Deep generative models

Overview

The purpose of generative models is creating/designing new concepts (Images, text, policies, objects, ideas, processes, policies,…) given historical (observed) data related to this concept under a constraint of zero-sum game (Nash equilibrium).

- *Training: Historical observed data on concept => Representation (Latent space)*
- *Prediction: Representation => new concept*

Concept/Target	Metric	Generator	Discriminator
Medical note	Accuracy x readability	Provider	Auditor
Medical claim	Revenue x acceptance	Provider	Insurance
ML pipeline	Latency x Failure rate	Data engineer	QA
Web UI	Usability	GUI designer	User
Prediction procedure	Accuracy	Medical staff	Medical staff
Dev process	Quality x time	Manager	Auditor

Generative models attempt to accomplish a task similar to model-based reinforcement learning using imitation or evolutionary techniques.

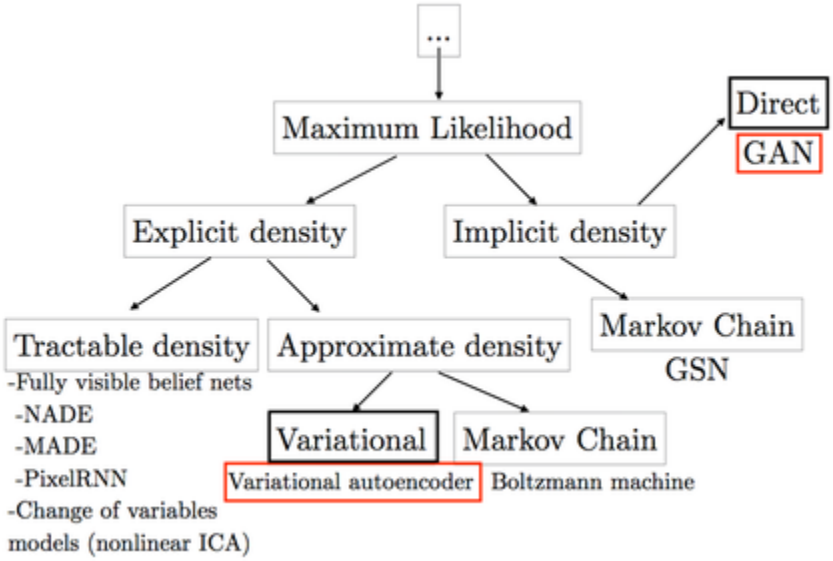
Concretely, a generative model takes a training set, consisting of samples drawn from a distribution $\mathbf{p}_{\mathbf{data}}$, and learns to represent an estimate of that distribution somehow. The result is a probability distribution $\mathbf{p}_{\mathbf{model}}$. The most commonly application of generative models is their ability to train and sample from generative models to extract or represent high-dimensional probability distributions.

However, there are many other use cases

- **Data set augmentation**: Case in which which the labels for many or even most training examples are missing. It is related to both semi-supervised learning and Transfer (multi-task) learning.
- **Deep reinforcement learning:** Generative models can be incorporated into reinforcement learning to support model-based algorithms.
- **Multi-modal outputs**: These models have the ability expressed output with multiple correct answers

Taxonomy

Generative models have been mainly developed around Markov chains that have dominate the landscape until mid to late 2010s



This page focuses on two variants of generative models:

- Variational auto-encoder (VAE)
- Generative adversarial network (GAN)

There are some key differences between Variational auto-encoders and generative adversarial network

	VAE	GAN
Objective	Latent space	Generator (decoder) model
Sample generation	Sample latent normal distribution and apply decoder	Generator process uniform random sampling of latent space
Support for model-based reinforcement learning	No	Yes

References:

- Tutorial Variational auto-encoder
- Tutorial Generative Adversarial Networks

Auto-encoders

Auto-encoder (AE)

