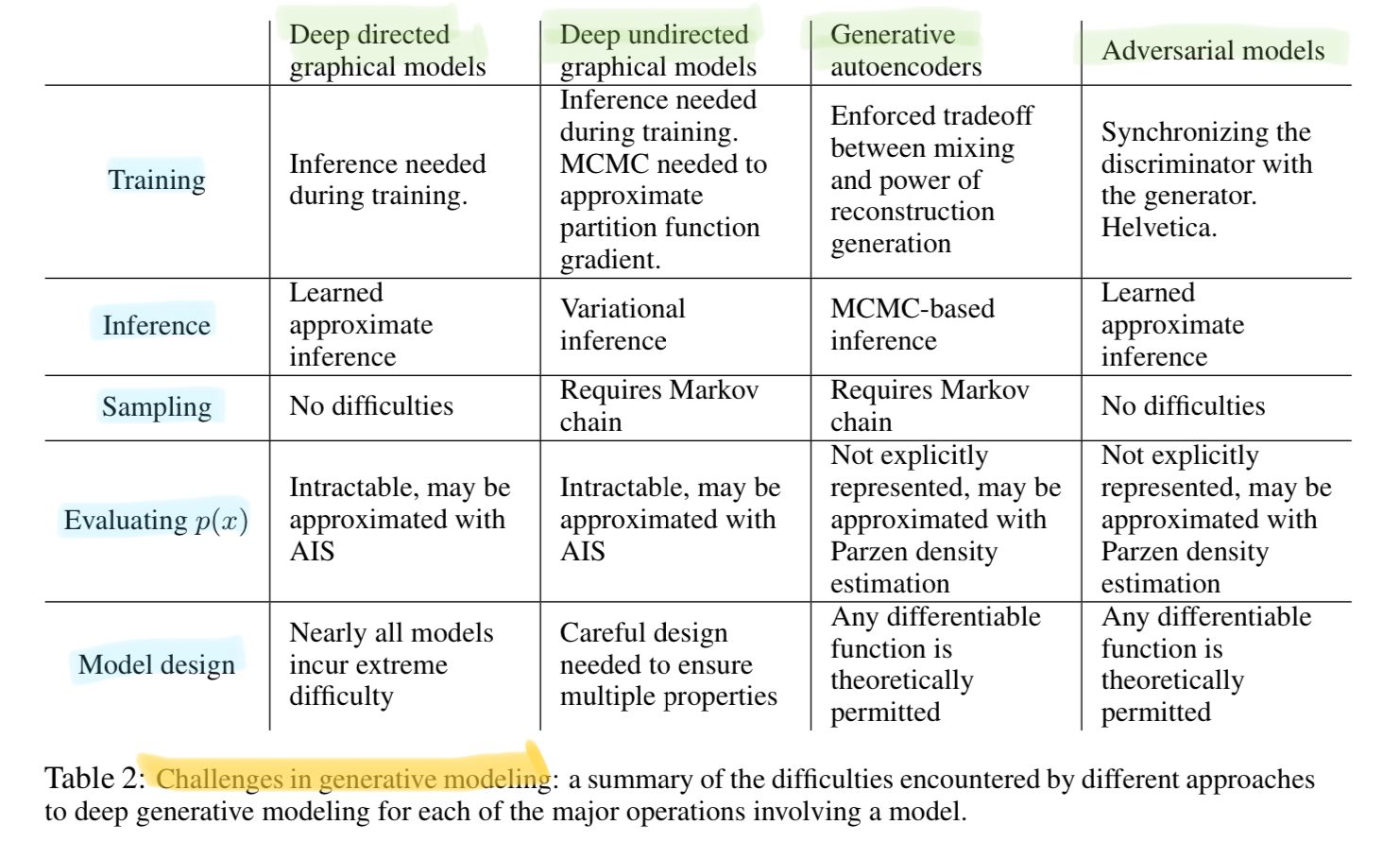
TFG Research

Generative Adversarial Networks (GAN)

* Generative model G, generates data imitating a sample.
* Discriminative model D, estimates the probability that a sample came from the training data rather than G.
* The idea is to create a minimal two-player, zero sum game where G creates a sample and D tries to distinguish them from the set.
* The models teach each other until reaching a point where the samples from G are virtually indistinguishable from the real ones.
  + This can be thought as a game where G are criminals trying to create falsified bills and D are the cops trying to catch them. The hopeful end is to get G to get away with the false cash without being caught.
* **Adversarial Nets:**
  + The adversarial modelling framework is most straightforward to apply when the models are both multilayer perceptrons.
  + Where:
    - V(G,D) is the value function of the minimal game.
    - Pdata(x) is the probability of a sample belonging to data x.
    - [log D(x)] is the result that model D is trained to maximise. Hence the max D.
    - Pz(z) is the probability of input noise variables.
    - [log(1-D(G(z)))] is the result that model G is trained to minimise. Hence the min G.
  + The ideal stopping point is when D(x) = 1/2, meaning it can no longer differentiate. This is because Pg = Pdata and as such, neither model can improve any longer.
  + Training: to avoid overfitting, we alternate between k steps of optimising D and one step of optimising G. This results in D being maintained near optimal solution, so long as G changes slowly enough.
    - Early in training, rather than making G minimise *log(1-D(G(z))),* we make it minimise *logD(G(z))* instead. This is done because the previous gradient saturates since G generates poor data and D is able to reject them with high confidence. This way results in the same dynamics but with a much stronger gradient early in training.
    - The gradient-based updates can use any standard gradient-based learning rule.
    - The optimal discriminator D for any given generator G is:
    - The global minimum of the virtual training criterion C(G) is achieved if and only if Pg = Pdata. At that point, C(G) = - log 4.
    - **If G and D have enough capacity, and at each step of the algorithm, the discriminator is allowed to reach its optimum given G, and Pg is updated so as to improve the criterion, then Pg converges to Pdata.**



