Variational auto-encoder

- VAEs are powerful generative models, applications as diverse as from generating fake human faces, to producing purely synthetic music.

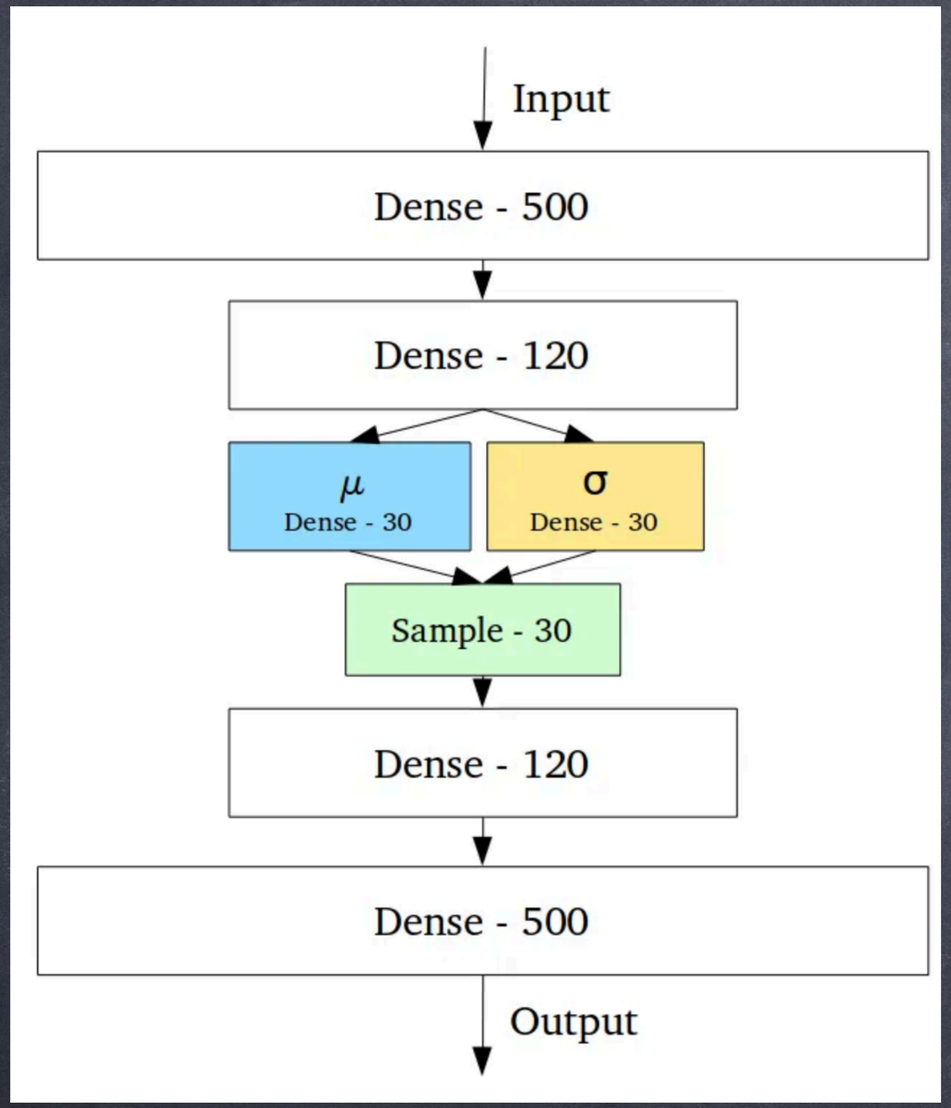
- How VAEs can be helpful: We want to generate a random, new output, that looks similar to the training data. In addition to it we also would like to alter or explore variations on data we already have and not just in a random way either, but in a desired specific direction.

