



קריית החינוך
פארק המדע
בית לערכים
למצינות ולחדשנות

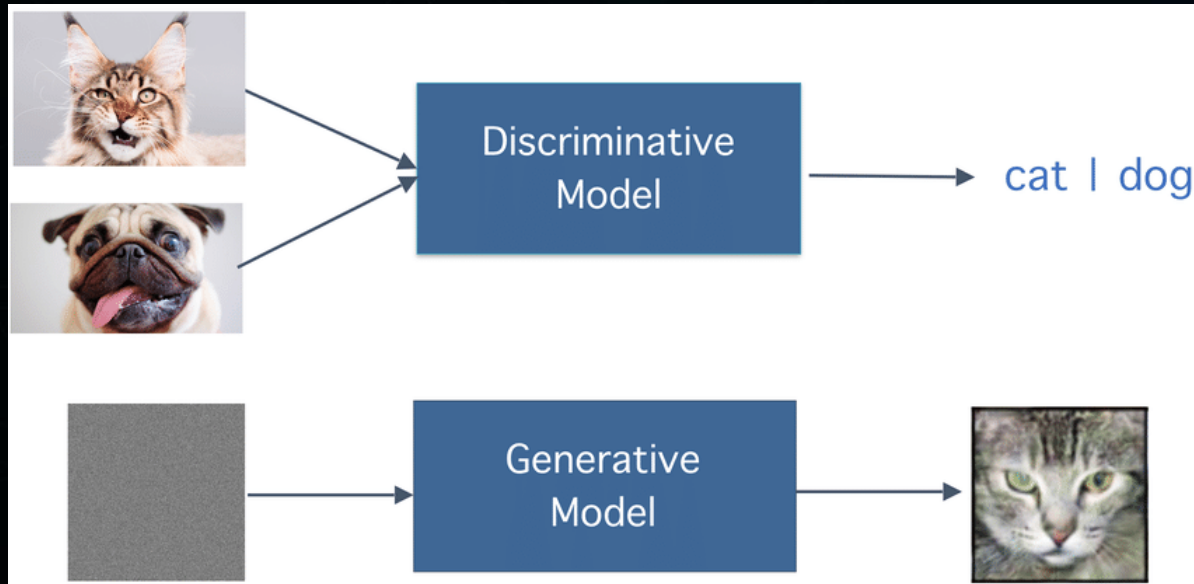
GAN | GENERATIVE ADVERSARIAL NETS

גלעד מרקמן

מחברת ב Google Colab

סוגי רשתות נוירונים

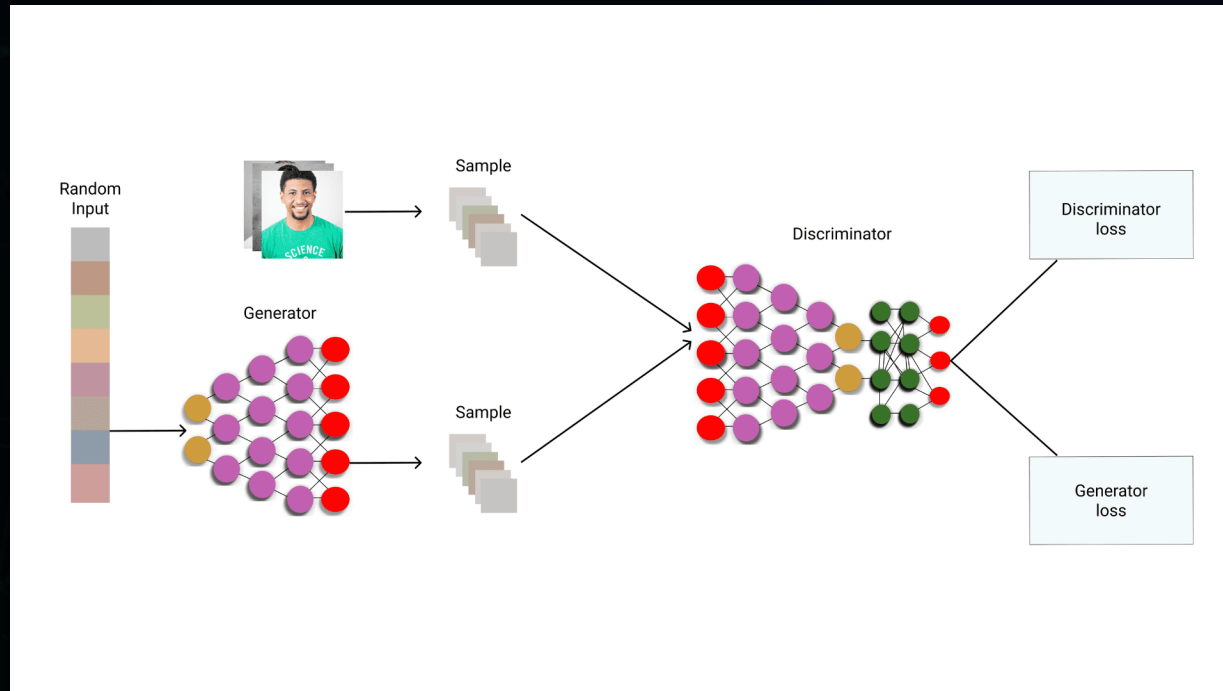
- ניתן לחלק את המודלים של למידת המכונה לשני סוגים עיקריים:
 - רשתות נוירונים מבדילות – Discriminative Neural Network
 - מודלים המקבלים נתונים (תמונות) ומסווגים אותם לקבוצות.
 - רשתות נוירונים יוצרות – Generative Neural Network
 - מודלים היוצרים נתונים חדשים, כגון יצירת תמונות חדשות.



