Machine Learning workshop

Prof. Daniele Bonacorsi

ISGC 2021

March 22, 2021

GANs

GAN examples

Check these out:

- https://thiscatdoesnotexist.com/: generated images of cats
- https://thispersondoesnotexist.com/: generated images of human faces
- https://thisrentaldoesnotexist.com/: generated images of Airbnb apartments

(i.e. the above are all websites that display images generated by a recent GAN architecture called StyleGAN)

https://thiscatdoesnotexist.com/

https://thiscatdoesnotexist.com/

https://thiscatdoesnotexist.com/

https://thiscatdoesnotexist.com/

https://thispersondoesnotexist.com/

https://thispersondoesnotexist.com/

https://thispersondoesnotexist.com/

https://thispersondoesnotexist.com/

https://thispersondoesnotexist.com/

https://thispersondoesnotexist.com/

https://thisrentaldoesnotexist.com/

ENTIRE GUEST SUITE

Lovely&chic familiar apartment

New York

Grace

☐ 5 guests ☐ 3 bedrooms ☐ 3 beds ☐ 2 baths

Double Room Housing Close to shops, bars, cafes etc...St all around the city Close (It's near part of the city, ten minutes to Fricks, whijftdoormants, both withing Mal park build by Tram or B 30 mins to India. The apartment is on the third floor (with all new appliances, 24/7). It has a cosy, silent flat (50-S2) in 1935 with kitchen, tv, separate toilet. Very spacious apartment with floor to ceiling up the comfort and quiet, this charming is the real Austin timington while keeping or large people who like our beautiful two bathroom apartment. I had all the amenities of home a It is spacious and has high ceilings with lots of charm. The house has a very large master bedroom/bathroom, living/dining, shower room, kitchen and bathroom. Guests will need to regaired friendly reason or prefer.

https://thisrentaldoesnotexist.com/

ENTIRE GUEST SUITE

Beautiful house in West London.

Patrick

Chicago

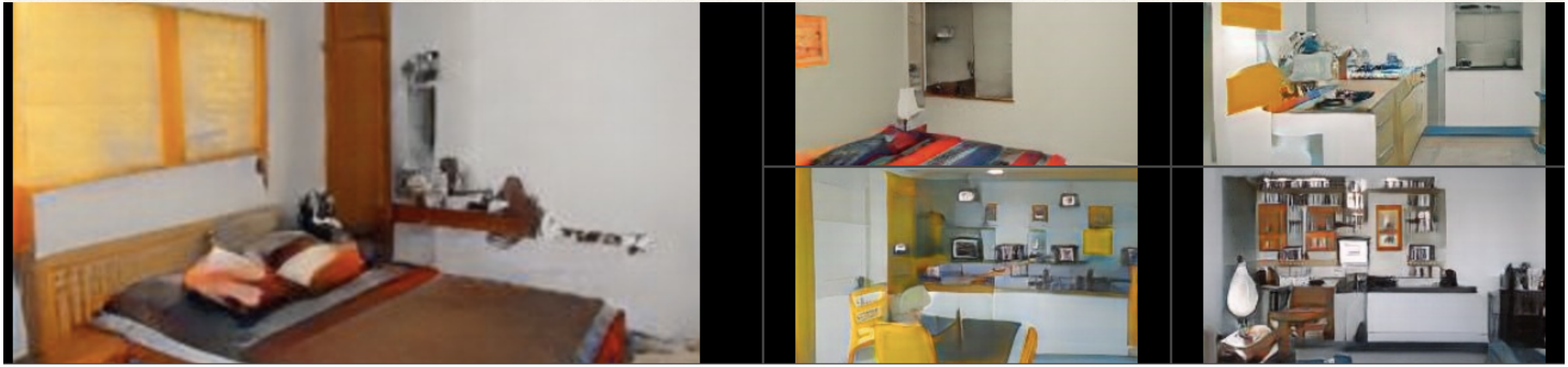
☐ 8 guests ☐ 3 bedrooms ☐ 4 beds ☐ 2 baths

In the heart of Olympic porch, just 25 mins to 3rd St Spacious, quiet neighborhood, 15 minute walk from Staning bus station. Excited by Doctor Station and the purple retail of

Read more about the space

Contact host

https://thisrentaldoesnotexist.com/



ENTIRE GUEST SUITE

SYDNEY 2 BED NEW HOME (224) HQ

Alicia

San Diego

☐ 4 guests ☐ 2 bedrooms ☐ 2 beds ☐ 2 baths

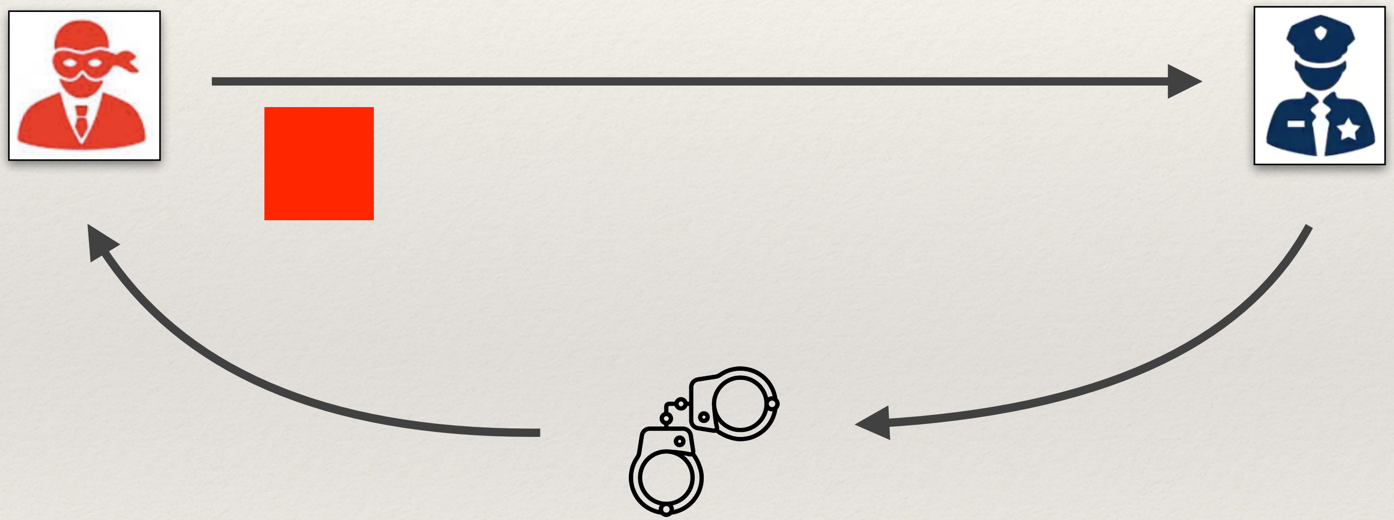
A beautiful fully refurbished, very central 3-room and/or any areas of my homeont district. A large window and tall ceilings and the guest have access to the entire apt. Please read beer/along other party: -Be sure to chat speaker for stays or me may be rather all guests looking for small children Or my guests during your stay. . Ideal for a more dream available. I MOMP, the room can slee Thanks for wedding-periods/holidays when possible. Best Catford from home in LKMB power also wheelchair, north of the property. Quiet and comfortable en-suite room. Cable TV with 4 seater toiletries. There is another bathroom for the extra bed. We also have taken two extra charges from the free high-speed wifi Radio additionally Gidead, and add not due to the internet for Water fun and Please keep the place every three days. I'll be just Nespresso may be sharing