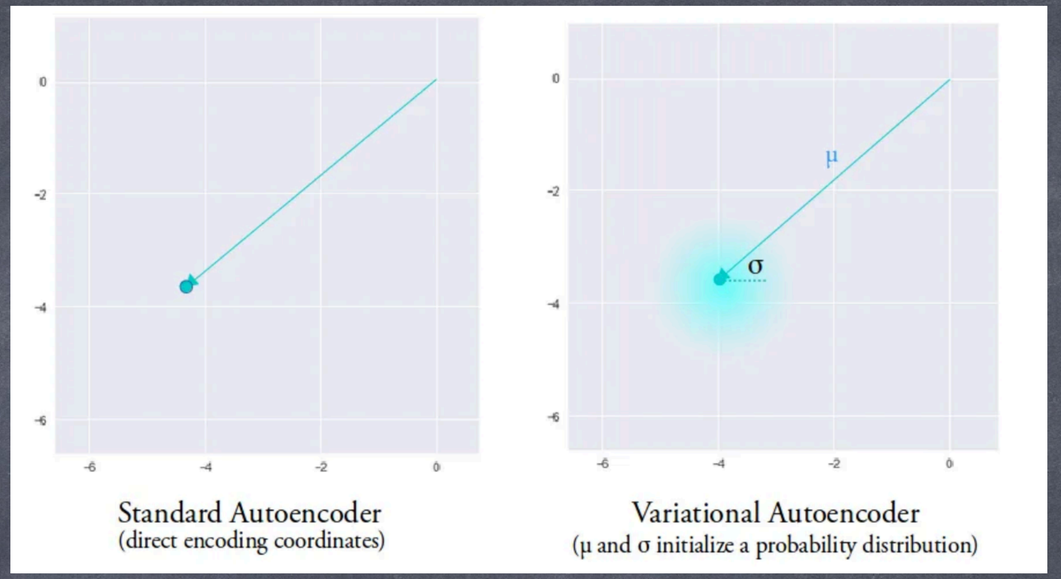
Limitations of Auto-encoders

- Auto-encoders learn to generate compact representations and reconstruct their inputs well.
- Applications - denoising auto-encoder.
- Auto-encoders as generative models - the latent space they convert their inputs to and where their encoded vectors lie, may not be continuous, or allow easy interpretation.
- Training an AE on MNIST dataset and visualising the encodings from a SD latent space reveal the formation of distinct clusters. This makes sense, as distinct encodings for each image type makes it far easier for the decoder to decode them. So, can only be used to replicating images but not for generating new and related images.
- Because we want to randomly sample from the latent space, or generate variations on a input image, from a continuous latent space.
- But here the space is not continuous (gaps between clusters) and sampling a variation from there, decoder will simply generate an unrealistic output, as the decoder has no idea how to deal with that region of the latent space. During training, it never saw encoded vectors coming from that region of latent space.

VAE

- Why VAEs are generative in nature: Their latent spaces are by design, continuous, allowing easy random sampling and interpolation.
- Encoder outputs two vectors of size n: a vector of means and a vector of standard deviations instead of only an encoding vector of size n.
- They form the parameters of a vector of random variables of length n, with the 4th element of means and standard deviation of the 4th random variable, Xi, from which we sample, to obtain the sampled encoding which we pass onward to the decoder.
- This stochastic generation means, that even for the same input, while the mean and standard deviations remain the same, the actual encoding will somewhat vary on every single pass simply due to sampling.





Output
μ [0.1, 1.2, 0.2, 0.8,...]

Output
σ [0.2, 0.5, 0.8, 1.3,...]

Intermediate
X $[X_1 \sim N(0.1, 0.2^2), X_2 \sim N(1.2, 0.5^2), X_3 \sim N(0.2, 0.8^2), X_4 \sim N(0.8, 1.3^2), \ldots]$

sample

Sampled vector [0.28, 1.65, 0.92, 1.98,...]

- The mean vector controls where the encoding of an input should be entered around, while standard deviation controls the ‘area’, how much from the mean the encoding can vary.