- עד כה הצגנו רק את הסוג הראשון.
- היום נציג דוגמה לסוג השני.

GAN – Generative Adversarial Nets

- בשנת 2014 פורסמה עבודה פורצת דרך המדגימה כיצד ניתן לבנות רשתות נוירונים יוצרות Goodfellow, Generative Adversarial Nets.
- האלגוריתם כולל שתי רשתות נוירונים המתחרות אחת בשניה ותוך כדי כך יוצרות תמונות לפי רצוננו.
- האלגוריתם כולל שתי רשתות:
 - רשת יוצרת - Generator.
 - רשת מאבחנת - Discriminator.

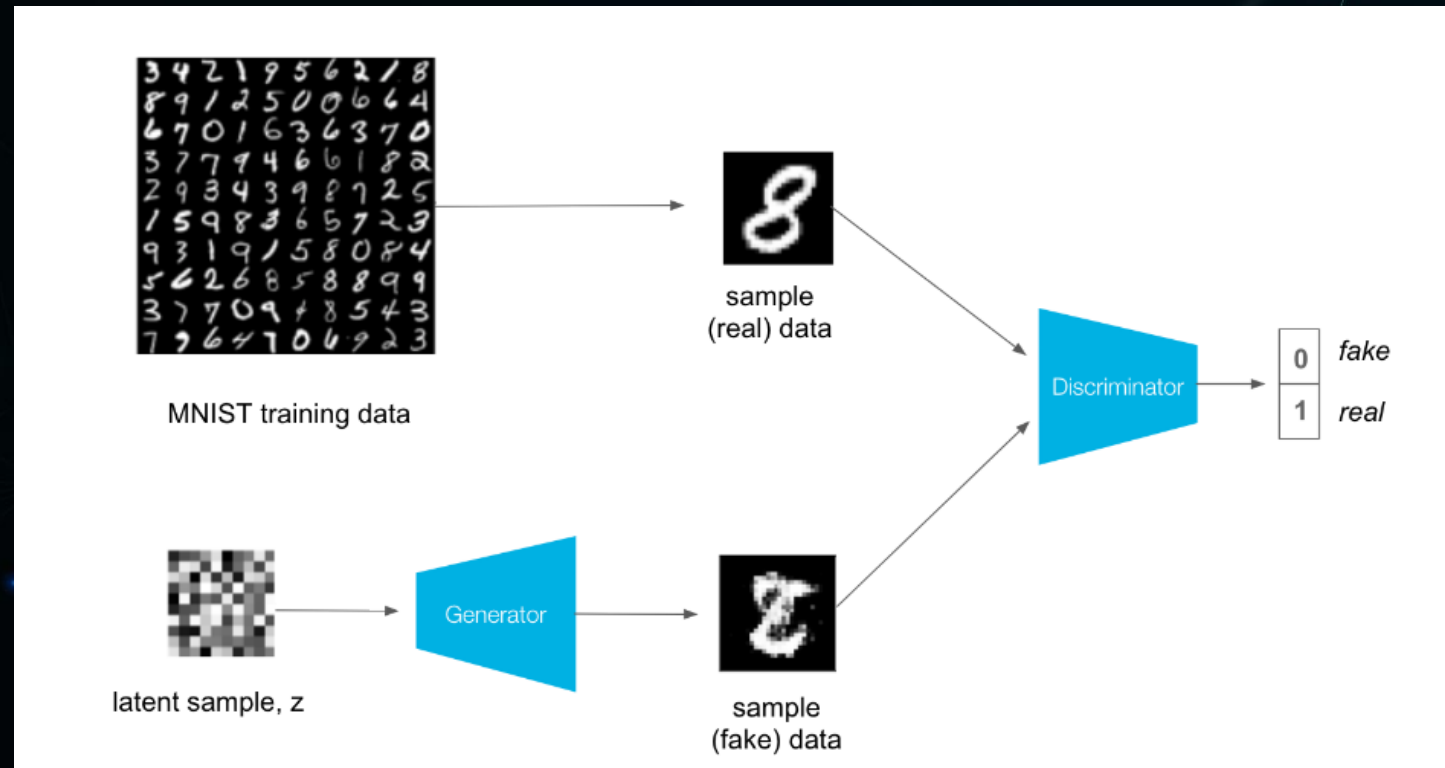


רשתות יוצרות ו- LLM