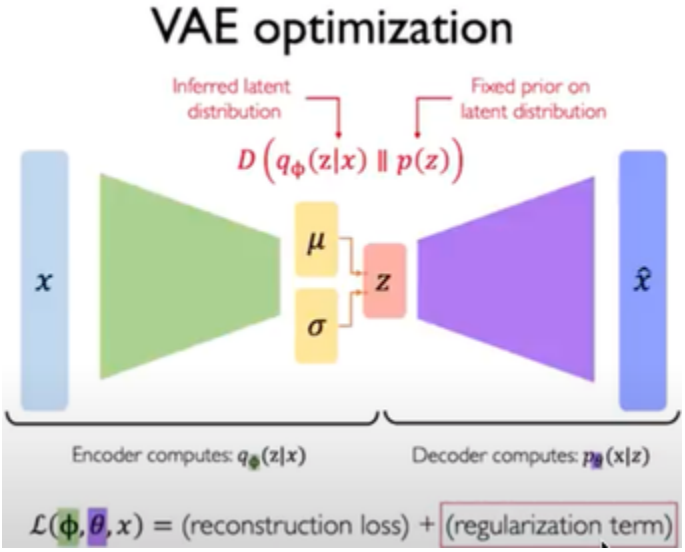
An autoencoder is a category of artificial neural network that learn an encoding or representation (latent space) in an unsupervised fashion. One purpose is to reduce the dimension of the original or observed input data. There are numerous variants of auto-encoder that constants the learned representations to assume useful properties.

Variational auto-encoder (VAE)

Overview

Variational auto-encoder (VAE) represents the latent space as a distribution, usually Gaussian for which the mean and standard deviation are learning in an unsupervised fashion. The basic concept is borrowed from Bayesian optimization.

VAE is also an alternative to Generative Adversarial Model.



The training flow is as follows

1. Encode observed data to generate latent, continuous variables
2. Constraint latent space to match a proposed distribution
3. Sample the distribution to generate latent samples
4. Decode the latent samples
5. Compute the reconstruction loss (ELBO)
6. Repeat 1 to 5 until reconstruction loss is small

Hyper parameters

- Regularization: Kullback-Leibler, Jensen-Shannon
- Distribution: Normal, Gumbel-Softmax, Log-normal
- Learning rate: 0.01 to 0.0001
- Optimizer: SGD, Adam
- Network layout: Number of hidden layer, number of node per layer, w/o bias

References

- Tutorial Variational auto-encoder

Convolutional Variational Auto-encoder (CVAE)

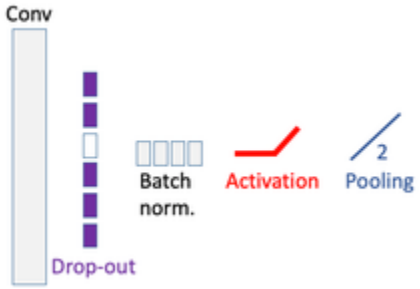
Overview

A convolutional variational Auto-encoder has 3 types of component

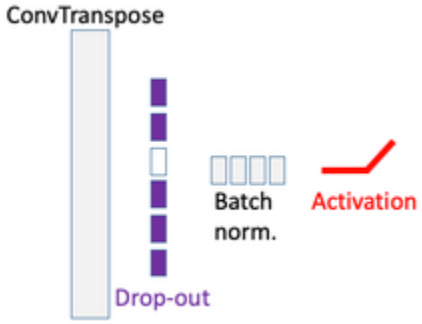
- Convolutional neural blocks (Encoder)
- Variational neural block (latent space collection and sampling)

- De-convolution neural blocks (Decoder)

