Generative Adversarial Networks

```
library(keras)
```

```
height <- 32
width <- 32
channels <- 3
```

GAN framework

In previous sessions a loss function based on a pre-trained classification network was defined. When Goodfellow et al. presented their article "Generative Adversarial Nets" in 2014 they took this idea one step further. Two networks are defined, one network called the *generator* and one network called the *discriminator*. The discriminator will, just as the CNN network did before, serve as a loss function for the generator but it will now be part of the training, in the sense of getting its weights updated. The name originates from the fact that the networks will compete against each other during training.

It consists of two networks, the generator and the discriminator. The input to the generator is a sample from some noise prior. The output of the generator is passed along to the discriminator. The discriminator is trained to separate fake samples from real samples and the generator is trained to fool the discriminator (Fig.1).

Latent space

The core idea behind representation learning is that instead of trying to model the high-dimensional sample space directly, we should instead describe each observation in the training set using some low-dimensional latent space and then learn a mapping function that can take a point in the latent space and map it to a point in the original domain. In other words, each point in the latent space is a representation of some high-dimensional image.

Figure 2. The cube represent the extremely high-dimensional space of all images. Representation learning tries to find the lower-dimensional manifold on which particular kinds of image lie (for example, the dog manifold).

Those characteristics that describe the visual content of a given semantic concept (for example: "dog") define a restriction in the space of all the images. Mathematically we say that a manifold is determined by each concept. Generative models based on deep learning learn to find these lower-dimensional manifolds.

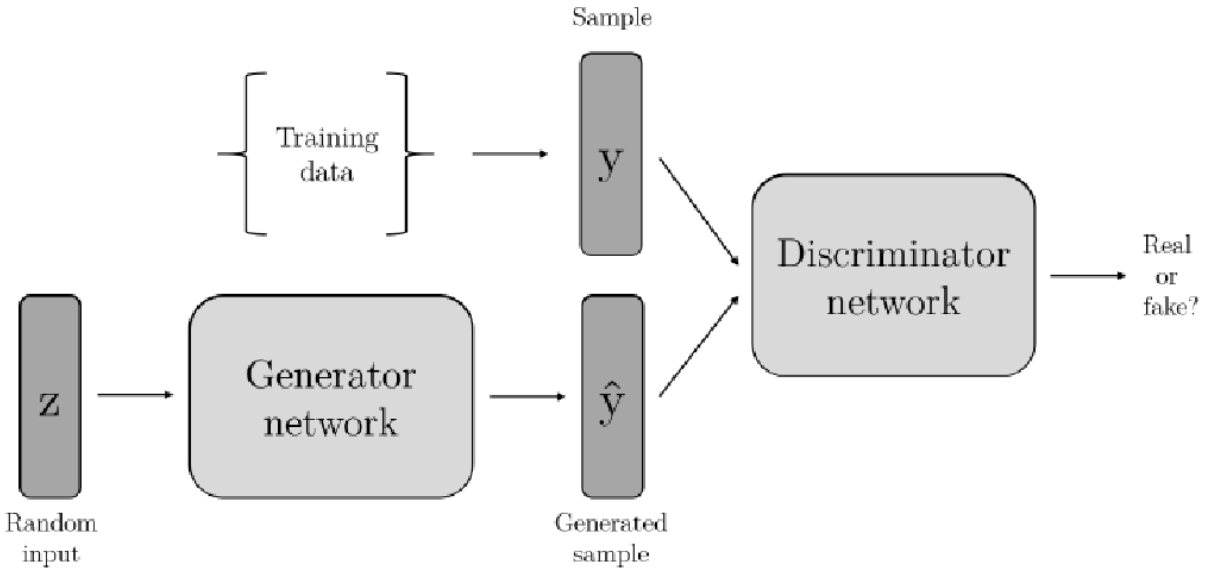
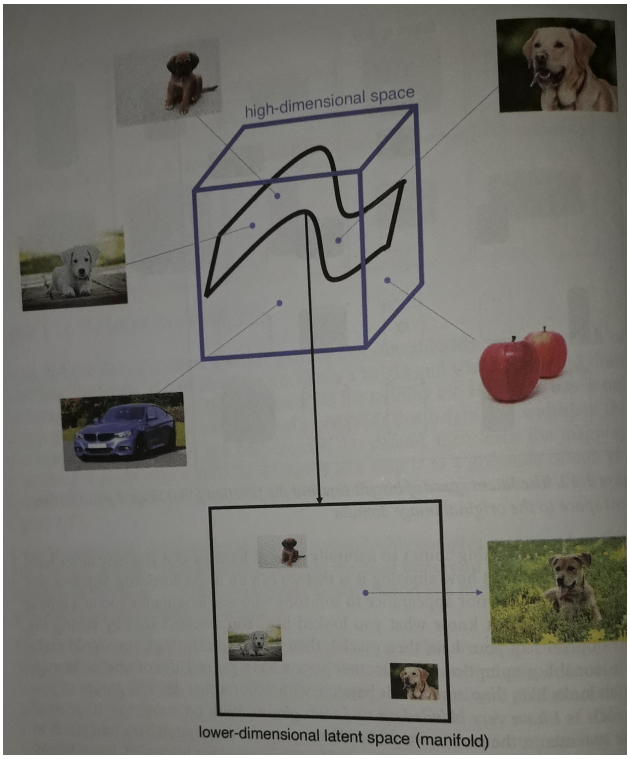