- האלגוריתם GAN אותו נלמד היום הינו האלגוריתם הבסיסי ללמידת מכונה יוצרת.
- מאז פרסומו בשנת 2014 היתה התקדמות מהירה ומדהימה בתחום, ופותחו מודלים מתקדמים של GAN, כגון CGAN (הכולל שכבות קונבולוציה), WGAN (הכולל שיפור במבנה של המודל) ועוד.
- לאחרונה פרצו לחיינו מודלים Large Language Models – LLM כמו BERT, GPT_n, DALL-E המבוססים על אלגוריתמים שונים המכונים Transformers. נושא זה חורג מסגרת הרצאה זו.

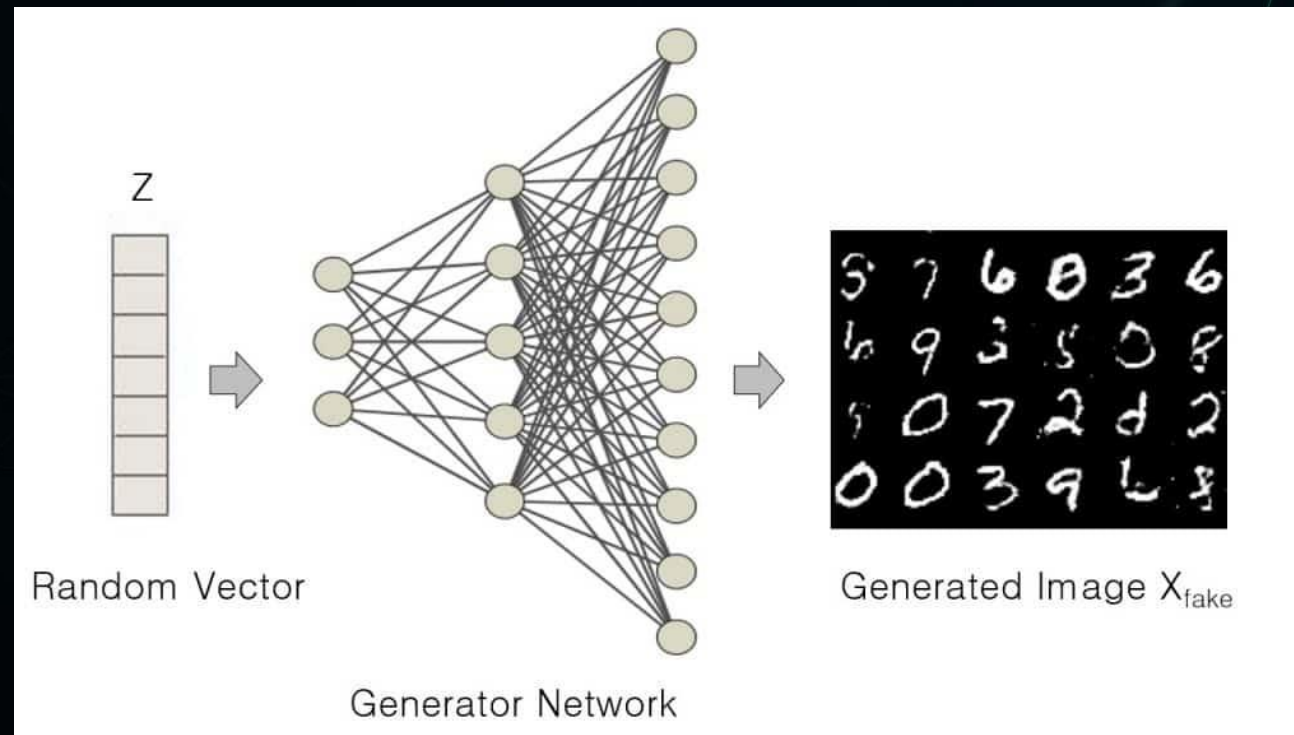
הדגמה של GAN - מודל היוצר תמונה של ספרות