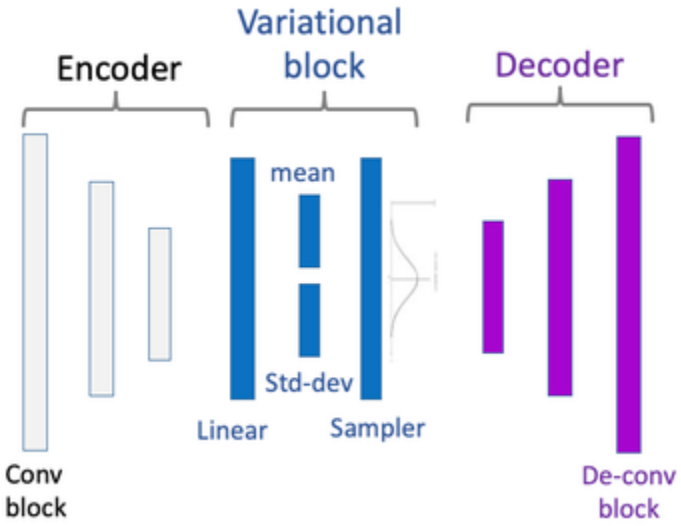
Convolutional block



De-convolutional block



Architecture



Estimation of output size of CNN blocks

For convolutional block without pooling, the size of the output $\mathbf{n_{out}}$ from a convolutional block is computed from the size of an input $\mathbf{n_{in}}$ given a stride, kernel and padding.

$$n_{out} = \left\lfloor \frac{n_{in} + 2p - k}{s} \right\rfloor + 1$$

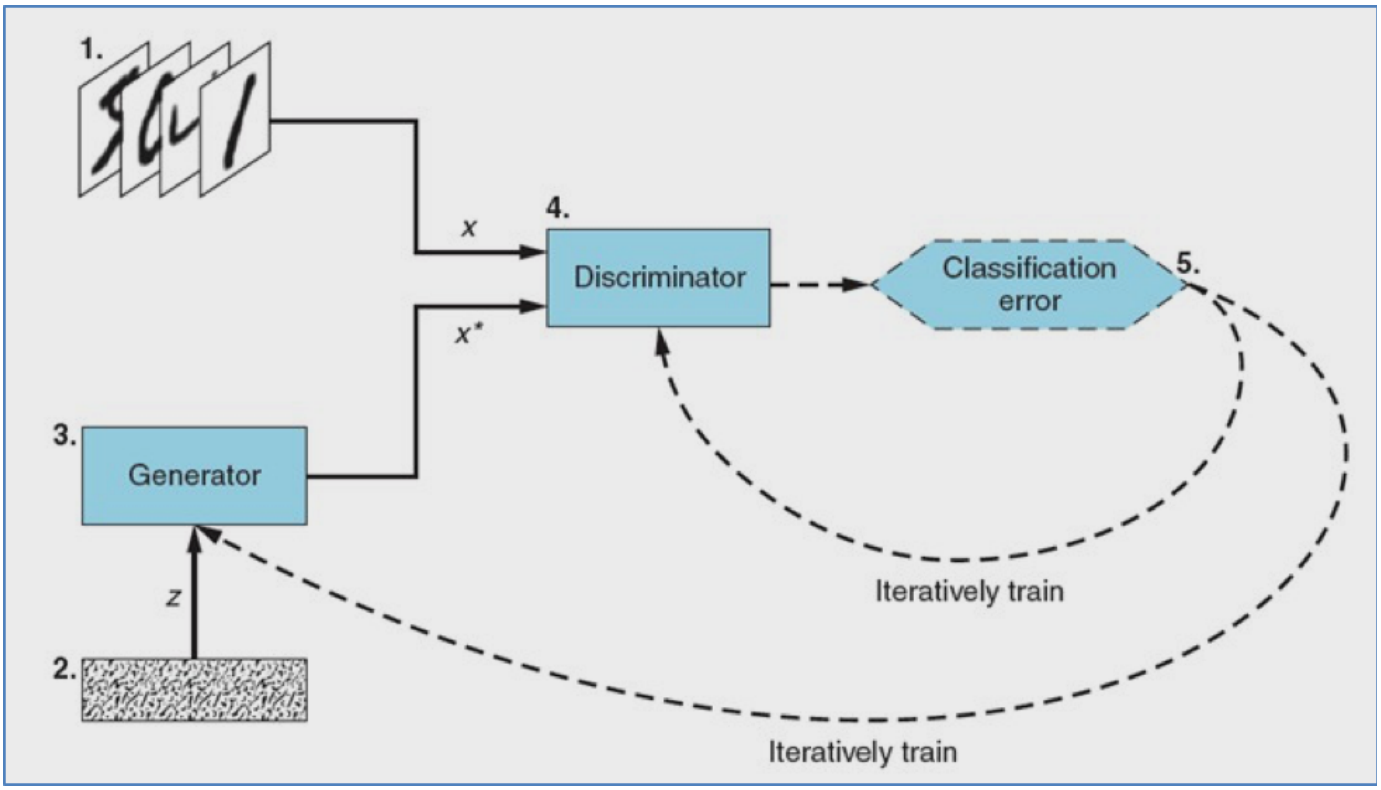
n_{in}: number of input features
n_{out}: number of output features
k: convolution kernel size
p: convolution padding size
s: convolution stride size