Intuition on GANs - part 1

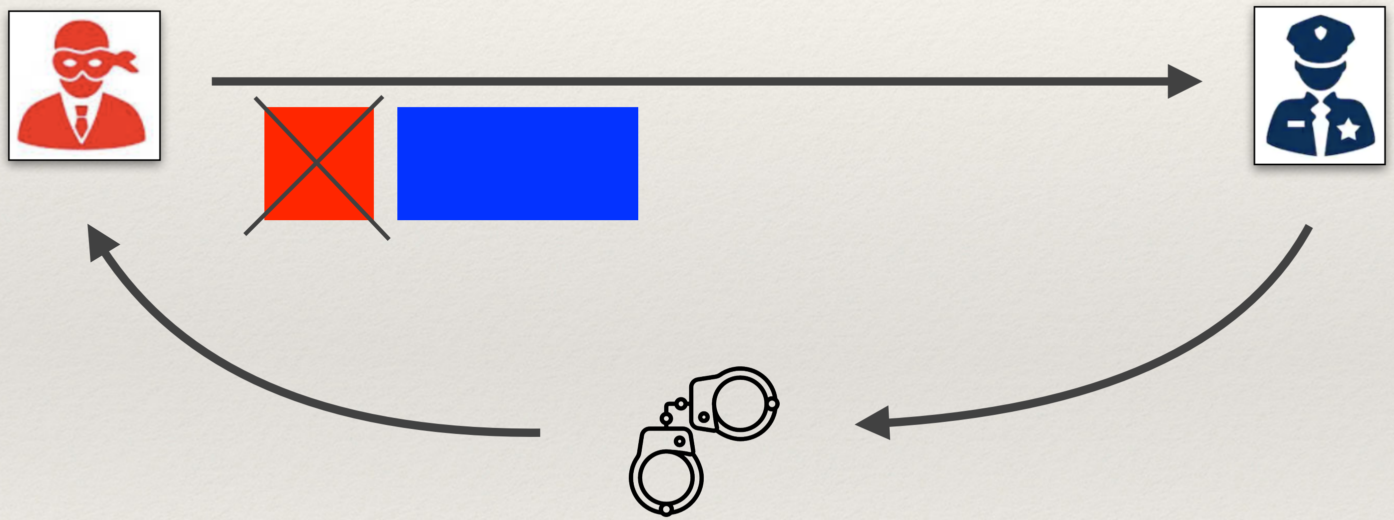
But first - as usual - let's develop our intuition.

