Deep generative neural networks Fundamentals & problem solving

Class 3

JAMAL TOUTOUH
jamal@uma.es
jamal.es
@jamtou

Class 3 - Learning Outcomes

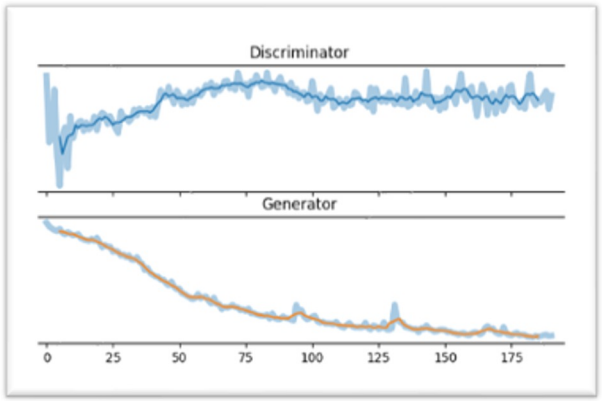
- Reviewing GANs training, pathologies, and remedies
- Introducing different types of GANs
- Developing GANs in python
- Creating data loaders to train deep learning models

GAN training, pathologies, and remedies

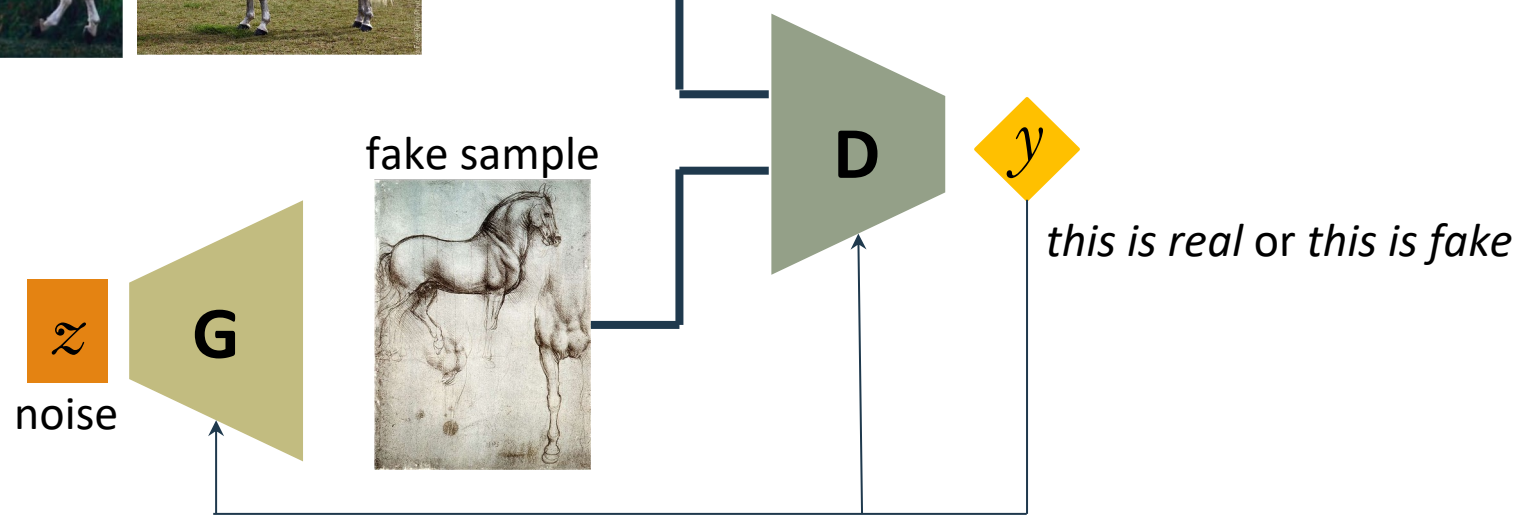
Generative Adversarial Networks

Generative Adversarial Networks: Build a generative model by raising an arms race between two neural networks, a **generator** and a **discriminator**

Goodfellow et al. 2014. **Generative Adversarial Nets**



real data



GAN Training. General Algorithm

Steps of the main training loop:

1. Train discriminator

1.1. Train discriminator on real data

1.1.1 Sample a batch of data from real dataset (x)

1.1.2 Get loss from the discriminator output with input x

1.2 Train the discriminator on data produced by the generator

1.2.1 Sample a batch of data from random latent space (z)

1.2.2 Get samples (x') from the generator with input z

1.2.3 Get loss from the discriminator output with input x'

1.3 Update discriminator weights according to the losses

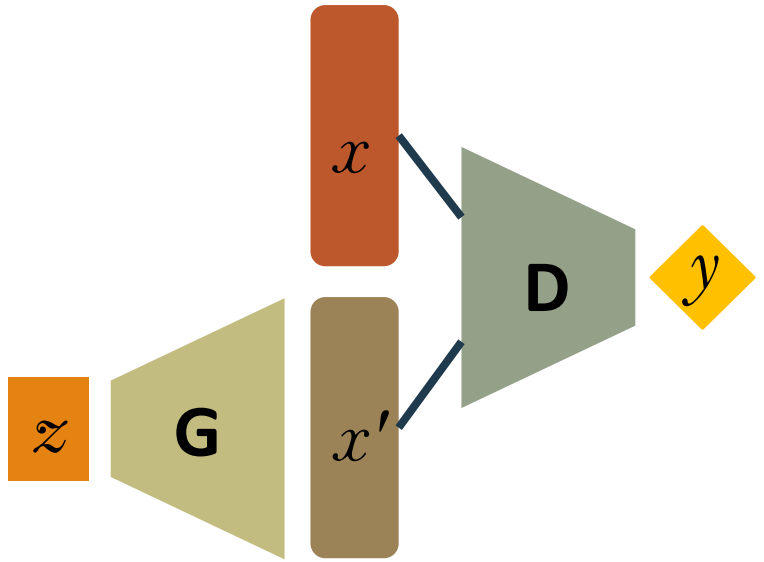
2. Train the generator

2.1 Sample a batch of data from random latent space (z)

2.2 Get samples (x') from the generator with input z

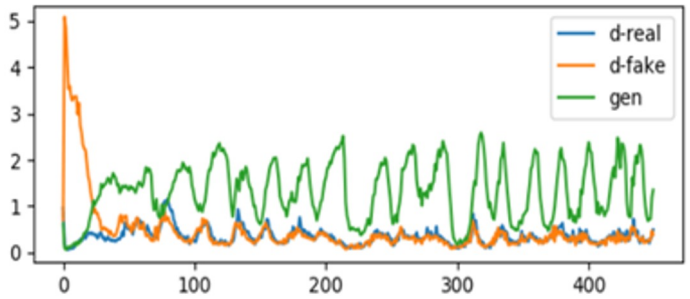
2.3 Get loss from the discriminator output with input x'

2.4 Update generator weights according to the losses



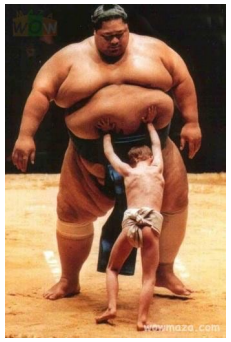
Training pathologies

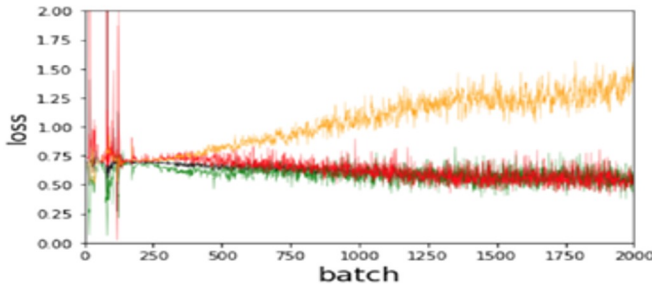
- **Non-convergence:** the model parameters oscillate, destabilize, and never converge



- **Mode collapse:** the generator collapses which produces limited varieties of samples

- **Diminished gradient:** the discriminator gets too successful that the generator gradient vanishes and learns nothing



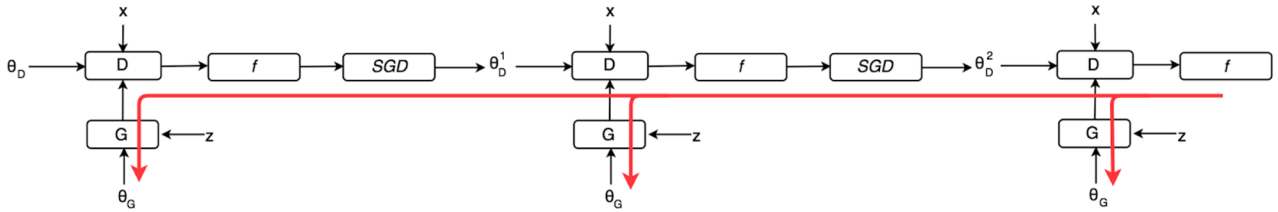


Proposed way to deal with training pathologies

- **Non-convergence:** various forms of regularization in the side of the discriminator
  - Adding noise to discriminator inputs
  - Penalizing discriminator weights