Similarly, the size of the output of a de-convolutional block is computed as

$$n_{out} = s[n_{in} - 1) - 2p + k$$

Generative Adversarial Models

Generative Adversarial Network (GAN)



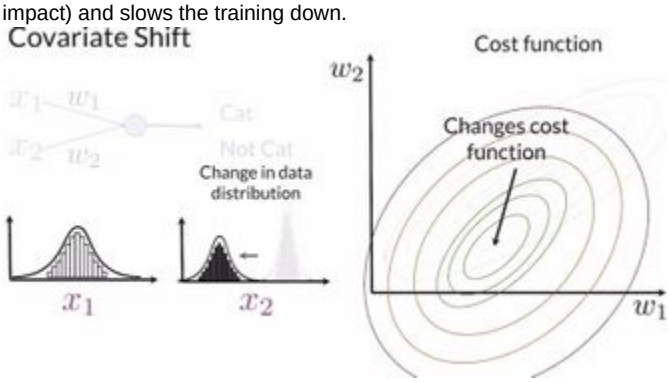
Loss function

Given a real value **x**, a generator **G** that predict latent space (**z**) **G(z)** and a discriminator (i.e. Binary classifier) **D(x, G(z))** probability, the loss function (minimal) is computed using the Binary Cross-Entropy formula:

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_z(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))].$$

Covariate shift

A **covariate shift** occurs when features have different distribution (mean and standard deviation). The difference in distribution skews the cost function (as some features have more impact) and slows the training down.



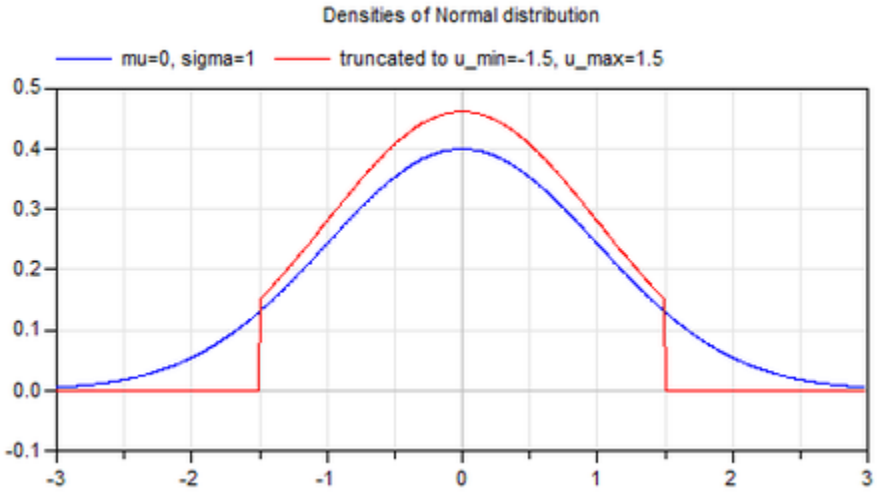
Batch normalization reduces all distributions as normal mean = 0 and standard deviation = 1. It smooths the cost function, speed up training. It applies to all hidden layers.