* + - Improved techniques for training GANs: these techniques are intended to encourage convergence when used to seek the Nash equilibrium. They are motivated by a heuristic understanding of the non-convergence problem. They lead to **improved** **semi-supervised** learning performance and **improved** **sample generation**.
      * *Feature matching*: we train the generator to match the expected value of the features on an intermediate layer of the discriminator.
      * *Mini batch discrimination*: the discriminator model looks at multiple examples in combination, rather than in isolation.
      * *Historical averaging*: we modify each player’s cost to include the historical average of the parameters, which can be updated in an online fashion. The last part means that this learning rule scales well to long time series. This technique is appropriate for low-dimensional, continuous non-convex games.
      * *One-sided label smoothing*: for the discriminator, we smooth the positive labels to an alpha value, but leaving negative labels set to 0. If we also smooth the negative labels then if Pdata is approximately zero and Pmodel is large, erroneous samples from Pmodel have no incentive to move nearer to the data.
      * *Virtual batch normalisation*: each example x is normalised based on the statistics collected on a **reference batch** of examples that are chosen once and fixed at the start of training, and on x itself. The reference batch is normalised using only its own statistics. VBN **is computationally expensive**.
    - **Advantages:**
    - Markov chains are not needed.
    - Only backdrop is used to obtain gradients.
    - No interference is needed during learning.
    - A wide variety of functions can be incorporated into the model.
    - Can represent very sharp, even degenerate distributions.
    - Components of the input are not copied directly into de generator’s parameters.
  + **Disadvantages:**
    - There is no explicit representation of *Pg(x).*
    - D and G must be well synchronised during training in a way where G must not be trained too much without updating D.

Conditional GAN (CGAN)

We can create a CGAN by feeding a condition *y* into both the generator and discriminator. This model can be used to learn a multi-modal model and generate descriptive tags which are not part of the training labels.

* Spatial Bilinear Pooling (SBP): provides multiplicative interaction between all elements of two vectors. When the dimension of an image is (*n* x *n* x *d),* the SBP performs cross product for each pixel (1 x 1 x *d*) of the image with *c* and then gathers the resulting vectors spatially to make a new image. Where *c* is a condition applied to the GAN. This model provides good and stable solutions that can fit conditional distributions *p(x|c)*.

Variational AutoEncoders (VAE)

VAEs are powerful **generative** models. They can alter or explore variations on existing data, and not just in a random way, but also in a desired, specific direction.

An auto-encoder network is made up of an encoder network and a decoder network. The encoder takes the input and converts it into a smaller, dense representation, which the decoder can use to convert it to the original input.

The output of the encoder is what is used to process into the desired output. AutoEncoders make the encoder generate encodings specifically useful for *reconstructing its own input*. The entire network is usually trained as a whole. The loss function is usually either the mean-squared error or cross-entropy between the output and the input, know as the *reconstruction loss*, which penalises the network for creating outputs different from the input.

Since the encoding has less units than the input, the encoder has to learn to keep as much of their relevant information as possible in the limited encoding, and intelligently discard the irrelevant parts. The decoder learns to take the encoding and properly reconstruct it into a full image.

The fundamental **problem** with normal AutoEncoders, for generation, is that the space where the encoded vectors lie may not be continuous or allow easy interpolation. If the space has discontinuities and you sample/generate a variation from there, the decoder will simply generate an unrealistic output, because the decoder has no idea how to deal with that region of the graph.

VAEs have one fundamental unique property that separates it from regular AutoEncoders, and it is this property that makes them so useful for generative modelling: their latent spaces are, *by design*, continuous, allowing easy random sampling and interpolation.

It achieves this by making its encoder not output encoding of vector size *n*, but rather, outputting *two* vectors of size *n*: a vector of means (nu), and another vector of standard deviations (sigma).

They form the parameters of a vector of random variables of length n, with the *i*th element of (nu) and (sigma) being the mean and standard deviation of the *i*th random variable, Xi, from which we sample, to obtain the sampled encoding which we pass to the decoder. This stochastic generation means that even for the same input, the encoding will vary on every single pass due to sampling, even if (nu) and (sigma) remain the same.

The mean vector (nu) controls where the encoding of an input should be cantered around, while the standard deviation (sigma) controls the “area”, how much from the mean the encoding can vary.

To allow smooth interpolation, we introduce the Kullback-Leibler divergence (KL divergence) into the loss function.

Minimising the KL divergence means optimising the probability distribution parameters (nu, sigma) to closely resemble that of the target distribution.

Then finally we use vector arithmetic to finalise the smooth interpolation. For example, if you wish to generate a new sample halfway between two samples, just find the difference between their mean vectors and add half the difference to the original and simply decode it.

To generate *specific features*, like generating glasses on a face, find two samples, one with glasses and one without, obtain their encoded vectors from the encoder and save the difference. Add this new “glasses” vector to any other face image and decode it.

TecoGAN (Temporally Coherent GAN)