- **Mode collapse:**
  - Unrolled GANs
  - W-loss

G^t loss incorporates $D^t + D^{t+1} + \ldots$



- **Diminished gradient:**
  - W-loss
  - Modified minimax loss

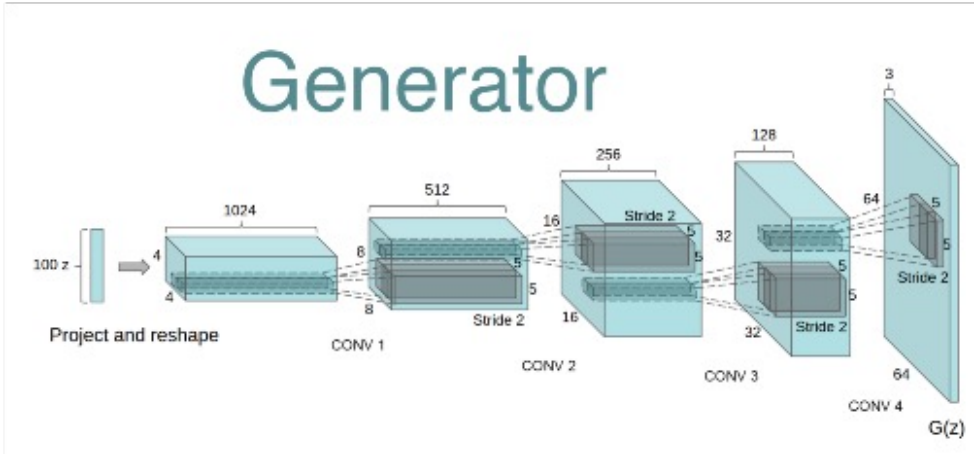
Learning variants

Name	Discriminator Loss	Generator Loss
Minimax GAN	$\mathcal{L}_D^{GAN} = -\mathbb{E}_{\mathbf{x}} \log D(\mathbf{x}) - \mathbb{E}_{\mathbf{z}} \log\left(1 - D(G(\mathbf{z}))\right)$	$\mathcal{L}_G^{GAN} = \mathbb{E}_{\mathbf{z}} \log(1 - D(G(\mathbf{z})))$
Non-Saturating GAN	$\mathcal{L}_D^{NSGAN} = \mathcal{L}_D^{GAN}$	$\mathcal{L}_G^{NSGAN} = -\mathbb{E}_{\mathbf{z}} \log D(G(\mathbf{z}))$
Least-Squares GAN	$\mathcal{L}_D^{LSGAN} = \mathbb{E}_{\mathbf{x}}(D(\mathbf{x}) - 1)^2 + \mathbb{E}_{\mathbf{z}} D(G(\mathbf{z}))^2$	$\mathcal{L}_G^{LSGAN} = \mathbb{E}_{\mathbf{z}}(D(G(\mathbf{z})) - 1)^2$
Wasserstein GAN	$\mathcal{L}_D^{WGAN} = -\mathbb{E}_{\mathbf{x}} D(\mathbf{x}) + \mathbb{E}_{\mathbf{z}} D(G(\mathbf{z}))$	$\mathcal{L}_G^{WGAN} = -\mathbb{E}_{\mathbf{z}} D(G(\mathbf{z}))$
WGAN-GP	$\mathcal{L}_D^{WGANGP} = \mathcal{L}_D^{WGAN} + \lambda \mathbb{E}_{\mathbf{x},\mathbf{z}}(\lVert \nabla D(\alpha \mathbf{x} + (1-\alpha) G(\mathbf{z})) \rVert_2 - 1)^2$	$\mathcal{L}_G^{WGANGP} = \mathcal{L}_G^{WGAN}$
DRAGAN	$\mathcal{L}_D^{DRAGAN} = \mathcal{L}_D^{GAN} + \lambda \mathbb{E}_{\mathbf{x} \sim p_{data} + \mathcal{N}(0,c)}(\lVert \nabla D(\mathbf{x}) \rVert_2 - 1)^2$	$\mathcal{L}_G^{DRAGAN} = \mathcal{L}_G^{GAN}$
BEGAN	$\mathcal{L}_D^{BEGAN} = \mathbb{E}_{\mathbf{x}} \lVert \mathbf{x} - AE(\mathbf{x}) \rVert_1 - k_t \mathbb{E}_{\mathbf{z}} \lVert G(\mathbf{z}) - AE(G(\mathbf{z})) \rVert_1$	$\mathcal{L}_G^{BEGAN} = \mathbb{E}_{\mathbf{z}} \lVert G(\mathbf{z}) - AE(G(\mathbf{z})) \rVert_1$

Introduction to different types of GANs

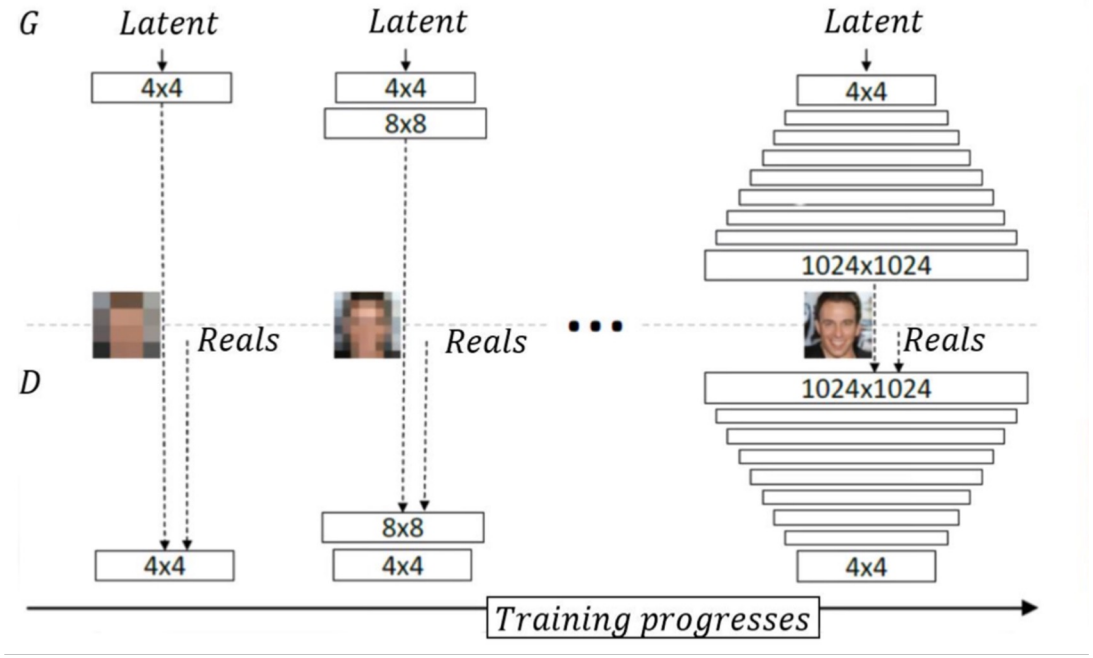
GAN models. Deep Convolutional GANs (DCGAN)

- Extends the GAN architecture with convolutional layers.
- Uses deep convolutional networks for both the generator and the discriminator.
- Helps in generating higher resolution and more realistic images.

