به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق

تمرین ششم

نام و نام خانوادگی	حسام اسداله‌زاده – مسعود طهماسبی
شماره دانشجویی	810198346 – 810198429
تاریخ ارسال گزارش	۱۴۰۱.۱۱.۰۷

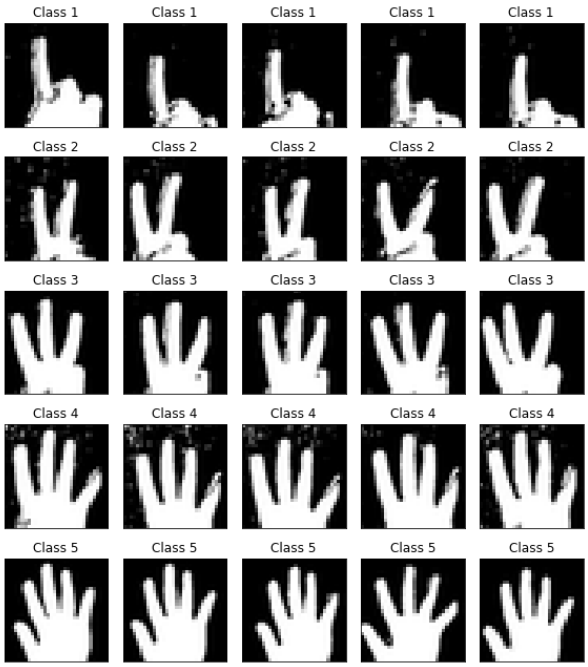
فهرست

شکل‌ها

پاسخ ۱ – شبکه‌های مولد تخاصمی کانولوشنال عمیق

۱-۱- پیاده‌سازی مولد تصویر با استفاده از شبکه‌های مولد تخاصمی کانولوشنال عمیق

مجموعه داده‌ی مورد نظر خوانده شد و تعدادی از داده‌های هر کلاس نمایش داده شد:



شکل 1. تعدادی از داده‌های هر ۵ کلاس مجموعه داده

ابتدا بخش generator به شکل زیر پیاده‌سازی شد:

```
def make_gen():
    g = keras.Sequential()
    g.add(layers.Dense(8*8*256, input_shape=[100]))
    g.add(layers.BatchNormalization())
    g.add(layers.LeakyReLU(0.2))
    g.add(layers.Reshape([8, 8, 256]))
    g.add(layers.BatchNormalization())
    g.add(layers.Conv2DTranspose(64, kernel_size=5, strides=2, padding="same"))
    g.add(layers.BatchNormalization())
    g.add(layers.LeakyReLU())
    g.add(layers.Conv2DTranspose(1, kernel_size=5, strides=2, padding="same", activation="tanh"))
    return g
```

سپس بخش discriminator نیز به شکل زیر پیاده‌سازی شد:

```
def make_disc():
    d = keras.Sequential()
    d.add(layers.Conv2D(64, kernel_size=5, strides=2, padding="same", input_shape=[32, 32, 1]))
    d.add(layers.LeakyReLU())
    d.add(layers.Dropout(0.3))
    d.add(layers.Conv2D(128, kernel_size=5, strides=2, padding="same"))
    d.add(layers.LeakyReLU())
    d.add(layers.Dropout(0.3))
    d.add(layers.Flatten())
    d.add(layers.Dense(1))
    return d
```

برای محاسبه‌ی loss مربوط به بخش generator و discriminator از تابع زیر استفاده شد:

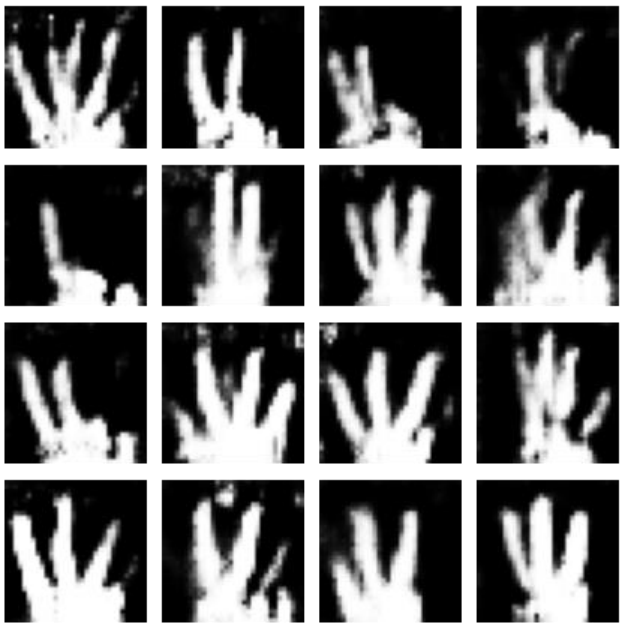
```
cross_entropy = tf.keras.losses.BinaryCrossentropy(from_logits=True)

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)

generator_optimizer = tf.keras.optimizers.Adam(1e-4)
discriminator_optimizer = tf.keras.optimizers.Adam(1e-4)
```

همچنین برای انجام back propagation و اعمال گرادیان‌ها از tensorflow استفاده کردیم:

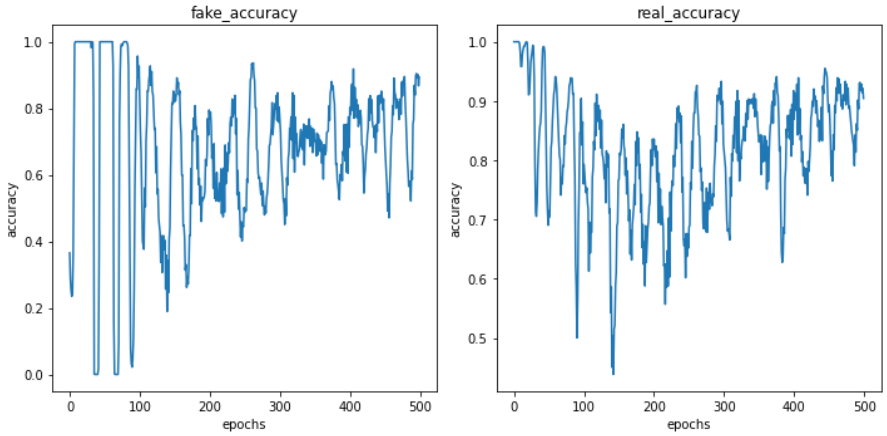
```
@tf.function
def train_step(images):
    noise = tf.random.normal([BATCH_SIZE, noise_dim])
    with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
      generated_images = generator(noise, training=True)

      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))
```

