Variational AutoEncoders (VAEs) and Generative Adversarial Networks (GANs).

Dr Haider Raza
Postdoctoral Research Fellow @IADS
University of Essex

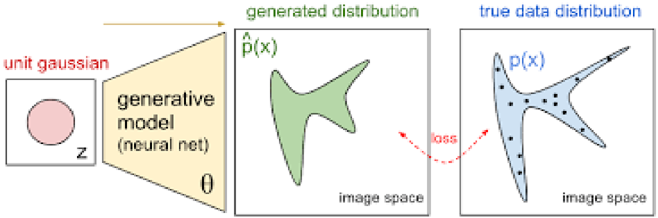
March 11, 2019

About

- We will be discussing Generative deep learning
  - Why generative models
- Creating entirely new images or edit existing ones is currently the most popular and successful application of creative AI
- We will review some high-level concepts pertaining to image generation, alongside implementations details relative to the two main techniques Variational AutoEncoders (VAEs) and Generative Adversarial Networks (GANs)
- The techniques we present here aren't specific to images—you could develop latent spaces of sound, music, or even text, using GANs and VAEs

Generative Models

- Given an observable variable X and a target variable Y, a generative model is a statistical model of the joint probability distribution on X $\times$ Y, P(X,Y)

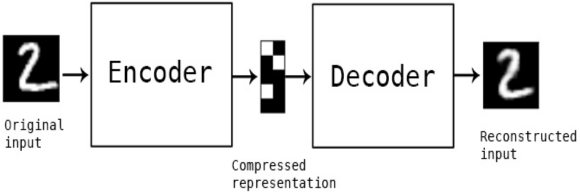


In other words, generative models model a distribution over high dimensional space

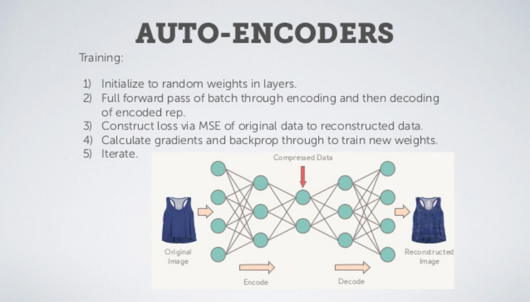
Auto-Encoders

An `autoencoder` is a type of artificial neural network used to learn efficient data coding in an unsupervised manner. It has two interesting properties:

1. The number of neurons is same in the input and output
2. We have a bottleneck (latent space) in one of these layers (low dimensional representation of data with less neurons)
3. The encoder acts as a way to compress the input data into fewer bits of information



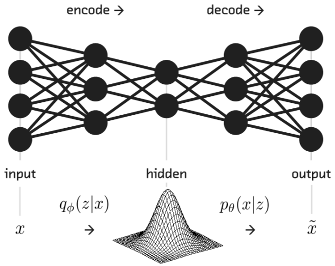
Auto-Encoders



Problems:

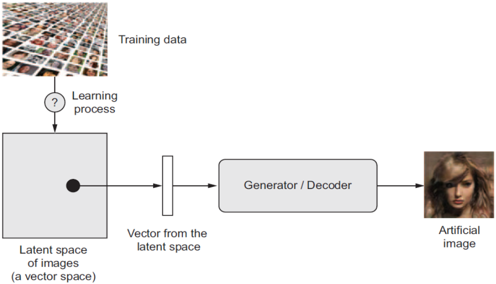
1. They will overfit unless large training data
2. Gradients diminish quickly so weigh updates get progressively smaller, the further we go from the output (vanishing gradient)

Sampling from latent spaces of images



1. The key idea of image generation is to develop a low-dimensional latent space of representations (which naturally is a vector space) where any point can be mapped to a realistic-looking image.
2. The module capable of realizing this mapping, taking as input a latent point and outputting an image (a grid of pixels), is called a generator (in the case of GANs) or a decoder (in the case of VAEs).

Sampling from latent spaces of images..



