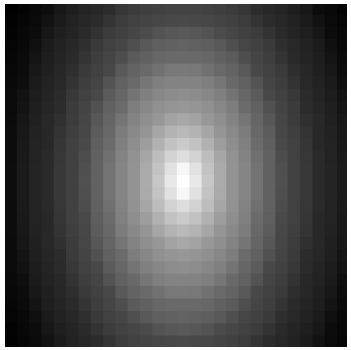
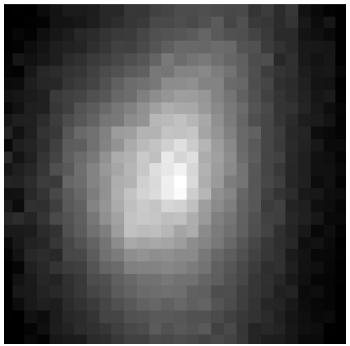
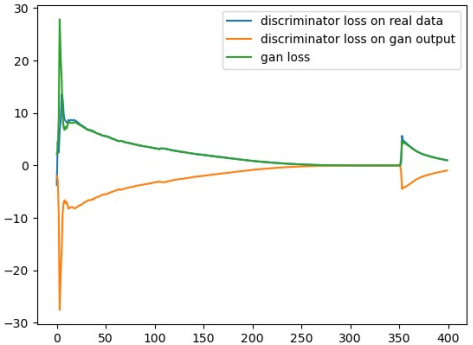
Train the Trainer – Generative Adversarial Networks







Caroline Heneka, Hamburg Observatory
Credits to: Jörn Bach (UHH)

Outline of this lecture

Part 1: Lecture on Generative Adversarial Networks (ca. 60 min)

1) GAN concept and architecture

2) Important types of GANs

3) Applications and performance

Part 2: Hands-on tutorial (ca. 30 min)
How to GAN galaxy clusters

1) Basic GAN concept and architecture

GAN = Generative Adversarial Network

- Introduced in 2014 by Goodfellow at al. (https://arxiv.org/abs/1406.2661)

- Two models (networks) are simultaneously trained, a generative and a discriminative model

- Framework: two-player game

"Adversarial training is like a really cool idea, it is like the coolest idea in machine learning in the last 20 years" (Yann LeCun, 2016)

1) Basic GAN concept and architecture

Examples of GAN applications

<also, your favourite examples here>

Credit: tutorials.one

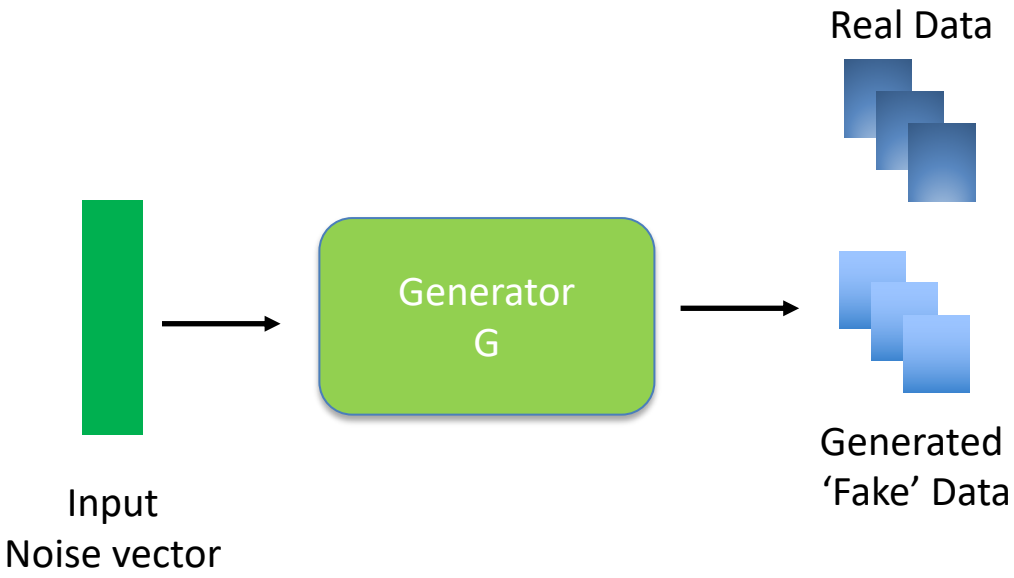
Image generation
Or ‘simulation’

Similar tasks:
De-noising, Inpainting

Credit: towardsdatascience.com

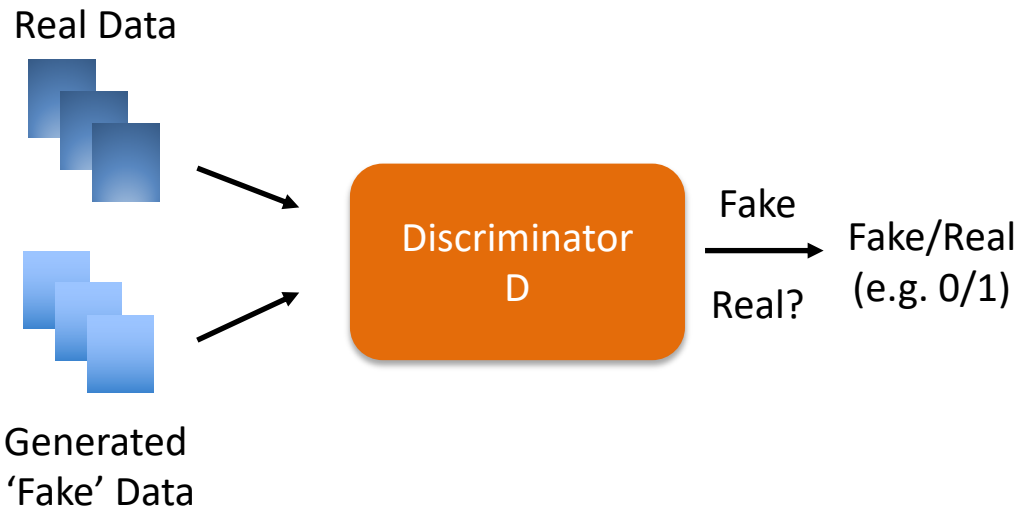
1) Basic GAN concept and architecture

GAN = **Generative** Adversarial Network



1) Basic GAN concept and architecture

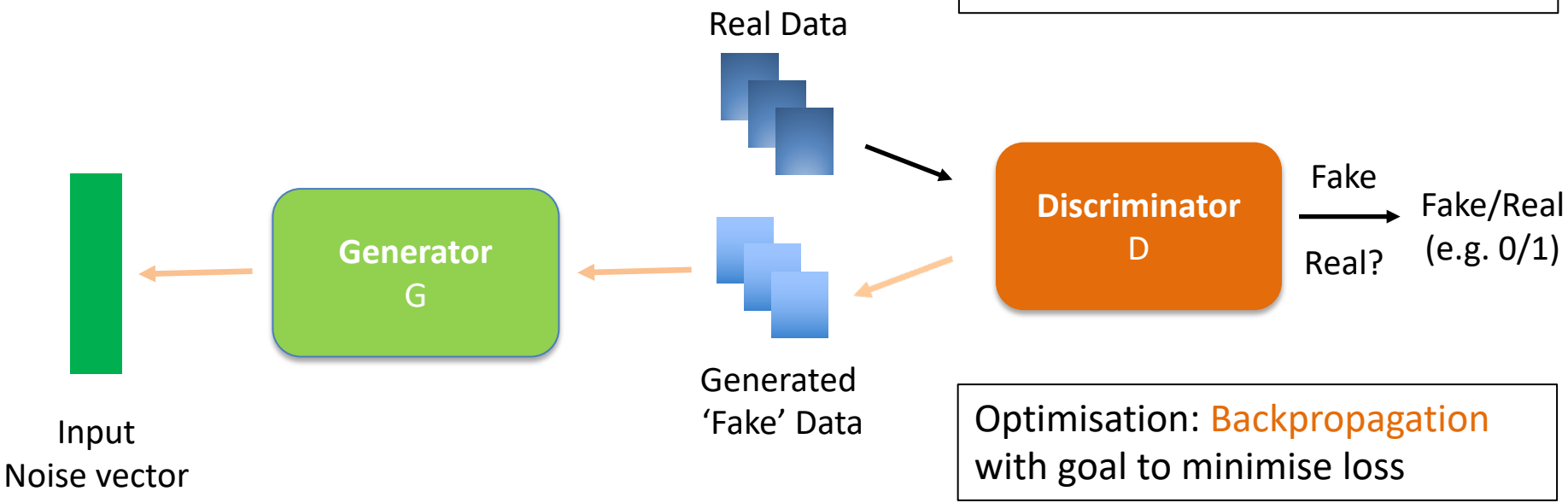
GAN = Generative **Adversarial** Network