۱-۲- ارزیابی شبکه

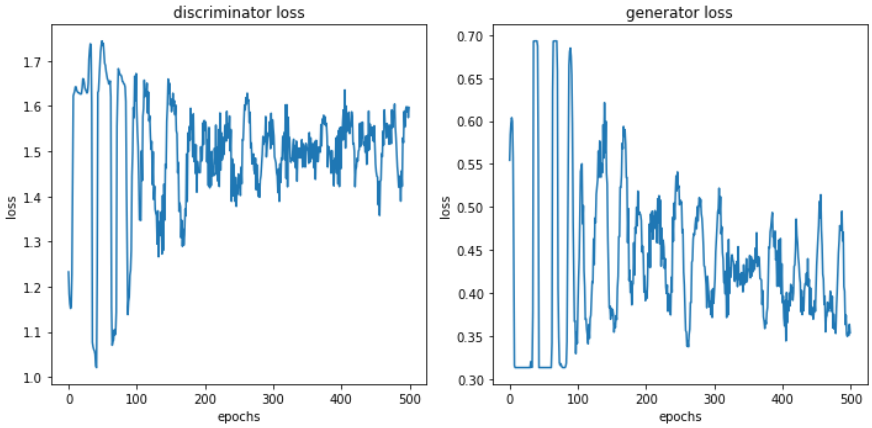
تصاویر تولید شده در ایپاک آخر:



نمودارهای loss و accuracy برای هر دو بخش generator و discriminator به شرح زیر است:



شکل 2. نمودار دقت برای داده‌های واقعی و جعلی (تولید شده توسط **generator**) مربوط به **discriminator**



شکل 3. نمودار تابع هزینه برای **generator** و **discriminator**

۱-۳- پایدارسازی شبکه

- تکنیک One-sided label smoothing:
  - استفاده از تکنیک Label smoothing برای جلوگیری از overfitting برای اولین بار در مقاله‌ی Rethinking the Inception Architecture for Computer Vision در سال ۲۰۱۵ مطرح شد. کلیت این ایده به این شکل بود که به جای استفاده از خود برچسب‌های داده، که معمولا به صورت one-hot هستند و روی ایندکس مربوط به شماره‌ی کلاس عدد یک قرار می‌گیرد، یک smoothing انجام شده یا به نوعی کمی noise به این برچسب واقعی اضافه شود:

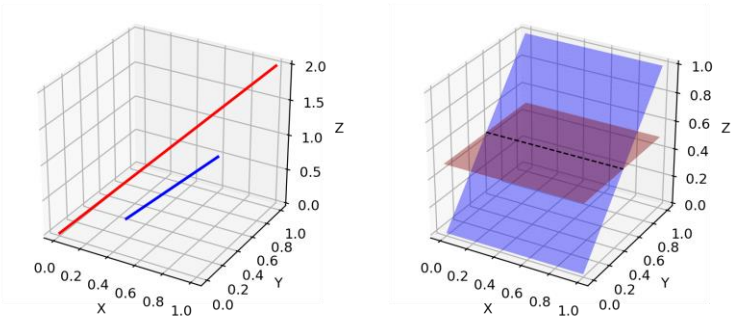
$$\epsilon \approx 0.1, K = \#of\ Classes \rightarrow [0 \dots 1 \dots 0] \Rightarrow [\frac{\epsilon}{K} \dots (1-\epsilon) \dots \frac{\epsilon}{K}]$$

  - حال برای شبکه‌های مولد متخاصمی، از One-sided label smoothing استفاده می‌شود به این معنی که فقط برای کلاس مثبت (یا ۱) از label smoothing استفاده می‌شود و برچسب داده‌های کلاس صفر به همان شکل باقی می‌ماند:

$$\begin{cases} Positive\ label: \quad smooth\ 1 \rightarrow \alpha\ (0 \ll \alpha < 1) \\ Negative\ label: \quad set\ as\ 0 \end{cases}$$

  - تعبیر این کار این است که مدل discriminator خیلی دقیق روی دیتای واقعی یا روی توزیع احتمال داده‌های واقعی مجموعه train به اصطلاح overfit نکند و اجازه‌ی تنوعی را برای داده‌های تولید شده به ما بدهد. ولی برای داده‌های با کلاس منفی این smoothing انجام نمی‌شود.

- تکنیک Add Noise:
  - در واقع برای استفاده از این تکنیک باید به داده‌های واقعی خود noise اضافه کنیم و سپس آن را وارد discriminator کنیم. این تکنیک به صورت مصنوعی تنوع داده‌های موجود در مجموعه آموزش را افزایش می‌دهد و به جلوگیری از overfitting کمک بسیاری می‌کند.
  - بررسی تئوری: همانطور که در مقاله‌ی Arjovsky and Bottou (2017) بیان شده:
    - The dimensions of many real-world datasets, as represented by p_r, only appear to be artificially high. They have been found to concentrate in a lower dimensional **manifold**.
  - یعنی توزیع داده‌های واقعی معمولا روی یک ابرصفحه موجود در فضاهای با ابعاد بالا پخش شده و در مورد تصاویر، معمولا دارای بعد کمتری از تعداد کل پیکسل‌های خود است.
  - از طرفی توزیع داده‌های تولید شده از generator هم دارای بعد کم هستند.
    - p_g lies in a low dimensional manifolds, too. Whenever the generator is asked to a much larger image like 64x64 given a small dimension, such as 100, noise variable input z, the distribution of colors over these 4096 pixels has been defined by the small 100-dimension random number vector and can hardly fill up the whole high dimensional space.
  - حال از آنجایی که هر دوی p_g و p_r روی یک low-dimensional manifold پخش هستند، به احتمال بسیار زیادی disjoint خواهند بود. از این رو، کار discriminator برای تفکیک داده‌های واقعی از داده‌های تولید شده توسط generator بسیار ساده خواهد شد.



شکل **4** . منیفولدهای کم‌بُعد در فضای با ابعاد بالا به سختی می‌توانند همپوشانی داشته باشند.

- To artificially “spread out” the distribution and to create higher chances for two probability distributions to have overlaps, one solution is to **add continuous noises** onto the inputs of the discriminator.

- یعنی برای "گسترش" مصنوعی توزیع و ایجاد احتمال بیشتری برای همپوشانی دو توزیع احتمال، یک راه‌حل، اضافه کردن نویزهای پیوسته به ورودی‌های discriminator است. با این تکنیک، کار discriminator برای تشخیص بین داده‌های جعلی و واقعی سخت‌تر می‌شود و یک نوع regularization برای آموزش این مدل است.