- המטרה: יצירת רשת נוירונים המקבלת מערך של 64 ערכים אקראיים ויוצרת תמונות שחור לבן של ספרות 0-9 בגודל 28×28 פיקסלים, הדומות לבסיס הנתונים MNIST.



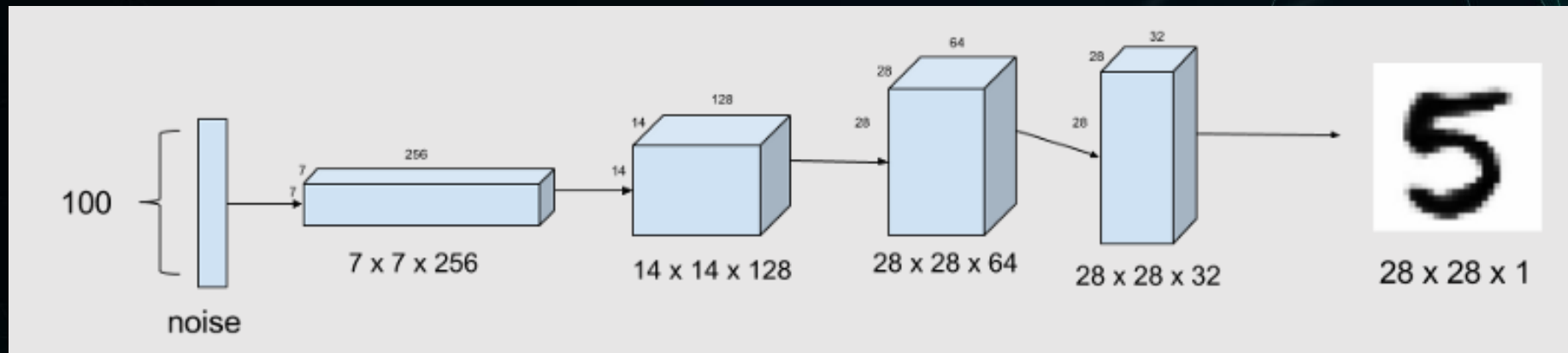
בניית ה Generator

- בניית ה Generator - המודל היוצר את התמונה הוא פשוט ביותר.
- נבנה מודל FNN המקבל קלט של 64 ערכים אקראיים, והפלט שלו יהיה $784=28*28$ ערכים בין $[-1, 1]$, אותם נהפוך לתמונה שחר לבן.