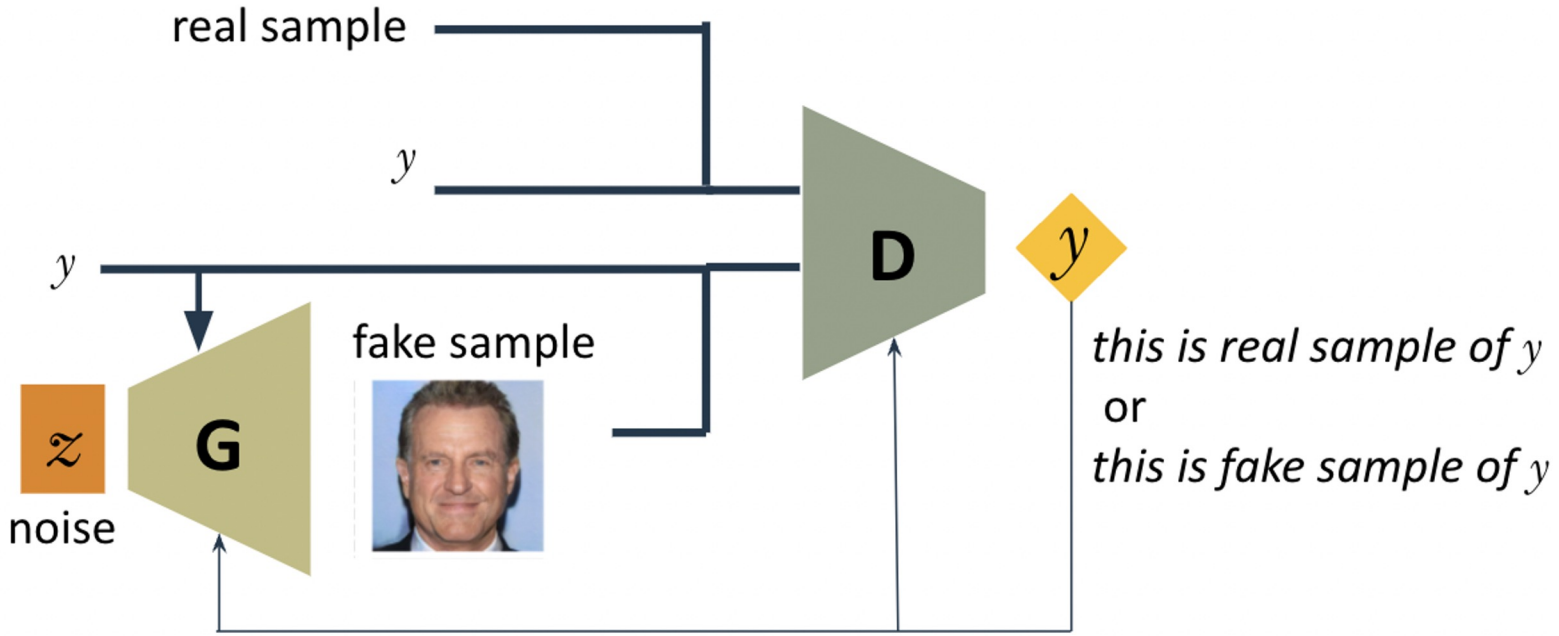


GAN models. Progressive GANs (PGGAN)

- PGGAN is a model developed by NVIDIA to improve the speed and stability of GAN training
- PGGAN generator produces high quality and high-resolution images
- During the training process the method adds new layers to the generator and discriminator



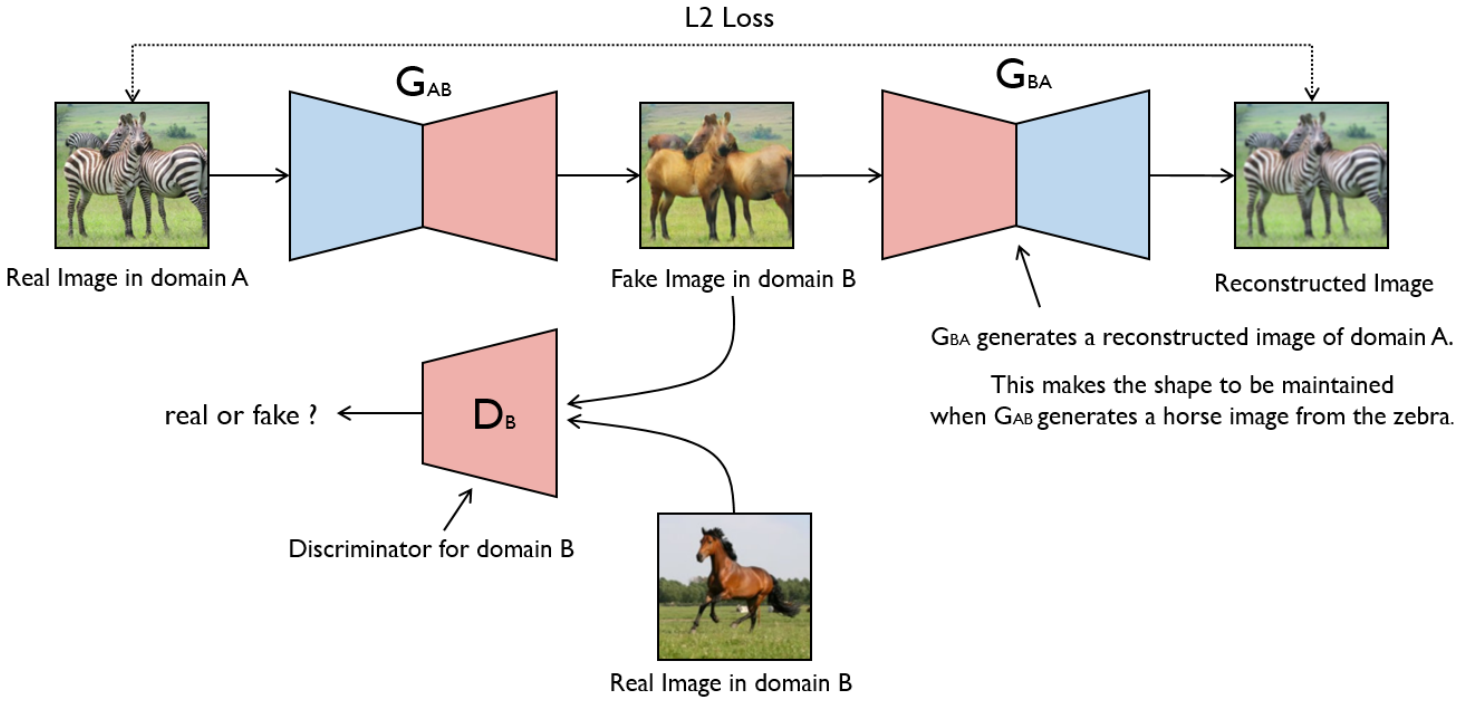
GAN models. Conditional GANs



$$\min_G \max_D V(D,G) = \mathop{\mathbb{E}}_{x \sim p_{data}(x)} [\log D(x,y)] + \mathop{\mathbb{E}}_{z \sim p_z(z)} [\log(1 - D(G(z,y),y))]$$

GAN models. Cycle GANs

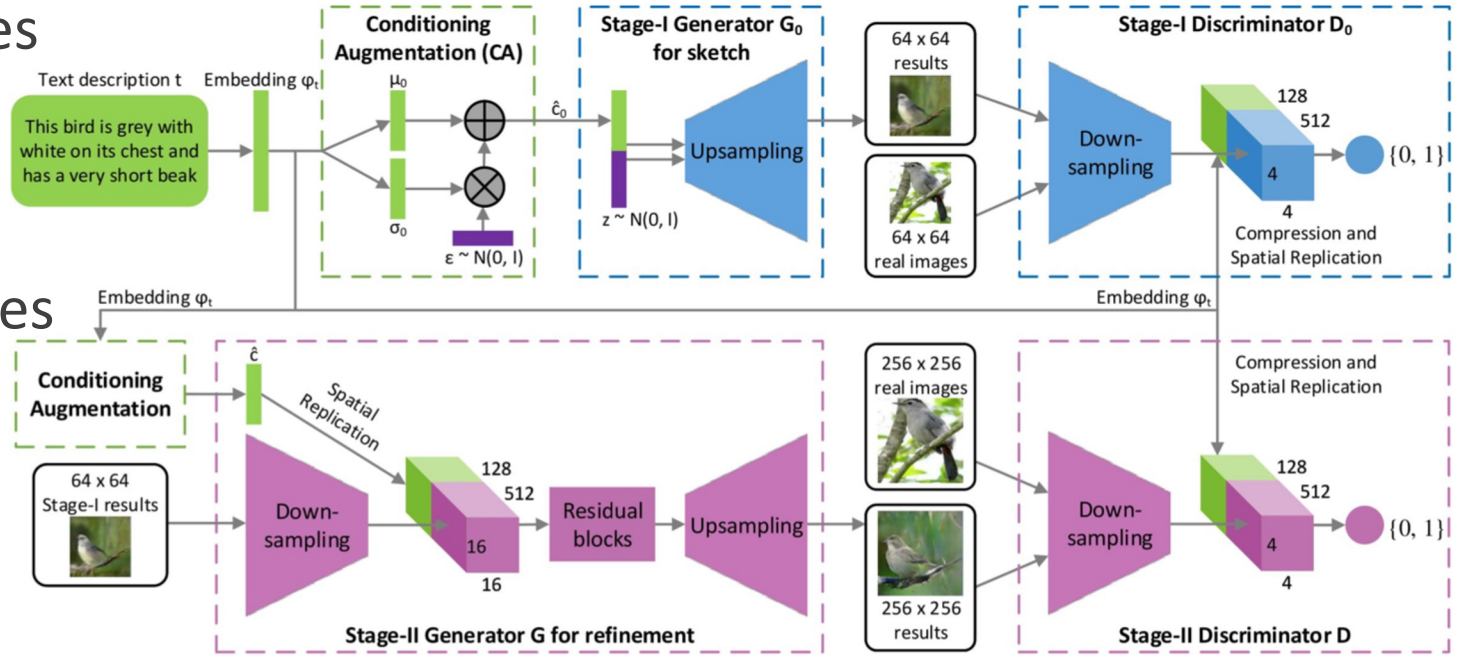
- Designed for image-to-image translation without paired training examples.
- Learns mappings between two domains without requiring matched input-output pairs during training.
- Useful for tasks like style transfer and image domain adaptation.