Sample
Training data
y
Discriminator network
Real or fake?
z
Generator network
ŷ
Random input
Generated sample

Figure 1:


high-dimensional space
lower-dimensional latent space (manifold)

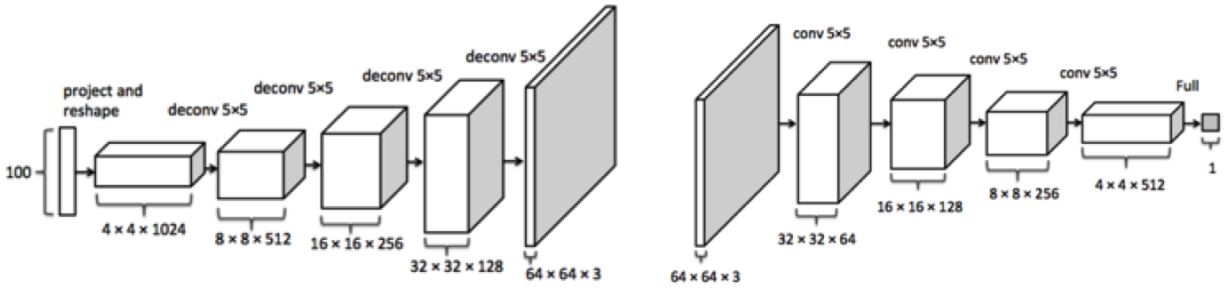


Figure 2:

Deep Convolutional GAN architecture

The discriminator

The goal of the discriminator is to predict if an image is real o fake. This is a supervised image classification task, so we can use stacked convolutional layers followed by a dense ouput layer.

```
discriminator_input <- layer_input(shape = c(height, width, channels))

discriminator_output <- discriminator_input %>%
  layer_conv_2d(filters = 64, kernel_size = 3,strides = 2,padding='same') %>%
  layer_activation_leaky_relu(alpha=0.2) %>%
  layer_dropout(rate = 0.4) %>%

  layer_conv_2d(filters = 64, kernel_size = 3, strides = 2,padding='same') %>%
  layer_activation_leaky_relu(alpha=0.2) %>%
  layer_dropout(rate = 0.4) %>%

  layer_flatten() %>%

  # Classification layer
  layer_dense(units = 1, activation = "sigmoid")

discriminator <- keras_model(discriminator_input, discriminator_output)
summary(discriminator)
```

```
## Model: "model"
## _________________________________________________________________________
## Layer (type)                 Output Shape              Param #
## =========================================================================
## input_1 (InputLayer)         [(None, 32, 32, 3)]       0
## _________________________________________________________________________
## conv2d (Conv2D)              (None, 16, 16, 64)        1792
## _________________________________________________________________________
## leaky_re_lu (LeakyReLU)      (None, 16, 16, 64)        0
## _________________________________________________________________________
## dropout (Dropout)            (None, 16, 16, 64)        0
## _________________________________________________________________________
## conv2d_1 (Conv2D)            (None, 8, 8, 64)          36928
## _________________________________________________________________________
## leaky_re_lu_1 (LeakyReLU)    (None, 8, 8, 64)          0
```

```
## _________________________________________________________________
## dropout_1 (Dropout)              (None, 8, 8, 64)              0
## _________________________________________________________________
## flatten (Flatten)                (None, 4096)                  0
## _________________________________________________________________
## dense (Dense)                    (None, 1)                     4097
## =================================================================
## Total params: 42,817
## Trainable params: 42,817
## Non-trainable params: 0
## _________________________________________________________________
```