מודל היוצר תמונה של ספרות

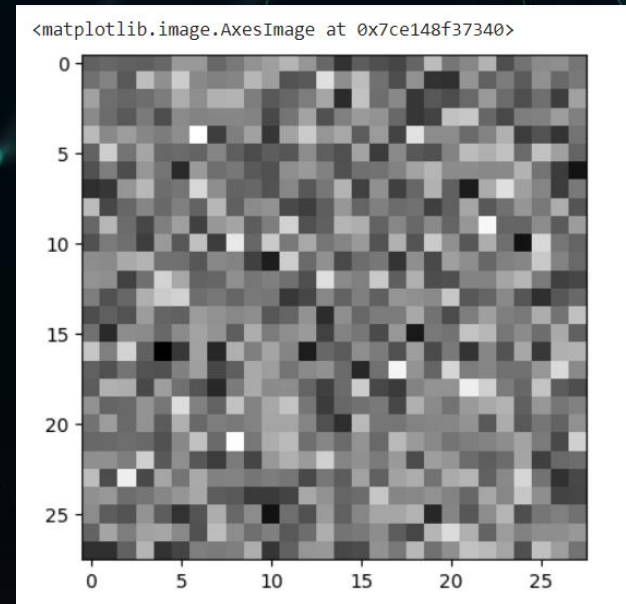
- אם נרצה ליצור תמונות איכותיות יותר או תמונות צבעוניות ניתן להשתמש גם במודלים עם שכבות קונבולוציה.
- על כך, נרחיב בהמשך.



Generator class

- נבנה רשת נוירונים FNN היוצרת תמונה שחור לבן בגודל $28*28$ פיקסלים:
 - הקלט בגודל 64.
 - שכבה אחת נסתרת.
 - פלט בגודל 784 עם ערכים $[-1,1]$ באמצעות אקטיבציה של Tanh.

```
class Generator(nn.Module):  
    def __init__(self):  
        super().__init__()  
        self.fc1 = nn.Linear(64,256)  
        self.fc2 = nn.Linear(256,256)  
        self.out = nn.Linear(256,784) # 28*28 = 784  
  
    def forward(self, x):  
        x = F.leaky_relu( self.fc1(x))  
        x = F.leaky_relu( self.fc2(x))  
        x = F.tanh(self.out(x))  
        return x
```

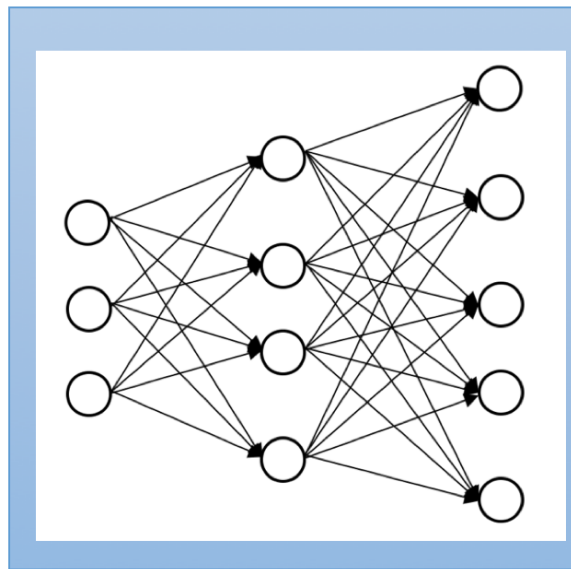


כיצד נאמן את ה Generator ?

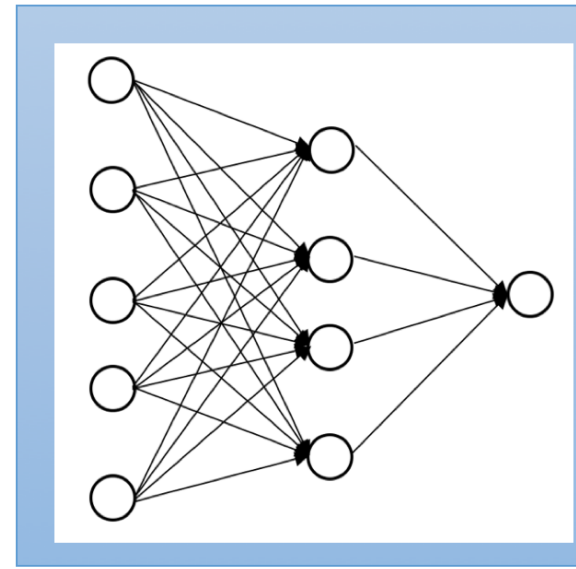
- יצרנו מודל המקבל ערכים רנדומליים (רעש – noise) ויוצר פלט של $28*28$ ערכים מספריים.
- כיצד נאמן את המודל כך שהפלט יהיה אכן תמונה של ספרה ולא פלט אקראי ?
- אנחנו זקוקים למודל שיקבל את הפלט מה Genetator ויאמר לנו האם זוהי אכן תמונה של ספרה או לא.
- באמצעות מודל זה נוכל לחשב את הטעות (loss) ולעדכן את הפרמטרים של הרשת כך שהפלט יהיה דומה יותר ויותר לתמונה של ספרה.
- למודל זה נקרא Discriminator.