GAN models. StackGAN

- The Text to Photo-realistic Image Synthesis with Stacked GANs takes a **text description** and then synthesizes high quality images

- To generate photo-realistic images StackGAN uses a sketch-refinement process:
  - Stage-I GAN → low-resolution images
    - Primitive shape and colors from text description
  - Stage-II GAN → photo-realistic images
    - Using Stage-I results and text descriptions

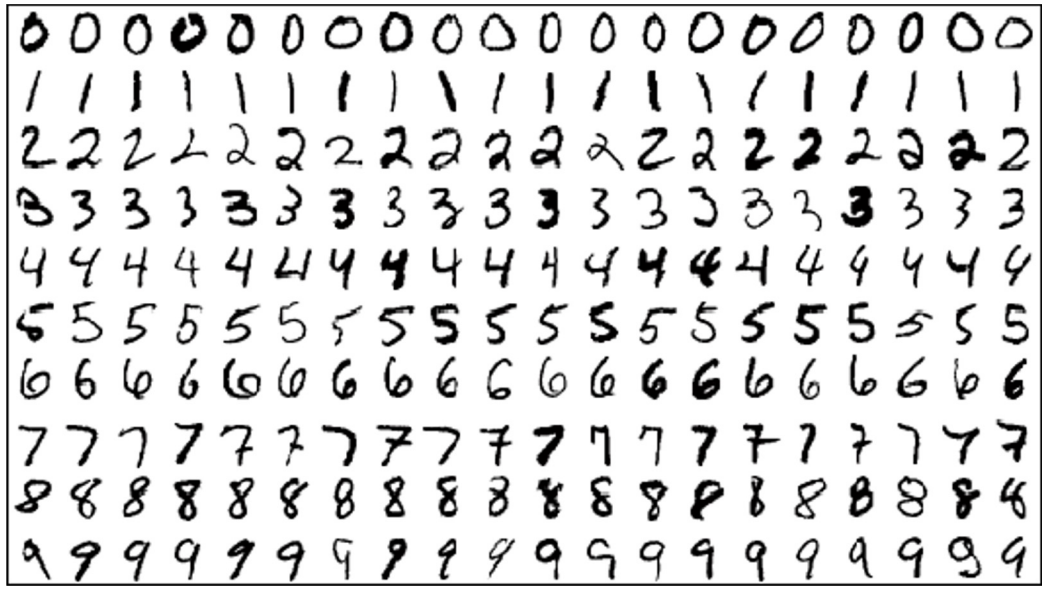


Developing GANs with Pytorch

ANNs in Pytorch

Deep learning. Example: MNIST - MLP vs. CNN

- Handwritten digits classification (MNIST repository).
- MLP to assign each image a label in the set {0,1,...,9}
- Training dataset, to train the ANN → 'what numbers 0 through 9 look like'.
- Testing dataset, never seen by the ANN during the training phase:
  - guarantee the ANN is not over-fitted to the training dataset,
  - assure the ANN can label independent items properly.



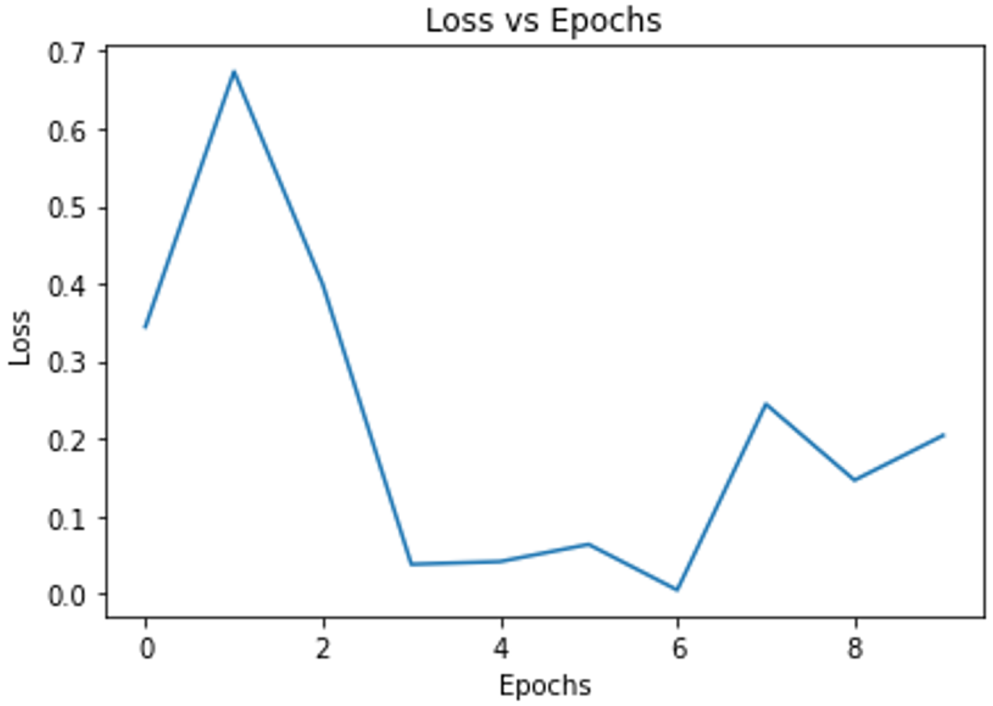
Deep learning. Example: MNIST - MLP vs. CNN

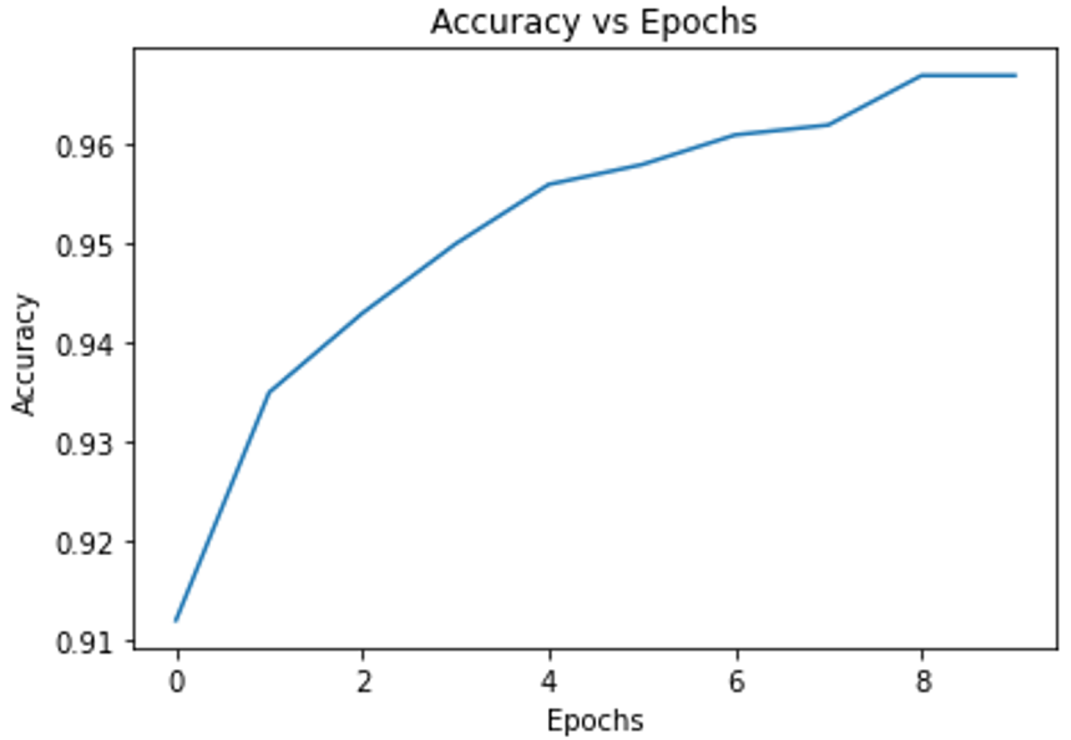
- Handwritten digits classification (MNIST repository).
- MLP to assign each image a label in the set {0,1,...,9}
- Training dataset, to train the ANN → 'what numbers 0 through 9 look like'.
- Testing dataset, never seen by the ANN during the training phase:
  - guarantee the ANN is not over-fitted to the training dataset,
  - assure the ANN can label independent items properly.