نتایج عملی: (یک فایل mp4 از تصاویر تولیدشده در هر epoch در فایل زیپ آپلود شده موجود است)

برای پیاده‌سازی label smoothing تابع loss را تغییر دادیم:

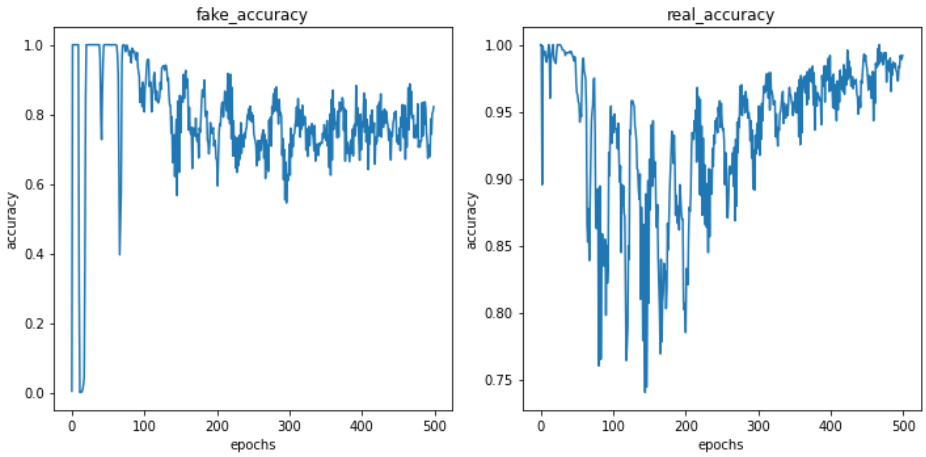
```
def smooth_discriminator_loss(real_output, fake_output):
    real_loss = cross_entropy(tf.ones_like(real_output,dtype = float)*smoothness_index, real_output)
    fake_loss = cross_entropy((tf.zeros_like(fake_output,dtype = float) + (1 - smoothness_index)), fake_output)
    total_loss = real_loss + fake_loss
    return total_loss

def smooth_generator_loss(fake_output):
    return cross_entropy(tf.ones_like(fake_output,dtype = float)*smoothness_index, fake_output)
```

در تابع loss، برچسب داده‌های واقعی را در smoothness_index ضرب می‌کنیم.

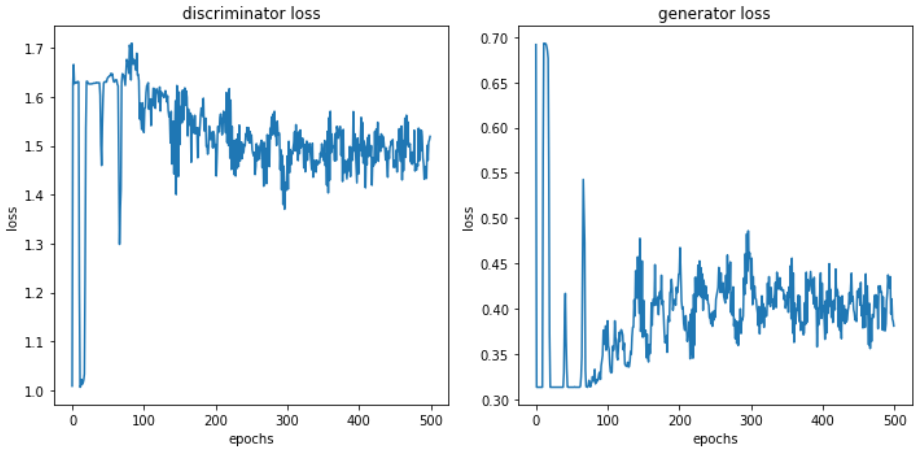
حال برای Add Noise نیز به ازای هر بچ از تصاویر به جای images، حاصلجمع آن با نویز رندوم را به discriminator می‌دهیم:

```
images + (tf.random.normal(shape=(images.shape[0],32,32),mean=0.0,stddev=np.random.uniform(0.0,
0.1),dtype=tf.float32))
```

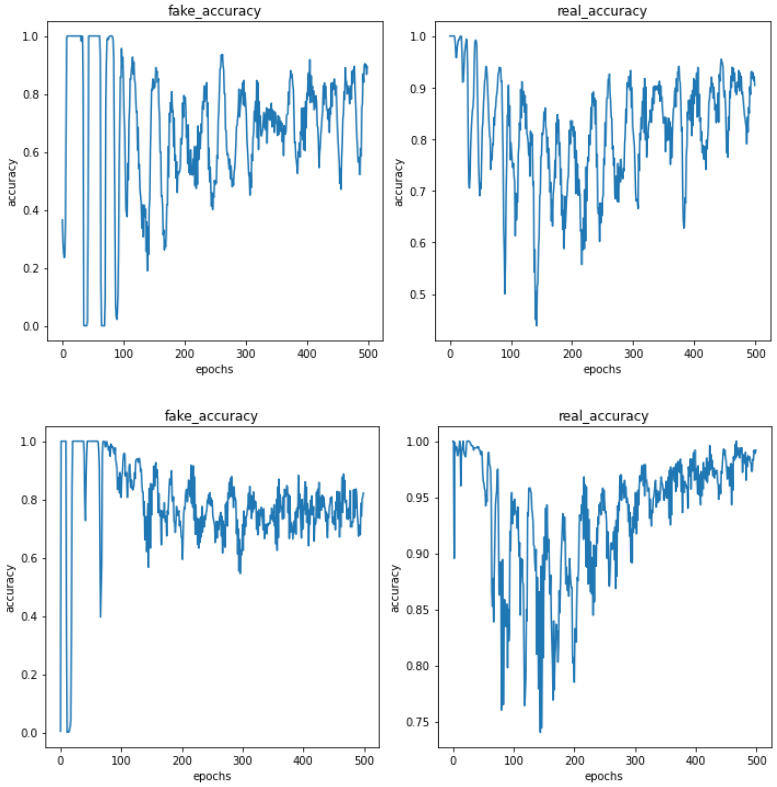
نتایج:



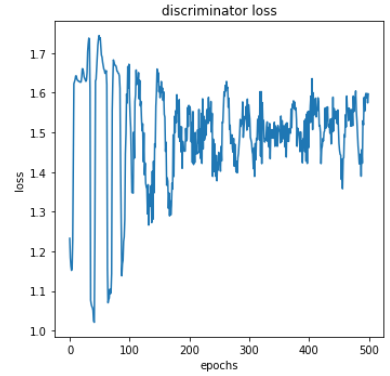
شکل 5. نمودار تغییرات دقت روی داده‌های واقعی و جعلی با استفاده از **Label Smoothing** و **Add Noise**

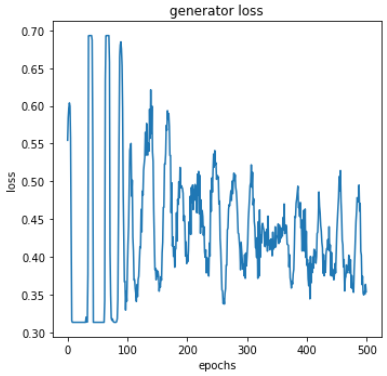


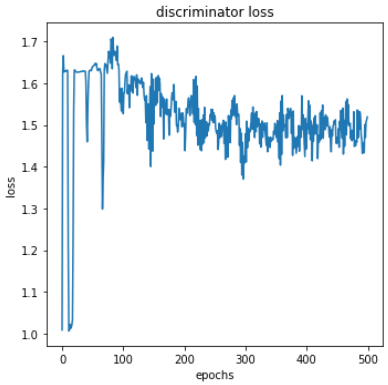
شکل 6. نمودار **loss** برای **discriminator** و **generator** با استفاده از **Label Smoothing** و **Add Noise**