1) Basic GAN concept and architecture

GAN = Generative Adversarial **Network**

Concept: Optimise generated wrt real data via adversarial network

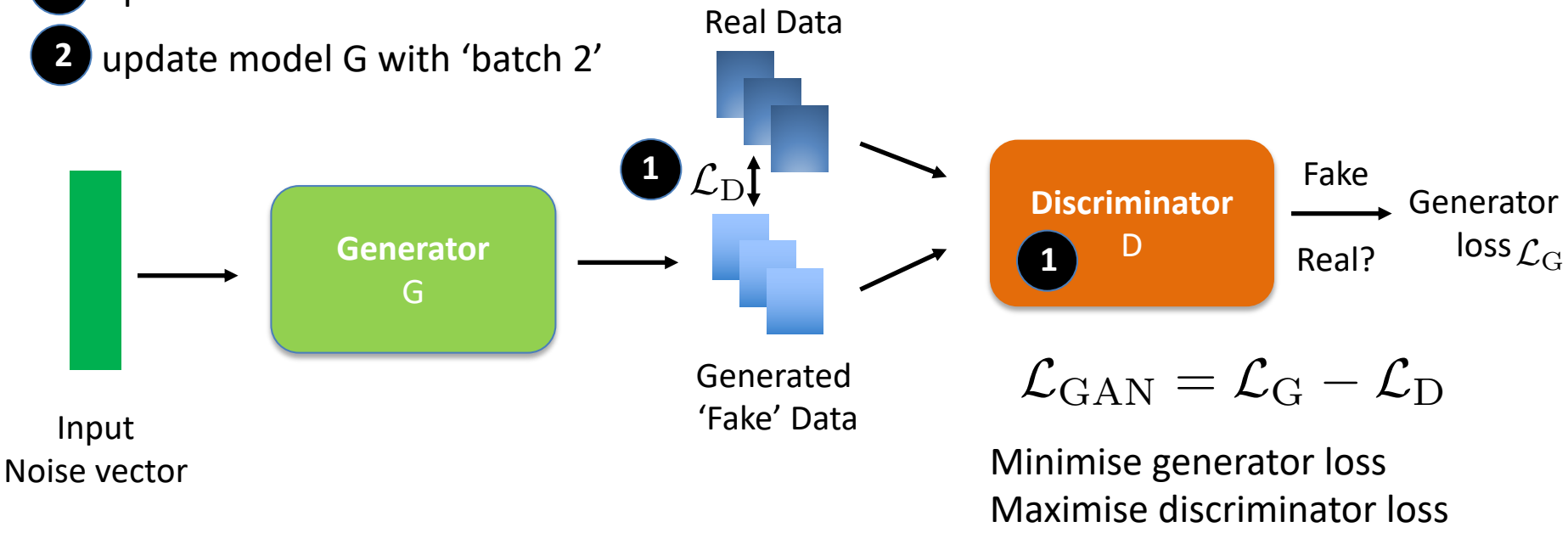


Optimisation: Backpropagation with goal to minimise loss

1) Basic GAN concept and architecture

Each training step we:

1. update model D with 'batch 1'
2. update model G with 'batch 2'



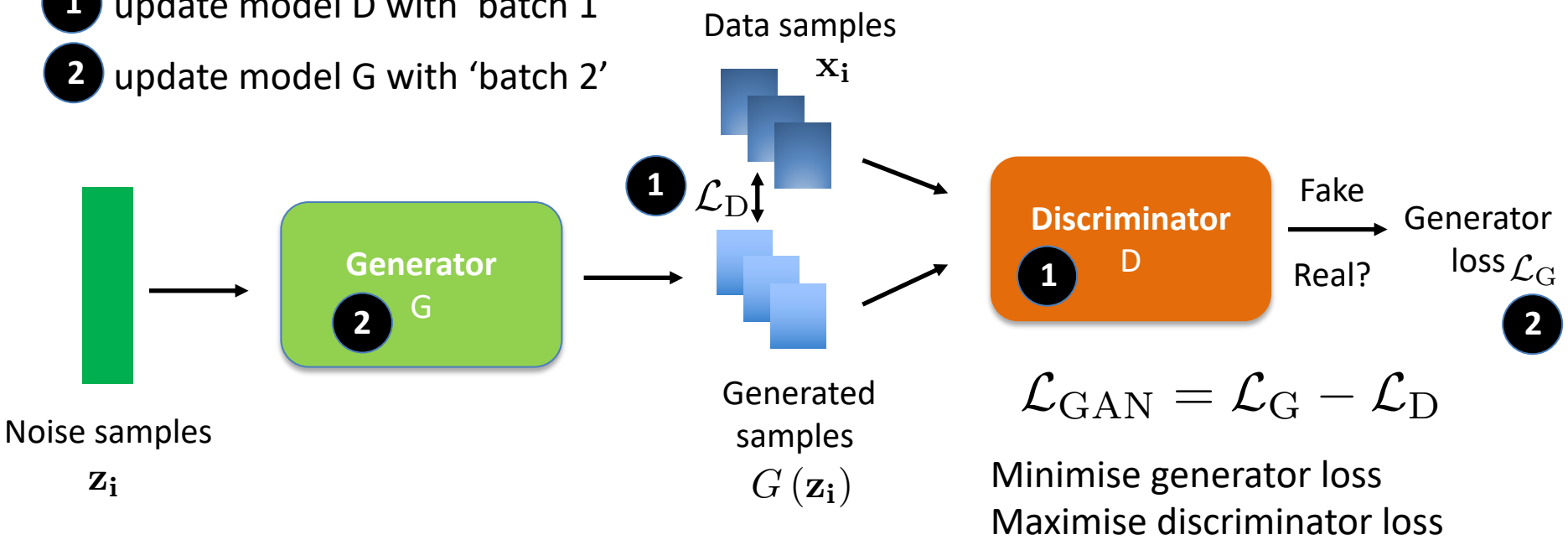
$$\mathcal{L}_{GAN} = \mathcal{L}_G - \mathcal{L}_D$$

Minimise generator loss
Maximise discriminator loss

1) Basic GAN concept and architecture

Each training step we:

1. update model D with 'batch 1'
2. update model G with 'batch 2'



$$\mathcal{L}_\mathrm{GAN} = \mathcal{L}_\mathrm{G} - \mathcal{L}_\mathrm{D}$$

Minimise generator loss
Maximise discriminator loss

2) Important types of GANs

A) MinMax GAN Assign label: 1 = real, 0 = fake

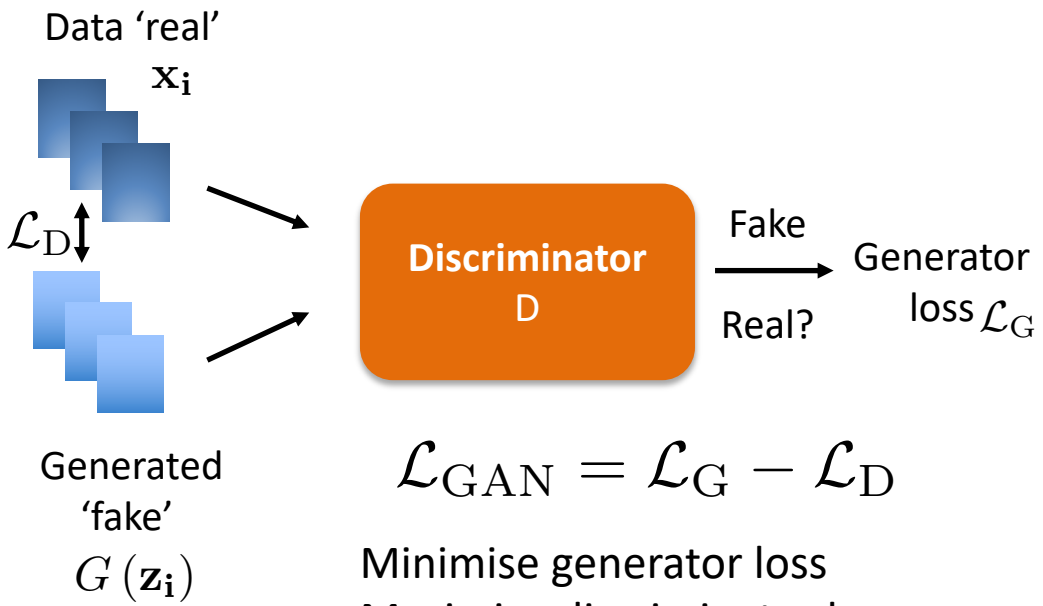
$\mathcal{L}_{\mathrm{D}}$ = binary cross-entropy:

$$\frac{1}{N}\sum_{i=1}^{N}\left[\log D\left(\mathbf{x_i}\right) + \log\left(1 - D\left(G\left(\mathbf{z_i}\right)\right)\right)\right]$$

Predicted p_1 — Predicted 1 - p_0

→ Maximise real vs. fake

'the standard binary classifier'



$$\mathcal{L}_{\mathrm{GAN}} = \mathcal{L}_{\mathrm{G}} - \mathcal{L}_{\mathrm{D}}$$

