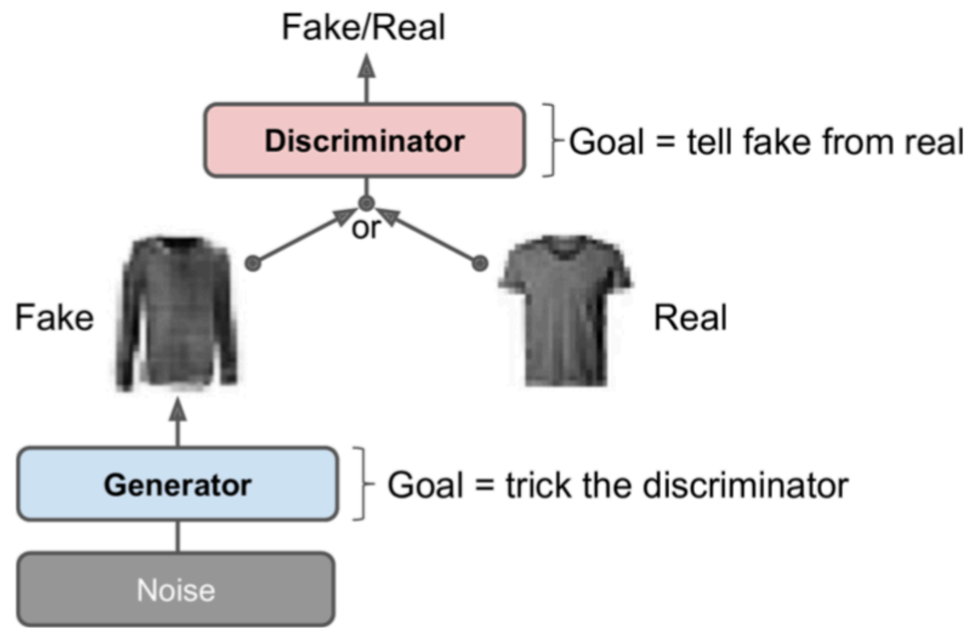
Generative Adversarial Networks (GANs)

Generator

- Takes a random distribution as input (Gaussian) and outputs some data - typically, an image. The random inputs are similar to codings
- Same functionality as a decoder in a VAE but it is trained differently.

Discriminator

- Takes either a fake image from the generator or a real image from the training set as input, and must guess whether the input image is fake or real.



- The generator & the discriminator have opposite training goals: the discriminator tries to tell fake images from real images, while the generator tries to produce images that look real enough to trick the discriminator.

- Each training iteration has two phases

- 1st phase (discriminator).

  A batch of real images (label=1) is sampled from the training set + an equal number of fake images (label=0) produced by the generator. Discriminator is trained on this batch for 1 step, using the binary cross-entropy loss. Backprop optimizes only the weights of the discriminator

- 2nd phase (generator)

  Generator produces another batch of fake images (label=1, real), and then the discriminator tells whether the images are fake or real. The weights of the discriminator are frozen during this step, so backprop only affects the weights of the generator.

- The generator never sees any real images - it only gets the discriminator gradients. 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.

- Let's look at the core code for a GAN built for Fashion MNIST

```
codings_size = 30

generator = keras.models.Sequential([
        keras.layers.Dense(100, activation="selu",
        input_shape=[codings_size]),
        keras.layers.Dense(150, activation="selu"),
        keras.layers.Dense(28 * 28, activation="sigmoid"),
        keras.layers.Reshape([28, 28])
])
```

```
discriminator = keras.models.Sequential([
      keras.layers.Flatten(input_shape=[28, 28]),
      keras.layers.Dense(150, activation="selu"),
      keras.layers.Dense(100, activation="selu"),
      keras.layers.Dense(1, activation="sigmoid")
  ])

  gan = keras.models.Sequential([generator, discriminator])
```