Deep learning. Example: MNIST - MLP vs. CNN

- **MLP** for handwritten digits classification: https://drive.google.com/file/d/1hHih46Z2Eth7sXCGHTgqv9CYr_PY8i5S/view?usp=sharing

Accuracy = 94.5%

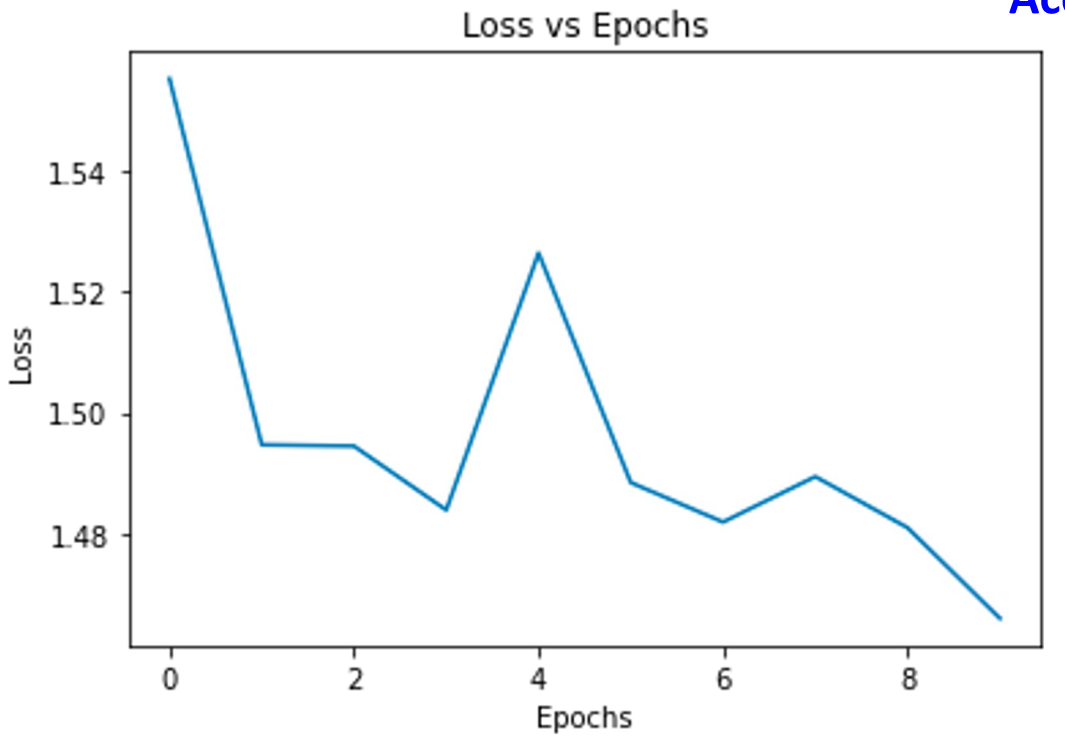


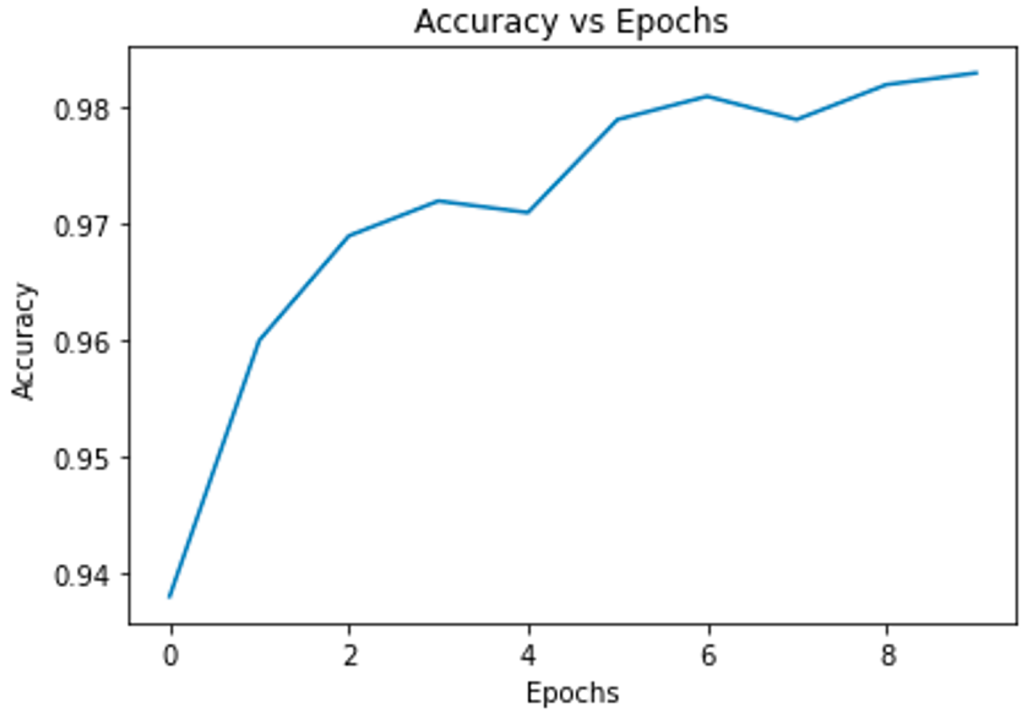


Deep learning. Example: MNIST - MLP vs. CNN

- **CNN** for handwritten digits classification: https://drive.google.com/file/d/12bAg-w9eWJChU-AXtRx8f42bdLNvIwGj/view?usp=sharing

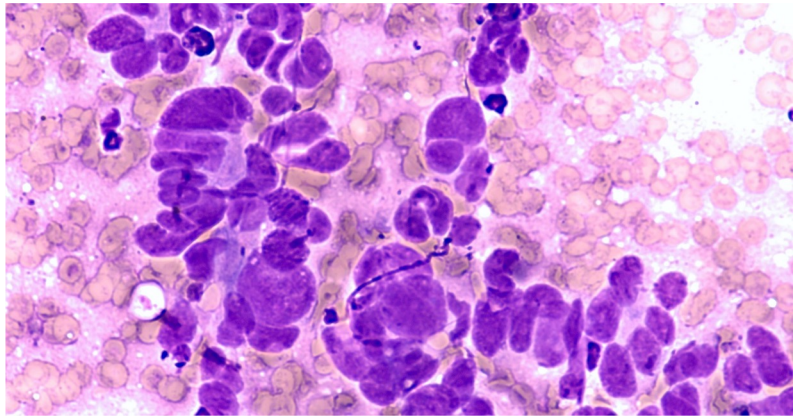
Accuracy = 98.3%





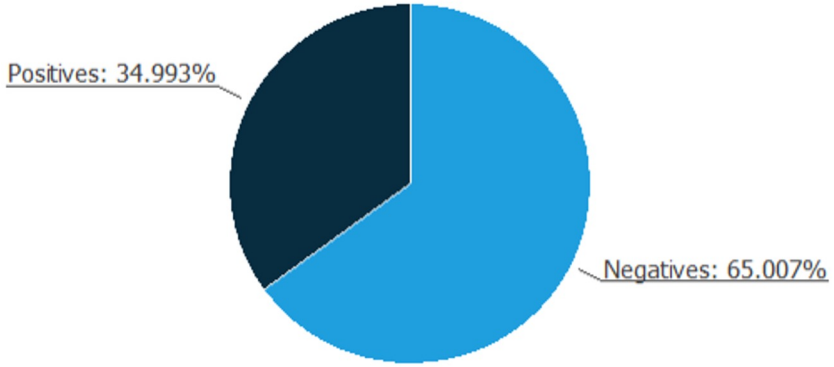
Deep learning. Sample applications (1)

- Breast cancer prediction
  - Assess whether a lump in a breast is malignant (**cancerous**) or benign (**non-cancerous**) from digitized images of a fine-needle aspiration biopsy.
  - The dataset contains 30 features from the images.



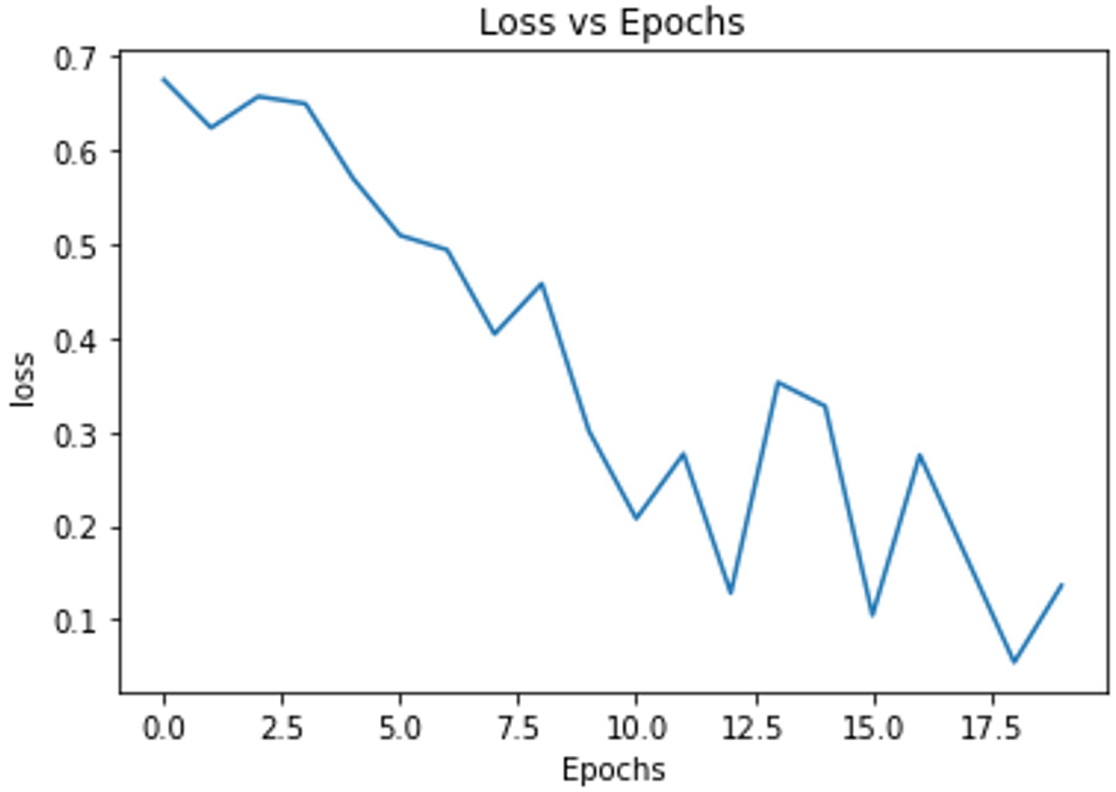
- Training dataset, to train the ANN
  - malignant or benign cases.
- Testing dataset, never seen by the ANN during the training phase:
  - guarantee not over-fitting the ANN to training dataset