Minimise generator loss
Maximise discriminator loss

2) Important types of GANs

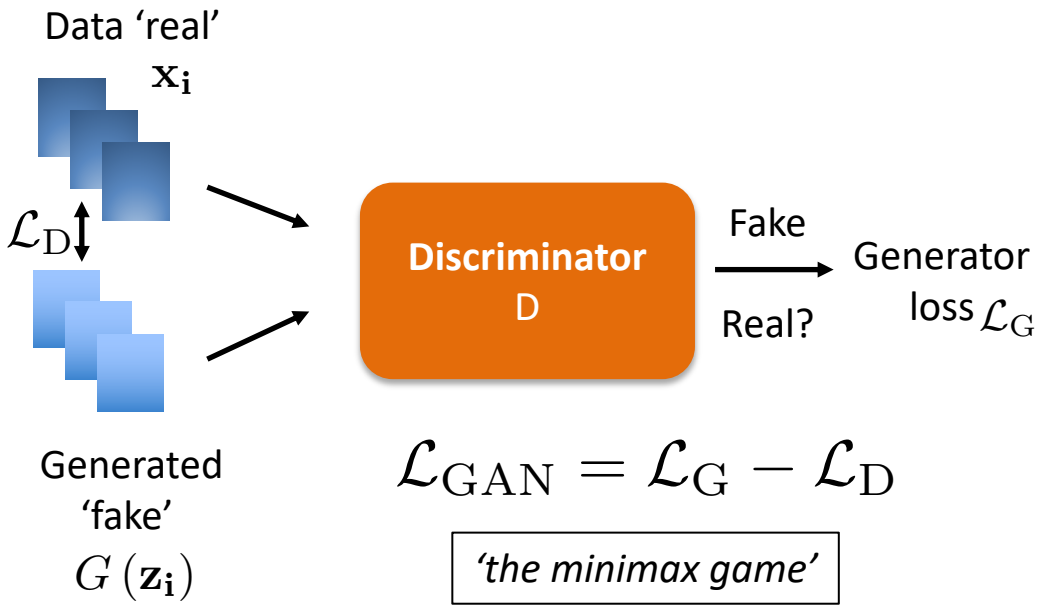
A) MinMax GAN Assign label: 1 = real, 0 = fake

$\mathcal{L}_{\mathrm{G}}$ = 'ability to mislead D':

$$\frac{1}{N}\sum_{i=1}^{N}\underbrace{\left[\log\left(1-D\left(G\left(\mathbf{z_i}\right)\right)\right)\right]}$$

Predicted
1 - p_0

⟶ Minimise D catching fake (inverted p_0)



$$\mathcal{L}_{\mathrm{GAN}} = \mathcal{L}_{\mathrm{G}} - \mathcal{L}_{\mathrm{D}}$$

'the minimax game'

2) Important types of GANs

A) MinMax GAN

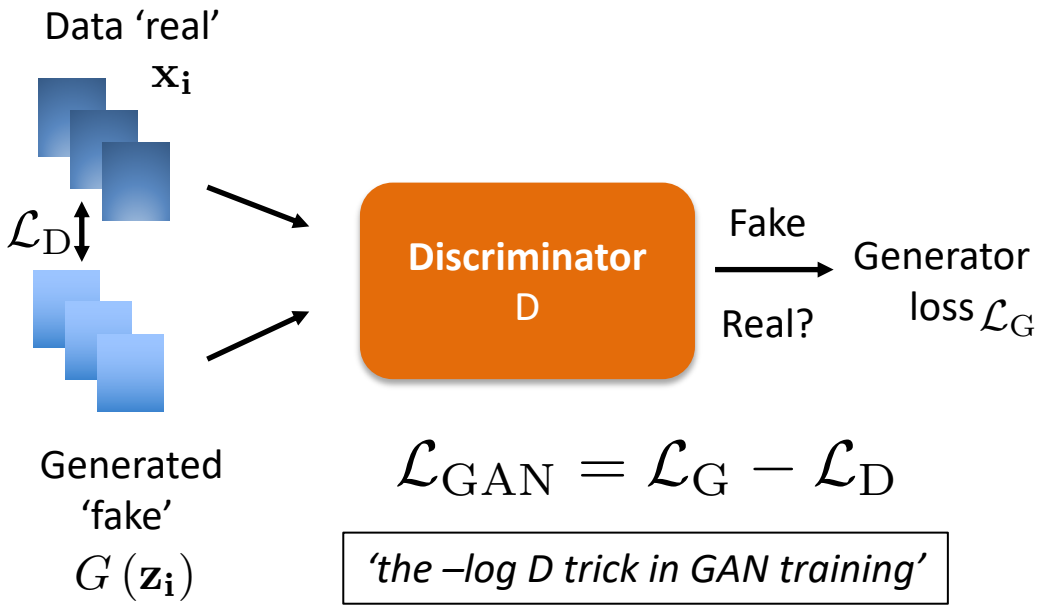
Problem: Small training gradient if D performs well and G poorly

$$\frac{1}{N}\sum_{i=1}^{N}\left[\log\left(1-D\left(G\left(\mathbf{z_i}\right)\right)\right)\right]$$

⟶ Maximise p_0 instead i.e. $D\left(G\left(\mathbf{z_i}\right)\right)$

[minimize $-\log D\left(G\left(\mathbf{z_i}\right)\right)$]

⟶ Stronger gradients

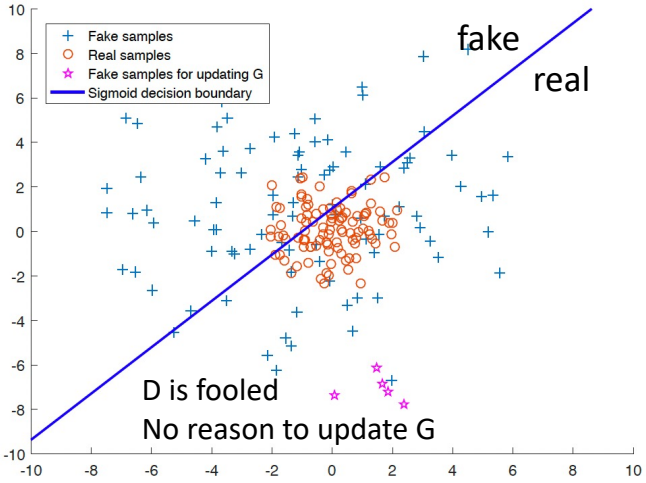


2) Important types of GANs

B) Least Squares GAN (LSGAN) or HOW correct is D?

Xudong Mao et al. 2016

MinMax GAN: binary



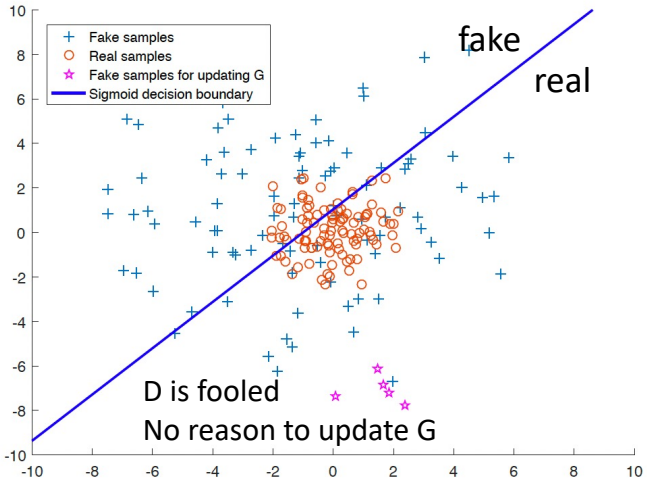
'vanishing gradient problem'
(loss saturation)

2) Important types of GANs

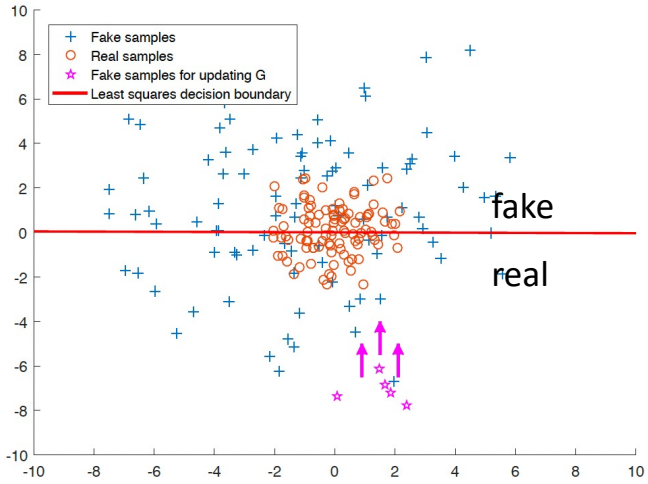
B) Least Squares GAN (LSGAN)

Xudong Mao et al. 2016

MinMax GAN: binary



LSGAN: least-squares



2) Important types of GANs

B) Least Squares GAN (LSGAN)