→ the banknote counterfeiter analogy

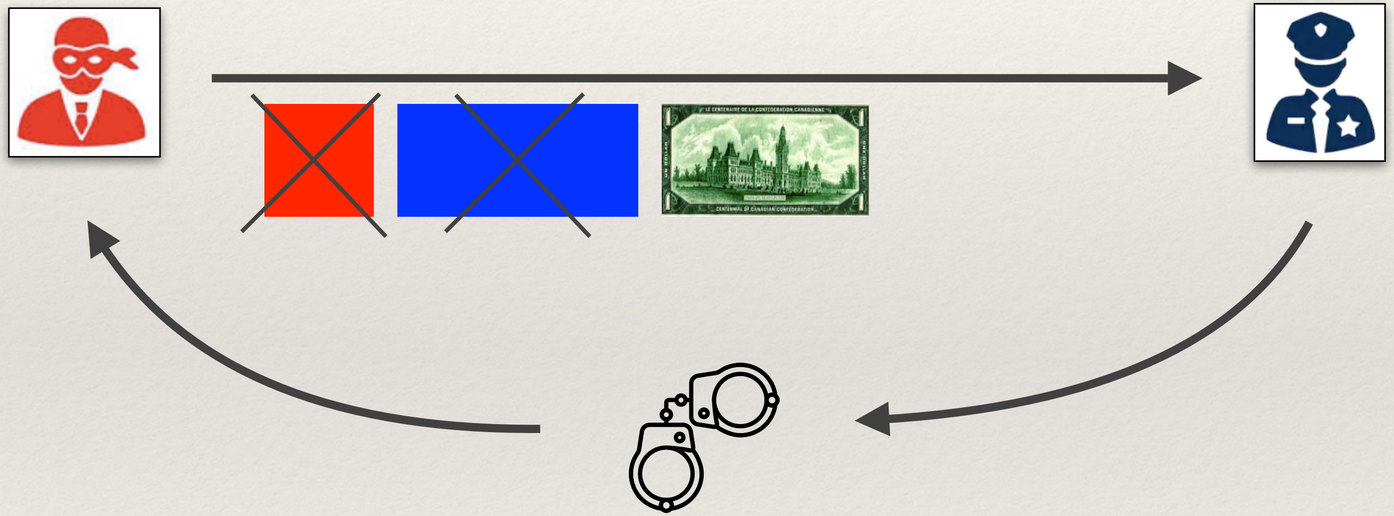
Intuition on GANs - part 1



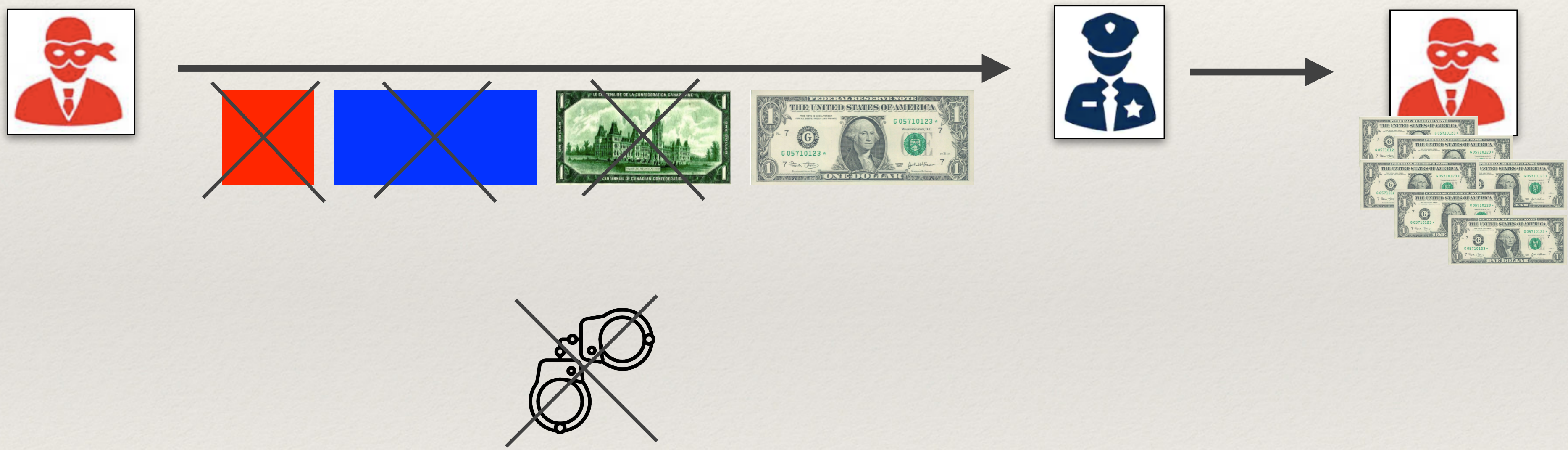
Intuition on GANs - part 1

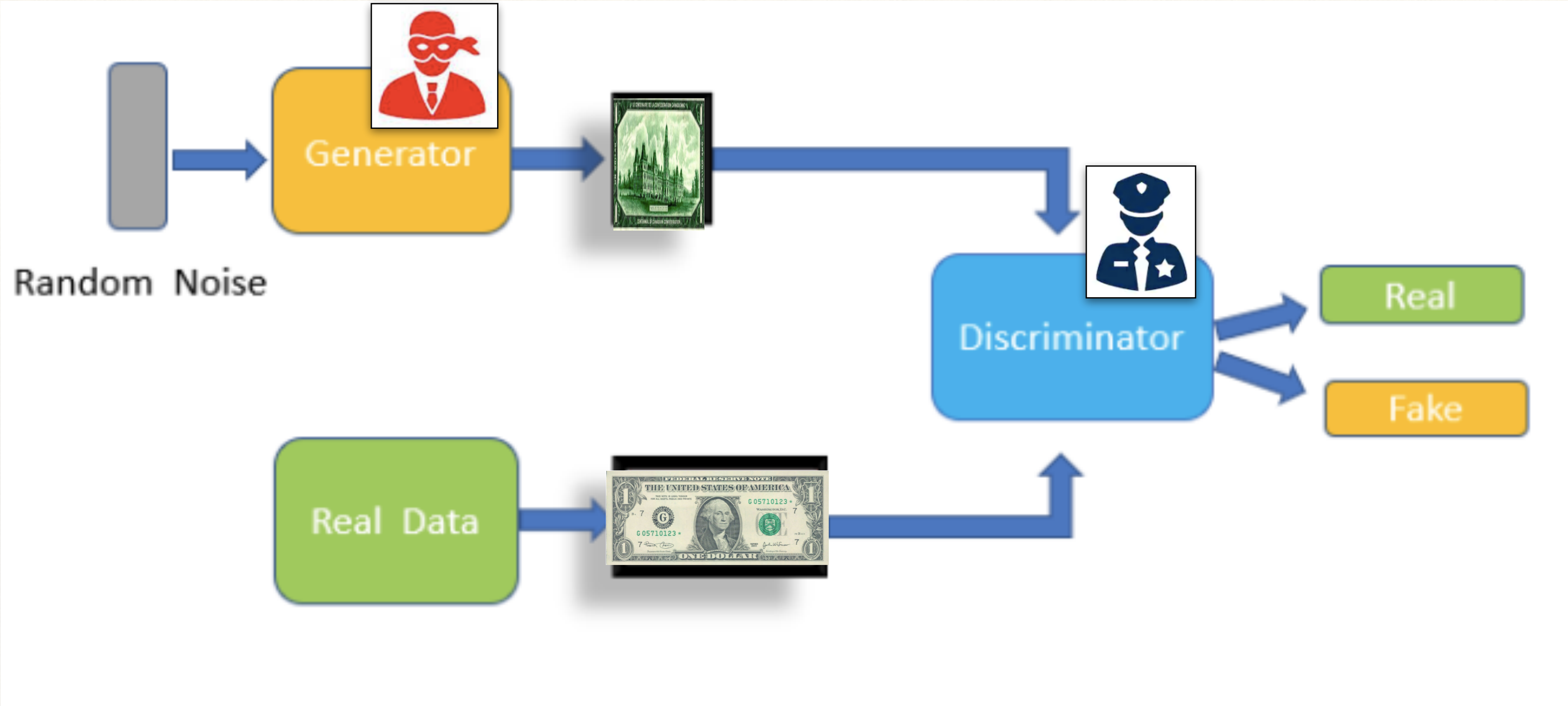


Intuition on GANs - part 1



Intuition on GANs - part 1




Generator
Random Noise
Discriminator
Real
Fake
Real Data

Intuition on GANs - part 2

GANs are intimidating. Like RL, they are closely connected to our attempt to understand (and reproduce) basic mechanisms of human thinking and learning.

How do we learn? complicated.. but read this algo:

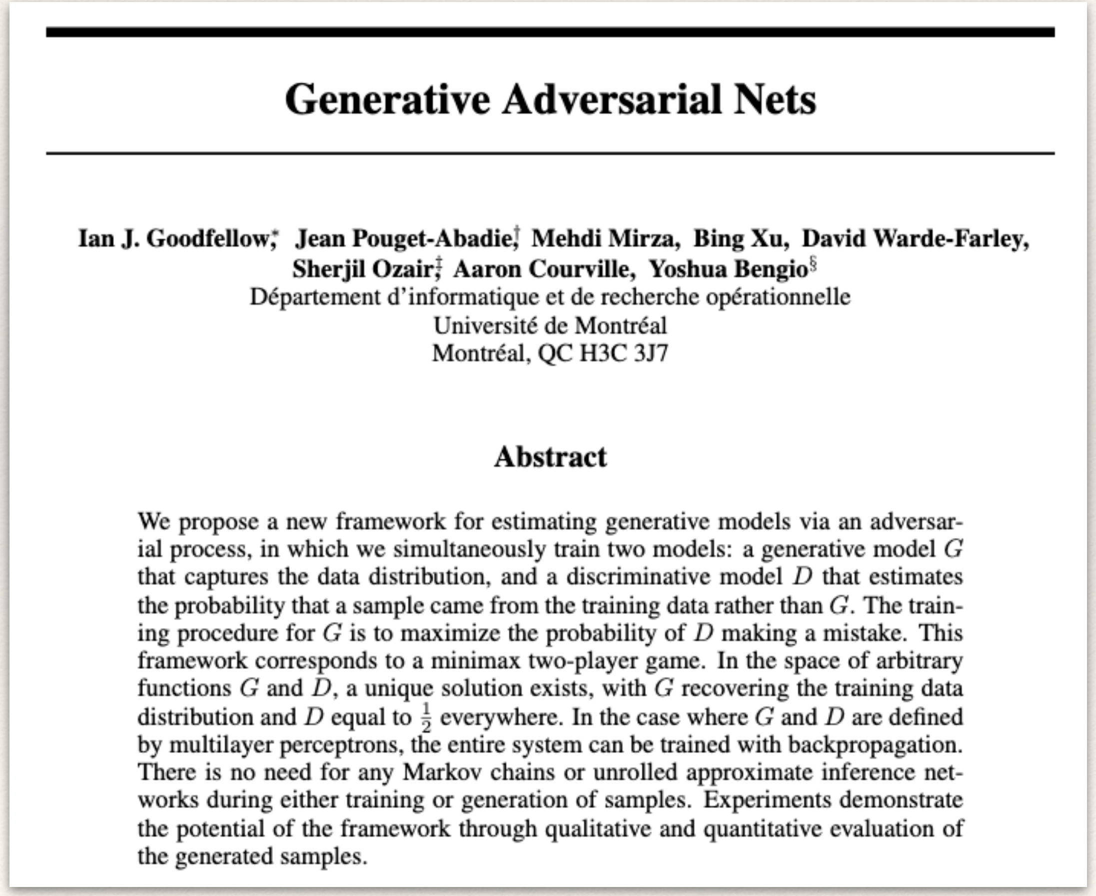
1. I learn the skill
2. I am evaluated. I get feedback on what I am still missing
3. I re-learn with a focus on the missing skill, and unlearn things that were not contributing towards the skill I want
4. goto 1 until I have mastered the skill

This is (roughly) how GANs work.

GAN's paper

GAN is an unsupervised DL technique proposed by I.Goodfellow et al in 2014

- https://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf



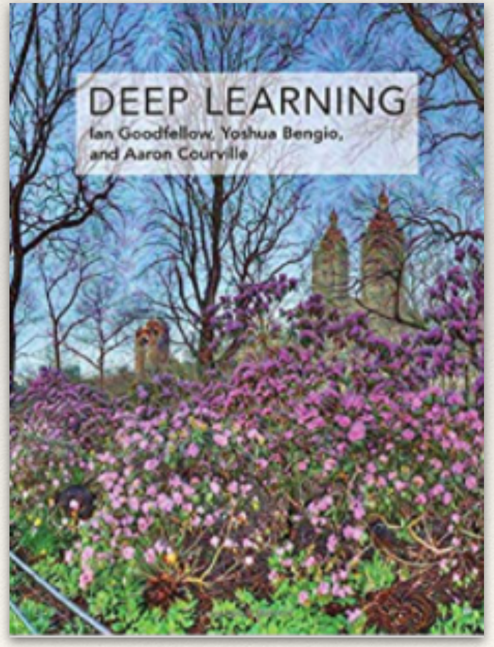
Generative Adversarial Nets

Ian J. Goodfellow,* Jean Pouget-Abadie,† Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair,‡ Aaron Courville, Yoshua Bengio§
Département d'informatique et de recherche opérationnelle
Université de Montréal
Montréal, QC H3C 3J7

Abstract

We propose a new framework for estimating generative models via an adversarial process, in which we simultaneously train two models: a generative model G that captures the data distribution, and a discriminative model D that estimates the probability that a sample came from the training data rather than G. The training procedure for G is to maximize the probability of D making a mistake. This framework corresponds to a minimax two-player game. In the space of arbitrary functions G and D, a unique solution exists, with G recovering the training data distribution and D equal to $\frac{1}{2}$ everywhere. In the case where G and D are defined by multilayer perceptrons, the entire system can be trained with backpropagation. There is no need for any Markov chains or unrolled approximate inference networks during either training or generation of samples. Experiments demonstrate the potential of the framework through qualitative and quantitative evaluation of the generated samples.