- This is included in a loop – note that the generator need to be compiled

The training loop needs to be build from scratch (without using a fit() method)

```
batch_size = 32
dataset = tf.data.Dataset.from_tensor_slices(X_train).shuffle(1000)
dataset = dataset.batch(batch_size, drop_remainder=True).prefetch(1)
```

```
def train_gan(gan, dataset, batch_size, codings_size, n_epochs=50):
    generator, discriminator = gan.layers
    for epoch in range(n_epochs):
        for X_batch in dataset:
            # phase 1 - training the discriminator
            noise = tf.random.normal(shape=[batch_size, codings_size])
            generated_images = generator(noise)
            X_fake_and_real = tf.concat([generated_images, X_batch], axis=0)
            y1 = tf.constant([[0.]] * batch_size + [[1.]] * batch_size)
            discriminator.trainable = True
            discriminator.train_on_batch(X_fake_and_real, y1)
            # phase 2 - training the generator
            noise = tf.random.normal(shape=[batch_size, codings_size])
            y2 = tf.constant([[1.]] * batch_size)
            discriminator.trainable = False
            gan.train_on_batch(noise, y2)

train_gan(gan, dataset, batch_size, codings_size)
```

The images generated by the GAN are a “pointillist” version of the original

They don't improve much after more epochs due to inherent problems with training GANs - mode collapse. This problem is tightly connected with the game theory phenomenon called Nash equilibrium.

Nash equilibrium (for games) in layman terms is:

“No player would be better off changing their own strategy, assuming the other players do not change theirs. “

Examples

- One equilibrium is reached when everyone drives on the left side of the road: no driver would be better off being the only one to switch sides. Another: when everyone drives on the right side of the road.

- In this example, there is a single optimal strategy once an equilibrium is reached (i.e., driving on the same side as everyone else), but a Nash equilibrium can involve multiple competing strategies (e.g., a predator chases its prey, the prey tries to escape, and neither would be better off changing their strategy).

- Mode collapse - the biggest difficulty for GAN and it is when the generator’s outputs gradually become less diverse.

- How can this happen?
  Suppose that the generator gets better at producing convincing shoes than any other class. It will fool the discriminator a bit more with shoes, and this will encourage it to produce even more images of shoes.

- Gradually, it will forget how to produce anything else. Meanwhile, the only fake images that the discriminator will see will be shoes, so it will also forget how to discriminate fake images of other classes.

- Eventually, when the discriminator manages to discriminate the fake shoes from the real ones, the generator will be forced to move to another class. It may then become good at shirts, forgetting about shoes, and the discriminator will follow. The GAN may gradually cycle across a few classes, never really becoming very good at any of them.

- Parameters may end up oscillating and becoming unstable. GANs are very sensitive to the hyperparameters: you may have to spend a lot of effort fine-tuning them.

Some approaches to avoid this problem with GANa are

- Store (fake) images produced by the generator in a replay buffer. Then use them later in the training and this way avoiding the discriminator overfitting the latest output from the generator

- Another technique is to measure how similar the images across a batch and reject batches (of fake images) that lack diversity. This is called mini-batch discrimination

Deep Convolutional GANs (DCGAN)

In 2015, improvements in the general GAN architecture were made

- Replace any pooling layers with convolutions with strides (in the discriminator) and transposed convolutions (in the generator).
- Use Batch Normalization in both the generator and the discriminator, except in the generator's output layer and the discriminator's input layer.
- Remove fully connected hidden layers for deeper architectures.
- Use ReLU activation in the generator for all layers except the output layer, which should use tanh.
- Use leaky ReLU activation in the discriminator for all layers.