The generator

The input to the generator is a vector, usually drawn form a multivariate standard normal distribution. The ouput is an image of the same size as an image in the original training data. The description may remind you of the decoder in an autoencoder; converting a set of deep descriptors to an image.The concept of mapping from a latent space back to the original domain is very common in generative modeling as it give us the ability to manipulate vectors in latent space to change high-level features of images in the original domain.

```
latent_dim <- 100
generator_input <- layer_input(shape = c(latent_dim))

generator_output <- generator_input %>%

  # First, transform the input into a 16x16 128-channels feature map
  layer_dense(units = 128 * 8 * 8) %>%
  layer_activation_leaky_relu(alpha = 0.2) %>%
  layer_reshape(target_shape = c(8, 8, 128)) %>%

  # Then, add a convolution layer
  layer_conv_2d_transpose(filters = 128, kernel_size = 4, strides = 2,
                padding = "same") %>%
  layer_activation_leaky_relu(alpha = 0.2) %>%

  # Upsample to 32x32
  layer_conv_2d_transpose(filters = 256, kernel_size = 4,
                          strides = 2, padding = "same") %>%
  layer_activation_leaky_relu(alpha = 0.2) %>%

  # Produce a 32x32 1-channel feature map
  layer_conv_2d(filters = channels, kernel_size = 7,
                activation = "tanh", padding = "same")

generator <- keras_model(generator_input, generator_output)
summary(generator)
```

```
## Model: "model_1"
## _________________________________________________________________
## Layer (type)                     Output Shape                  Param #
## =================================================================
## input_2 (InputLayer)             [(None, 100)]                 0
## _________________________________________________________________
## dense_1 (Dense)                  (None, 8192)                  827392
```

```
## _________________________________________________________________
## leaky_re_lu_2 (LeakyReLU)        (None, 8192)              0
## _________________________________________________________________
## reshape (Reshape)                (None, 8, 8, 128)         0
## _________________________________________________________________
## conv2d_transpose (Conv2DTranspos (None, 16, 16, 128)       262272
## _________________________________________________________________
## leaky_re_lu_3 (LeakyReLU)        (None, 16, 16, 128)       0
## _________________________________________________________________
## conv2d_transpose_1 (Conv2DTransp (None, 32, 32, 256)       524544
## _________________________________________________________________
## leaky_re_lu_4 (LeakyReLU)        (None, 32, 32, 256)       0
## _________________________________________________________________
## conv2d_2 (Conv2D)                (None, 32, 32, 3)         37635
## =================================================================
## Total params: 1,651,843
## Trainable params: 1,651,843
## Non-trainable params: 0
## _________________________________________________________________
```

Training the GAN

We can train the discriminator by creating a training set where some of the images are randomly selected real observations from the training set and some are outputs from the generator. Let us supose that the response would be 1 for the true images and 0 for the generated images. If we treat this a supervised learning task, we can train the discriminator to learn how to describe the differences between the original and generated images, outputting values near 1 for true images and values near 0 for the fake images.

Training the generator is more difficult as there is no training set that tells us the true image that a particular point in the latent space should be mapped to.

Instead, we only want the image that is generated to fool the discriminator, that is, when the image is fed as input to the discriminator, we want the output to be close to 1.

Therefore, to train the generator, we must first connect it to discriminator to create a Keras model (the GAN) that we can train. Specifically, we feed the output from the generator (a 32x32x3 image) into the discriminator so that the ouput for this combined model is the probability that the generated image is `real`, according to the discrimator. We can train this combined model by creating training batches consisting of randomly sampled 100-dimensional latent vectors as input and a response which is set to 1, since we want to train the generator to produce images that the discriminator thinks are real.

The loss function is then just the binary cross-entropy loss between the output from the discriminator and the reponse vector 1.

CRUCIALLY, we must freeze the weights of the discriminator while we are training the combined model, so that only the generator weights are updated.

If we do not freeze the discriminator weights, the discriminator will adjust so that it is more likely to predict generated images as real, which is not the desired outcome. We want generated images to be predict close to 1 (real) because the generator is strong, not because the discriminator is weak (see Fig. 4).

Latent vector arithmetic

One benefit of mapping images into a lower-dimensional space is that we can perform arithmetic on vectors in this latent space that has visual analogue when decoded back into the original image space.