1. They are built on top of standard function approximators (neural networks), and can be trained with SGD
2. VAEs are great for learning latent spaces that are well structured, where specific directions encode a meaningful axis of variation in the data

Concept vectors for image editing

Given a latent space/embedding space of representations, certain directions in the space may encode interesting axes of variation in the original data

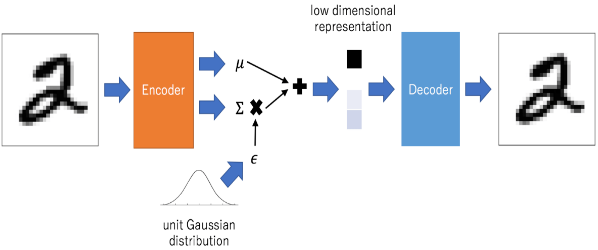
1. In a `latent space` of images of faces, for instance, there may be a `smile vector s`, such that if `latent point z` is the embedded representation of a certain face, then `latent point z + s` is the embedded representation of the same face, smiling.
2. There are `concept vectors` for essentially any independent dimension of variation in image space—in the case of faces, you may discover vectors for adding `sunglasses to a face`, `turning a male face into as female face, and so on`.

Concept vectors for image editing

An example of a `smile vector`, a `concept vector` discovered by Tom White from the Victoria University School of Design in New Zealand, using `VAEs` trained on a dataset of faces of celebrities (the CelebA dataset)

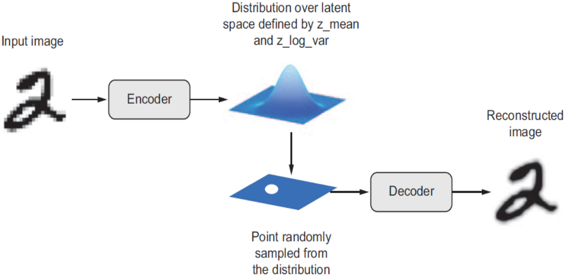
Variational autoencoders (VARs)

1. **VARs** are modern take on **autoencoders**. A type of network that aims to encode an input to a low-dimensional latent space and then decode it back—that mixes ideas from deep learning with Bayesian inference
2. A **VAE**, turns the image into the parameters of a statistical distribution: a **mean** and a **variance**. Assuming the input image has been generated by a statistical process and that the randomness of this process should be taken into accounting during encoding and decoding



VARs...

3. The VAE then uses the **mean** and **variance** parameters to randomly sample one element of the distribution, and decodes that element back to the original input
4. The stochasticity of this process improves robustness and forces the latent space to encode meaningful representations everywhere: every point sampled in the latent space is decoded to a valid output



VARs Algorithm

In technical terms, here's how a VAE works:

1 An encoder module turns the input samples `input_img` into two parameters in a latent space of representations, `z_mean` and `z_log_variance`.

2 You randomly sample a point `z` from the latent normal distribution that's assumed to generate the input image, via `z = z_mean + exp(z_log_variance) * epsilon`, where `epsilon` is a random tensor of small values.

3 A decoder module maps this point in the latent space back to the original input image.

1. **`epsilon`** is random, the process ensures that every point that's close to the latent location where you encoded **`input_img (z-mean)`** can be decoded to something similar to **`input_img`**, thus forcing the latent space to be continuously meaningful
2. Two close points in latent space will decode to highly similar images.
3. Parameters of a VAE are trained via two loss functions: a **`reconstruction loss`** that forces the decoded samples to match the initial inputs, and a **`regularization loss`** that helps learn well-formed latent spaces and reduce overfitting to the training data

VARs Conclusion

1. Image generation with deep learning is done by learning latent spaces that capture statistical information about a dataset of images. By sampling and decoding points from the latent space, you can generate never-before-seen images.
2. VAEs result in highly structured, continuous latent representations. For this reason, they work well for doing all sorts of image editing in latent space: face swapping, turning a frowning face into a smiling face, and so on.