Xudong Mao et al. 2016

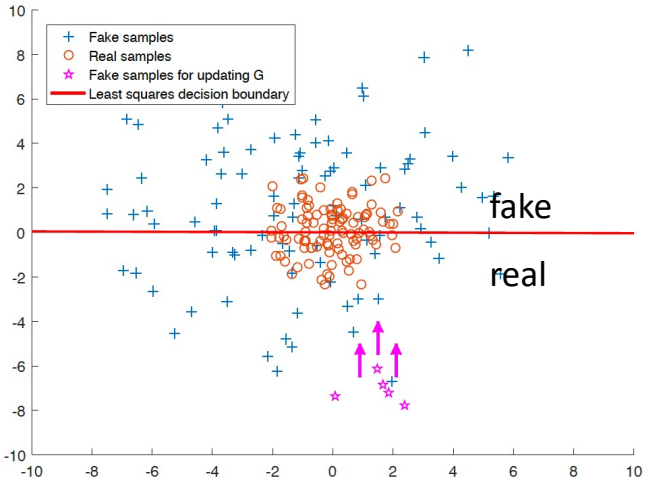
$\mathcal{L}_{\mathrm{D}}$ = mean-squared loss (MSE), L2

1 = real, 0 = fake

$$\frac{1}{N}\sum_{i=1}^{\mathrm{N}}\left[\left(D\left(\mathbf{x_i}\right)-1\right)^2+\underbrace{\left(D\left(G\left(\mathbf{z_i}\right)\right)-0\right)^2}_{-\mathcal{L}_{\mathrm{G}}}\right]$$

→ Stronger gradients
More stable training

LSGAN: least-squares



2) Important types of GANs

C) Wasserstein GAN (WGAN)

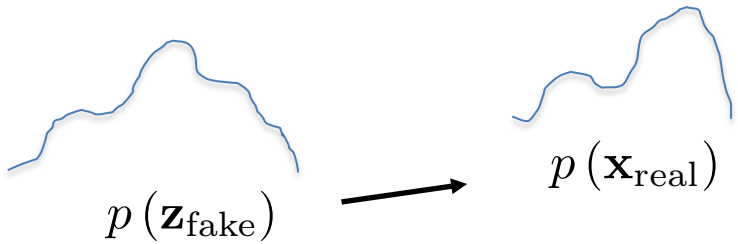
or scoring ‘realness’ vs. ‘fakeness’

Arjovsky, Chintala, Bottou 2017

Idea: Minimise distance between distribution of training vs. generated data

WGAN score = Wasserstein or Earth Mover distance (EMD)

EMD = cost to transport, or transform, one probability distribution to another



2) Important types of GANs

C) Wasserstein GAN (WGAN)

Discriminator → Critic

Arjovsky, Chintala, Bottou 2017 have shown:

Seeking convergence, not equilibrium

EMD can be approximated by maximising **critic** loss:

$$\mathcal{L}_{\mathrm{D}} = \bar{f}_w(\mathbf{x}) - \bar{f}_w(G(\mathbf{z}))$$

(D)

$$\mathcal{L}_{\mathrm{G}} = -\bar{f}_w(G(\mathbf{z}))$$

f_w = family of parametrised functions over critic weights w

Reminder: $\mathcal{L}_{\mathrm{GAN}} = \mathcal{L}_{\mathrm{G}} - \mathcal{L}_{\mathrm{D}}$

2) Important types of GANs

C) Wasserstein GAN (WGAN)

Discriminator → Critic

Arjovsky, Chintala, Bottou 2017 have shown:

Seeking convergence, not equilibrium

EMD can be approximated by maximising **critic** loss:

$$\mathcal{L}_{\mathrm{D}} = D(\mathbf{x}) - D(G(\mathbf{z}))$$

GAN. Note that the fact that $\mathcal{W}$ is compact implies that all the functions f_w will be K-Lipschitz for some K that only depends on $\mathcal{W}$ and not the individual weights, therefore approximating (2) up to an irrelevant scaling factor and the capacity of the 'critic' f_w. In order to have parameters w lie in a compact space, something simple we can do is clamp the weights to a fixed box (say $\mathcal{W} = [-0.01, 0.01]^l$) after

* A more mathematical excursion based on arXiv:1701.07875 could be implemented here

2) Important types of GANs

C) Wasserstein GAN (WGAN)

Discriminator → Critic

Arjovsky, Chintala, Bottou 2017 have shown:

Seeking convergence, not equilibrium

EMD can be approximated by maximising **critic** loss.

In practice implemented as: loss = $\bar{y}_{\text{true}} * \bar{y}_{\text{pred}}$

Labels **y**: -1 = real, 1 = fake (or vice-versa)

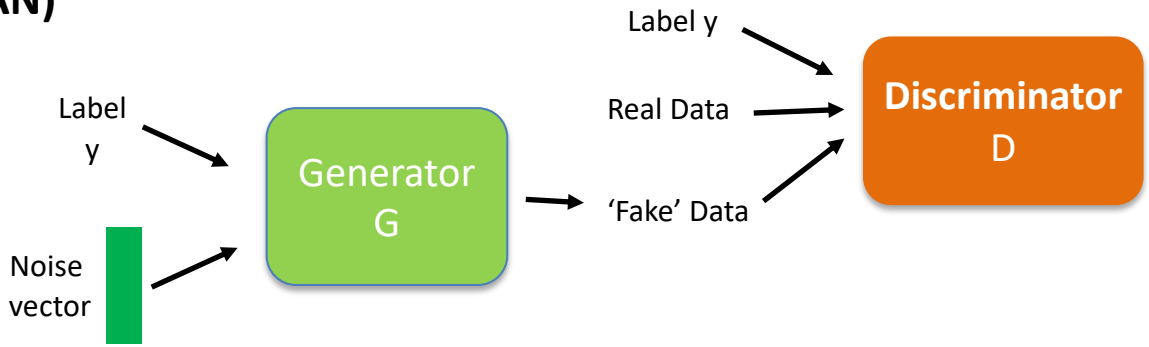
Some further practical points:
- Linear activation in output layer
- Clipping of weights w (critic) after each update to [-c,c], c<1
- Often: update critic several times per generator update (per iteration)

2) Important types of GANs

D) Conditional GAN (cGAN)

Mirza & Osindero 2014

Uses labels to train **both** G and D:



E) GANs with regulatory loss term

$$\mathcal{L}_{\mathrm{tot}} = \mathcal{L}_{\mathrm{GAN}} + \gamma \mathcal{L}_{\mathrm{reg}}$$

$$\mathcal{L}_{\mathrm{GAN}} = \mathcal{L}_{\mathrm{G}} - \mathcal{L}_{\mathrm{D}}$$

$\mathcal{L}_{\mathrm{reg}}$ = regulatory loss term (e.g. symmetry)

γ = regulatory strength

2) Important types of GANs

Ad C) Wasserstein GAN (WGAN)

Arjovsky, Chintala, Bottou 2017

Mode collapse:
G learns to map several input z to same output

→ Loss of multi-modality

→ Low diversity of generated images

Seeking convergence, not equilibrium

more stable training, better

Convergence

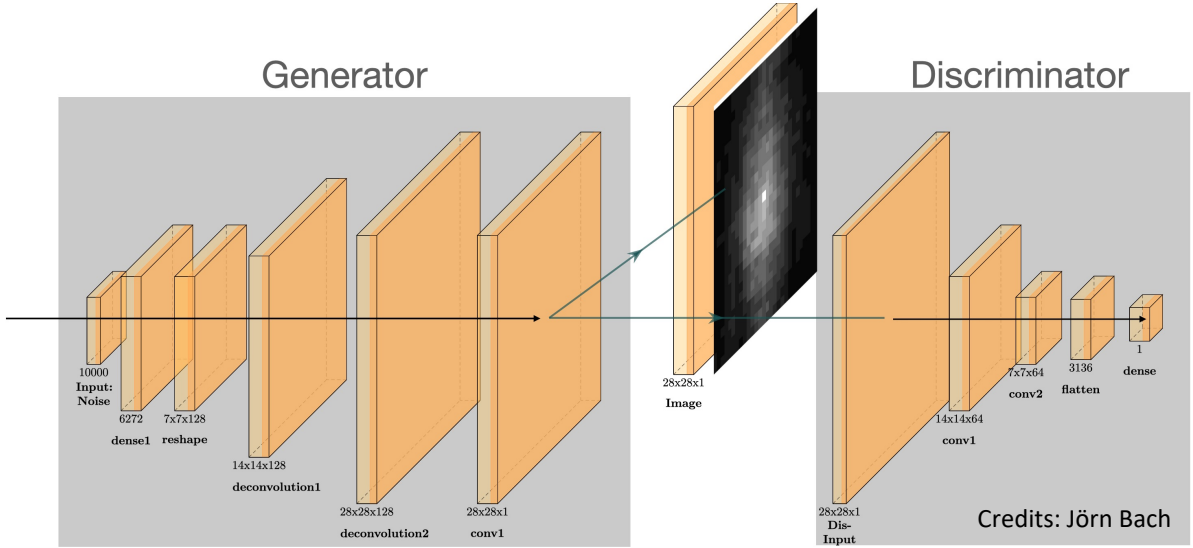
better against:

Problem of 'mode collapse'

3) Applications and performance

How to GAN galaxy clusters
[See also: exercise Part 2]

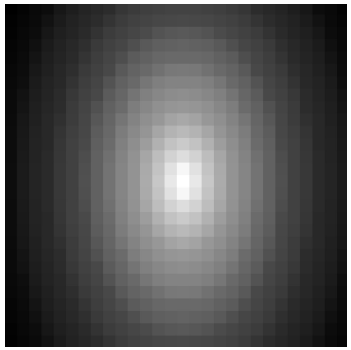
Scientific use case:
Fast, reliable way to simulate galaxy cluster images (in X-rays, at different cosmologies)



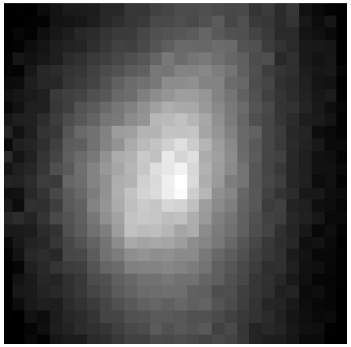
3) Applications and performance

How to GAN galaxy clusters
[See also: exercise Part 2]

A)



B)



Please vote:

1. A) real B) fake

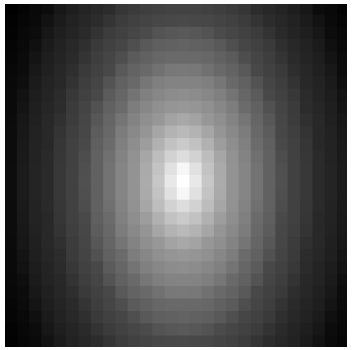
2. A) fake B) real

Unit: EM (emissivity)

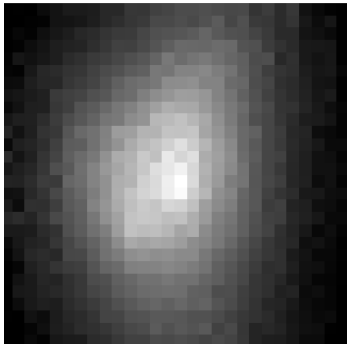
3) Applications and performance

How to GAN galaxy clusters
[See also: exercise Part 2]

A)



B)



Please vote:

1. A) real B) fake

2. A) fake B) real

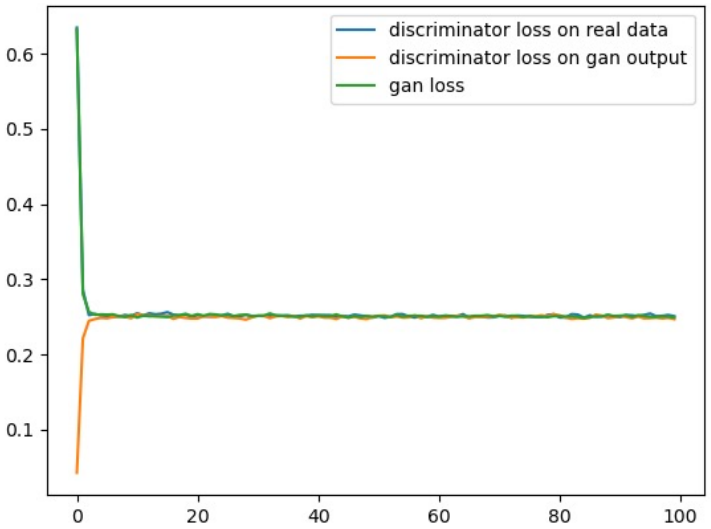
Unit: EM (emissivity)

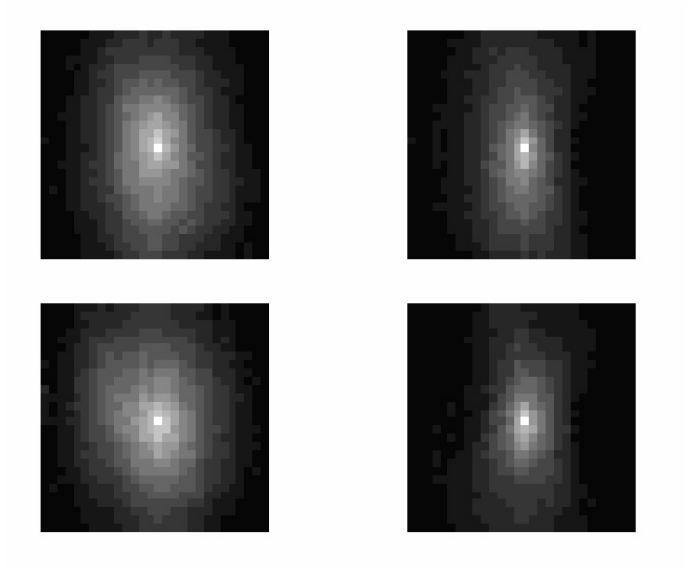
Solution: 1. insofar, as B) is GAN-generated, A) is coming from the 'real' sample of simulated training data

3) Applications and performance

Failure modes of GANs: (non-WGAN)

Convergence

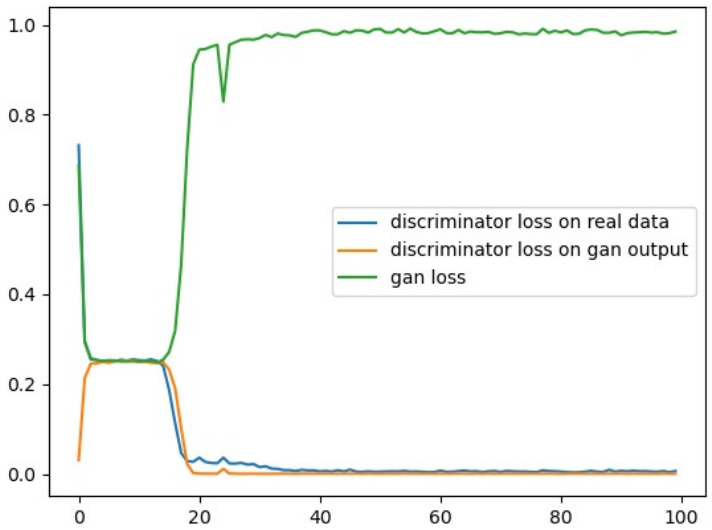


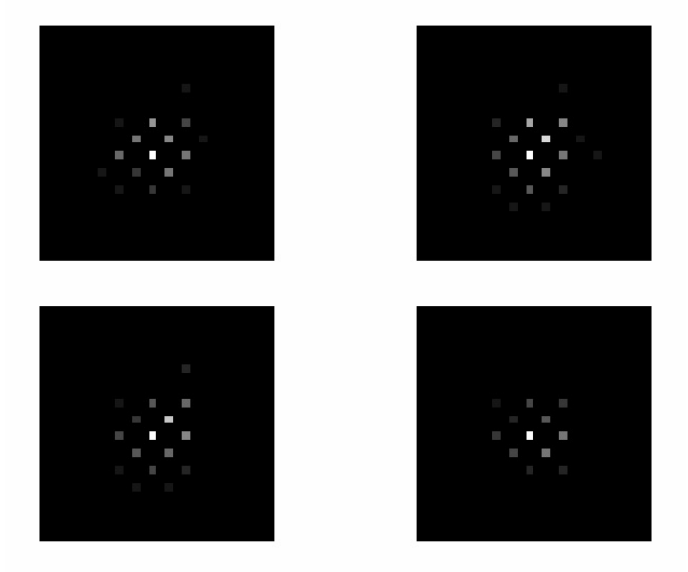


3) Applications and performance

Failure modes of GANs:
(non-WGAN)