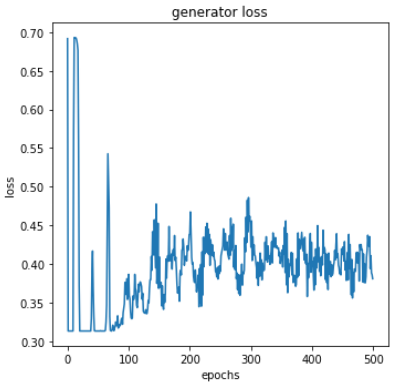


با مقایسه تصاویر بالا (بدون Label Smoothing و Add Noise) و پایین (با استفاده از این تکنیک‌ها) مشاهده می‌شود که فرآیند آموزش بسیار پایدارتر شده است و بازه تغییرات دقت محدودتر و پایدارتر شده است. این موضوع در نمودار تغییرات هزینه نیز به وضوح قابل مشاهده است:


discriminator loss
loss
epochs

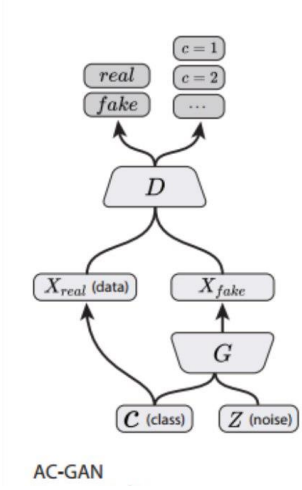

generator loss
loss
epochs


discriminator loss
loss
epochs

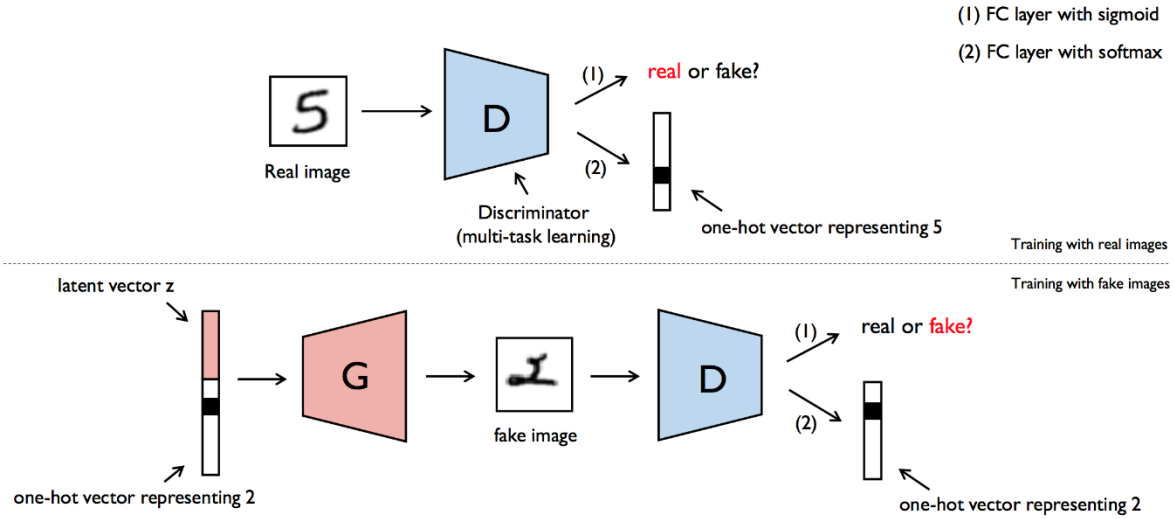

generator loss
loss
epochs

پاسخ ۲ – شبکه متخاصم مولد طبقه‌بند کمکی و شبکه Wasserstein

۲-۱- شبکه متخاصم مولد طبقه‌بند کمکی



شکل 7. **Conditional Image Synthesis with Auxiliary Classifier GANs**



شکل 8. نحوه آموزش **AC-GAN** برای داده‌های واقعی و جعلی

مدل Auxiliary Classifier GAN، یا به اختصار AC-GAN، توسعه‌ای از GAN شرطی است که discriminator را تغییر می‌دهد تا به جای دریافت برچسب کلاس یک تصویر به عنوان ورودی، بتواند آن را پیش‌بینی کند.

Generative Adversarial Network (GAN)

$$X_{fake} = G(z)$$

$$P(\underbrace{S}_{\text{source}} \mid X) = D(X)$$

$$L = E[\log P(S = real \mid X_{real})] + E[\log P(S = fake \mid X_{fake})]$$

شکل**9** . مدلسازی **GAN**

Auxiliary Classifier GAN (AC-GAN)

$$X_{fake} = G(\underbrace{c}_{\text{class label}}, \underbrace{z}_{\text{noise}})$$

$$P(S \mid X),\ P(C \mid X) = D(X)$$

$$L_S = E[\log P(S = real \mid X_{real})] + E[\log P(S = fake \mid X_{fake})]$$

$$L_C = E[\log P(C = c \mid X_{real})] + E[\log P(C = c \mid X_{fake})]$$

$L_S \rightarrow$ log-likelihood of the correct source

$L_C \rightarrow$ log-likelihood of the correct class

$$D^* = \arg\max_D L_S + L_C$$

$$G^* = \arg\max_G L_C - L_S$$

شکل**10** . مدلسازی **AC-GAN**

آموزش مدل GAN در AC-GAN به گونه‌ای تغییر می‌کند که generator به عنوان ورودی هم یک نقطه تصادفی در فضای پنهان و هم یک برچسب کلاس می‌گیرد و سعی می‌کند یک تصویر برای آن کلاس تولید کند. Discriminator هم به عنوان ورودی یک تصویر می‌گیرد و باید مثل قبل، واقعی یا جعلی بودن تصویر را طبقه‌بندی کند. به علاوه، در AC-GAN مدل تفکیک‌کننده باید کلاس مربوط به تصویر ارائه شده را نیز پیش‌بینی کند.

برای پیاده‌سازی این شبکه ابتدا معماری generator را تغییر می‌دهیم:

```
def make_generator(noise,labels):
    g = keras.layers.Concatenate()([noise, labels])
    g = (layers.Dense(8*8*256,activation="relu", input_shape=[105,]))(g)
    g = (layers.BatchNormalization())(g)
    g = (layers.Reshape([8, 8, 256]))(g)
    g = (layers.BatchNormalization())(g)
    g = (layers.UpSampling2D())(g)
    g = (layers.Conv2D(128, kernel_size=3, padding="same"))(g)
    g = (layers.Activation("relu"))(g)
    g = (layers.BatchNormalization())(g)
    g = (layers.UpSampling2D())(g)
    g = (layers.Conv2D(64, kernel_size=3, padding="same"))(g)
    g = (layers.Activation("relu"))(g)
    g = (layers.BatchNormalization())(g)
    g = (layers.Conv2D(1, kernel_size=3, padding='same'))(g)
    out = (layers.Activation("tanh"))(g)

    return keras.models.Model([noise,labels],out)
```

تغییر مهمی که معماری داشته این بوده که input_shape از ۱۰۰ به ۱۰۵ افزایش یافته. چون برچسب داده‌ها نیز (شامل ۵ کلاس) باید به عنوان ورودی به مدل generator داده شود.