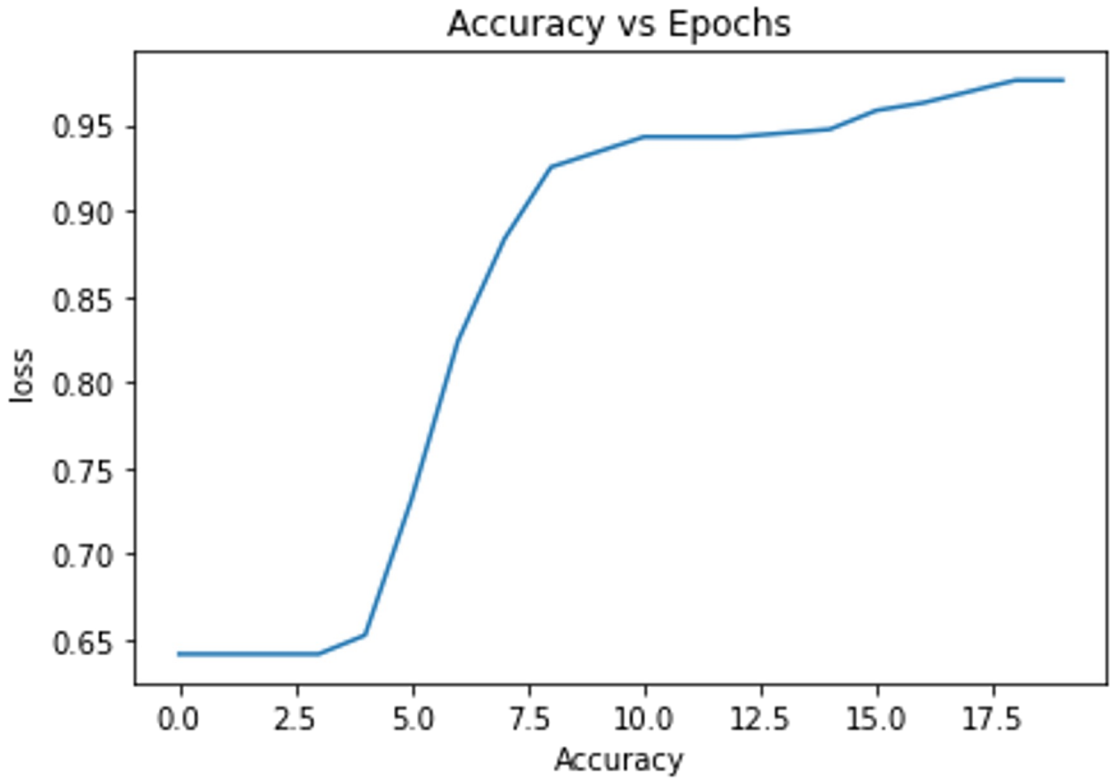
https://drive.google.com/file/d/1tW1UymqY9gq_UrTY9uUnsC1jwUo_ufJx/view?usp=sharing



Deep learning. Sample applications (1)

- Breast cancer prediction: results





ANNs in Pytorch. Exercise

- Create a multiclass classifier

https://colab.research.google.com/drive/1ymTiE-RJesWbfMljrtQAOBQfuAvmeJld?usp=sharing

GAN Training. General Algorithm

Steps of the main training loop:

1. Train discriminator

1.1. Train discriminator on real data

1.1.1 Sample a batch of data from real dataset (x)

1.1.2 Get loss from the discriminator output with input x

1.2 Train the discriminator on data produced by the generator

1.2.1 Sample a batch of data from random latent space (z)

1.2.2 Get samples (x') from the generator with input z

1.2.3 Get loss from the discriminator output with input x'

1.3 Update discriminator weights according to the losses

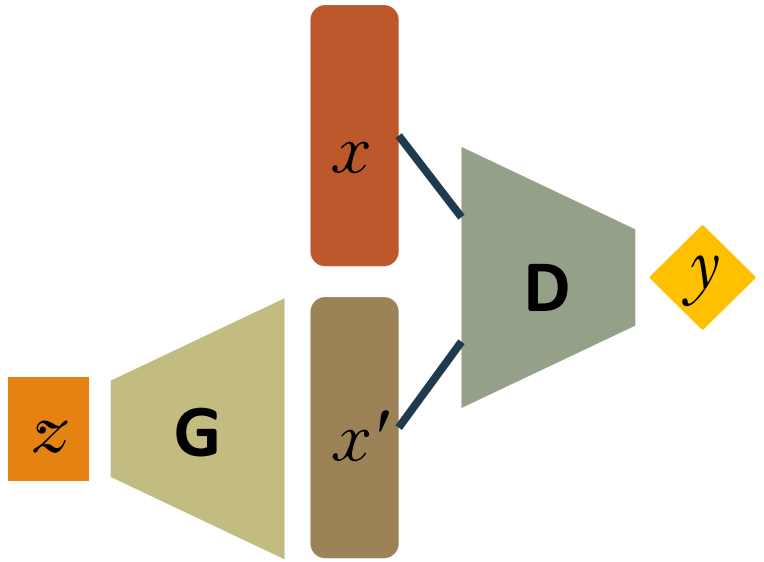
2. Train the generator

2.1 Sample a batch of data from random latent space (z)

2.2 Get samples (x') from the generator with input z

2.3 Get loss from the discriminator output with input x'

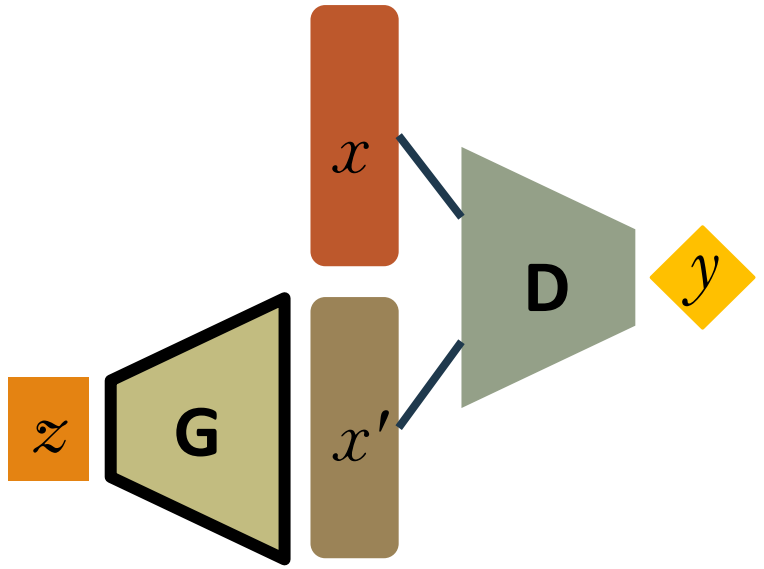
2.4 Update generator weights according to the losses



GAN Training. General code

0. Create ANNs

```
class Generator(nn.Module):
    """
    Class that defines the the Generator Neural Network
    """
    def __init__(self, input_size, hidden_size, output_size):
        super(Generator, self).__init__()

        self.net = nn.Sequential(
            nn.Linear(input_size, hidden_size),
            nn.SELU(),
            nn.Linear(hidden_size, hidden_size),
            nn.SELU(),
            nn.Linear(hidden_size, output_size),
            nn.SELU(),
        )

    def forward(self, x):
        x = self.net(x)
        return x
```