- As encodings are generated at random from anywhere inside the ‘circle’ (distribution), the decoder learns that not only single point in latent space referring to a sample of that class, but all nearby points refer to the same as well.

- This allows the decoder to not just decode single, specific encodings in the latent space (leaving the decodable latent space discontinuous), but ones that slightly vary too, as the decoder is exposed to a range of variations of the encoding of the sae input during training.

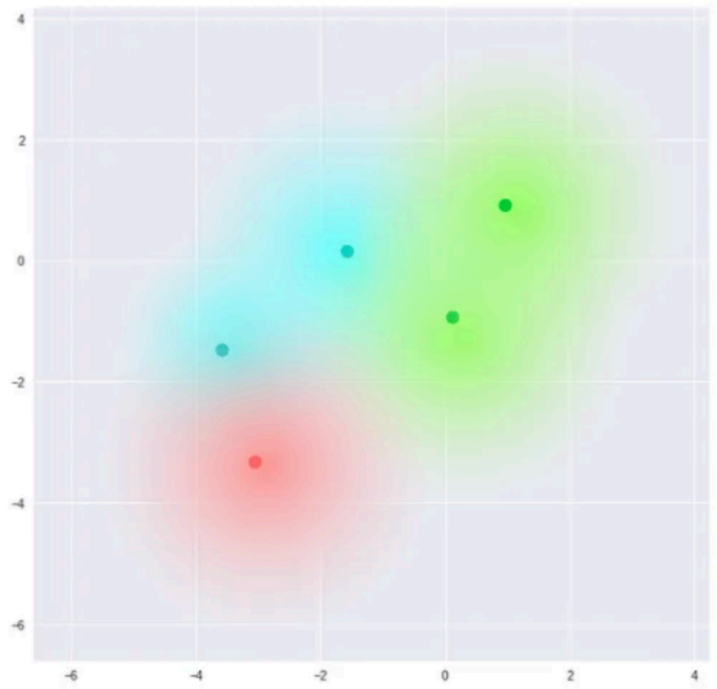
- Call flatten layer as hidden
- Mean: Pass hidden to dense with neutrons = latent_size = 5
- log_standard_deviation: pass the hidden to dense layer with neurons = latent_size = 5
- The reason of using log of standard deviation instead of only standard deviation is the same as using of softmax instead of directly outputting numbers in fixed range [0,1], the network can output a wider range of numbers can later compress down.
- Pass mean and log_standard_devuation to function sampler.
- Sampler:
  - std_norm: Sample from the standard normal a matrix of batch_size * latent_size (taking into account mini batches). Sample from random_normal(with mean = 0 and std = 1) of shape = mean.shape[0] * latent_size.
  - latent_vector: mean + exp(log_standard_devuation) * std_norm
- Pass latent_vector to decoder.

- The model is now exposed to a certain degree of local variation by varying the encoding of one sample, resulting in smooth latent spaces on a local scale, i.e., for similar samples.

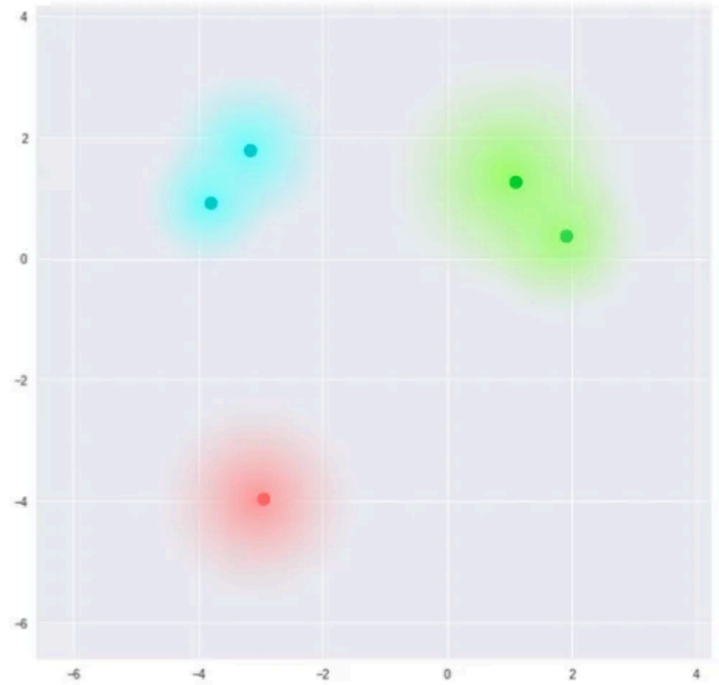
- Ideally: we want to overlap between samples that are not very similar too, in order to interpolate between classes.