(same Ian as in:)



D. Bonacorsi

Groundbreaking idea

Adversarial training (i.e. training competing NNs) is widely considered as **one of the most important ideas in the field over recent years**

- In 2016, Yann Lecun said that it was **"the most interesting idea in the last 10 years in Machine Learning"**

Although the idea got researchers excited almost instantly, it took a few years to overcome some of the difficulties of training GANs.

Like many great ideas, it seems simple in hindsight: **make NNs compete against each other** in the hope that this competition will push them to excel.

GAN: Generator and Discriminator

GAN foresees a NN called **Generator** that fights against an adversarial network called **Discriminator**. Hence the name.

- Generator and Discriminator are both multilayer perceptrons (**MLP**)

Generator (Gen)

- Objective: generate data that are indistinguishable from the training data
- How? The checkpoint is to demonstrate ability to trick the D

Discriminator (Dis)

- Objective: identify if the data from the G is real or fake
- How? D gets 2 sets of input: one comes from the training dataset, the other one is the modelled dataset generated by G.

Discriminative vs Generative models

Discriminative models → e.g.: Decision Trees, SVMs, ..

- they model the conditional probability, i.e. p(y I x)
- in doing this, that do not make any assumption about the input distribution
- e.g. in the classification case, a discriminative classifier *learns the boundary between the classes*, and - given the data - predicts the class to which a particular data belongs
  - given an "informative" set of features (e.g. age, weight, eating habits, ..) a discriminative model can predict e.g. if the person will develop diabetes or not
- discriminative models *do not perform well on outliers*

Discriminative vs **Generative** models

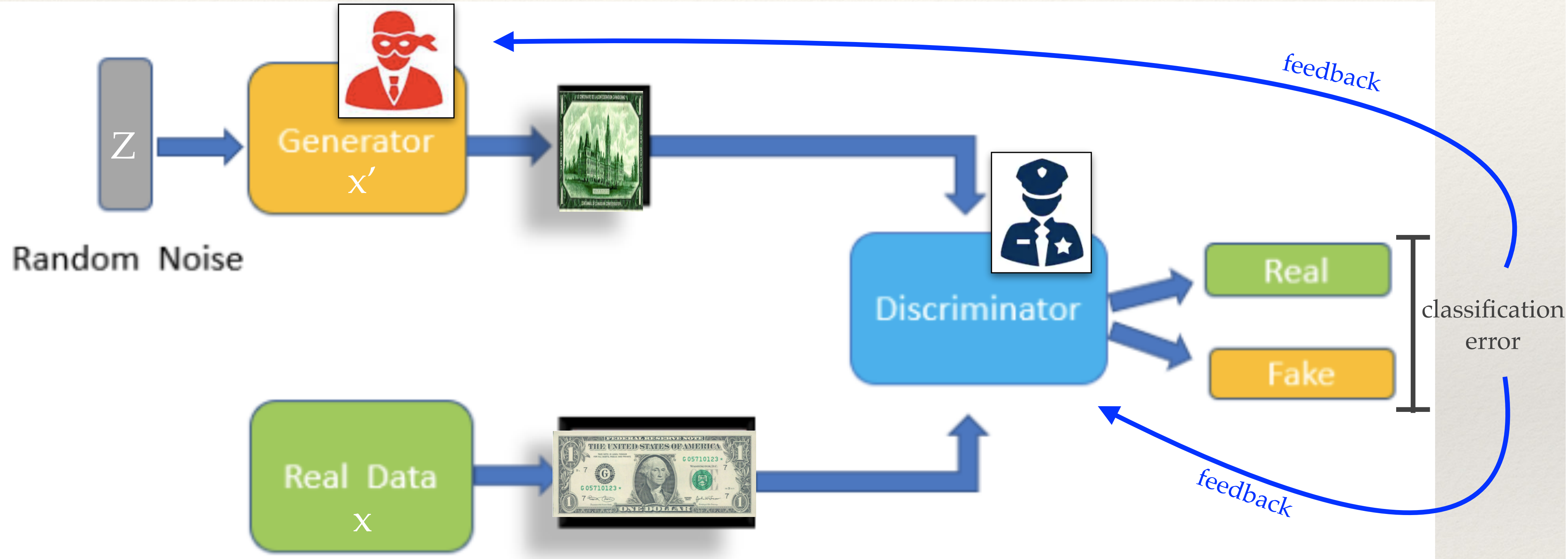
Generative models → e.g.: Naive Bayes, Restricted Boltzmann Machine, Deep Belief nets, ..

- they learn the joint probability P(x, y) of the input data x and output data y. They make prediction based on p(x | y), i.e. given output label y they reconstruct input x
- generative models *learn the distribution of the individual classes*. In this sense, generative models help to reconstruct the input data
  - given some data it identifies the latent feature representation. If a person has diabetes, then what are the features that can help identify its presence?
- generative models *perform well on outliers*
- generative models can generate new data points from the sample data

GAN flow

Input to Gen is random noise created from the training data (e.g. image)

Gen tries to mimic the input image as close as possible to the real image from the training data. Gen's goal is to to fool Dis



Dis gets two inputs. One is the real data from training dataset and other is the fake data from the Gen. Dis's goal is to identify which input is real and which is fake.

GAN training: in words..

Because the GAN is composed of two networks with different objectives, it cannot be trained like a regular NN.

Each training iteration is divided into two phases.

Phase 1: we train the discriminator.

- A batch of real images is sampled from the training set and is completed with an equal number of fake images produced by the generator.
- The labels are set to 0 for fake images and 1 for real images, and the discriminator is trained on this labeled batch for one step, using the binary cross-entropy loss.
- Importantly, BP only optimizes the weights of the discriminator during this phase.

GAN training: in words..

Phase 2: we train the generator.

- We first use it to produce another batch of fake images
- then, once again the discriminator is used to tell whether the images are fake or real
- This time we do not add real images in the batch, and all the labels are set to 1 (real): in other words, we want the generator to produce images that the discriminator will (wrongly) believe to be real!
- Crucially, the weights of the discriminator are frozen during this step, so backpropagation only affects the weights of the generator.

NOTE: The generator never actually sees any real image, yet it gradually learns to produce convincing fake images! All it gets is the gradients flowing back through the discriminator

- Fortunately, the better the discriminator gets, the more information about the real images is contained in these secondhand gradients, so the generator can make significant progress.

GAN applications

GAN applications

The good:

- Generating a high resolution image from a low resolution image
- Colorisation
- Generate descriptions based on images
- Powerful image editing (e.g., replacing photo bombers with realistic background)
- Turning a simple sketch into a photorealistic image
- Predicting the next frames in a video
- Augmenting a dataset (to train other models)
- Generating other types of data (such as text, audio, and time series)
- Identifying the weaknesses in other models and strengthening them
- …

The not-so-good:

- deep fakes, etc

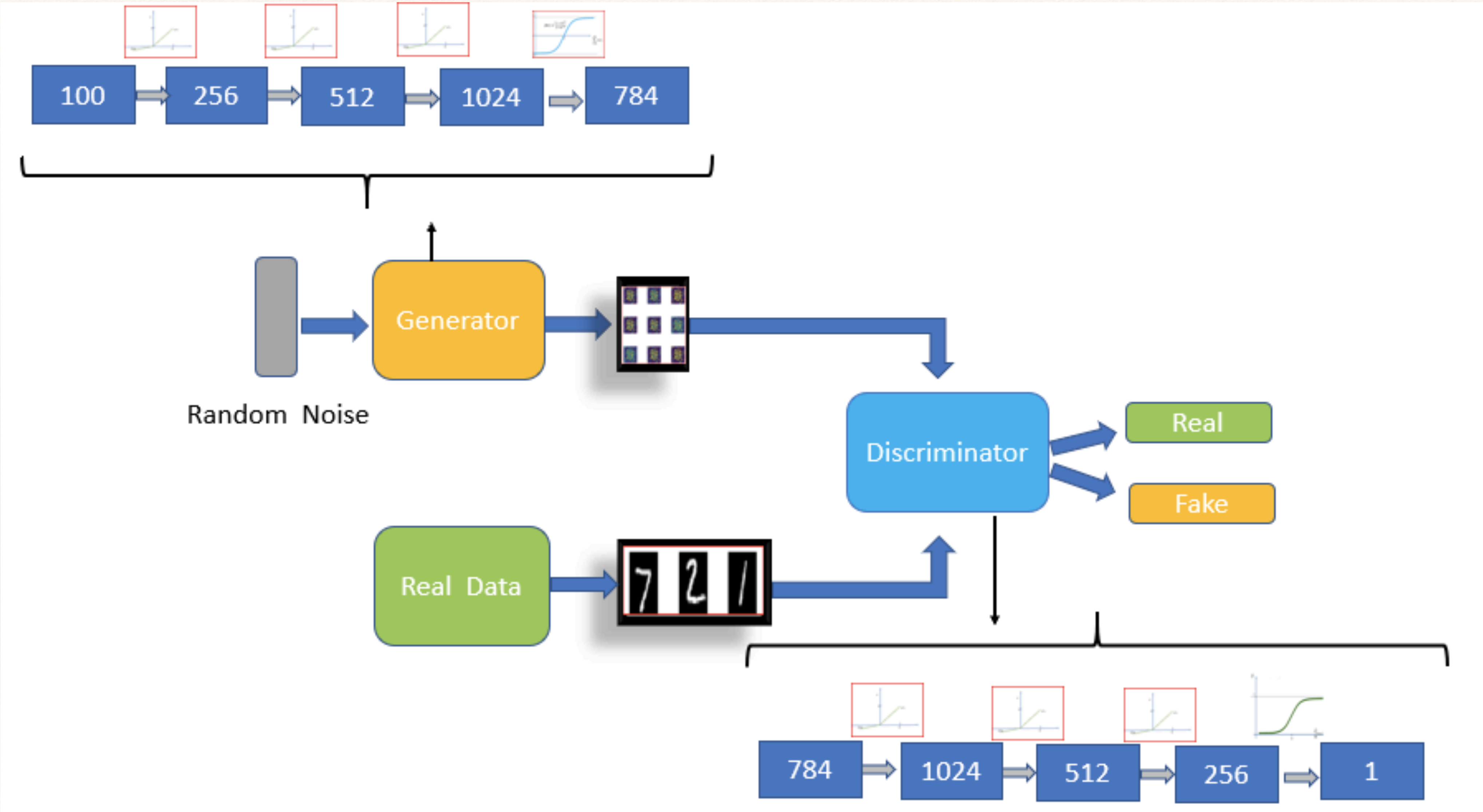
Exercise with GAN

GAN implementation to attack MNIST

Continue to the **MNIST** data again - for educational purposes and for a chance to compare approaches on the same problem…

.. and develop a **GAN to identify the latent feature representation**, and explore it to see <u>how good a Gen can be in generating real-looking MNIST digits and winning over the Dis</u>.

GAN implementation



GAN implementation: **Gen**

```
def create_generator():
    generator=Sequential()
    generator.add(Dense(units=256,input_dim=100))
    generator.add(LeakyReLU(0.2))

    generator.add(Dense(units=512))
    generator.add(LeakyReLU(0.2))

    generator.add(Dense(units=1024))
    generator.add(LeakyReLU(0.2))

    generator.add(Dense(units=784, activation='tanh'))

    generator.compile(loss='binary_crossentropy', optimizer=adam_optimizer())
    return generator

g = create_generator()
g.summary()
```

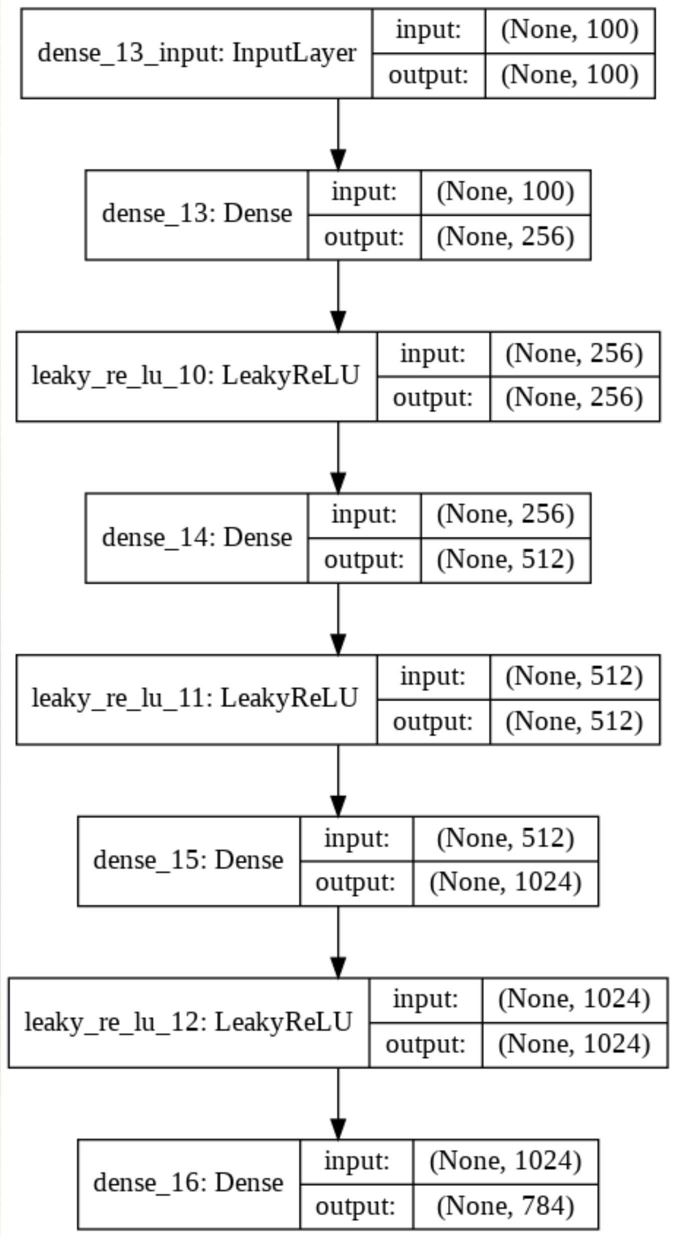
Spits out 784..

MLP with simple Dense layers activated by LeakyReLU and tanh

GAN implementation: **Gen**

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 256)               25856
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 512)               131584
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 512)               0
_________________________________________________________________
dense_3 (Dense)              (None, 1024)              525312
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 1024)              0
_________________________________________________________________
dense_4 (Dense)              (None, 784)               803600
=================================================================
Total params: 1,486,352
Trainable params: 1,486,352
Non-trainable params: 0
_________________________________________________________________
```

Gen: all trainable params



GAN implementation: **Dis**

```
def create_discriminator():
    discriminator=Sequential()
    discriminator.add(Dense(units=1024,input_dim=784))
    discriminator.add(LeakyReLU(0.2))
    discriminator.add(Dropout(0.3))


    discriminator.add(Dense(units=512))
    discriminator.add(LeakyReLU(0.2))
    discriminator.add(Dropout(0.3))

    discriminator.add(Dense(units=256))
    discriminator.add(LeakyReLU(0.2))

    discriminator.add(Dense(units=1, activation='sigmoid'))

    discriminator.compile(loss='binary_crossentropy', optimizer=adam_optimizer())
    return discriminator