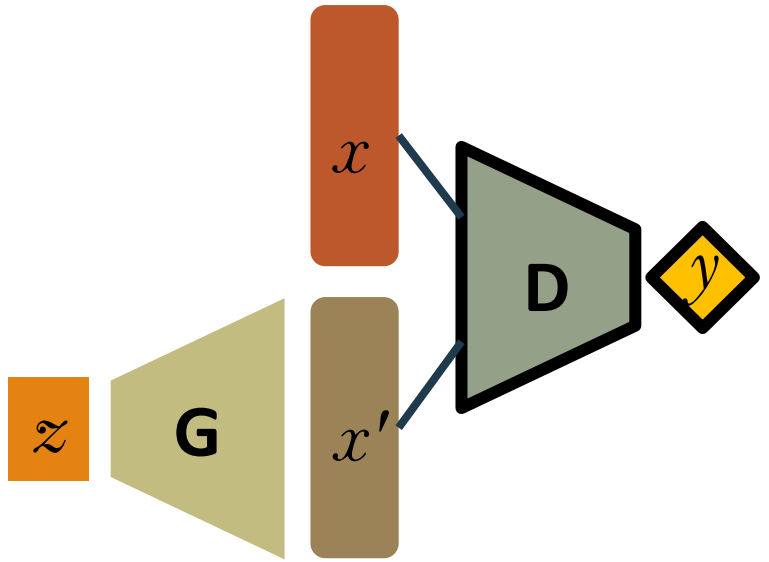



```
# Creating the GAN generator
generator_input_size = 50  # Input size of the generator (latent space)
generator_hidden_size = 150
generator_output_size = vector_length
generator = Generator(input_size=generator_input_size, hidden_size=generator_hidden_size,
                      output_size=generator_output_size)
```

GAN Training. General code

0. Create ANNs

```
class Discriminator(nn.Module):
    """
    Class that defines the the Discriminator Neural Network
    """
    def __init__(self, input_size, hidden_size, output_size):
        super(Discriminator, self).__init__()

        self.net = nn.Sequential(
            nn.Linear(input_size, hidden_size),
            nn.ELU(),
            nn.Linear(hidden_size, hidden_size),
            nn.ELU(),
            nn.Linear(hidden_size, output_size),
            nn.Sigmoid()
        )

    def forward(self, x):
        x = self.net(x)
        return x
```

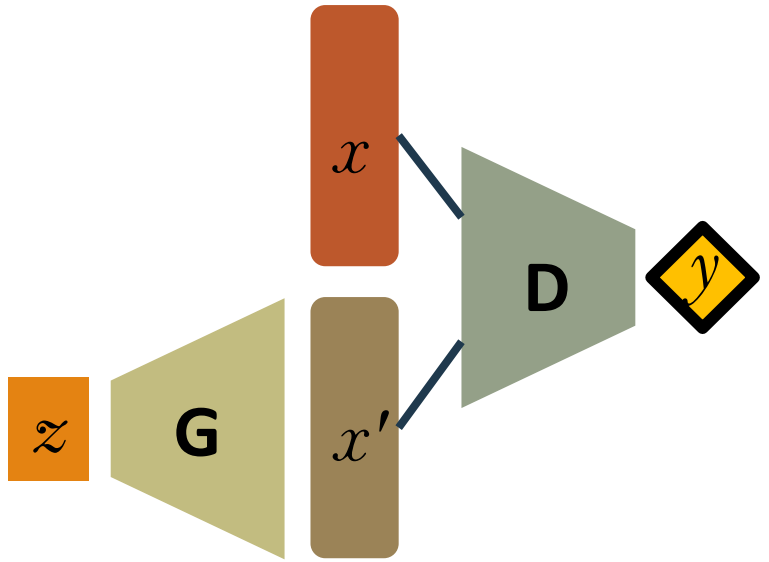



```
# Creating the GAN discriminator
discriminator_input_size = vector_length
discriminator_hidden_size = 75
discriminator_output_size = 1
discriminator = Discriminator(input_size=discriminator_input_size, hidden_size=discriminator_hidden_size,
                              output_size=discriminator_output_size)
```

GAN Training. General code

0. Labels to compare

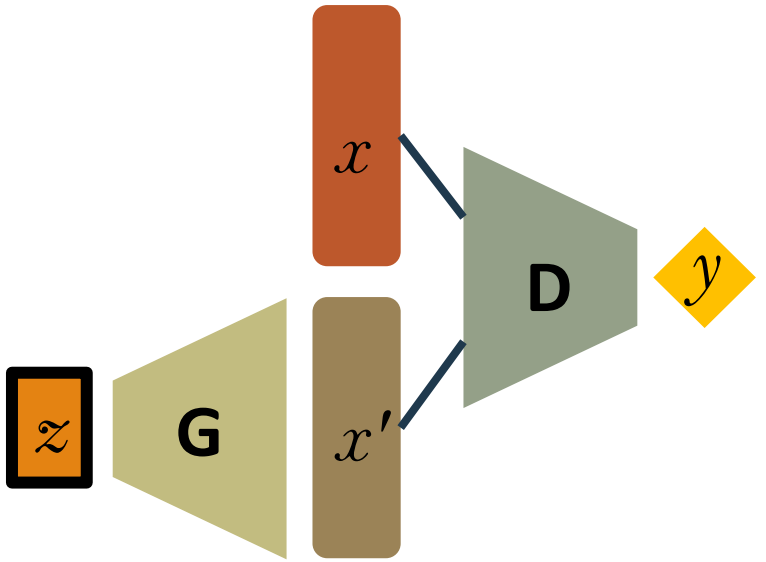
```python
def real_data_target(size):
    """
    Creates a tensor with the target for real data with shape = size
    :param size: Size of the tensor (batch size).
    :return: Tensor with real label value (ones) with shape = size
    """
    data = Variable(torch.ones(size, 1))
    if torch.cuda.is_available(): return data.cuda()
    return data


def fake_data_target(size):
    """
    Creates a tensor with the target for fake data with shape = size
    :param size: Size of the tensor (batch size).
    :return: Tensor with fake label value (zeros) with shape = size
    """
    data = Variable(torch.zeros(size, 1))
    if torch.cuda.is_available(): return data.cuda()
    return data
```



GAN Training. General code

0. Latent space reading

```python
def read_latent_space(batch_size, latent_vector_size):
    """
    Creates a tensor with random values fro latent space  with shape = size
    :param size: Size of the tensor (batch size).
    :return: Tensor with random values (z) with shape = size
    """
    z = torch.rand(batch_size, latent_vector_size)
    if torch.cuda.is_available(): return z.cuda()
    return z
```



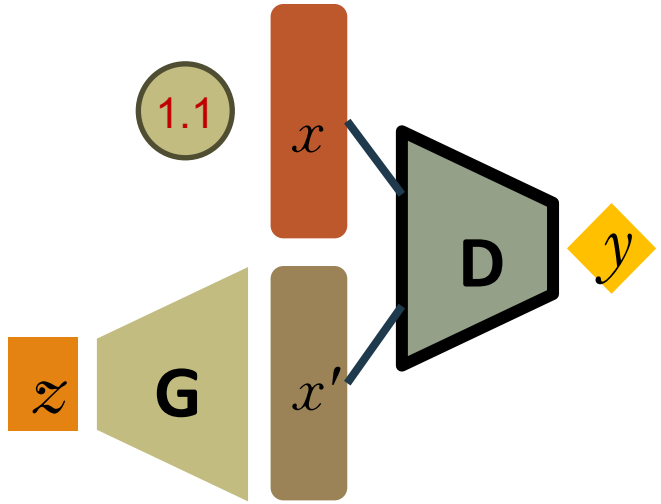
GAN Training. General code

1. Train discriminator

```
# 1. Train the discriminator
discriminator.zero_grad()
# 1.1 Train discriminator on real data
input_real = get_data_samples(batch_size)
discriminator_real_out = discriminator(input_real.reshape(batch_size, 2))
discriminator_real_loss = discriminator_loss(discriminator_real_out, real_data_target(batch_size))
discriminator_real_loss.backward()
# 1.2 Train the discriminator on data produced by the generator
input_fake = read_latent_space(batch_size)
generator_fake_out = generator(input_fake).detach()
discriminator_fake_out = discriminator(generator_fake_out)
discriminator_fake_loss = discriminator_loss(discriminator_fake_out, fake_data_target(batch_size))
discriminator_fake_loss.backward()
# 1.3 Optimizing the discriminator weights
discriminator_optimizer.step()
```

1.1

1.2

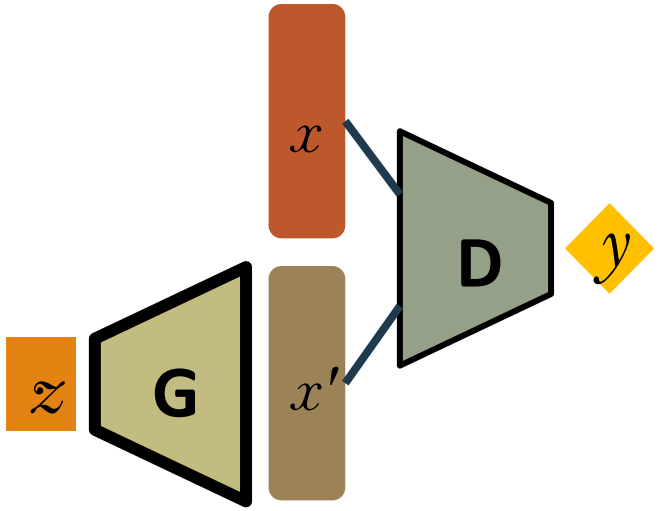


GAN Training. General code

2. Train discriminator

```
# 2. Train the generator
if batch_number % freeze_generator_steps == 0:
  generator.zero_grad()
  # 2.1 Create fake data
  input_fake = read_latent_space(batch_size)
  generator_fake_out = generator(input_fake)
  # 2.2 Try to fool the discriminator with fake data
  discriminator_out_to_train_generator = discriminator(generator_fake_out)
  discriminator_loss_to_train_generator = generator_loss(discriminator_out_to_train_generator,
                                                         real_data_target(batch_size))
  discriminator_loss_to_train_generator.backward()
  # 2.3 Optimizing the generator weights
  generator_optimizer.step()
```