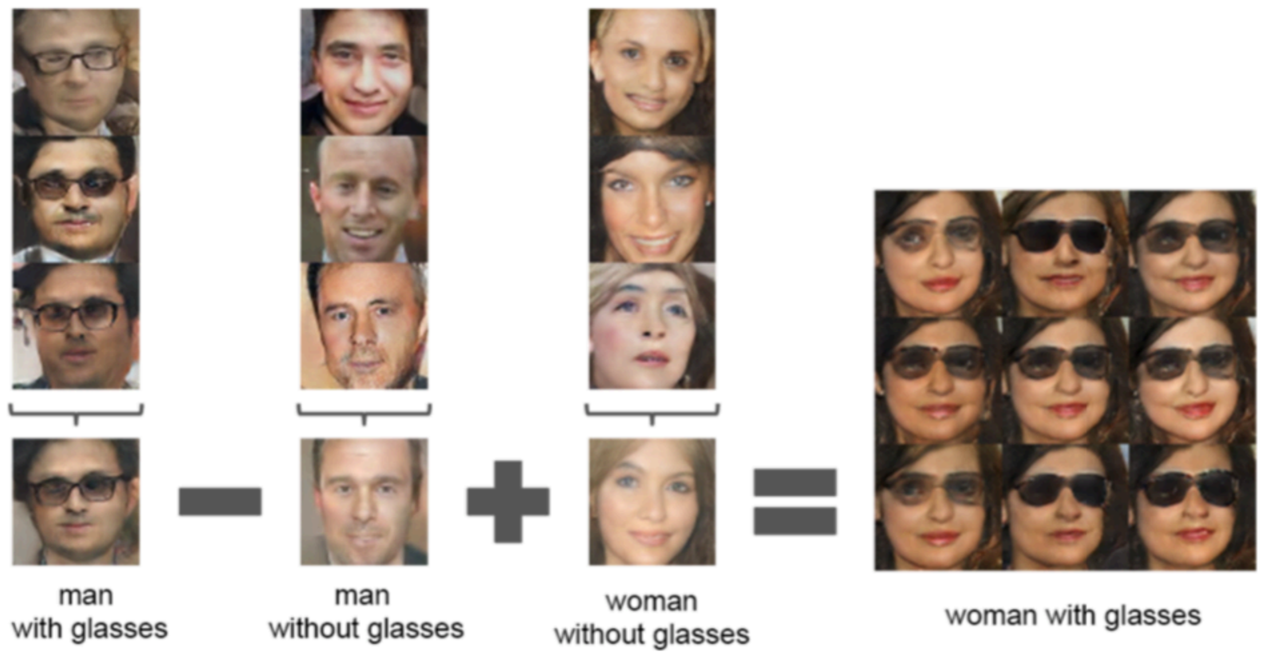
See an example of thus suggested DCGAN for Fashion MNIST

```python
codings_size = 100

generator = keras.models.Sequential([
    keras.layers.Dense(7 * 7 * 128, input_shape=[codings_size]),
    keras.layers.Reshape([7, 7, 128]),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2DTranspose(64, kernel_size=5, strides=2, padding="same",
                                 activation="selu"),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2DTranspose(1, kernel_size=5, strides=2, padding="same",
                                 activation="tanh")
])
discriminator = keras.models.Sequential([
    keras.layers.Conv2D(64, kernel_size=5, strides=2, padding="same",
                        activation=keras.layers.LeakyReLU(0.2),
                        input_shape=[28, 28, 1]),
    keras.layers.Dropout(0.4),
    keras.layers.Conv2D(128, kernel_size=5, strides=2, padding="same",
                        activation=keras.layers.LeakyReLU(0.2)),
    keras.layers.Dropout(0.4),
    keras.layers.Flatten(),
    keras.layers.Dense(1, activation="sigmoid")
])
gan = keras.models.Sequential([generator, discriminator])
```

- The images on right generated by the network look quite realistic
- But there is more - such a DCGAN can learn meaningful latent image representations.

- Out of many DCGAN generated images, 9 are picked (top left)
- The codings in each of the 3 groups (men w/wo glasses and women without glasses) are averages
- Then “man with glasses” – “man w/o glasses” + “women w/o glasses”:



- This “face arithmetic” is similar to the word embedding one

- DCGANs can make mistakes too – e.g. generating pants images where one leg is shorter or there is a third leg, see two slides up.