Convergence

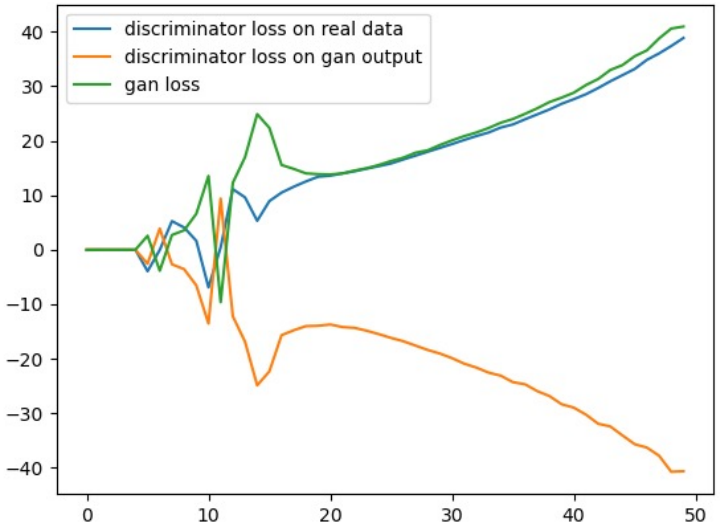


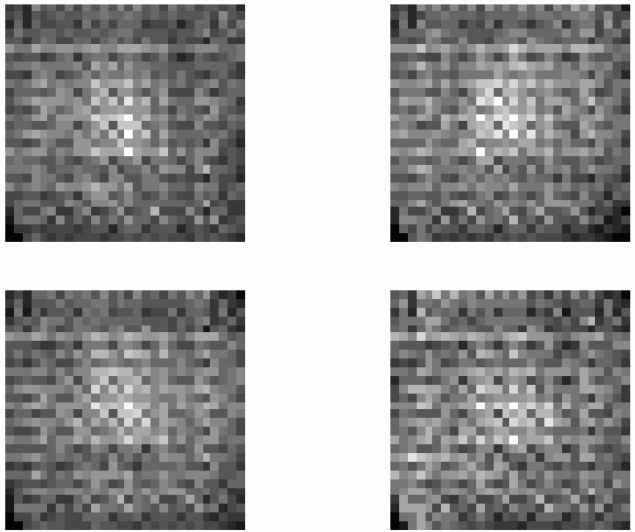


3) Applications and performance

Failure modes of GANs: (WGAN)

Convergence ✗

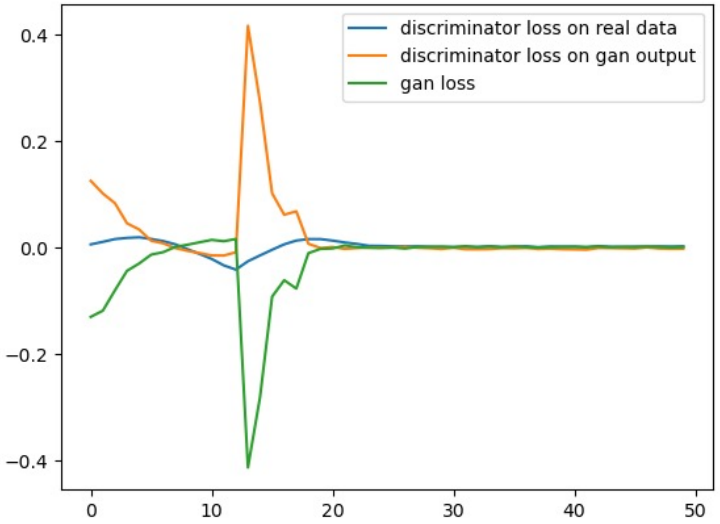


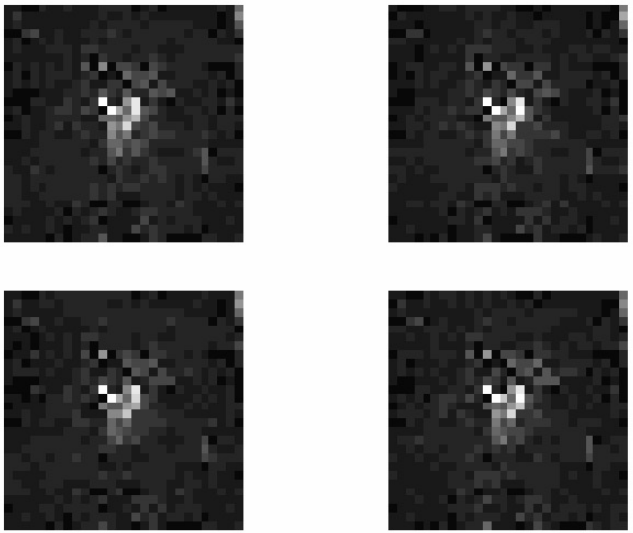


3) Applications and performance

Failure modes of GANs:

'mode collapse'

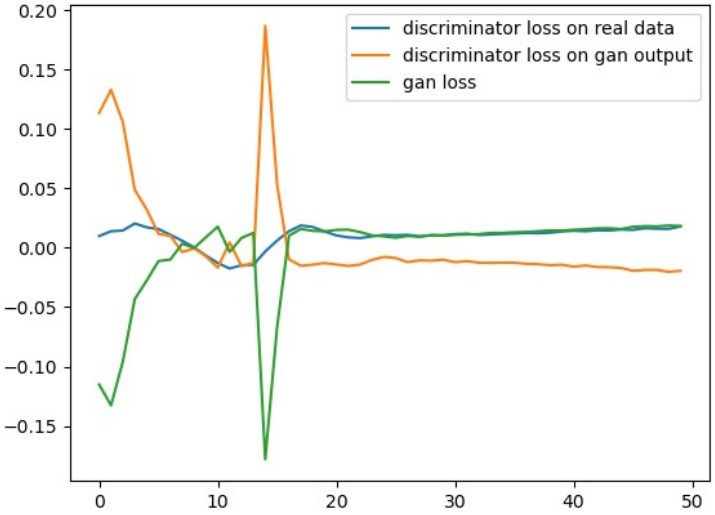


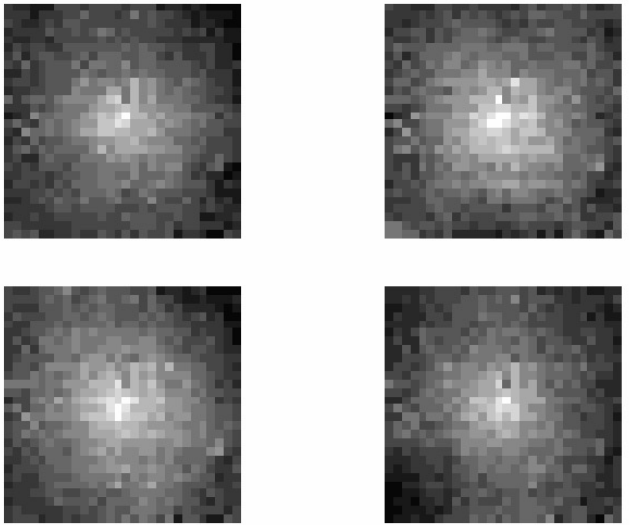


3) Applications and performance

Failure modes of GANs: *'mode collapse'*

survived

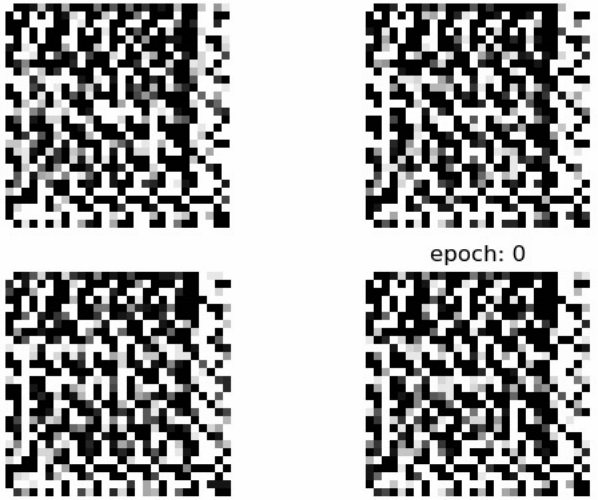




Summary of this lecture

1) We learned about the general GAN concept of a MinMax adversarial two-player game to generate data close to a given data distribution.
2) An overview of important types of GANs was given:
   MinMax GAN, LS-GAN, WGAN,
   as well as further possible additions, such as cGAN & regulatory loss terms.
3) Applications and performance: Failure modes of GANs at the example of the generation of images of galaxy clusters in X-rays.

Part 2: Hands-on tutorial (ca. 30 min)
How to GAN galaxy clusters



Gif of GAN training in progress

Please start binder image here:
https://mybinder.org/v2/gh/csheneka/GAN-tutorial/HEAD

OR via github here: https://github.com/csheneka/GAN-tutorial

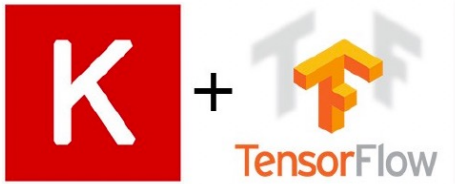
* For GPU-acces open in google Colab + adjust data-loading

Extra data: https://cloud.hs.uni-hamburg.de/s/Syq84Zwo7CC3adm

Our setup today

The tutorial provides examples of neural network models written in Python, using the Keras library and TensorFlow tensor ordering convention.

Keras provides a high level API to create deep neural networks and train them using numerical tensor libraries (backends) such as TensorFlow, CNTK or Theano.



Credit: A. Boucaud

Step-by-step tutorial walkthrough

How to (LS)GAN galaxy clusters

In this tutorial we will learn how to implement a Least-Squares GAN using tensorflow.keras.