Discriminator

- מבנה ה-Discriminator הפוך ממבנה ה-Generator. הוא מקבל תמונה (מערך בגודל 28×28) ומוציא פלט בין 0 ל-1:
 - ערך קרוב ל-1 התמונה קרובה לספרה אמיתית (True).
 - ערך קרוב ל-0 התמונה רחוקה מספרה אמיתית (Fake).



Generator network

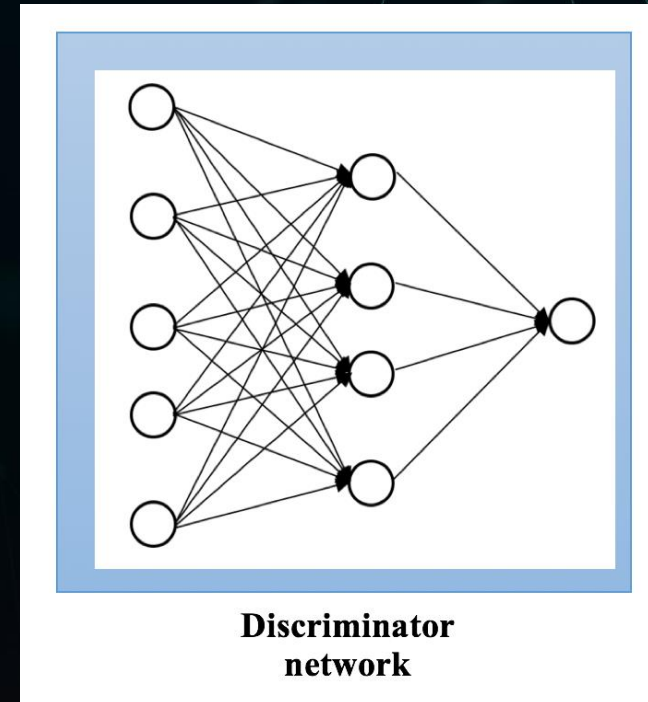


Discriminator network

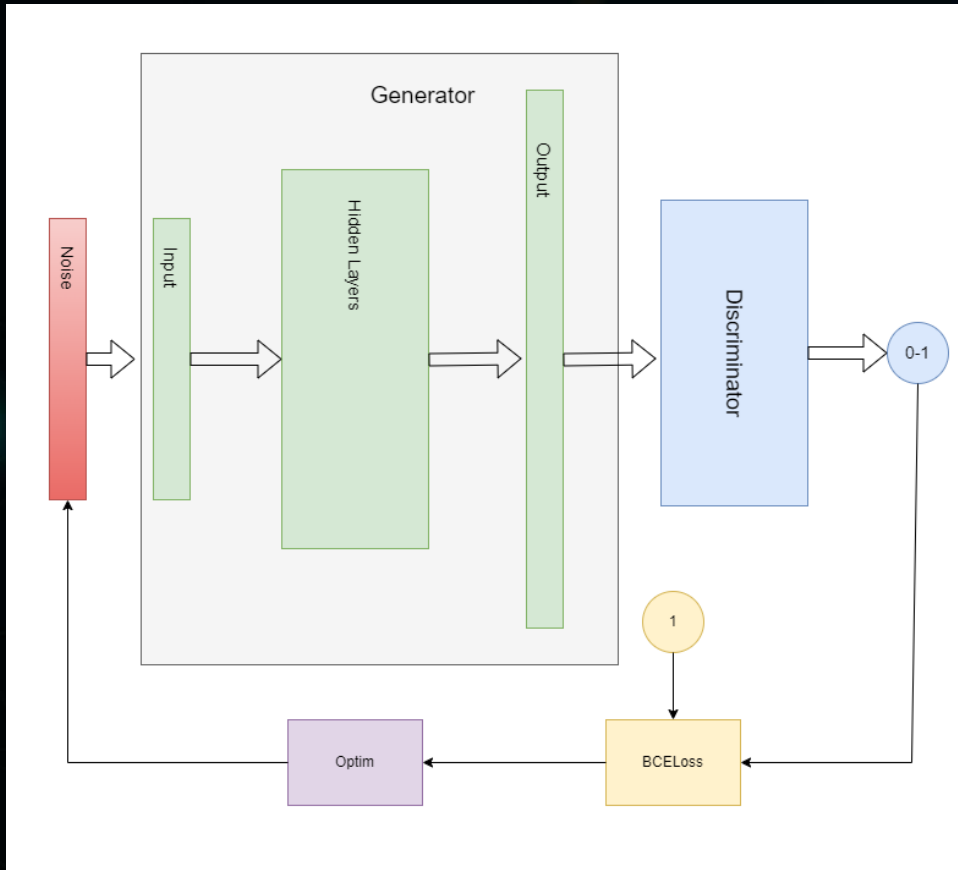
Discriminator class

- נבנה רשת נוירונים FNN כך:
 - הקלט בגודל $28*28 = 784$.
 - שכבה אחת נסתרת.
 - פלט בגודל 1 עם ערכים בין 0 ל-1 באמצעות אקטיבציה של Sigmoid.

```
class discriminator(nn.Module):  
    def __init__(self):  
        super().__init__()  
  
        self.fc1 = nn.Linear(28*28,256)  
        self.fc2 = nn.Linear(256, 256)  
        self.out = nn.Linear(256,1)  
  