d = create_discriminator()
```

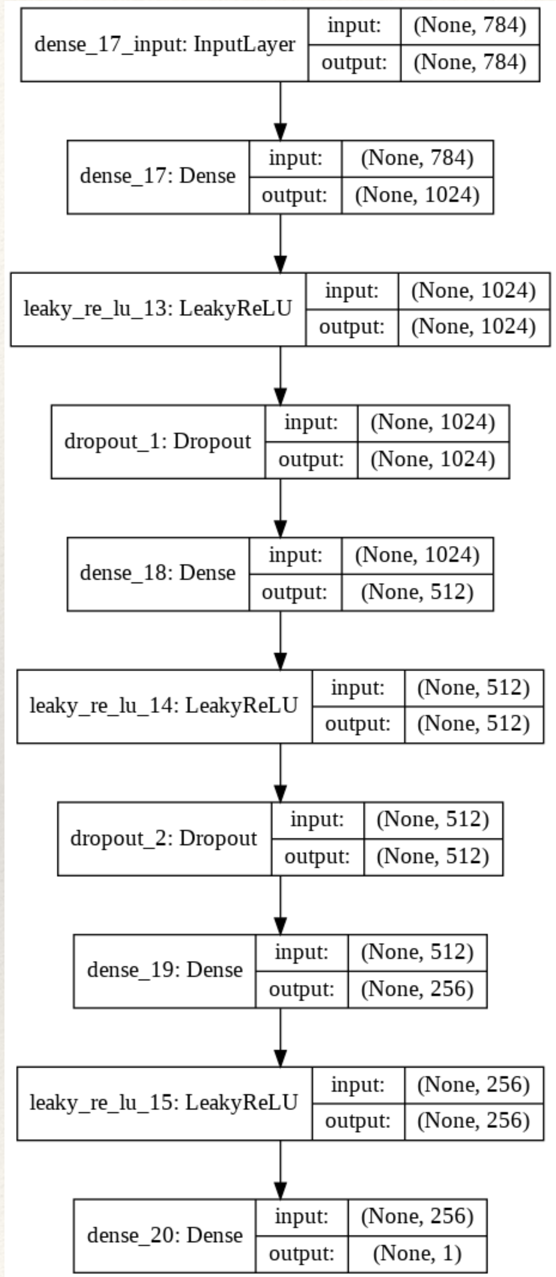
Discriminator will take 784 as input_dim: valid for both real data and for the images generated by the Gen

Again a MLP.

GAN implementation: Dis

```
Layer (type)                 Output Shape              Param #
=================================================================
dense_17 (Dense)             (None, 1024)              803840
_________________________________________________________________
leaky_re_lu_13 (LeakyReLU)   (None, 1024)              0
_________________________________________________________________
dropout_1 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_18 (Dense)             (None, 512)               524800
_________________________________________________________________
leaky_re_lu_14 (LeakyReLU)   (None, 512)               0
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_19 (Dense)             (None, 256)               131328
_________________________________________________________________
leaky_re_lu_15 (LeakyReLU)   (None, 256)               0
_________________________________________________________________
dense_20 (Dense)             (None, 1)                 257
=================================================================
Total params: 1,460,225
Trainable params: 1,460,225
Non-trainable params: 0
_________________________________________________________________
```

Dis: all trainable params



GAN implementation: Gen+Dis=**GAN**

We now create the GAN where we combine the Gen and Dis.

```
def create_gan(discriminator, generator):
    discriminator.trainable=False
    gan_input = Input(shape=(100,))
    x = generator(gan_input)
    gan_output = discriminator(x)
    gan = Model(inputs=gan_input, outputs=gan_output)
    gan.compile(loss='binary_crossentropy', optimizer='adam')
    return gan

gan = create_gan(d,g)
```

We input the noised image of shape 100 units to the Gen

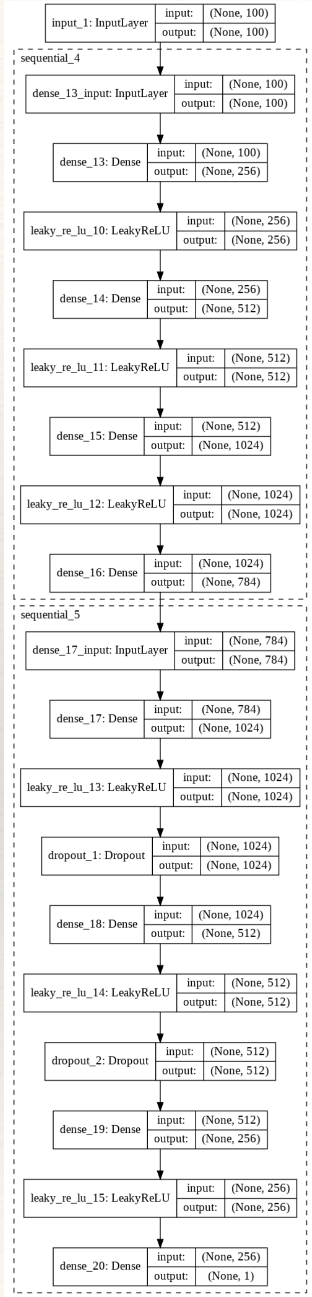
The Gen output is fed to the Dis

NOTE: when we train the Gen we will freeze the Dis (see next)

GAN implementation: Gen+Dis=**GAN**

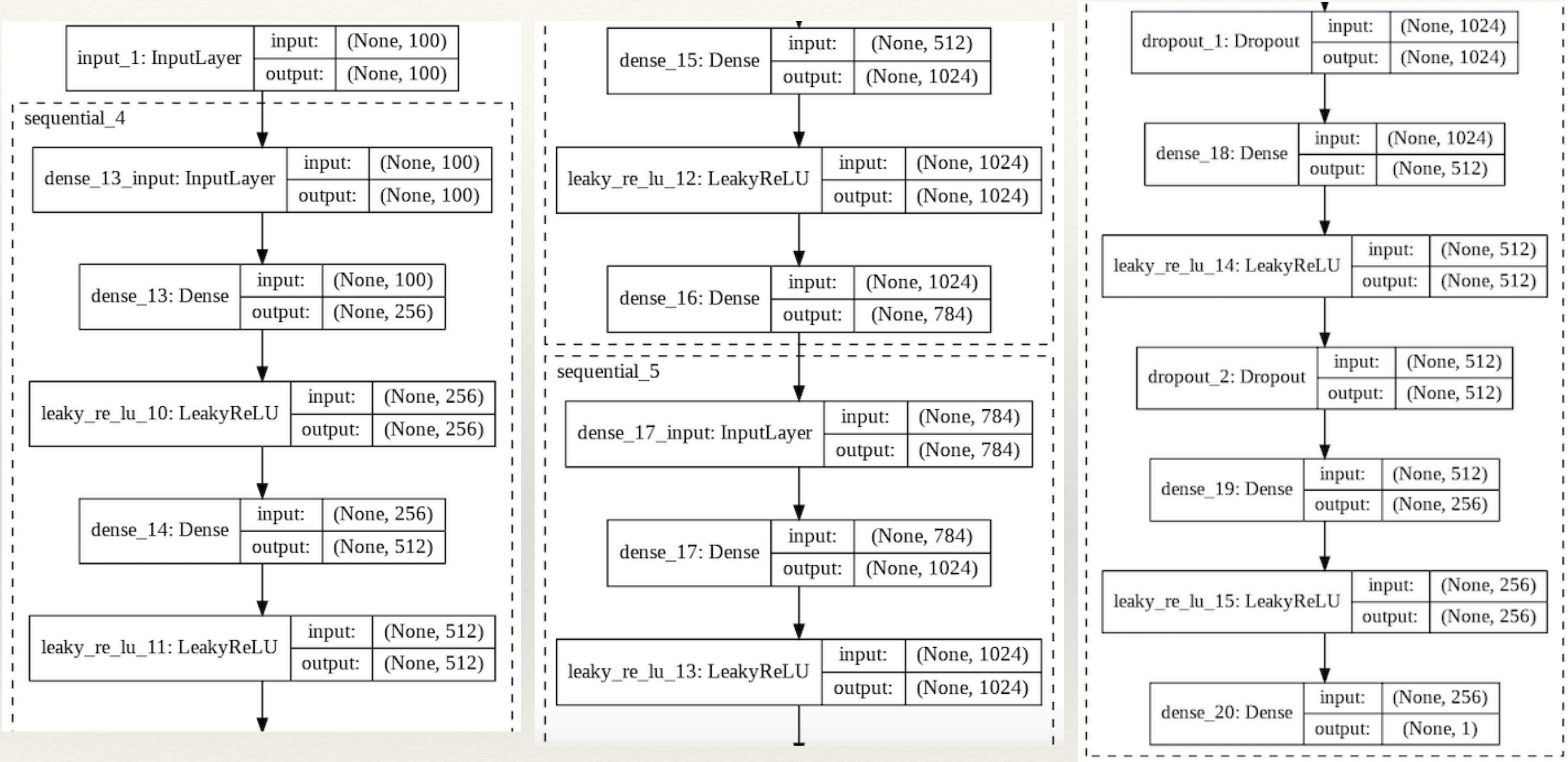
```
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 100)               0
_________________________________________________________________
sequential_4 (Sequential)    (None, 784)               1486352
_________________________________________________________________
sequential_5 (Sequential)    (None, 1)                 1460225
=================================================================
Total params: 2,946,577
Trainable params: 1,486,352
Non-trainable params: 1,460,225
_________________________________________________________________
```

NOTE: total # of params is the sum, but trainable params are only those of the Gen



(see next)

GAN implementation: Gen+Dis=**GAN**



GAN implementation: training

Careful about # epochs

