی با واریانس یک و میانگین صفر) به عنوان خروجی تحویل میدهد

Conv2DTranspose: این لایه عملیاتی شبیه به کانولوشون را انجام میدهد و خروجی آن تصویری با ابعاد بزرگتر از تصویر ورودی است.

همچنین از BinaryCrossEntropy به عنوان تابع هزینه استفاده شده است

*) فرایند یادگیری این شبکه بسیار زمان گیر بوده و نتایج نهایی پس از 50 تا epoch به دست آمده است احتمالا صرف وقت و افزایش تعداد epoch ها باعث بهبود تصاویر ساخته شده خواهد شد اما به هرصورت Generator قادر نخواهد بود به تمامی اعضای فضای نویز پاسخ قابل تشخیصی بدهد لذا به منظور مشخص شدن کارایی Generator ،پس از آموزش، تعداد زیادی ورودی نویز به آن اعمال شده و بهترین خروجی ها انتخاب شده است..

بخش اول:طراحی شبکه Discriminator

فرایند یادگیری شبکه های Generator و Discriminator به صورت همزمان و در یک حلقه اتفاق می افتد در طی این حلقه Discriminator باید نحوه تشخیص تصاویر جعلی از واقعی را یادبگیرد و Generator نیز باید ساختن تصاویری جعلی ولی مشابه تصاویر دیتاست را بیاموزد. بدیهی است که وظیفه Generator سختر و پیچیده تر است و لذا زمان بیشتری برای یادگیری میطلبد. درحالیکه وظیفه Discriminator به راحتی و حتی توسط یک شبکه عصبی ساده قابل فراگیریست. لذا Discriminator باید به گونه ای طراحی شود که در طی این فرایند طولانی دچار Overfitting نشود بدین منظور در ساختار آن از تعداد قابل ملاحظه ای لایه Dropout استفاده شده است. این لایه با حذف تعدادی از واحدهای لایه قبل از خود، باعث تحریک شبکه به یادگیری بیشتر میشود در عین حال overfitting را نیز کاهش میدهد. از طرفی چون لایه خروجی شبکه discriminator دارای فعال ساز سیگموید است، امکان اشباع شدن و لذا محو شدگی گرادیان وجود دارد که این امر نیز میتواند ناشی از بزرگ شدن اعداد در فرایند محاسبه گرادیان باشد به منظور کاهش این پدیده ، از لایه های BatchNormalization استفاده شده است. این لایه خروجی های لایه قبل از خود را (با فرض گوسی بودن) به حالت نرمال استاندارد (یعنی توزیع نرمال با میانگین صفر و واریانس یک) تبدیل میکند که این امر باعث کوچک شدن اعداد و لذا کاهش پدیده محو شدگی گرادیان میشود.

ساختار شبکه Discriminator:

این شـبکه یک عکس 28*28 را دریافت کرده و با عبور آن از لایه های کانولوشـنی و فعالسـاز های غیر خطی و نهایتا یک لایه با فعال ساز سیگموید، خروجی صفر یا یک را تولید کرده که جعلی یا حقیقی بودن عکس دریافتی را مشخص میکند

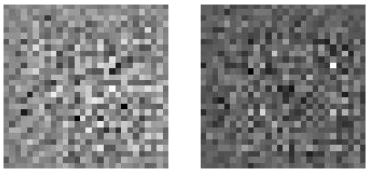
ساختار ای شبکه به صورت زیر است:

```
Model: "discriminator"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 28, 28, 1)         0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 14, 14, 64)        1664
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 14, 14, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 14, 14, 64)        0
_________________________________________________________________
batch_normalization_1 (Batch (None, 14, 14, 64)        256
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 7, 7, 128)         204928
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 7, 7, 128)         0
_________________________________________________________________
dropout_2 (Dropout)          (None, 7, 7, 128)         0
_________________________________________________________________
```

بخش دوم :طراحی شبکه شبکه Generator

شـبکه Generator، نویز یونیفرم 256 بعدی را به عنوان ورودی دریافت کرده و خروجی آن باید عکسـی مشـابه به عکسـهای دیتاسـت مذکور باشـد. این شـبکه باید نویز ورودی را که یک بردار سـتونی 256 بعدی است به یک تصویر 28*28 تبدیل کند بدین منظور ابتدا نویز ورودی از یک لایه Dense با 7*7*256 واحدی عبور داده شده و خروجی این لایه از یک لایه فعالساز LeakyReLU عبور داده میشود و سپس ابعاد ورودی تغییر میکند از ای مرحله‌به بعد از لایه Conv2DTranspose‌به منظور Upsampling اسـتفاده شـده اسـت همچنین لایه های Dropoutو BatchNormalization نیز به منظور جلوگیری از محو شـدن گرادیان و اشـباع شـدن تابع خروجی به تعداد زیاد اسـتفاده شـده است!....