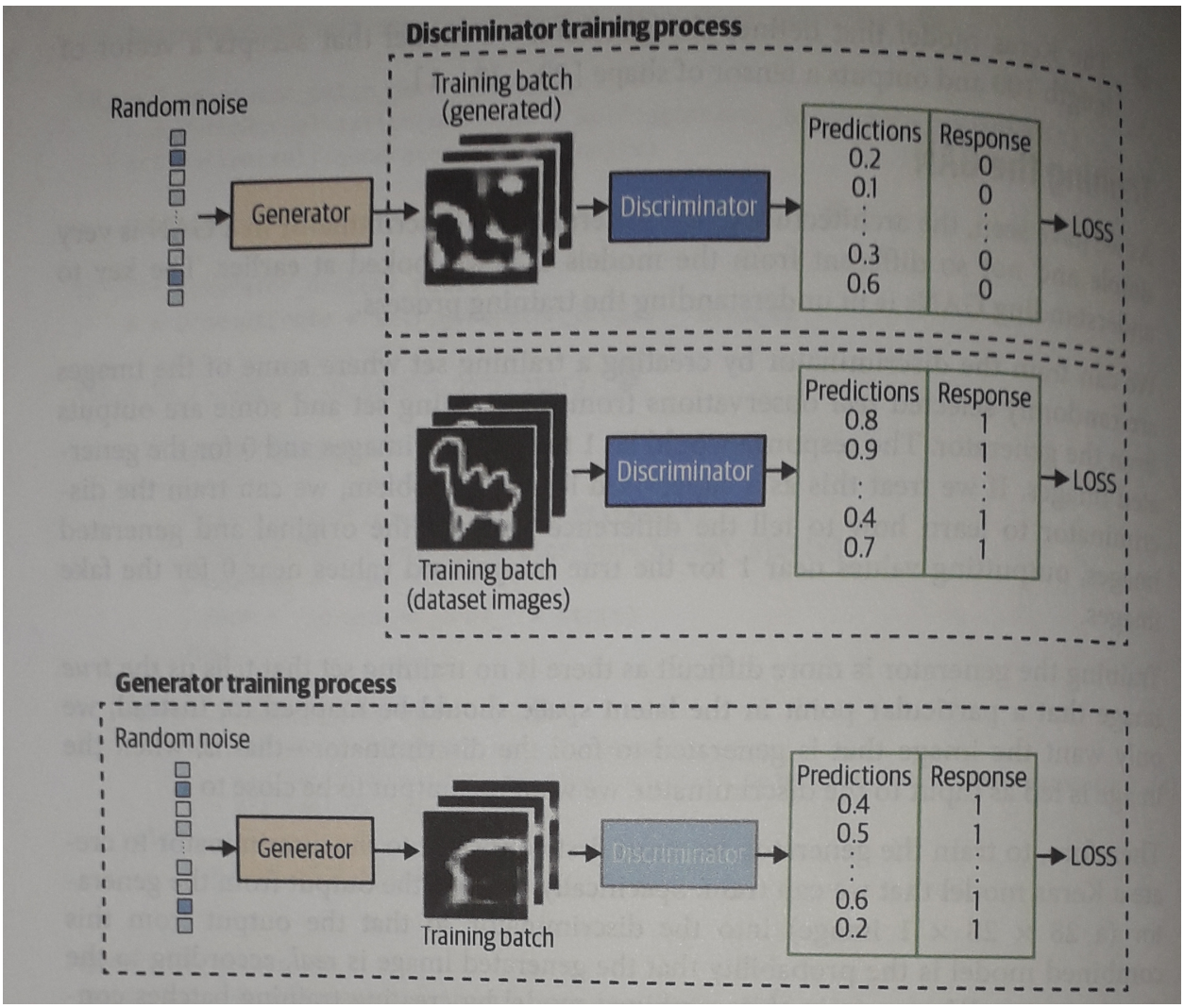

Discriminator training process
Random noise
Generator
Training batch
(generated)
Discriminator
Predictions
0.2
0.1
0.3
0.6
Response
0
0
0
0
LOSS
Discriminator
Predictions
0.8
0.9
0.4
0.7
Response
1
1
1
1
LOSS
Training batch
(dataset images)
Generator training process
Random noise
Generator
Training batch
Discriminator
Predictions
0.4
0.5
0.6
0.2
Response
1
1
1
1
LOSS

Figure 3:

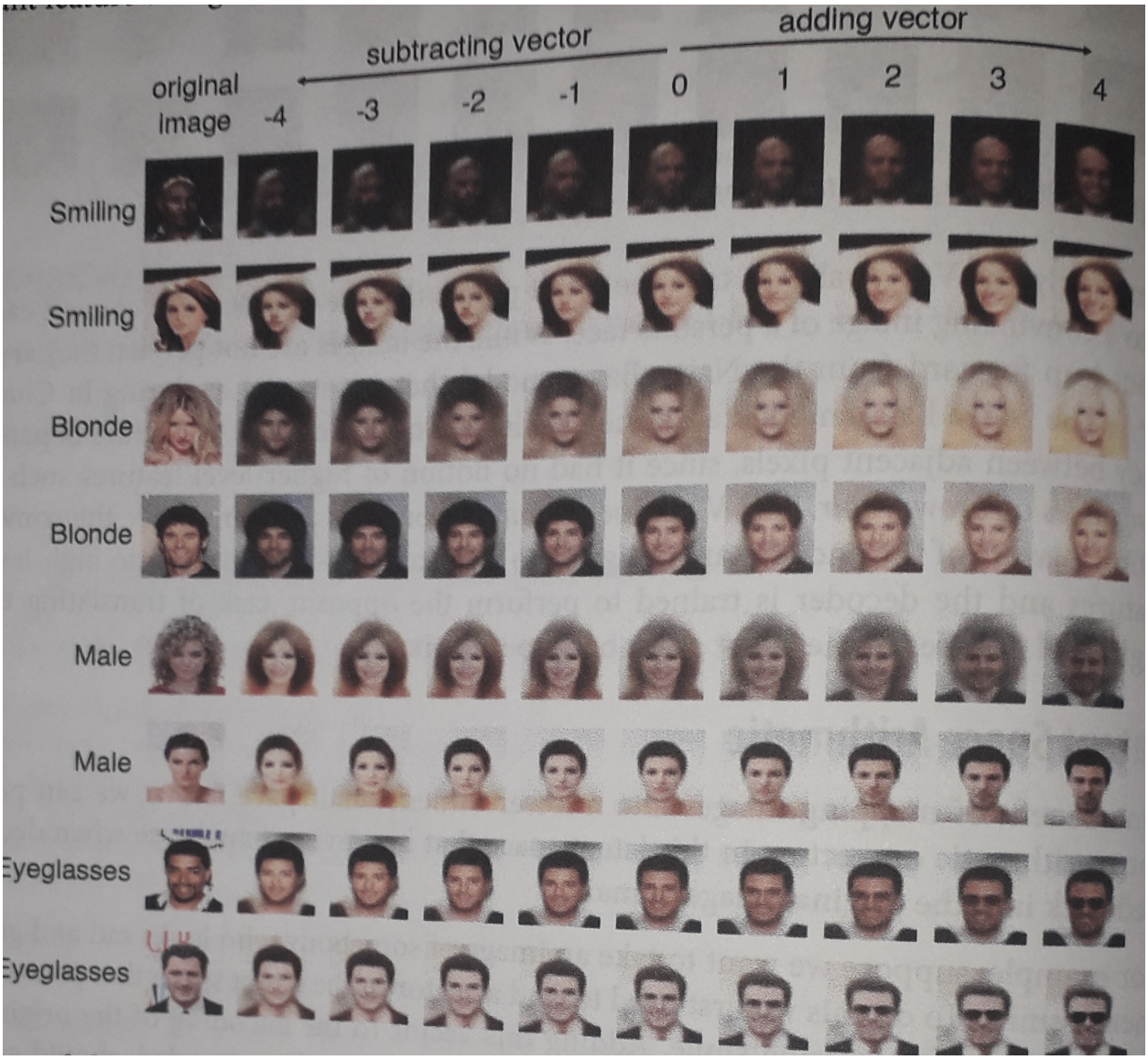

subtracting vector
adding vector
original image
-4
-3
-2
-1
0
1
2
3
4
Smiling
Smiling
Blonde
Blonde
Male
Male
Eyeglasses
Eyeglasses

Figure 4:

For example, supose we want to take an image of somebody who looks sad and give them a smile. To do this we first need to find a vector in the laten space that points in the direction of increasing smile. Adding this vector to the encoding to the original image in the latent space will give us a new point which, when decoded, should give us a more smiley version of the original image.

So how can we find the `smile` vector? Each image in the CelebA dataset is labeled with attributes, one of which is `smiling`. If we take the average position of encoded images in the latent space with the attribute `smiling` and subtract the average position of encoded images that do not have the attribute `smiling`, we will obtain the vector that points from `not smiling` to `smiling`, which is exactly what we need.

Conceptually, we are performing the following vector operation in the latent sapce, where α is a factor that determines how much of the feature vector is added or subtracted:

$$z_{new} = z + \alpha * feature\,vector$$

Figure 5 shows several images that have been encoded into the latent space. We then add or subtract multiples of a certain vector (smile, blonde, male, eyeglasses) to obtain different versions of the image, with only the relevant feature changed.

References

Foster, D. (2019). Generative deep learning: teaching machines to paint, write, compose, and play. O'Reilly Media.

http://mmlab.ie.cuhk.edu.hk/projects/CelebA.html

https://heartbeat.fritz.ai/stylegans-use-machine-learning-to-generate-and-customize-realistic-images-c943388dc672

https://www.youtube.com/watch?v=kSLJriaOumA