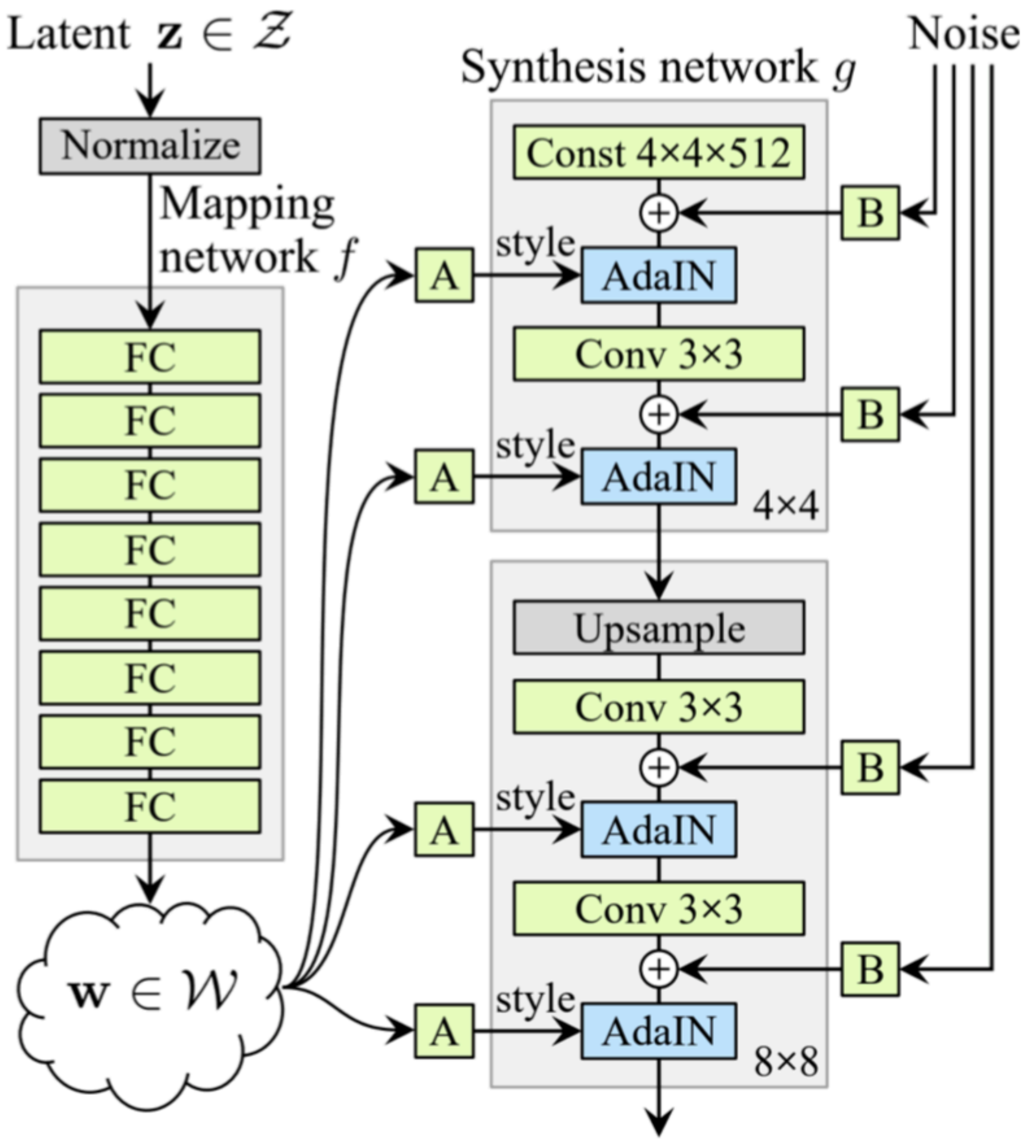
- To eliminate these overall inconsistences, new networks were invented

  - Progresive Growing GANs (Nvidia) – read

  - Style GANs (Nvidia)

    - Style transfer techniques in the generator were added. The discriminator and the loss function were not modified

    - Look for a schema of the modifications of the generator part of the GAN on next page and read detailed explanations in the text


Latent $\mathbf{z} \in \mathcal{Z}$
Normalize
Mapping network f
FC
FC
FC
FC
FC
FC
FC
FC
$\mathbf{w} \in \mathcal{W}$
Synthesis network g
Noise
Const 4×4×512
B
style
A
AdaIN
Conv 3×3
B
style
A
AdaIN
4×4
Upsample
Conv 3×3
B
style
A
AdaIN
Conv 3×3
B
style
A
AdaIN
8×8