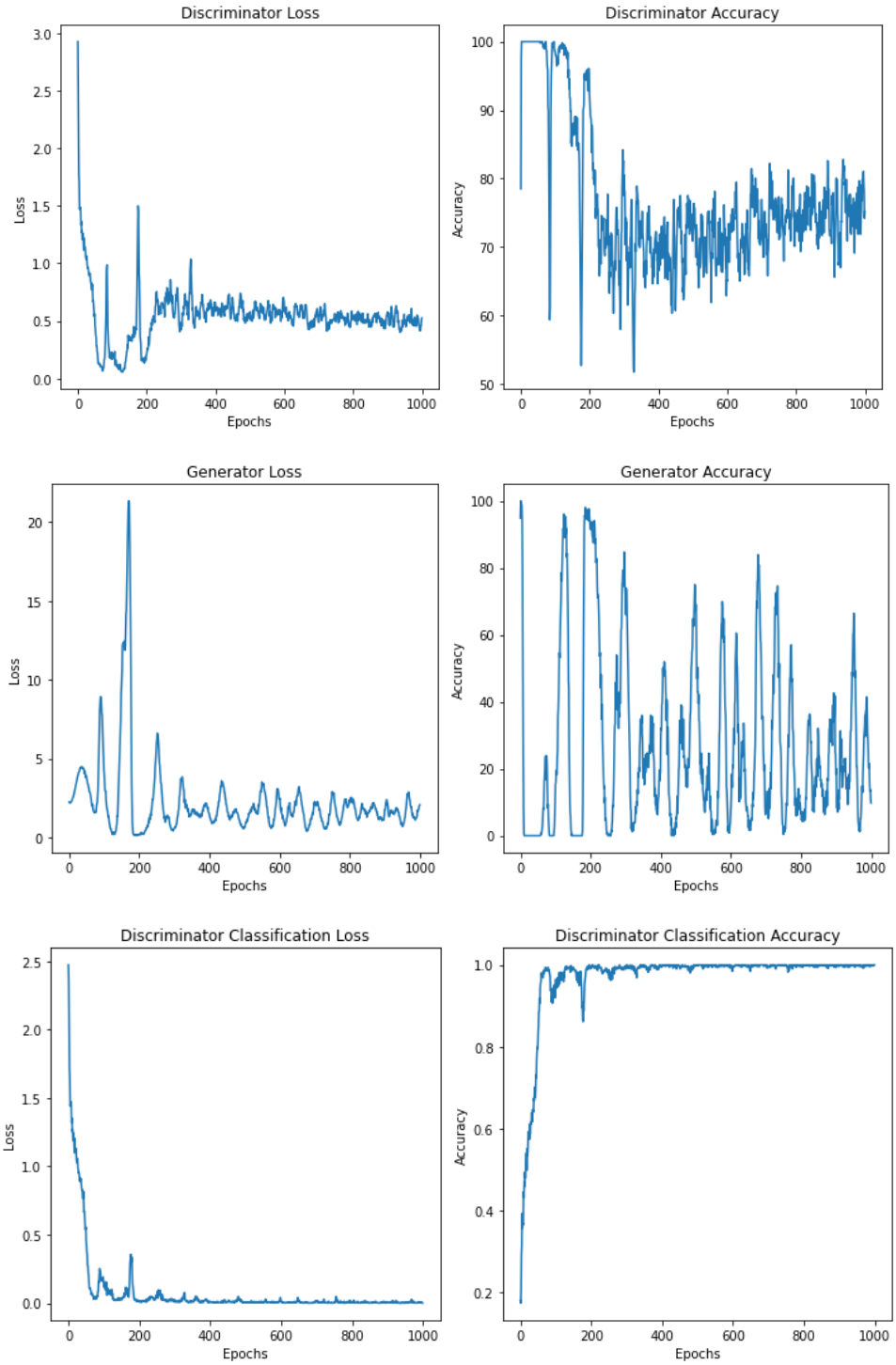
```
def make_discriminator(input):
    d = (layers.Conv2D(32, (3,3), strides=(2,2), padding='same', input_shape=[32, 32, 1]))(input)
    d = (layers.LeakyReLU(0.2))(d)
    d = (layers.Dropout(0.5))(d)
    d = (layers.Conv2D(64, (3,3), padding='same'))(d)
    d = (layers.BatchNormalization())(d)
    d = (layers.LeakyReLU(0.2))(d)
    d = (layers.Dropout(0.5))(d)
    d = (layers.Conv2D(128, (3,3), strides=(2,2), padding='same'))(d)
    d = (layers.BatchNormalization())(d)
    d = (layers.LeakyReLU(0.2))(d)
    d = (layers.Dropout(0.5))(d)
    d = (layers.Conv2D(256, (3,3), padding='same'))(d)
    d = (layers.BatchNormalization())(d)
    d = (layers.LeakyReLU(0.2))(d)
    d = (layers.Dropout(0.5))(d)
    out = (layers.Flatten())(d)

    out1 = layers.Dense(1, activation='sigmoid')(out)
    out2 = layers.Dense(5, activation='softmax')(out)

    return keras.models.Model(input, [out1, out2])
```

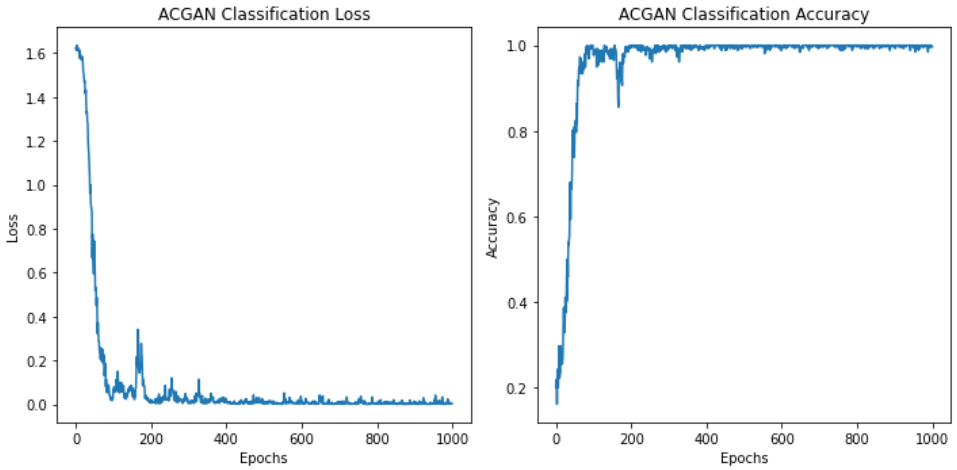
تغییر مهمی که discriminator داشته نیز این بوده که در out1 مانند گذشته واقعی یا جعلی بودن داده را پیش‌بینی می‌کند و در out2 کلاس (یا برچسب) مربوط به داده‌ی ورودی را پیش‌بینی می‌کند.