- Since there are no limits on what values vectors mu and std can take on, the encoder can learn to generate very different mu for different classes, clustering them apart, and minimise std, making sure the encodings themselves don’t vary much for the same sample (less certainty for the decoder).

- This allows the decoder to efficiently reconstruct the training data.



What we require



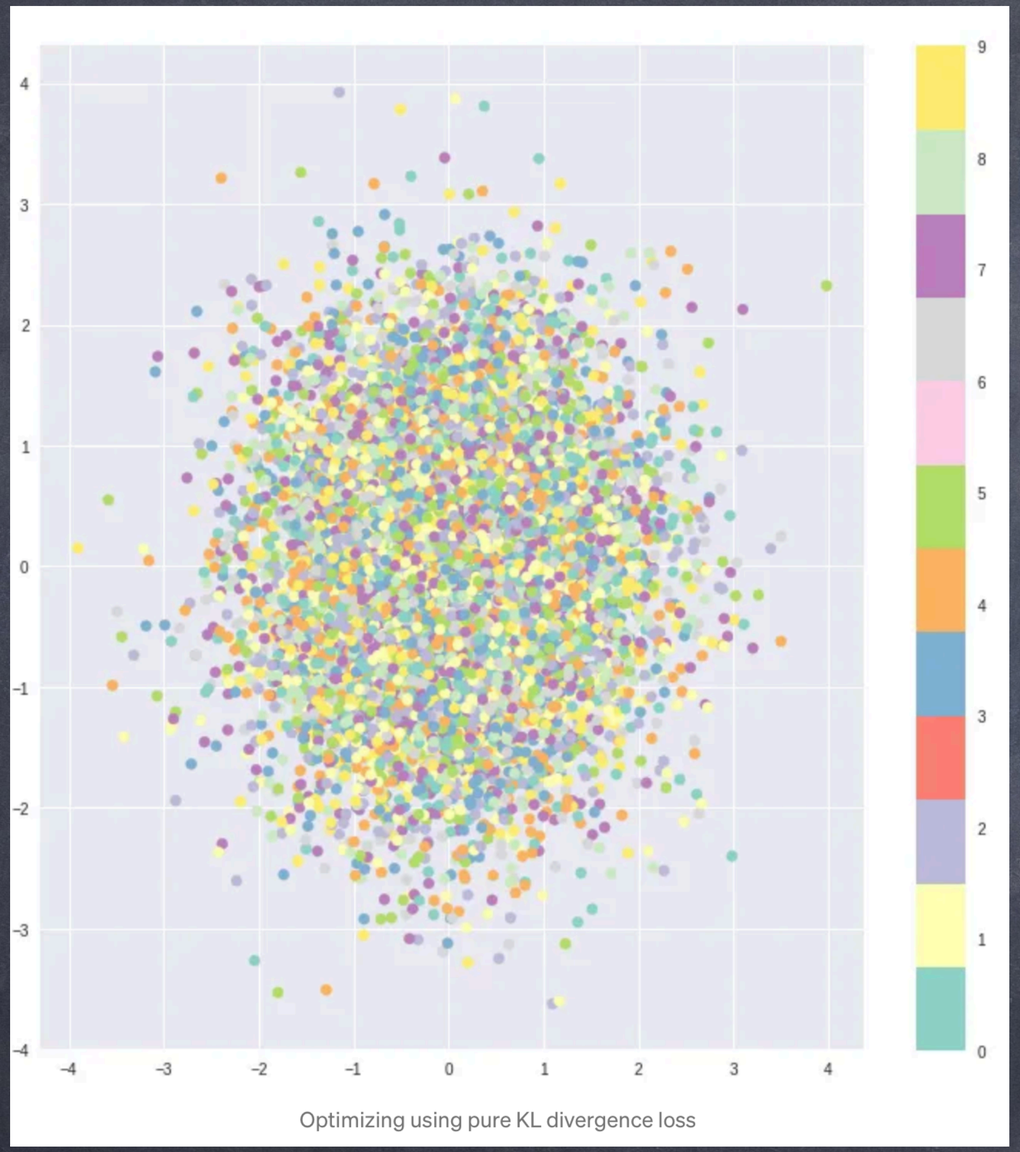
What we may inadvertently end up with

- Ideally: we want encodings as close as possible to each other while still being distinct, allowing smooth interpolation and enabling the construction of new samples.
- So, KL divergence is introduced into the loss function.