Step-by-step, we are going to:

1) Explore images of galaxy clusters as measured at X-ray wavelengths

2) Prepare our data for GAN training

3) Set up both the generator and discriminator model

4) Generate the noise that the GAN will use to create images from

5) Train our LSGAN model and explore possible improvements

Step-by-step tutorial walkthrough

Data

The data we are using were generated (Comparat et al. 2020, arXiv:2008.08404) to simulate exptected imaging of galaxy clusters at X-ray wavelengths for the eROSITA telescope (https://erosita.mpe.mpg.de/).

The provided file 'imagedata_cuts.npy' is a (small) dataset for storage on github.

All data is available here: https://www2011.mpe.mpg.de/~comparat/eROSITA_mock/

Step-by-step tutorial walkthrough

Let’s get started!

First we import some basic packages that we will need:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
from tensorflow import keras
import scipy
```

Step-by-step tutorial walkthrough

1) Load data from Sample File with Galaxy Cluster Images

The data file prepared is a smaller subset of the full data available.

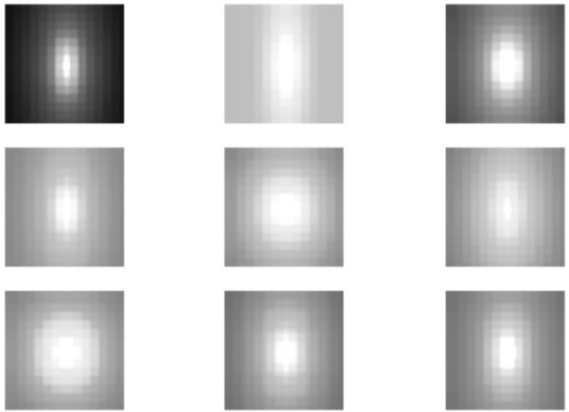
Question: What do you notice about the data? What are minima / maxima? What needs to be done in order to be able to input it to a network?

Example data loading and plotting:

```
data = np.load('imagedata_cuts.npy')
N_image = data.shape[0]
print('No. of images: ', N_image)

# let's have a look at some images
i=0
n_axis = 3
n_samples = n_axis*n_axis

for i in range(n_samples):
    plt.subplot(n_axis, n_axis, 1 + i)
    plt.axis('off')
    #print(data[np.random.randint(0,1000)].shape)
    plt.imshow(data[np.random.randint(0,1000)], cmap='gray',norm=cm.colors.LogNorm(vmin=3e-7,vmax=0.025))
plt.show()
```



Step-by-step tutorial walkthrough

2) Rescale images

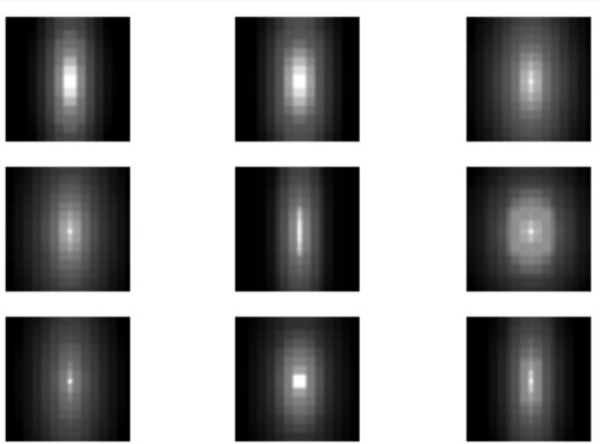
Task: Normalise the images in such a way they can be useful for network training.

Minmax log-normalisation:

```
j=0
scale = True
resizedimages = []
eps = 1e-14
im_size = 28

# we chose a minmax log-scaling here
while j< N_image:
    img = data[j]+eps
    if scale == True:
        img = -np.log(img)
        mincrop = np.min(img)
        maxcrop = np.max(img)
        if (maxcrop-mincrop)!=0.0:
            img = 1.0-1.0*(img-mincrop)/(maxcrop-mincrop)
    imgsave = np.array(img)
    del(img)
    resizedimages.append(imgsave)
    j=j+1
```

```
# let's have a look again at images rescaled to [0,1]
i=0
for i in range(n_samples):
    plt.subplot(n_axis, n_axis, 1 + i)
    plt.axis('off')
    #print(data[np.random.randint(0,1000)].shape)
    plt.imshow(resizedimages[np.random.randint(0,1000)], cmap='gray')
plt.show()
```



Step-by-step tutorial walkthrough

3a) Generator model

Task: Set up a generator model based on convolutional layers suitable to generate images of the desired dimension. Have a look at the model summary.

```python
def generator_model(latent_dim):
    model = keras.models.Sequential()
    # start with 7x7 image
    n_nodes = latent_dim * 7 * 7
    model.add(keras.layers.Dense(n_nodes, input_dim=latent_dim))
    model.add(keras.layers.LeakyReLU(alpha=0.2))
    model.add(keras.layers.Reshape((7, 7, latent_dim)))
    # upsample to 14x14
    model.add(keras.layers.Conv2DTranspose(latent_dim, (4,4), strides=(2,2), padding='same'))
    model.add(keras.layers.LeakyReLU(alpha=0.2))
    # upsample to 28x28
    model.add(keras.layers.Conv2DTranspose(latent_dim, (4,4), strides=(2,2), padding='same'))
    model.add(keras.layers.LeakyReLU(alpha=0.2))
    model.add(keras.layers.Conv2D(1, (7,7), activation='sigmoid', padding='same'))
    model.add(keras.layers.Activation('tanh'))
    return model
```

```python
latent_size = 128
g = generator_model(latent_size)
g.summary()
```

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 6272)	809088
leaky_re_lu (LeakyReLU)	(None, 6272)	0
reshape (Reshape)	(None, 7, 7, 128)	0
conv2d_transpose (Conv2DTran	(None, 14, 14, 128)	262272

Step-by-step tutorial walkthrough

3b) Discriminator model

Task: Set up a discriminator model suitable to generate images of the right dimension. How many parameters does the model have? Choose a final activation that makes sense for a mse loss.

```
def discriminator_model(in_shape=(im_size,im_size,1)):
    model = keras.models.Sequential()
    model.add(keras.layers.Conv2D(64, (3,3), strides=(2, 2), padding='same', input_shape=in_shape))
    model.add(keras.layers.LeakyReLU(alpha=0.2))
    model.add(keras.layers.Dropout(0.4))
    model.add(keras.layers.Conv2D(64, (3,3), strides=(2, 2), padding='same'))
    model.add(keras.layers.LeakyReLU(alpha=0.2))
    model.add(keras.layers.Dropout(0.4))
    model.add(keras.layers.Flatten())
    model.add(keras.layers.Dense(1, activation='linear'))
    opt = keras.optimizers.Adam(lr=0.00001, beta_1=0.5)
    model.compile(loss='mse', optimizer=opt, metrics=['accuracy'])
    return model
```

```
# check the model summary
d = discriminator_model()
d.summary()
# compile the discriminator
d_opt = keras.optimizers.Adam(learning_rate=1e-4, beta_1=0.5)
d.compile(loss='mse', optimizer=d_opt, metrics=['accuracy'])
```

Note: The discriminator is individually compiled, while the generator is only compiled as part of the full GAN model later on.

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 14, 14, 64)	640
leaky_re_lu_3 (LeakyReLU)	(None, 14, 14, 64)	0
dropout (Dropout)	(None, 14, 14, 64)	0
conv2d_2 (Conv2D)	(None, 7, 7, 64)	36928

Step-by-step tutorial walkthrough

4) Noise and sample generation

Task: Generate random noise to be used as a generator input. Prepare for the generation of real and fake samples.

Example noise and sample preparation:

```python
# generate points in latent space as input for the generator
def generate_latent_points(latent_dim, n_samples):
    # generate points in the latent space
    x_input = np.random.randn(latent_dim * n_samples)
    # reshape into a batch of inputs for the network
    x_input = x_input.reshape(n_samples, latent_dim)
    return x_input
```

```python
# use the generator to generate n fake examples
# with class labels 0 for generated images)
def generate_fake_samples(g_model, latent_dim, n_samples):
    # generate points in latent space
    x_input = generate_latent_points(latent_dim, n_samples)
    # predict outputs
    X = g_model.predict(x_input)
    # add class labels (0)
    y = np.zeros((n_samples, 1))
    return X, y

# draw and label real sample
def generate_real_samples(dataset, n_samples):
    # choose random instances
    n_samples = int(n_samples)
    lendata = int(len(dataset))
    ix = np.random.randint(0, lendata, n_samples)
    dataset = np.array(dataset)
    # retrieve selected images
    X = dataset[ix]
    X = np.expand_dims(X, axis = -1)
    # generate 'real' class labels (1)
    y = np.ones((n_samples, 1))
    return X, y
```

Step-by-step tutorial walkthrough

4) Noise and sample generation

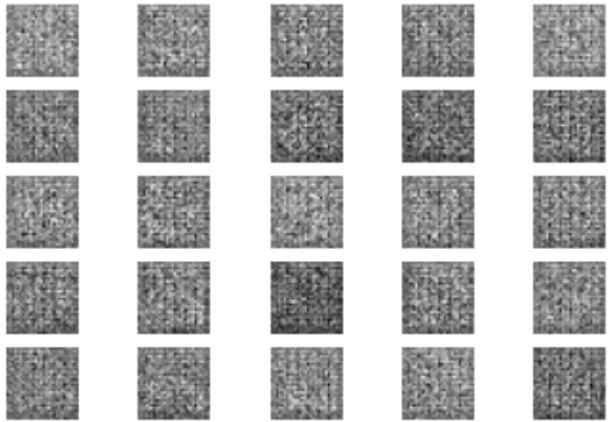
Task: Generate random noise to be used as a generator input. Prepare for the generation of real and fake samples.

Example noise and sample preparation:

```
# generate and plot generated noise samples
g_model = generator_model(latent_size)

n_samples = 25
X, _ = generate_fake_samples(g_model, latent_size, n_samples)

for i in range(n_samples):
    plt.subplot(5, 5, 1 + i)
    plt.axis('off')
    plt.imshow(X[i, :, :, 0], cmap='gray_r')
plt.show()
print(X[1,:,:,0].shape)
```



Step-by-step tutorial walkthrough

5) GAN model and training

Task: Define the GAN as a sequential model of G and D, combining the discriminator and the generator model, and choosing the appropriate loss again for our LSGAN.