خروجی Generator قبل از آموزش به صورت زیر است...ساختار شبکه نیز در زیر قابل مشاهده است.



```
Model: "generator"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         (None, 100)               0
_________________________________________________________________
dense_2 (Dense)              (None, 12544)             1266944
_________________________________________________________________
dropout_3 (Dropout)          (None, 12544)             0
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 12544)             0
_________________________________________________________________
batch_normalization_3 (Batch (None, 12544)             50176
_________________________________________________________________
reshape_1 (Reshape)          (None, 7, 7, 256)         0
_________________________________________________________________
conv2d_transpose_1 (Conv2DTr (None, 7, 7, 128)         819200
_________________________________________________________________
batch_normalization_4 (Batch (None, 7, 7, 128)         512
_________________________________________________________________
dropout_4 (Dropout)          (None, 7, 7, 128)         0
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 7, 7, 128)         0
_________________________________________________________________
conv2d_transpose_2 (Conv2DTr (None, 14, 14, 128)       409600
_________________________________________________________________
batch_normalization_5 (Batch (None, 14, 14, 128)       512
_________________________________________________________________
dropout_5 (Dropout)          (None, 14, 14, 128)       0
_________________________________________________________________
leaky_re_lu_5 (LeakyReLU)    (None, 14, 14, 128)       0
_________________________________________________________________
conv2d_transpose_3 (Conv2DTr (None, 14, 14, 128)       409600
_________________________________________________________________
batch_normalization_6 (Batch (None, 14, 14, 128)       512
_________________________________________________________________
dropout_6 (Dropout)          (None, 14, 14, 128)       0
_________________________________________________________________
leaky_re_lu_6 (LeakyReLU)    (None, 14, 14, 128)       0
_________________________________________________________________
conv2d_transpose_4 (Conv2DTr (None, 28, 28, 1)         6272
=================================================================
Total params: 2,963,328
Trainable params: 2,937,472
Non-trainable params: 25,856
```

بخش سوم : ساختار شبکه GAN

این شبکه یک بردار نویز 256 بعدی را دریافت کرده و با عبور آن زا Generator یک تصویر 28*28*1 میسازد و سپس با اعمال این تصویر به عنوان ورودی Discriminator آنرا اعتبار سنجی میکند و نهایتا بر اساس آن پارامتر های Generator را آپدیت میکند (در طی این فرایند Discriminator تغییر نمیکند)

ساختار آن به صورت زیر است:

```
Model: "gan"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         (None, 100)               0
_________________________________________________________________
generator (Model)            (None, 28, 28, 1)         2963328
_________________________________________________________________
discriminator (Model)        (None, 1)                 213633
=================================================================
Total params: 3,176,961
Trainable params: 2,937,472
Non-trainable params: 239,489
```

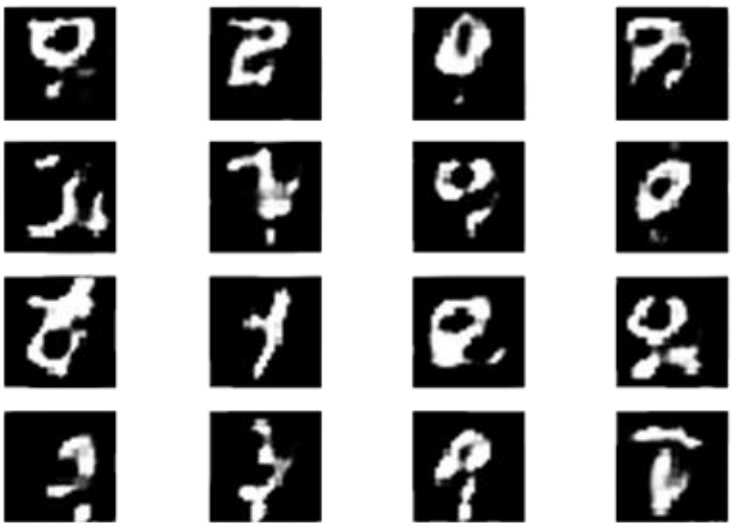
بخش چهارم : آموزش شبکه

به منظور آموزش شبکه از دو حلقه تودرتو استفاده شده است حلقه اول به تعدادepochs اجرا شده و حلقه دوم به تعداد batch های موجود در دیتاست بدین ترتیب از تعداد زیادی از نمونه های موجود در دیتاست و احتمال تکرا بالای یک نمونه، در فرایند آموزش استفاده میشود. در هر بار اجرای حلقه دوم، یک بچ از عکسهای موجود در در دیتاست و یک بچ از عکسهای تولیدی توسط Generator برای آموزش Discriminator و نیز آپدیت generator استفاده میشود.

هماطور که پیشتر گفته شده Generator سعی میکند یک توزیع احتمال مشترک در ابعاد نویز (در فضای نویز) به تصاویر دیتاست نسبت دهد لذا از لحاظ تئوری باید بتواند به هر برداری از فضای نویز (فضایی با ابعاد نویز ورودی که در اینجا یک فضای 256 بعدیست) یک تصویر مناسب نسبت دهد اما لزوما این تصویر معنای خاصی ندارد!...بنابراین به منظور نشان دادن کیفیت Generator تعداد زیادی (16 بار و هربار 64 ورودی) به generator اعمال شده و بهترین خروجی ها انتخاب شده است.

فرایند یادگیری Generator بسیار طولانی بوده و نتیجه قابل قبول پس از حد اقل 50 تا epoch و صرف زمانی نزدیک به یک روز و در پلتفرم colab حاصل شده است.

تصاویر زیر پس از 10 تا epoch به دست آمده است:



پس از 50 تا epoch تصاویر حاصله قابل تشخیص تر شده اند