نتایج آموزش مدل به شرح زیر است:

خروجی ایپاک آخر (از چپ به راست، برچسب‌های ۱ تا ۵ تولید شده):

نمودارهای تغییرات دقت و تابع هزینه:



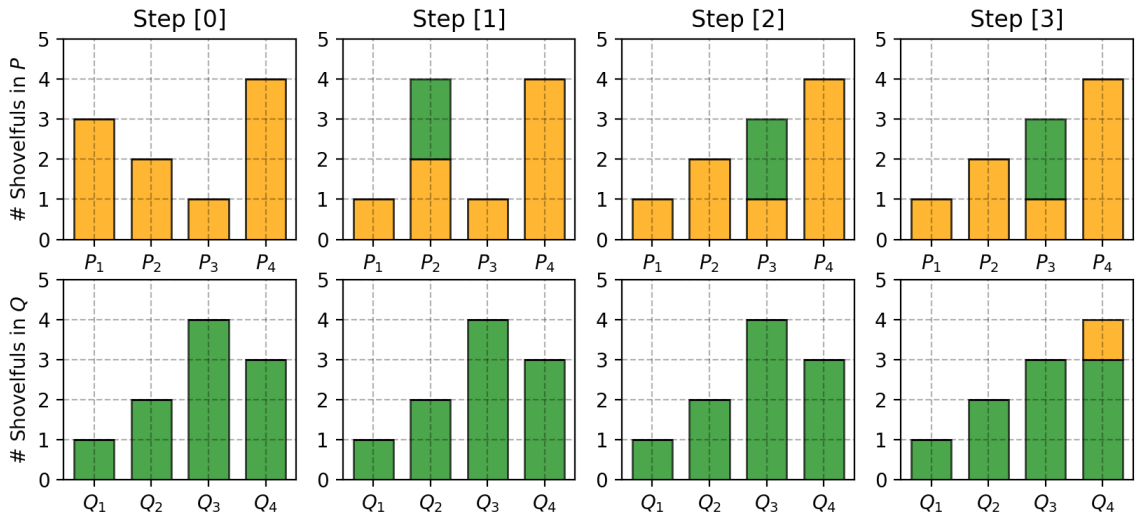


۲-۲- شبکه متخاصم مولد Wasserstein

ابتدا به توضیح Wasserstein distance می‌پردازیم:

این فاصله یک معیار برای میزان مشابهت یا تفاوت (در واقع فاصله) دو توزیع احتمال می‌باشد. از این فاصله به عنوان Earth Mover's distance نیز یاد می‌شود[1].

برای درک بهتر این معیار، به توضیح مثال زیر می‌پردازیم:



در این مثال، دو توزیع احتمال گسسته P و Q را در نظر بگیرید که مانند Step [0] دارای ۴ مقدار اولیه با وزن احتمال موجود در شکل هستند. حال می‌خواهیم این دو توزیع را به گونه‌ای تغییر دهیم که دقیقا مانند هم شوند. برای این کار ابتدا باید ۲ واحد از P_1 به P_2 منتقل کنیم. سپس این دو واحد را از P_2 به P_3 و در نهایت یک واحد از Q_3 به Q_4 منتقل می‌کنیم. مشاهده می‌شود که این دو توزیع احتمال در نهایت

[1] because informally it can be interpreted as the minimum energy cost of moving and transforming a pile of dirt in the shape of one probability distribution to the shape of the other distribution.

دقیقا مانند هم شده‌اند و فاصله‌ی Wasserstein این دو برابر ۲+۲+۱ یا ۵ می‌باشد. حال برای توزیع‌های **پیوسته**، این فرمول کمی پیچیده‌تر خواهد بود:

$$W(p_r, p_g) = \inf_{\gamma \sim \Pi(p_r, p_g)} E_{(x,y)\sim\gamma}\left[||x - y||\right]$$

چرا Wasserstein بهتر از KL و JS است؟

برای یادآوری ابتدا دو معیار ذکر شده را می‌بینیم:

روش‌های سنتی از روش maximum likelihood برای مدل کردن مدل‌های generative استفاده می‌کردند که معادل کمینه کردن Kullback-Leibler divergence بین توزیع داده‌های واقعی $p_r(x)$ و داده‌های تولیدشده توسط generator یعنی $p_g(x)$ استفاده می‌کردند.

$$KL(P_r||P_g) = \int_{\mathcal{X}} p_r(x) \log \frac{p_r(x)}{p_g(x)} \mathrm{dx} = E_{P_r}\left[\log \frac{p_r(x)}{p_g(x)}\right].$$

این روش دو ویژگی جالب و نه چندان خوب داشت:

- اگر $p_r(x) > p_g(x)$: پس x نقطه‌ای با احتمال بیشتر روی داده واقعی است و معیار KL هزینه بالایی برای این داده‌ها در نظر می‌گیرد.
- اگر $p_r(x) < p_g(x)$: پس x نقطه‌ای با احتمال بیشتر روی داده جعلی یا تولید شده از generator است و معیار KL هزینه کمی برای این داده‌ها در نظر می‌گیرد.

برای حل این موضوع از معیار Jensen-Shannon divergence استفاده شد:

$$JS(P_r, P_g) = KL(P_r||P_m) + KL(P_g||P_m)$$
$$P_m = \frac{p_r + p_g}{2}$$

و فرآیند آموزش مانند زیر است:

The training procedure is shown as follows,

- We first train a discriminator D to maximize $L(D, g_\theta)$,

  $L(D, g_\theta) = \mathbb{E}_{x\sim\mathbb{P}_r}[\log D(x)] + \mathbb{E}_{x\sim\mathbb{P}_g}[\log(1 - D(x))]$. The optimal discriminator has the form, $D^*(x) = \frac{p_r(x)}{p_r(x)+p_g(x)}$, and $L(D^*, g_\theta) = 2JSD(\mathbb{P}_r||\mathbb{P}_g) - 2\log 2$.

- Once we get the optimal discriminator D^*, we minimize $L(D^*, g_\theta)$ with repsect to θ, which is equivalent to minimizing the Jensen-Shanon divergence $JSD(\mathbb{P}_r||\mathbb{P}_g)$.