Mode collapse

Modes are maximum probability in the distribution of features, which correspond to classes. For instance, digit classification has 10 modes, one mode per digit. Model collapse happens when the generator gets stuck in few less modes or even one mode

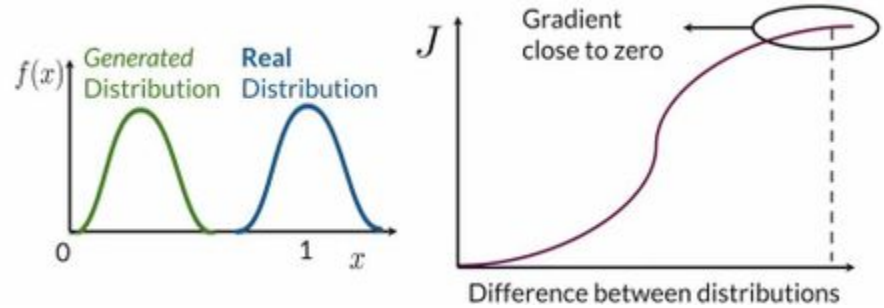
GAN Truncation trick

What the truncation trick does is resamples the noise vector until it falls within some bounds of the normal distribution. In fact, it samples from a truncated normal distribution where the tails are cut off at different values (red line in graph is truncated normal, blue is original). You can tune these values and thus tune fidelity/diversity.



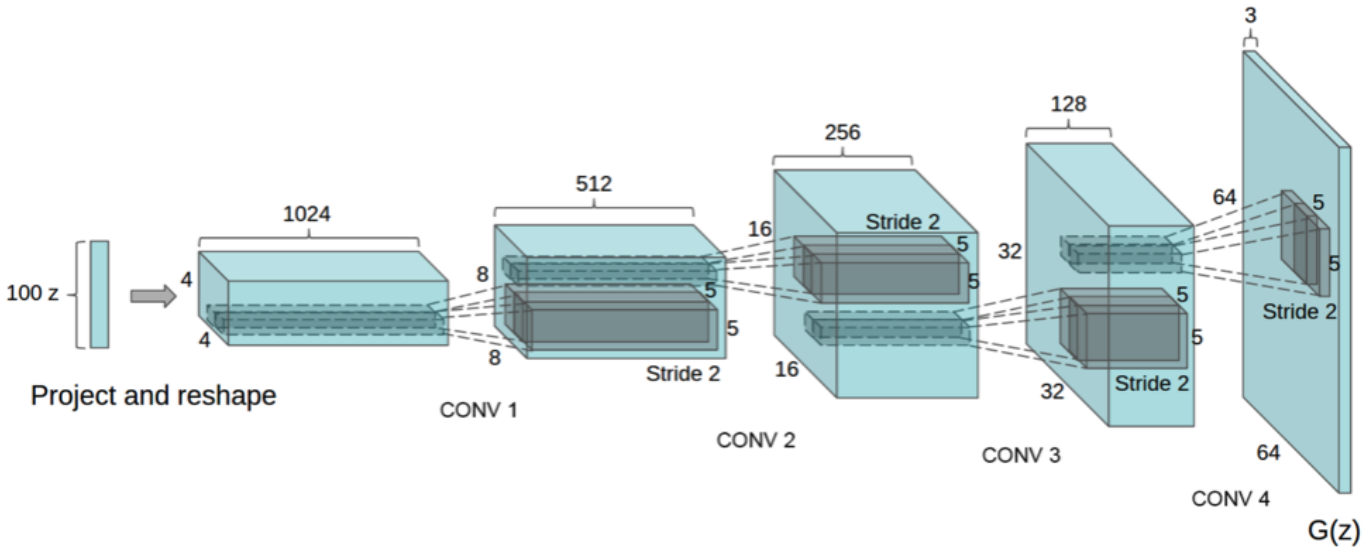
BCE loss saturation

The discriminator has a lot easier task than the generator as it just need to apply a binary classification. The generator must produce complex output. At the beginning of the training, the discriminator has no problem in distinguishing between the generator and real distribution. But at later stages, the **BCE loss saturate (vanishing gradients) and the discriminator cannot provide the generator with useful feedback.**



Deep-convolutional GAN (DCGAN)