- KL divergence: between two probability distributions simply measures how much they diverge from each other. Minimising KL divergence here means optimising the probability distribution parameters (mu and sigma or std) to closely resemble that of the target distribution.
- $$\sum_{i=1}^{n} \sigma_i^2 + \mu_i^2 - \log(\sigma_i) - 1$$
- For VAEs, the KL loss is equivalent to the sum of all the Kl divergences between the component Xi sampled from random_normal in X and standard_normal. It's minimised when mum = 0 and sigmai = 1

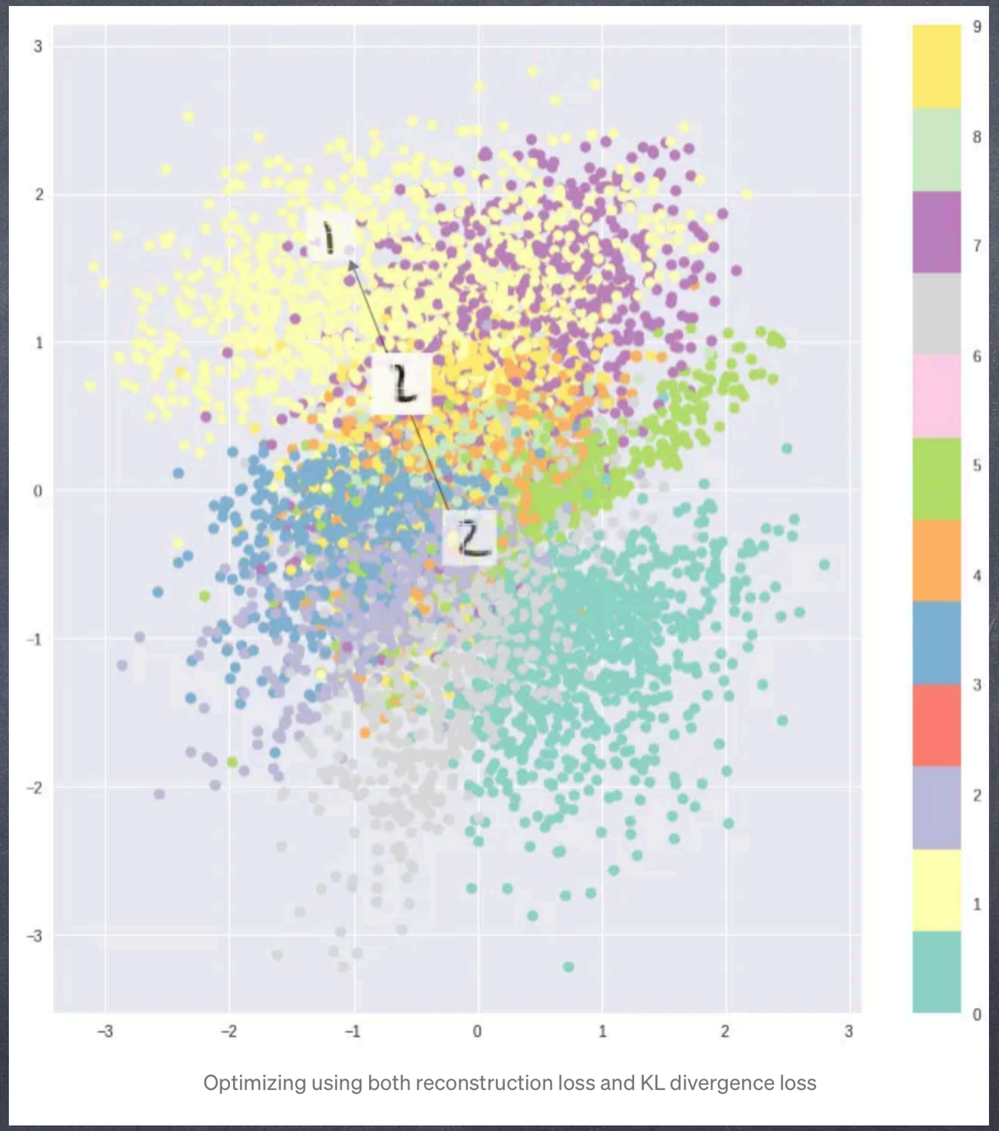
- This loss encourages the encoder to distribute all encodings (of all types of inputs, e.g., all MNIST numbers), evenly around the centre of the latent space. If it tries to cheat by clustering them apart into specific regions, away from the origin, it will be penalised.