پس به طور خلاصه ۳ معیار استفاده شده در آموزش GAN ها به شرح زیر است:

- Kullback-Leibler (KL) divergence

  $KL\left(\mathbb{P}_r \| \mathbb{P}_g\right)=\int \log \left(\frac{P_r(x)}{P_g(x)}\right) P_r(x) d \mu(x)$ $\mathbb{P}_r, \mathbb{P}_g$ are assumed to be absolutely continuous, and therefore admits densities, with respect to a same measure μ defined on $\mathcal{X}$.

- Jensen-Shannon (JS) divergence

  $JS\left(\mathbb{P}_r, \mathbb{P}_g\right)=KL\left(\mathbb{P}_r \| \mathbb{P}_m\right)+KL\left(\mathbb{P}_g \| \mathbb{P}_m\right)$ where $\mathbb{P}_m$ is the mixture $\left(\mathbb{P}_r+\mathbb{P}_g\right) / 2$. The divergence is symmetrical and always defined as we can choose $\mu=\mathbb{P}_m$ (choose μ as the measure of $\mathbb{P}_m$?)

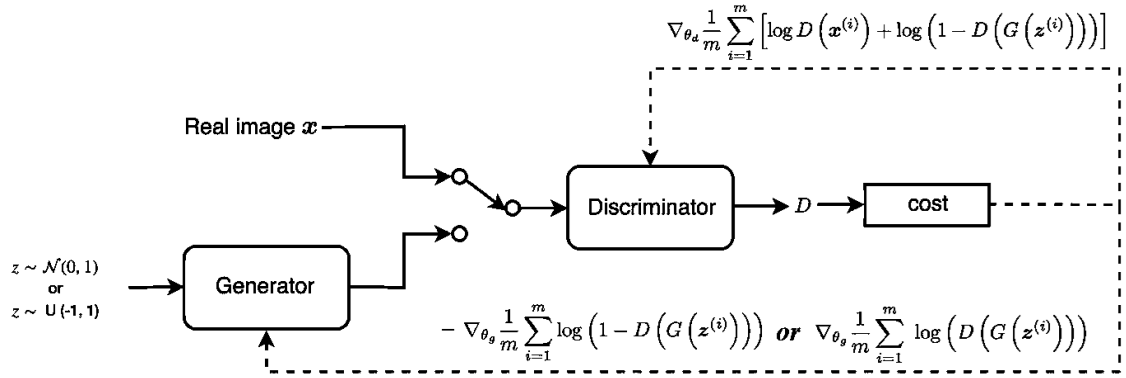
- Earth-Mover (EM) distance or Wasserstein-1

  $W\left(\mathbb{P}_r, \mathbb{P}_g\right)=\inf _{\gamma \in \Pi\left(\mathbb{P}_r, \mathbb{P}_g\right)} \mathbb{E}_{(x, y) \sim \gamma}[\|x-y\|_2]$ where $\Pi\left(\mathbb{P}_r, \mathbb{P}_g\right)$ denotes the set of all joint distributions $\gamma(x, y)$ whose marginals are respectively $\mathbb{P}_r$ and $\mathbb{P}_g$. Intuitively, $\gamma(x, y)$ indicates how much mass must be transported from x to y, while $\|x-y\|_2$ indicates the transportation cost.

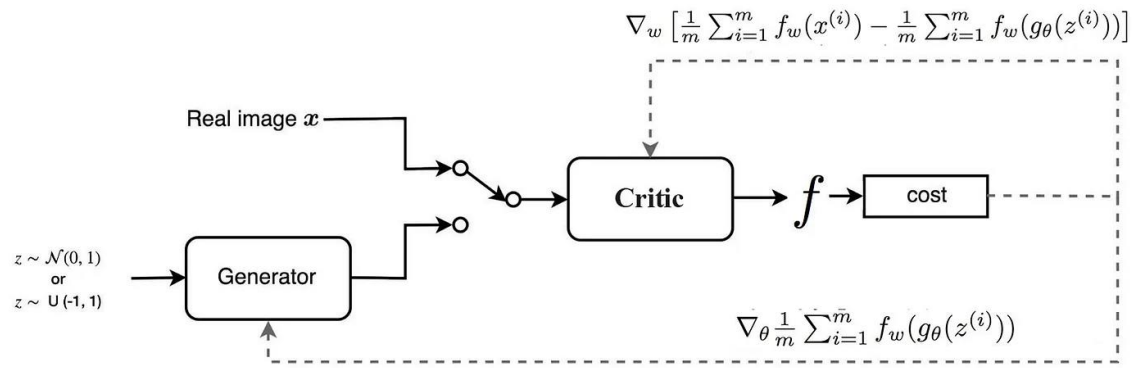
در نهایت با ساده‌سازی‌های ریاضی (Kantorovich-Rubinstein duality) تابع Loss شبکه WGAN به شکل زیر قابل بازنویسی است:

$$\max_{w \in \mathcal{W}} E_{x \sim P_r}[f_w(x)] - E_{z \sim p_{data}(z)}\big[f_w\big(g_\theta(z)\big)\big].$$

برای مقایسه، ابتدا مدل GAN را نمایش می‌دهیم:



ولی مدل WGAN نهایی به شکل زیر است:



چند نکته در رابطه با شبکه‌ی WGAN قابل بیان است:

- در این شبکه، discriminator به جای پیش‌بینی احتمال متعلق بودن یک تصویر به کلاس واقعی یا جعلی، یک score خروجی می‌دهد.
- این score می‌تواند مانند قبل به عنوان معیاری برای واقعی بودن عکس در نظر گرفته شود.
- اسم شبکه discriminator به critic تغییر می‌کند تا این تغییر را در خود نمایش دهد.
- شبکه discriminator سعی در یاد گرفتن پارامتر w برای بهترین تابع f_w را دارد. همچنین تابع loss به شکل فاصله‌ی Wasserstein بین p_r و p_g تغییر کرده است. توضیحات دقیق‌تر به زبان انگلیسی در بند بعد آمده:

- Thus the "**discriminator**" is not a direct critic of telling the fake samples apart from the real ones anymore. Instead, it is trained to learn a K-Lipschitz[1] continuous function to help **compute Wasserstein distance**. As the loss function decreases in the training, the Wasserstein distance gets smaller and the generator model's output grows closer to the real data distribution.

[1] در آنالیز ریاضی، پیوستگیِ لیپشیتس شکل قویتری از پیوستگی برای توابع است که در آن تابع از نظر سرعت تغییرات محدود می‌باشد. یعنی گرادیان تابع لیپشیتس دارای حد می‌باشد که این حد برابر K (ثابت لیپشیتز) است.