```
def define_lsgan(g_model, d_model):
    # make weights in the discriminator not trainable
    d_model.trainable = False
    # connect them
    model = keras.models.Sequential()
    # add generator
    model.add(g_model)
    # add the discriminator
    model.add(d_model)
    # compile model
    opt = keras.optimizers.Adam(lr=0.002, beta_1=0.5)
    model.compile(loss='mse', optimizer=opt)
    return model
```

Note: We set trainable = False to keep discriminator model and thus weights fixed during generator training.

Step-by-step tutorial walkthrough

5) GAN model and training

Some auxiliaries, can be expanded upon (also as a task), e.g. saving and plotting loss curves.

```python
def save_plot(examples, epoch, n=2):
    # plot images
    for i in range(n * n):
        # define subplot
        plt.subplot(n, n, 1 + i)
        # turn off axis
        plt.axis('off')
        # plot raw pixel data
        plt.imshow(examples[i, :, :, 0], cmap='gray')
    # save plot to file
    filename = 'generated_plot_e%03d.png' % (epoch+1)
    plt.savefig(filename)
    plt.close()
```

```python
def summarize_performance(epoch, g_model, d_model, dataset, latent_dim, n_samples=100):
    # prepare real samples
    X_real, y_real = generate_real_samples(dataset, n_samples)
    # evaluate discriminator on real examples
    _, acc_real = d_model.evaluate(X_real, y_real, verbose=0)
    # prepare fake examples
    x_fake, y_fake = generate_fake_samples(g_model, latent_dim, n_samples)
    # evaluate discriminator on fake examples
    _, acc_fake = d_model.evaluate(x_fake, y_fake, verbose=0)
    # save plot
    save_plot(x_fake, epoch)
    # save the generator model tile file
    filename = 'generator_model_n%03d.h5' % (epoch + 1)
    g_model.save(filename)
```

Step-by-step tutorial walkthrough

5) GAN model and training

A GAN training function:

```
def train(g_model, d_model, gan_model, dataset, latent_dim, n_epochs=20, n_batch=500):
    bat_per_epo = int(dataset.shape[0] / n_batch)
    half_batch = int(n_batch / 2)

    for i in range(n_epochs):

        # enumerate batches over the training set
        for j in range(bat_per_epo):
            # get randomly selected 'real' samples
            X_real, y_real = generate_real_samples(dataset, half_batch)
            # generate 'fake' examples
            X_fake, y_fake = generate_fake_samples(g_model, latent_dim, half_batch)

            # create training set for the discriminator
            X, y = np.vstack((X_real, X_fake)), np.vstack((y_real, y_fake))
            # update discriminator model weights
            d_loss, _ = d_model.train_on_batch(X, y)

            # prepare points in latent space as input for the generator
            X_gan = generate_latent_points(latent_dim, n_batch)
            # create labels for the fake samples
            y_gan = np.ones((n_batch, 1))

            # update (train) the generator
            g_loss = gan_model.train_on_batch(X_gan, y_gan)

            # summarize loss on this batch
            print('epoch %d, batch %d/%d, loss d=%.3f, loss g=%.3f' % (i+1, j+1, bat_per_epo, d_loss, g_loss))
        save_plot(X_fake, i)
```

Note: D is updated first via d_model, then G is updated via (full) gan_model.

Step-by-step tutorial walkthrough

5) GAN model and training

Task: Train the GAN. Judge when the training starts to converge. What can and should be improved in order to create better images?

```
latent_dim = 100
# create the discriminator
d_model = discriminator_model()
# create the generator
g_model = generator_model(latent_dim)
# create the gan
gan_model = define_lsgan(g_model, d_model)

# load image data
dataset = np.array(resizedimages)

# train model
n_epochs = 10
train(g_model, d_model, gan_model, dataset, latent_dim, n_epochs)
```

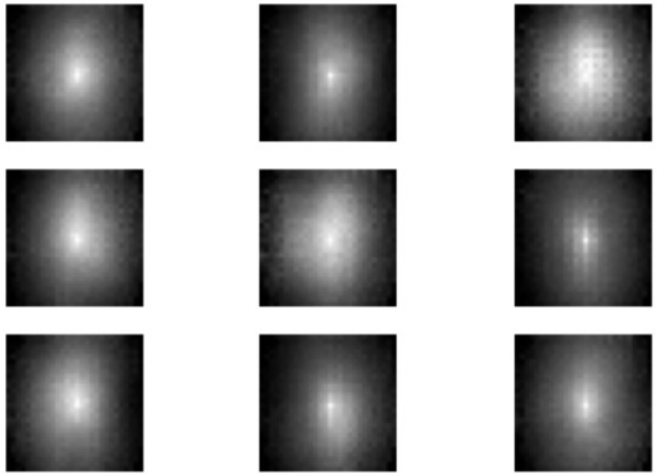
```
epoch 1, batch 1/7, loss d=0.446, loss g=0.877
epoch 1, batch 2/7, loss d=0.439, loss g=0.842
epoch 1, batch 3/7, loss d=0.444, loss g=0.793
epoch 1, batch 4/7, loss d=0.453, loss g=0.765
epoch 1, batch 5/7, loss d=0.454, loss g=0.727
epoch 1, batch 6/7, loss d=0.456, loss g=0.707
epoch 1, batch 7/7, loss d=0.456, loss g=0.677
epoch 2, batch 1/7, loss d=0.458, loss g=0.663
epoch 2, batch 2/7, loss d=0.454, loss g=0.658
epoch 2, batch 3/7, loss d=0.460, loss g=0.621
epoch 2, batch 4/7, loss d=0.456, loss g=0.627
epoch 2, batch 5/7, loss d=0.459, loss g=0.633
epoch 2, batch 6/7, loss d=0.460, loss g=0.610
epoch 2, batch 7/7, loss d=0.461, loss g=0.582
epoch 3, batch 1/7, loss d=0.454, loss g=0.614
epoch 3, batch 2/7, loss d=0.460, loss g=0.566
epoch 3, batch 3/7, loss d=0.469, loss g=0.535
epoch 3, batch 4/7, loss d=0.468, loss g=0.564
epoch 3, batch 5/7, loss d=0.472, loss g=0.514
epoch 3, batch 6/7, loss d=0.476, loss g=0.508
epoch 3, batch 7/7, loss d=0.473, loss g=0.481
epoch 4, batch 1/7, loss d=0.472, loss g=0.495
epoch 4, batch 2/7, loss d=0.473, loss g=0.481
epoch 4, batch 3/7, loss d=0.482, loss g=0.477
epoch 4, batch 4/7, loss d=0.477, loss g=0.469
```

Step-by-step tutorial walkthrough

5) GAN model and training

Task: Have a look at images generated with your trained model. Optional: Change the training function to pass on loss curves and plot these.

```
# generate samples for model just trained OR saved model
n_axis = 3
n_samples = n_axis*n_axis
latent_dim =100
do_load = True
if do_load:
    latent_dim = 10000
    g_model = keras.models.load_model('lsgan.h5')
    # or load a previous old model of yours here, e.g.:
    #g_model = keras.models.load_model('generator_model_n020.h5')
X, _ = generate_fake_samples(g_model, latent_dim, n_samples)
# plot the generated samples
for i in range(n_samples):
    # define subplot
    plt.subplot(n_axis, n_axis, 1 + i)
    # turn off axis labels
    plt.axis('off')
    plt.imshow(X[i, :, :, 0], cmap='gray')
plt.show()
```



Bonus: Compare results with the ones based on the model 'lsgan.h5' provided. Changes made in order to improve the model include: a) larger training sample, b) larger latent space, c) adaptive learning rate, d) smaller batch size, e) more epochs.