Overview



Stride and padding

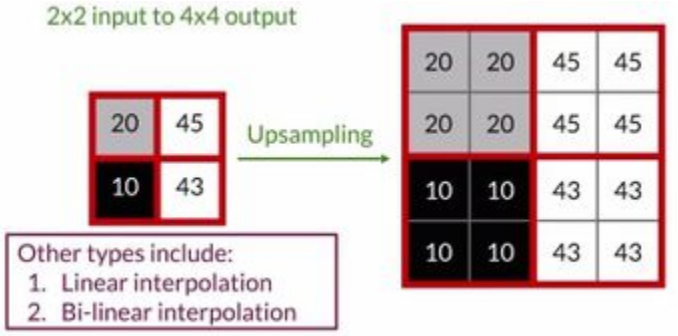
The larger stride smooths the key features of images or data sets. Padding (> 0) reduces overlap when applying kernel to an existing image or dataset and give more emphasis to edge features or outliers.

Pooling

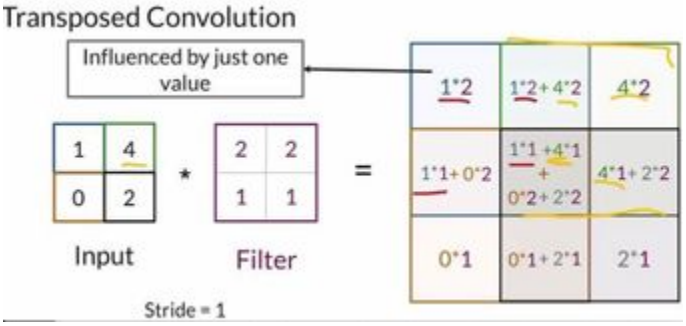
The purpose is to reduce the dimension of the problem (i.e. blurring an image while keep the most salient features). It reduces the computation load. It is more effective when there is no overlap. These are not learned parameters but just a rule/formula.

Deconvolution

This operation is opposite of Pooling. It interpolates the data or pixel. Like pooling it is a rule or formula not learned. Up-sampling the nearest neighbors. Deconvolution or transposed convolution applied a 'learned' up-sampling to expand the size the data (number of features or pixels). It is however affected by overlap when the filter is applied to the input data (check-board effect). This is because with DCGAN, you use convolutions which don't depend on the number of pixels on an image. However, the number of channels is important to determine the size of the filters.



Max pooling (MaxPool1d, MaxPool2d) then Batch normalization (BatchNorm1d, BatchNorm2d) are used after ConvTranspose2d. The last layer may or may not be fully connected (Linear) and usually do not have a Batch normalization.



Contrary to convolutional neural network used as supervised learning technique, Deconvolutional neural networks learn unsupervised. De-convolutional layers are transposed convolutional layer.

$$\boldsymbol{y}_i^{L-1} = \sum_{i=1}^{m_i^L} \boldsymbol{k}_{ij}^L * \boldsymbol{y}_i^L$$

The convolution layer **L** with a kernel (filer) **k**, a feature map $\mathbf{m_{L-1}}$ produces a features map $\mathbf{y_{L-1}}$ defined as

$$\boldsymbol{y}_i^L = \boldsymbol{b}_i^L + \sum_{j=1}^{m_i^{L-1}} \boldsymbol{k}_{ij}^L * \boldsymbol{y}_i^{L-1}$$

Evaluation of representation learning

One common technique for evaluating the quality of unsupervised representation learning algorithms is to apply them as a feature extractor on supervised datasets and evaluate the performance of linear models fitted on top of these features.

We compare the result of the linear classifier after a DCGAN and a simple K-Means. The classifier using features extracted from DCGAN should outperform the same classifier using the features extracted from K-mean

Best practices

Best practices for designing a Deep Convolutional Generative Adversarial Network

- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
- Use batchnorm in both the generator and the discriminator.
- Remove fully connected hidden layers for deeper architectures.
- Use ReLU activation in generator for all layers except for the output, which uses Tanh.
- Use LeakyReLU activation in the discriminator for all layers.

Wasserstein GAN with Gradient penalty (WGAN-GP)

Earth mover's distance

It is a function of the distance between the generated and real distribution and the amount to be moved. It does not have flat region (at 0 and 1) when the distributions are very different.