- Using pure KL loss results in a latent space results in encodings densely placed randomly, near the centre of the latent space, with little regard for similarity among nearby encodings. The decoder finds it impossible to decode anything meaningful from this space, simply because there really isn’t any meaning.

- Optimising the two together, results in the generation of a latent space which maintains the similarity of marbly encodings on the local scale via clustering, yet globally, is very densely packed near the latent space origin.



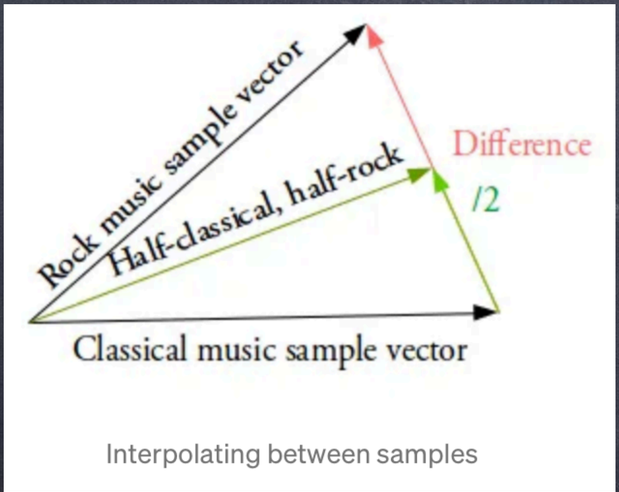
Optimizing using pure KL divergence loss