Generative Adversarial Networks (GANs)

Generative adversarial networks (GANs) by Goodfellow et al 2014., are an alternative to VAEs for learning latent spaces of images. They enable the generation of fairly realistic synthetic images by forcing the generated images to be statistically almost indistinguishable from real ones

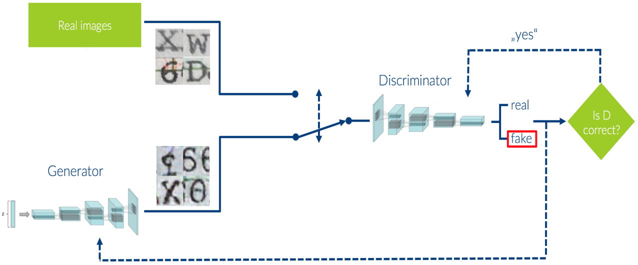
Example: taken from François Chollet book

1. A `forger` trying to create a fake `Picasso` painting
2. At first, the `forger` is pretty bad at the task
3. He mixes some fakes with authentic `Picassos` and present to `art dealer`
4. `Art dealer` makes an authenticity check and provide feedback about what makes a `Picasso` look like `Picasso`
5. The `forger` goes back to his studio to prepare some `new fakes` and as time goes, `forger` becomes expert in `making fake` and `dealer` becomes expert in `spotting fakes`
6. In the end, they have on their hands some excellent fake `Picassos`

GANs..

GAN is a combination of: a forger network and an expert network, each being trained to best the other. As such, a GAN is made of two parts:

1. `Generator network`: Takes as input a `random vector` (a random point in the `latent space`), and `decodes` it into a synthetic image
2. `Discriminator network (or adversary)`: Takes as input an `image` (real or synthetic), and `predicts` whether the image came from the `training set` or was created by the `generator network`



GANs

1. GAN is a dynamic system, where the optimization process is seeking not a minimum, but an equilibrium between two forces.
2. For this reason, GANs are notoriously difficult to train - getting a GAN to work requires lots of careful tuning of model architecture and training parameters.

Latent space dwellers. Images generated by Mike Tyka using a multi-staged GAN trained on a dataset of faces

A schematic GAN implementation (Keras)

The specific implementation is a `deep convolutional GAN (DCGAN)`: a GAN where the generator and discriminator are deep convnets. In particular, it uses a `Conv2DTranspose` layer for image upsampling in the `generator`

1. A `generator network` maps vectors of shape `(latent_dim,)` to images of shape `(32, 32, 3)`
2. A `discriminator` network maps images of shape `(32, 32, 3)` to a binary score estimating the probability that the image is real
3. A `gan` network chains the generator and the discriminator together: `gan(x) = discriminator(generator(x))`. Thus this `gan` network maps latent space vectors to the `discriminator's` assessment of the realism of these latent vectors as decoded by the `generator`
4. You train the `discriminator` using examples of `real` and `fake` images along with `"real"/"fake"` labels, just as you train any regular image-classification model.
5. To train the `generator`, you use the gradients of the generator's weights with regard to the loss of the gan model. This means, at every step, you move the weights of the generator in a direction that makes the discriminator more likely to classify as "real" the images decoded by the generator. In other words, you train the generator to fool the discriminator#

A bag of tricks for GANs

1. Use `tanh` as the last activation in the `generator`, instead of `sigmoid`, which is more commonly found in other types of models
2. It sample points from the latent space using a `normal distribution` (Gaussian distribution), not a uniform distribution
3. `Stochasticity` is good to induce robustness. In finding dynamic equilibrium they are likely to get stuck in all sort of ways. Introduce randomness in two ways: by using `dropout` in the `discriminator` and by adding `random noise` to labels for the `discriminator`
4. `Sparse gradients` can hinder GAN training. In deep learning, sparsity is often a desirable property, but not in `GANs`. Two things can induce `gradient sparsity`: `max pooling` operations and `ReLU` activations. Instead of max pooling, it is recommended using `strided convolutions` for downsampling, and using a `LeakyReLU` layer instead of a `ReLU` activation. It's similar to ReLU, but it relaxes sparsity constraints by allowing small negative activation values
5. In generated images, it's common to see checker-board artifacts caused by unequal coverage of the pixel space in the `generator`. To fix this, use a `kernel size` that's divisible by the `stride size` whenever we use a strided `Conv2DTranpose` or `Conv2D` in both the `generator` and the `discriminator`