The problem with BCE loss is that as a discriminator improves, it would start giving more extreme values between zero and one, values closer to one and closer to zero.

As a result, this became less helpful feedback back to the generator. The **generator** would **stop learning** due to **vanishing gradient** problems. With Earth mover's distance, however, there's no such ceiling to the zero and one: The cost function continues to grow regardless of how far apart these distributions are. The gradient of this measure won't approach zero and as a result, GANs are less prone to vanishing gradient problems and from vanishing gradient problems, mode collapse.

W-loss and 1-Lipschitz continuity

Gradient penalty regularization is easier to implement than weights clipping to enforce 1-Lipschitz continuity because clipping the weights requires a lot of tuning.

$$\min_G \max_C \left\{ \mathbb{E}[c(x)] - \mathbb{E}[c(g(z)] + \lambda \left(\mathbb{E}\left[\left\| \nabla c\left(\epsilon + (1-\epsilon) g(z)\right) \right\|_2 - 1 \right] \right)^2 \right\}$$

The regularization is applied to the weighted sum of the real sample and the fake sample predicted by the generator. The W-Loss is computed as the difference between the expected critic of the real labels and the expected critic value of the fake label predicted by the *generator + regularization term*

WGAN best practices

- **Sample distribution**: Define p(x~) sampling uniformly between pairs of points sampled from the data distribution p_R and generator distribution p_g
- **Penalty coefficient**: lambda ~ 10
- **Critic batch normalization**: None
- **Two-side penalty**: The 1-L continuity forces the norm of the gradient toward 1 (two-side) instead of < 1 (one-side)

References

- From GAN to WGAN
- Towards Generalized Implementation of Wasserstein Distance in GANs

Spectral-normalized GAN (SN-GAN)

SN-GAN normalized the weights of the discriminator using **their spectral norm to enforce 1-L continuity constraint.** The spectral norm of a matrix W is the matrix's largest singular value (SVD), sigma(W). SVD is a generalization of the Eigen-decomposition.

The SVD decomposition is approximated by computing U and V through power iteration starting with a random initialization. Spectral normalization prevents the transformation of each layer from becoming too sensitive in one direction and avoid exploding gradients. It only applies to the discriminator. The only difference between a SN-GAN and traditional GAN is the discriminator, in the last output layer.

References

- Spectral Normalization for Generative Adversarial Networks
- Why Spectral Normalization Stabilizes GANs: Analysis and Improvements

Gumbel-Softmax GAN (GS-GAN)

Gumbel-Softmax distribution

The purpose of the Gumbel-Softmax is to replace non-differentiable categorical samples with differentiable approximation during training, - Gumbel-Softmax estimator.

The Gumbel-Max trick is defined as drawing samples z from a categorical distribution with probabilities and Gumbel distribution. It is approximated by the Gumbel-Softmax distribution.

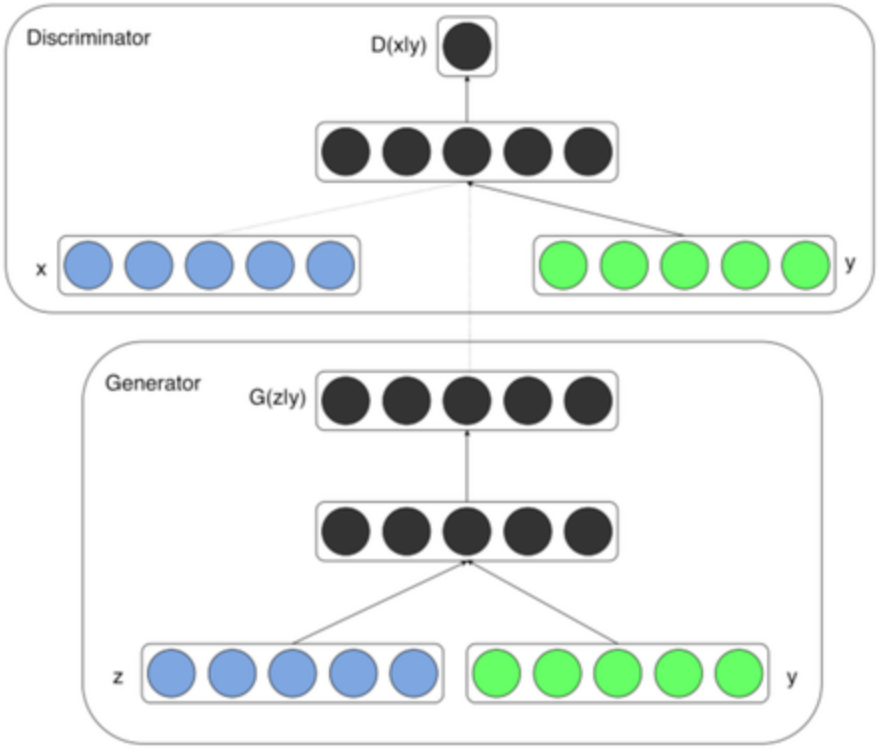
Practically, the Gumbel-Softmax can be sampled using inverse transform sampling with