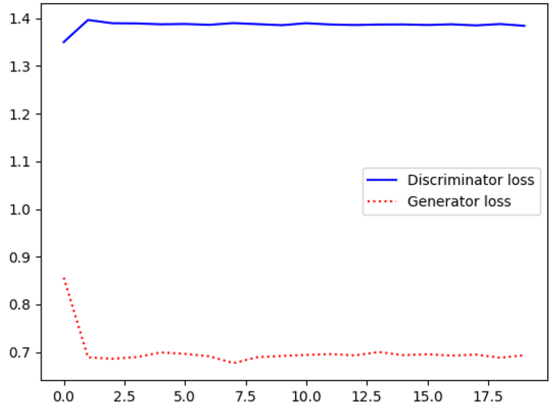
2

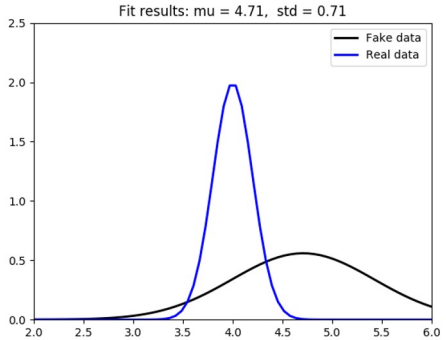


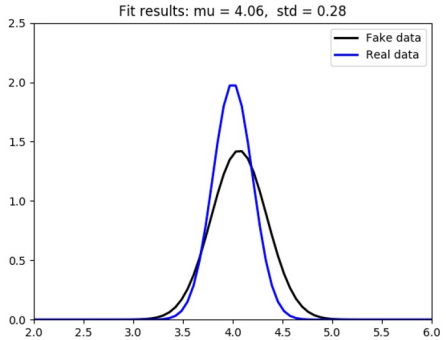
GAN Training. Source code example 1

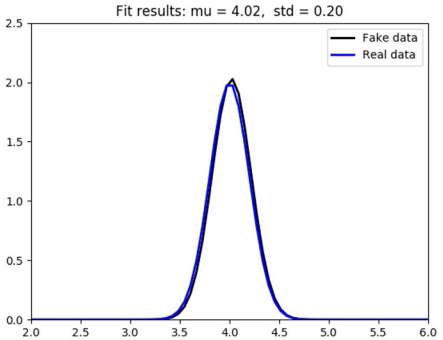
Example: Train a generator to create vectors of a given size that contains float numbers that follow a normal distribution given the mean and the standard deviation

- Real dataset samples: Vectors of real numbers that follow a normal distribution
- Source code: *https://drive.google.com/file/d/195onuuhVksZenHGEdMqY5h8PY7TN1CzR/view?usp=sharing*





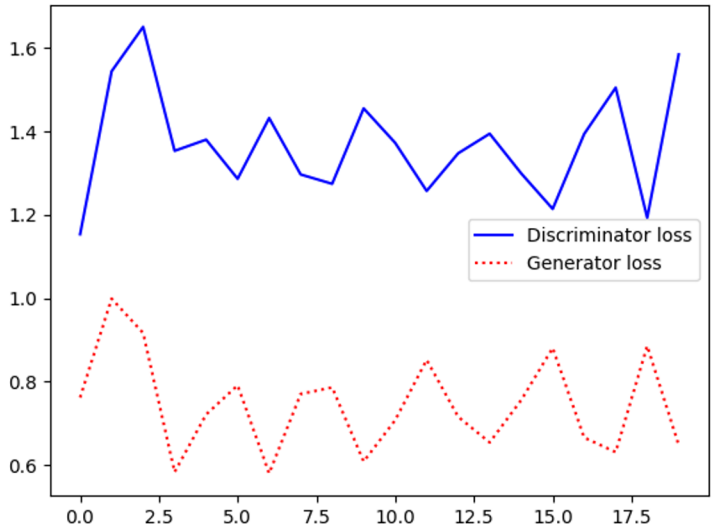


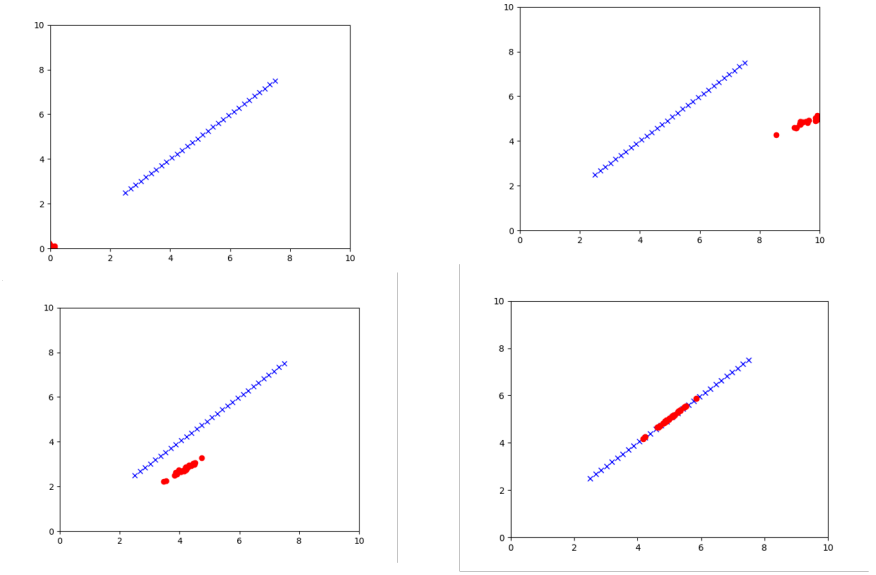


GAN training. Source code example 2

Example: Train a generator to create 2D points (x, y) that belong to a line in the 2D space

- Real dataset samples: Points (x, y) that belong to the line
- Source code: *https://drive.google.com/file/d/1bfBQ8CZyi9Ht5Nr7rgAPAD9E6Xcg5pQP/view?usp=sharing*





- Test *freezing* the generator and increasing the number of epochs

GAN training. Source code example 3

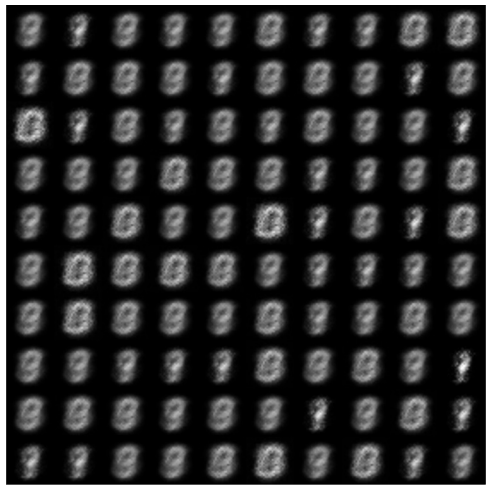
Example: Train a generator to create samples of handwritten digits of MNIST dataset.

The MNIST dataset is one of the most common datasets used for image classification and generation. It contains 60,000 training images and 10,000 testing images of handwritten digits (from 0 to 9)

- Real dataset samples: Digits from MNIST dataset
- Source code: https://drive.google.com/file/d/1eqUsRaVcvp4urSZLQU4JupTVrEGn0QQk/view?usp=sharing

Real data

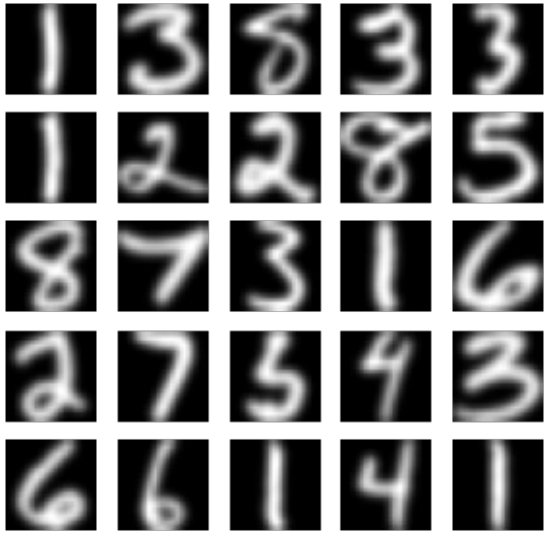
Generated data during training



GAN training. Exercise

- Create a GAN to generate USPS samples

https://colab.research.google.com/drive/1ZmfkwxYzAndwjyC27Ij8ogMS2L9LEeL1?usp=sharing



Thanks!

Comments?

JAMAL TOUTOUH
jamal@uma.es
jamal.es
@jamtou

Sergio Nesmachnow
sergion@fing.edu.uy