- الگوریتم نهایی برای آموزش شبکه‌ی WGAN به شرح زیر است:

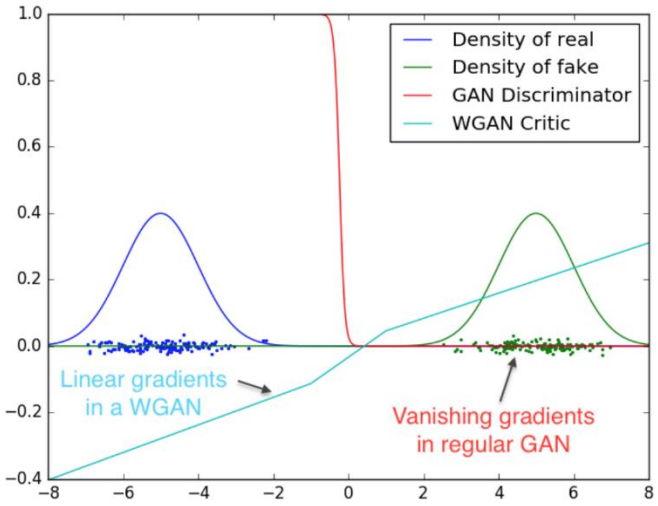
Algorithm 1 WGAN, our proposed algorithm. All experiments in the paper used the default values $\alpha = 0.00005$, $c = 0.01$, $m = 64$, $n_{\text{critic}} = 5$.

Require: : α, the learning rate. c, the clipping parameter. m, the batch size. n_{critic}, the number of iterations of the critic per generator iteration.
Require: : w_0, initial critic parameters. θ_0, initial generator's parameters.

while θ has not converged **do**
  for $t = 0, ..., n_{\text{critic}}$ **do**
    Sample $\{x^{(i)}\}_{i=1}^{m} \sim \mathbb{P}_r$ a batch from the real data.
    Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
    $g_w \leftarrow \nabla_w \left[\frac{1}{m}\sum_{i=1}^{m} f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))\right]$
    $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
    $w \leftarrow \text{clip}(w, -c, c)$
  end for
  Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
  $g_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))$
  $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
end while

شکل 11. الگوریتم آموزش شبکه‌ی **WGAN**

از مزایای استفاده از تابع هزینه‌ی Wasserstein غلبه بر موضوع vanishing gradients می‌باشد.



شکل 12. تفاوت گرادیان‌ها در تابع هزینه **GAN** و **WGAN**

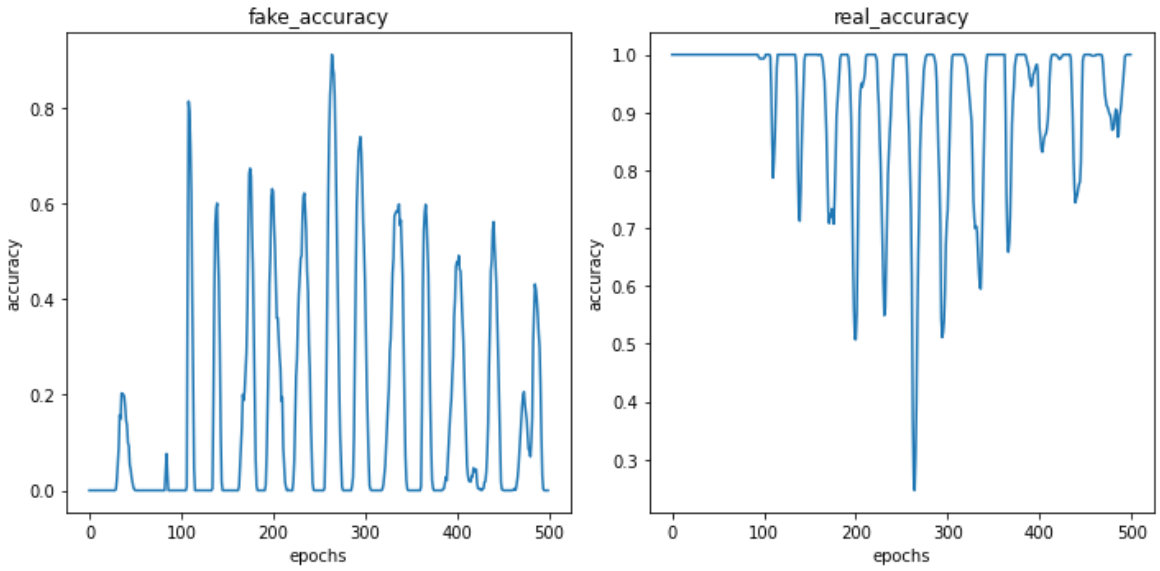
نتایج پیاده‌سازی WGAN:

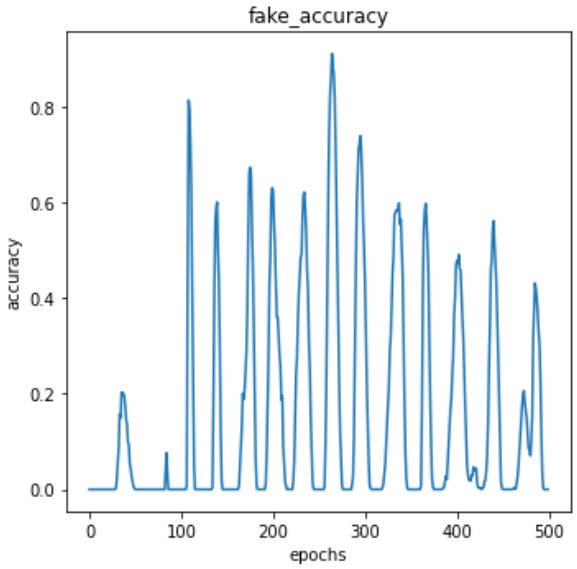
تغییرات نحوه محاسبه تابع loss:

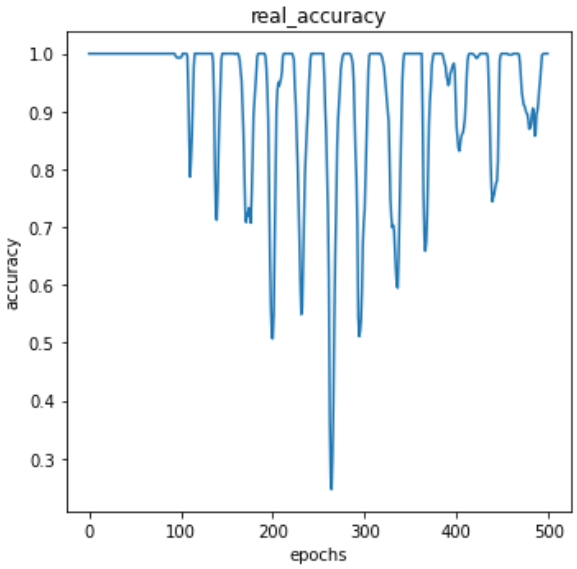
```
disc_loss = tf.math.reduce_mean(fake_output) - tf.math.reduce_mean(real_output)
gen_loss = (-1)*tf.math.reduce_mean(fake_output)
```

تصاویر تولید شده در ایپاک آخر:

نمودار تغییرات دقت و تابع هزینه:




fake_accuracy
accuracy
epochs
0.0
0.2
0.4
0.6
0.8
0
100
200
300
400
500


real_accuracy
accuracy
epochs
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
0
100
200
300
400
500