The Generator

1. A `generator` model that turns a vector (from the latent space—during training it will be sampled at random) into a candidate image.
2. Issue with GAN that generator get stuck with generated images that look like noise. `Solution`: Use `dropout` on both side `discriminator` and `generator`

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 32)	0
dense_1 (Dense)	(None, 32768)	1081344
leaky_re_lu_1 (LeakyReLU)	(None, 32768)	0
reshape_1 (Reshape)	(None, 16, 16, 128)	0
conv2d_1 (Conv2D)	(None, 16, 16, 256)	819456
leaky_re_lu_2 (LeakyReLU)	(None, 16, 16, 256)	0
conv2d_transpose_1 (Conv2DTr	(None, 32, 32, 256)	1048832
leaky_re_lu_3 (LeakyReLU)	(None, 32, 32, 256)	0
conv2d_2 (Conv2D)	(None, 32, 32, 256)	1638656
leaky_re_lu_4 (LeakyReLU)	(None, 32, 32, 256)	0
conv2d_3 (Conv2D)	(None, 32, 32, 256)	1638656
leaky_re_lu_5 (LeakyReLU)	(None, 32, 32, 256)	0
conv2d_4 (Conv2D)	(None, 32, 32, 3)	37635

Total params: 6,264,579
Trainable params: 6,264,579
Non-trainable params: 0

The Discriminator

A `discriminator` model that takes as input a candidate `image (real or synthetic)` and classifies it into one of two classes: `"generated image"` or `"real image"` that comes from the training set

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	(None, 32, 32, 3)	0
conv2d_5 (Conv2D)	(None, 30, 30, 128)	3584
leaky_re_lu_6 (LeakyReLU)	(None, 30, 30, 128)	0
conv2d_6 (Conv2D)	(None, 14, 14, 128)	262272
leaky_re_lu_7 (LeakyReLU)	(None, 14, 14, 128)	0
conv2d_7 (Conv2D)	(None, 6, 6, 128)	262272
leaky_re_lu_8 (LeakyReLU)	(None, 6, 6, 128)	0
conv2d_8 (Conv2D)	(None, 2, 2, 128)	262272
leaky_re_lu_9 (LeakyReLU)	(None, 2, 2, 128)	0
flatten_1 (Flatten)	(None, 512)	0
dropout_1 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 1)	513

Total params: 790,913
Trainable params: 790,913
Non-trainable params: 0

The Adversarial Network

1. Finally, you'll set up the GAN, which chains the generator and the discriminator
2. This model will move the generator in a direction that improves its ability to fool the discriminator
3. Training `gan` will update the weights of `generator` in a way that makes `discriminator` more likely to predict "real" when looking at fake images
4. It's very important to note that you set the `discriminator` to be frozen during training: its weights won't be updated when training `gan`
5. If the `discriminator` weights could be updated during this process, then you'd be training the `discriminator` to always predict "real," which isn't what you want!

How to train your DCGAN

For each epoch, you do the following

1. Draw random points in the latent space (random noise)
2. Generate images with `generator` using this random noise.
3. Mix the generated images with real ones
4. Train `discriminator` using these mixed images, with corresponding targets: either "real" (for the real images) or "fake" (for the generated images)
5. Draw new random points in the latent space
6. Train `gan` using these random vectors, with targets that all say "these are real images." This updates the weights of the `generator` (only, because the `discriminator` is frozen inside `gan`) to move them toward getting the `discriminator` to predict "these are real images" for generated images: this trains the `generator` to fool the `discriminator`

GANs Conclusion

1. A GAN consists of a generator network coupled with a discriminator network
2. GANs are difficult to train, because training a GAN is a dynamic process rather than a simple gradient descent process with a fixed loss landscape
3. GANs can potentially produce highly realistic images. But unlike VAEs, the latent space they learn doesn't have a neat continuous structure and thus may not be suited for certain practical applications,