    def forward(self,x):  
        x = F.leaky_relu(self.fc1(x))  
        x = F.leaky_relu(self.fc2(x))  
        x = F.sigmoid(self.out(x))  
        return x
```



אימון ה Generator



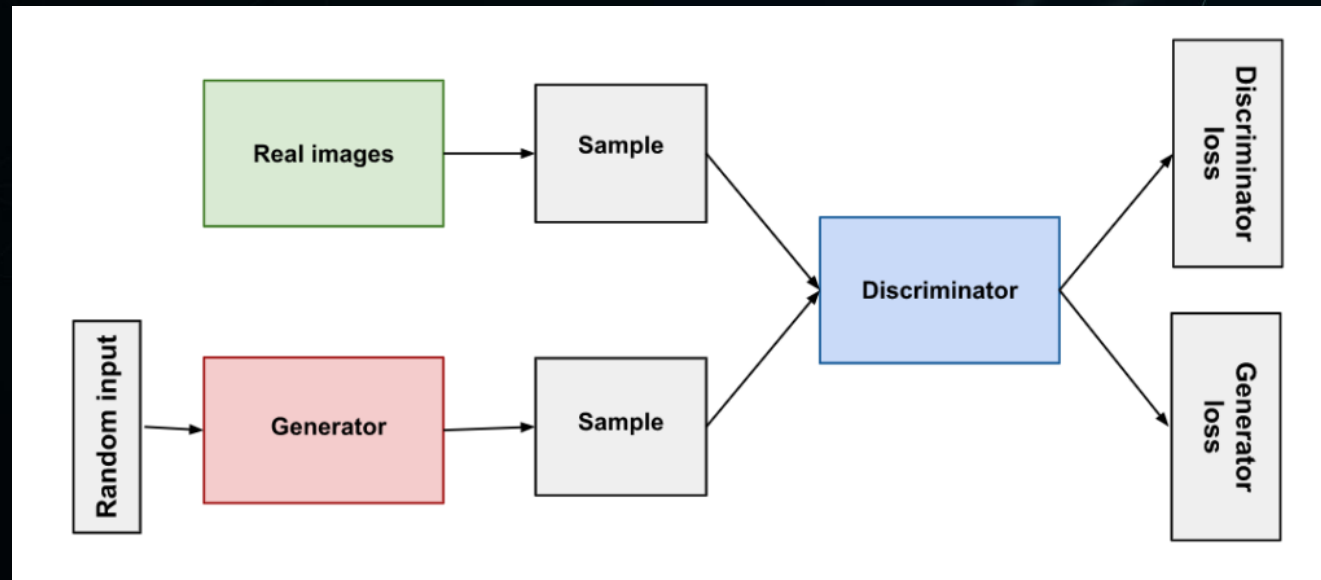
- המודל מייצר תמונה מסויימת מערכים רנדומליים.
- הפלט של המודל (התמונה) נכנס ל Discriminator אשר מחזיר לנו ערך בין 0 – 1, כאשר:
 - ערך קרוב ל- 1 התמונה דומה לספרה אמיתית.
 - ערך קרוב ל- 0 התמונה אינה של ספרה אמיתית.
- פונקציית המחיר BCELoss מקבלת שני ערכים:
 - פלט של ה Discriminator בין 0 ל- 1.
 - הערך 1 (target) – המייצג תמונה אמיתית.
- באמצעות פונקציית המחיר ו- SGD משנים את המשקלים (W) של המודל כך שהפלט של המודל יביא אותנו לתוצאה קרובה יותר ל- 1 ב Discriminator המייצגת תמונה אמיתית.