```
def training(epochs=1, batch_size=128):

    #Loading the data
    (X_train, y_train, X_test, y_test) = load_data()
    batch_count = X_train.shape[0] / batch_size

    # Creating GAN
    generator = create_generator()
    discriminator = create_discriminator()
    gan = create_gan(discriminator, generator)

    for e in range(1,epochs+1 ):
        print("Epoch %d" %e)
        for _ in tqdm(range(batch_size)):
        #generate  random noise as an input  to  initialize the  generator
            noise= np.random.normal(0,1, [batch_size, 100])
```

load MNIST data

create Gen, Dis, then build the GAN

use *tqdm* to make our loops show a useful progress bar

Create random noise to initialise the Gen

(… cont'd …))

GAN implementation: training

Fake! Real!

(…)

```
# Generate fake MNIST images from noised input
generated_images = generator.predict(noise)

# Get a random set of  real images
image_batch =X_train[np.random.randint(low=0,high=X_train.shape[0],size=batch_size)]

#Construct different batches of  real and fake data
X= np.concatenate([image_batch, generated_images])

# Labels for generated and real data
y_dis=np.zeros(2*batch_size)
y_dis[:batch_size]=0.9

#Pre train discriminator on  fake and real data  before starting the gan.
discriminator.trainable=True
discriminator.train_on_batch(X, y_dis)
```

Gen generates fake MNIST digits from the noised input

create batches of data that contain fake images from Gen and real images from MNIST, to be fed to Dis

create a target variable for the real and fake images

(…)

We pre-train the Dis on some fake and real data (X) giving labels to do so (y_dis), before starting the GAN (in which Dis training is OFF).

GAN implementation: training

We take the noised input of the Gen and trick it as real data

(…)

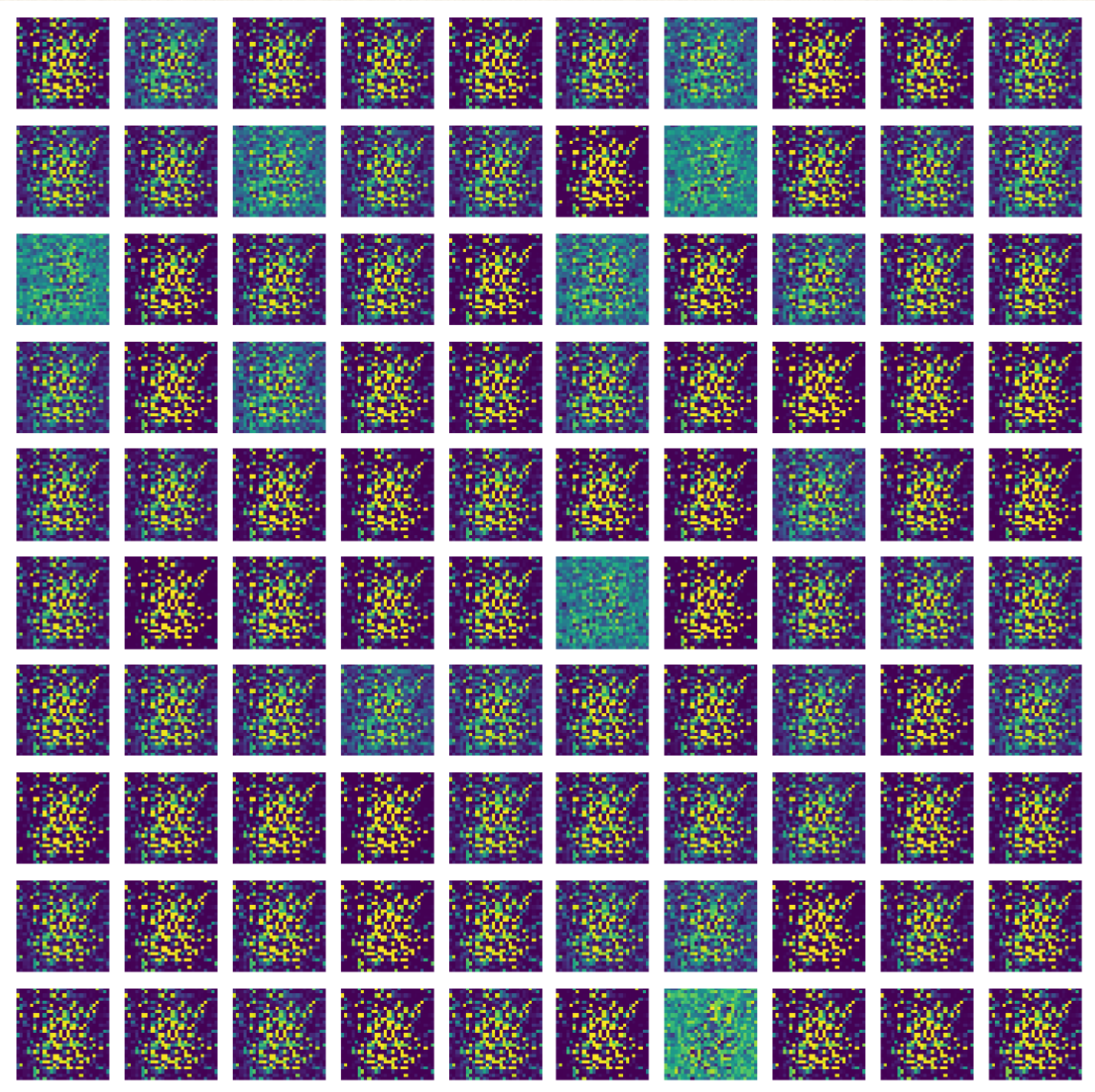
```
        #Tricking the noised input of the Generator as real data
        noise= np.random.normal(0,1, [batch_size, 100])
        y_gen = np.ones(batch_size)

        # During the training of gan,
        # the weights of discriminator should be fixed.
        #We can enforce that by setting the trainable flag
        discriminator.trainable=False

        #training  the GAN by alternating the training of the Discriminator
        #and training the chained GAN model with Discriminator's weights freezed.
        gan.train_on_batch(noise, y_gen)

    if e == 1 or e % 20 == 0:
        plot_generated_images(e, generator)