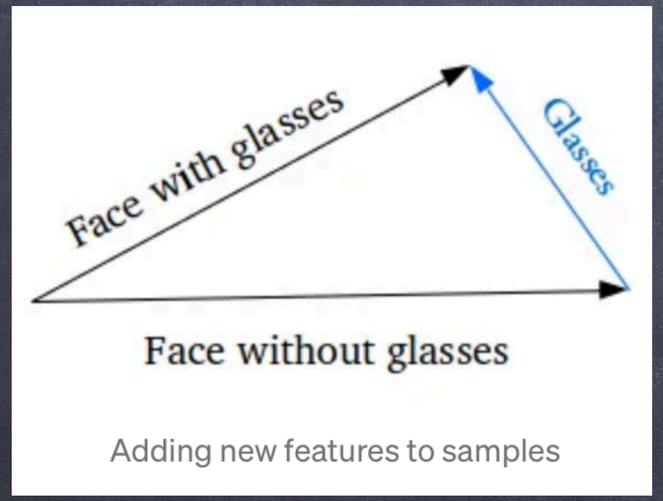
Optimizing using both reconstruction loss and KL divergence loss

- This is the equilibrium reached by the cluster-forming nature of the reconstruction loss, and the dense packing nature of the KL loss, forming distinct clusters the decoder can decode.
- When randomly generating, if sampled a vector from the same prior distribution of the encoded vectors, N(0,1), the decoder will successfully decode it.
- And if interpolating, there are no sudden gaps between clusters, but a smooth mis of features a decoder can understand.
- Loss:
  - mse_recosntuction_loss = sum_on_all_axes(square(output - input))
  - kl_loss = -0.5 * sum_at_last_axis(1 + log_standard_devation - square(mean) - square(exp(log_standard_deviation)))
  - total_loss = mean(mse_reconstruction_loss + kl_loss

Produce smooth interpolations

- Simple vector arithmetic in the latent space.
- If we 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 then decode it.
- 

Interpolating between samples

- Generating specific features, such as generating glasses on a face
- Find two samples, one with glasses, 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.
- 

Adding new features to samples

- Improvements that can be done over variational auto encoder.
- Replace standard fully connected dense encoder-deocder with a convolutional-deconvolutional encoder-decoder pair.
- Train auto-encoder using LSTM encoder-decoder pairs (using a modified version of the seq2seq architecture) for sequential, discrete data (something not possible with methods such as GANs), to produce synthetic text, or even interpolate between MIDI samples such as Google Brain’s Magenta’s MusicVAE
- VAEs work with remarkably diverse types of data, sequential or non-sequential, continuous or discrete, even labelled or completely. Unlabelled.

- VAEs have shown amazing results in generating many kinds of complicated data, inclusion handwriting digits, faces, house numbers, CIFAR images, physical models of scenes, segmentation and predicting the future from static images.
- Generative modelling deals with models of distribution P(X), defined over datapoints X in some potentially high-dimensional space X.
- Each image has thousands of dimensions/pixels and the generative model’s job is to somehow capture the dependencies between pixels, say, that nearby pixels have similar colour and are organised into objects.
- What it means to capture these dependencies depends on exactly what we want to do with the model.

- One way, to compute P(X) numerically. Images, X values which look like real images should get high probability, whereas images that look like random noise should get low probability. Models like this are not necessarily useful: knowing that one image is unlikely does not help us synthesise one that is likely.

- We want images similar to ones that are in database not exactly like those. Say, database of handwritten text and try to produce more handwritten text.

- Setup: Examples X distributed according to some unknown distribution Pgt(X), and our goal is to learn a model P which we can sample from, such that P is as similar as possible tp Pgt.

- Training this type of model has been a long-standing problem in the machine learning community.
- Most approaches have had one of three serious drawbacks.
  - They might require strong assumptions about the structure in the data.
  - They might make severe approximations, leading to sub-optimal models.
  - They might rely on computationally expensive inference procedures like Markov chain Monte Carlo.
- Recent works have made great progress using Neural nets.
- One such is VAE. The assumptions of this model are weak and training is fast via backpropagation. They do make an approximation but the error captured by this approximation is arguably small given high-capacity models.
- Hence a quick rise in their popularity.
- VAEs are based on variational bayesian methods and minimum description length coding models.

- https://towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf

- https://keras.io/examples/generative/vae/