אימון Discriminator

- החידוש הגדול ב-GAN הינו האלגוריתם לאימון ה-Discriminator, והשילוב שלו עם אימון ה-Generator.
- כדי לאמן Discriminator עלינו לקבל שני סוגים של תמונות:
 - תמונות עם $Label=1$; תמונות אמיתיות של ספרות. תמונות אילו נקבל מהמאגר MNIST.
 - תמונות עם $Label=0$; תמונות שאינן של ספרות.
- הרעיון החדשני ב-GAN הוא שאת התמונות שאינן של ספרות ניקח מהפלט של ה-Generator, ולא סתם תמונות שרירותיות שאינן של ספרות.
- למעשה אנחנו מאמנים את ה-Discriminator לאבחן בין:
 - תמונות של ספרות MNIST (True). $Label = 1$.
 - תמונות של ה-Generator (Fake). $Label = 0$.

אימון משולב של רשת GAN

- ה Discriminator מתאמן לאבחן בין תמונות אמיתיות לתמונות של ה Generator שנחשבות לא אמיתיות (Fake).
- ה Generator מתאמן כדי ליצור תמונות שה Discriminator יחשוב שהן אמיתיות. למעשה הוא מנסה "לעבוד" על ה Discriminator.
- למעשה יש כאן מאבק בין ה Generator שמנסה להשתפר וליצור תמונות שיעבדו על ה Discriminator, וה Discriminator שמתאמן כל הזמן לאתר תמונות של ה – Generator.



GAN-Trainer code

```
for epoch in range(epochs):
    for i, (real_images, labels) in enumerate(train_loader):

        # create minibatches of REAL and FAKE images
        real_images = real_images.view(-1,28*28).to(device)
        noise = torch.randn(batch_size,64).to(device)
        fake_images = gnet(noise) # output of generator

        # labels used for real and fake images
        real_labels = torch.ones(batch_size,1).to(device)
        fake_labels = torch.zeros(batch_size,1).to(device)
```

MNIST LOADER

dnet – Discriminator
gnet - Generator

```
# loss function (same for both phases of training)
loss = nn.BCELoss()

# optimizers (same algo but different variables b/c different parameters)
d_optimizer = torch.optim.Adam(dnet.parameters(), lr=lr)
g_optimizer = torch.optim.Adam(gnet.parameters(), lr=lr)
```

```
### ----- Train the discriminator ----- ###

# forward pass and loss for REAL pictures
pred_real = dnet(real_images) # REAL images into discriminator
d_loss_real = loss(pred_real,real_labels) # all labels are 1

# forward pass and loss for FAKE pictures
pred_fake = dnet(fake_images) # FAKE images into discriminator
d_loss_fake = loss(pred_fake,fake_labels) # all labels are 0

# compute loss (using combined losses) and collect
d_loss = (d_loss_real + d_loss_fake) / 2

# backprop
d_optimizer.zero_grad()
d_loss.backward()
d_optimizer.step()
```

```
### ----- Train the generator ----- ###

# create fake images and compute loss
fake_images = gnet(torch.randn(batch_size,64).to(device))
pred_fake = dnet(fake_images)

# compute and collect loss and accuracy
g_loss = loss(pred_fake,real_labels)

# backprop
g_optimizer.zero_grad()
g_loss.backward()
g_optimizer.step()
```

CGAN – Convolution GAN

- בשיעור הבא יצירת GAN הכולל שכבות קונבולוציה ליצירת תמונות ברמה גבוהה יותר.