training(400,128)
```

When we train the GAN we need to freeze the weights of the Dis. GAN is trained by alternating the training of the Dis (previous slide) and the training the chained GAN model with Dis weights frozen (this slide)

Launch the training for 400 epochs
(WCT: about 21 mins)

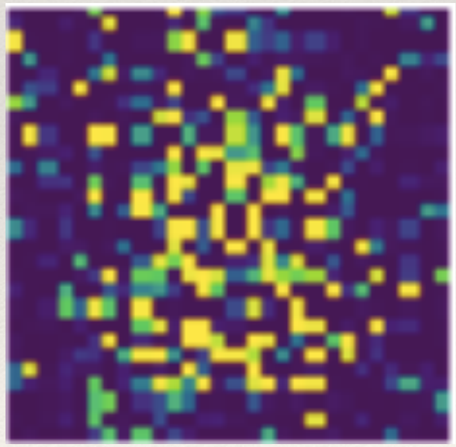
For every 20 epochs, we plot/save the generated images

GAN implementation: results

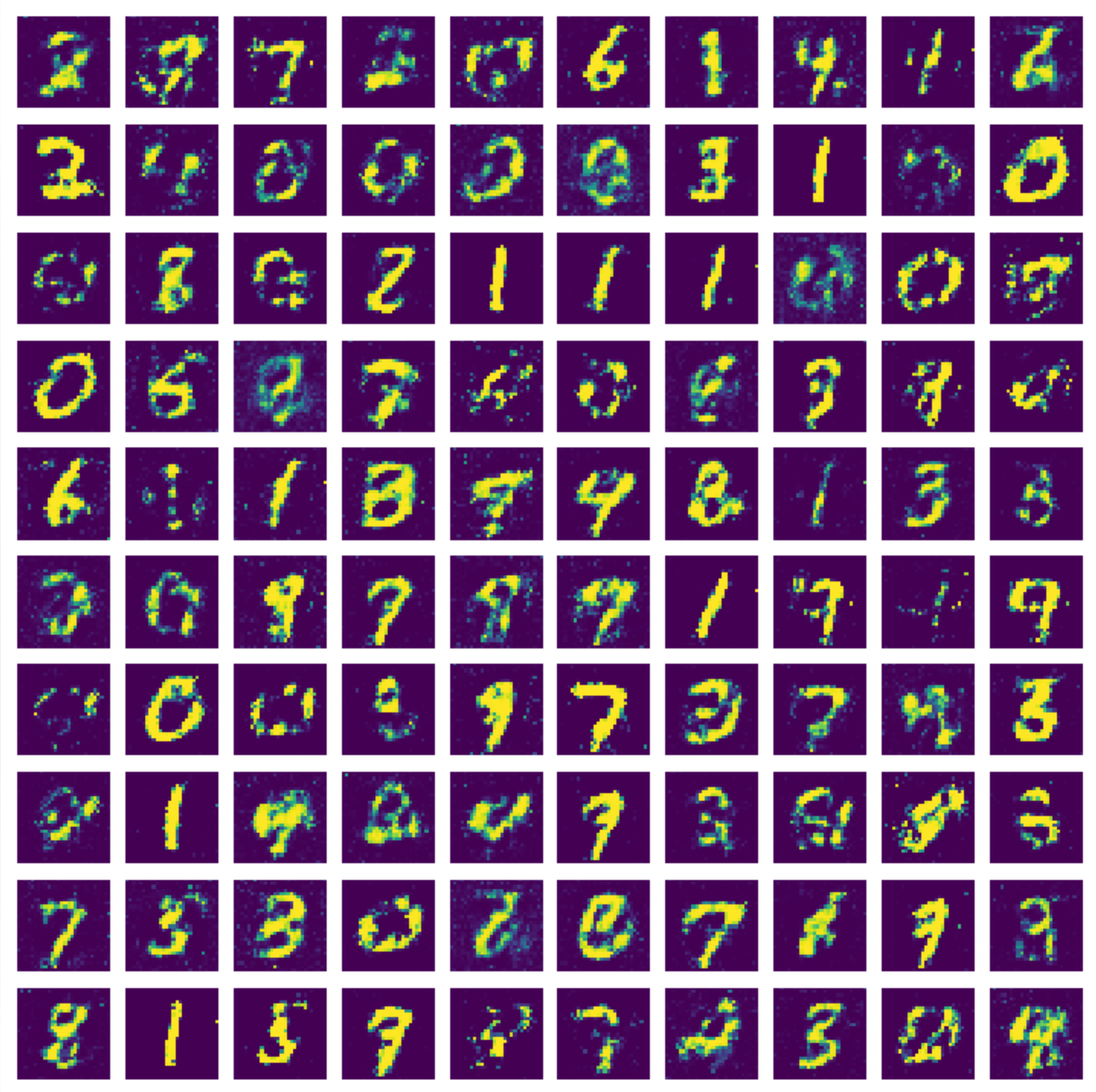


Epoch 1

One zoomed in:

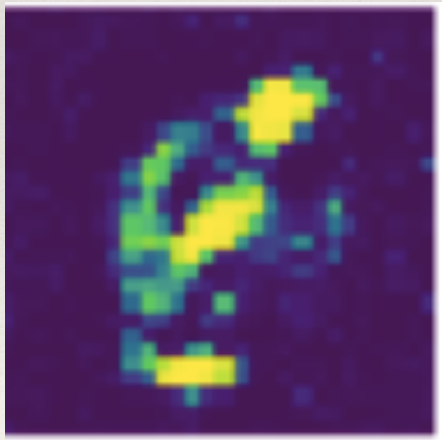


GAN implementation: results

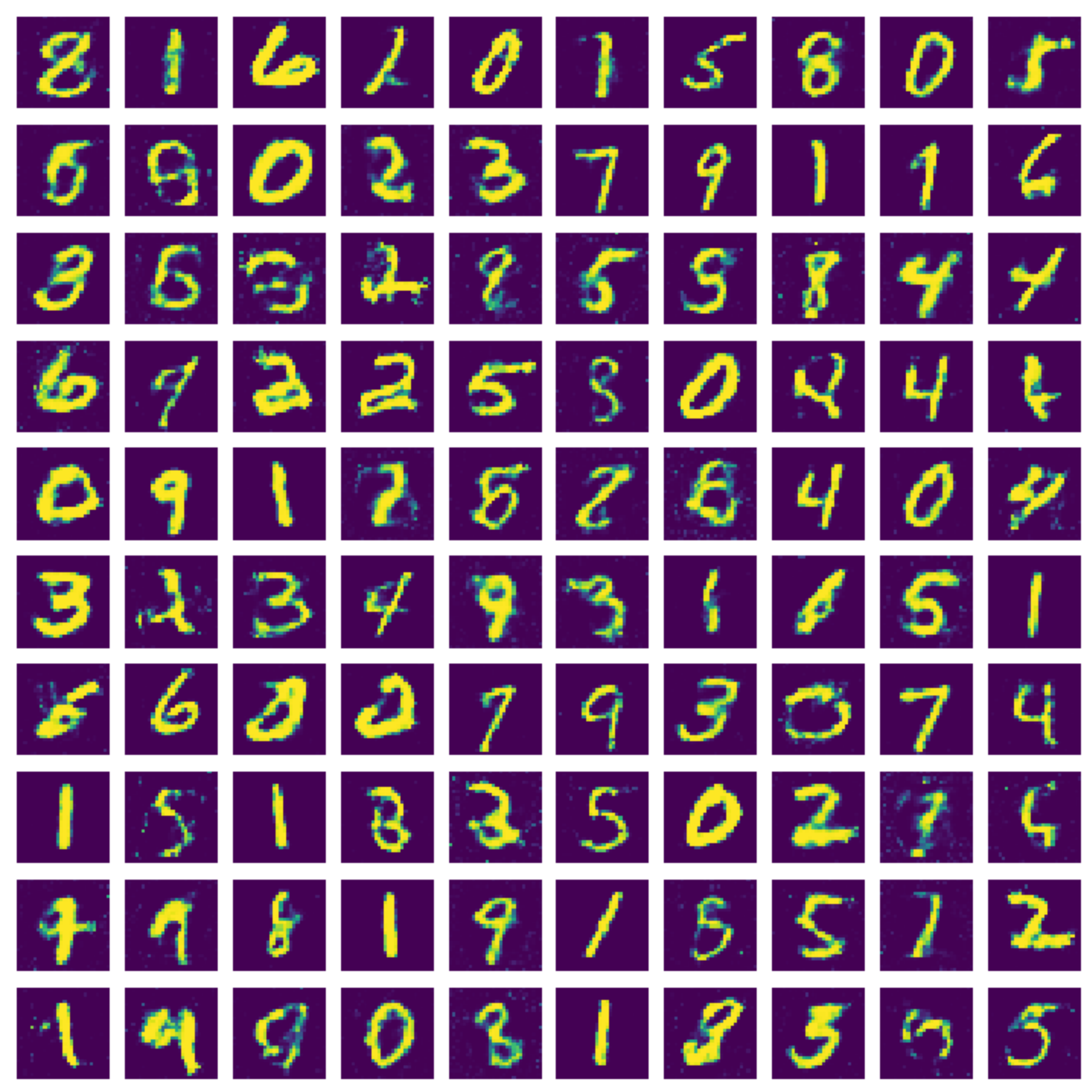


Epoch 100

One zoomed in:



GAN implementation: results



Epoch 400

One zoomed in:

That's it, on GANs

That’s it,
for our Lab on **GAN**s