u ~ U(0,1) and g = -log(-log(u))

Conditional GAN (CGAN)

Generative adversarial nets can be extended to a conditional model if both the generator and discriminator are conditioned on some extra information (label, context,…) **y**. The prior input noise $\mathbf{p_z(z)}$, and **y** are combined in joint hidden representation. The minimax expression becomes:

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x}|\boldsymbol{y})] + \mathbb{E}_{\boldsymbol{z}\sim p_z(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z}|\boldsymbol{y})))].$$



The main difference between a conditional and an unconditional GAN is that a conditional GAN is supervised and requires labels (class). Unconditional GAN do not require labels

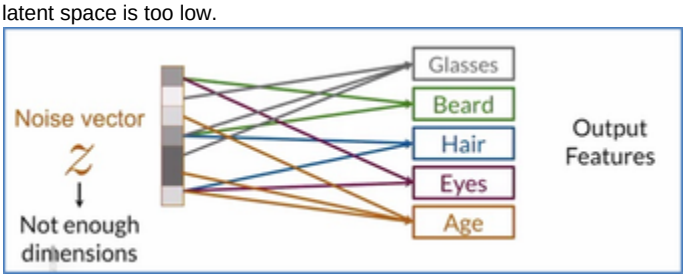
Conditional	Unconditional
Examples from the classes you want	Examples from *random classes*
Training dataset needs to be labeled	Training dataset *doesn't need to be labeled*

Reference

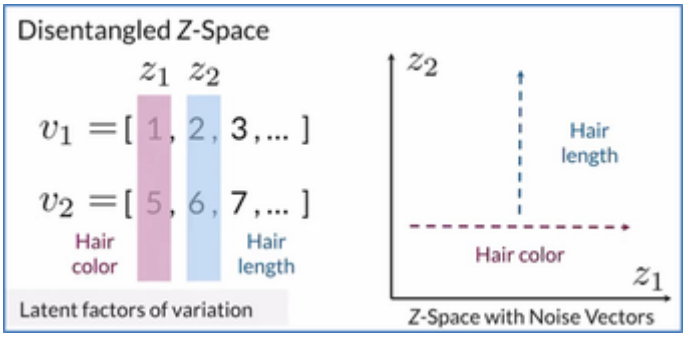
Conditional Generative Adversarial Networks

Controllable Generative Adversarial Networks (Ctrl-GAN)

Although conditional GAN allows you to guide the generation of samples by specifying a class (or target label) it does not allow to extract with features can be manipulate to modify the generated output. It is not possible to fully control which features to select or focus on. It is usually caused because of dimension of the latent space is too low.



Controlling the GAN means generating an interpolation in the Z-space (latent space or representation) between two vectors and estimate the direction of the change associated with a given feature. Attempt to control on feature in the latent space affects other features because of the compressed, low dimension space.



Classifiers can be used to find directions in the Z-space for a given features. The binary classifier should be able to accurately detect the given feature. Only the noise vector, output of the generator has updated but the generator weights are not affected. The ‘un-entanglement’ is performed on the trained generator.

There is a significant difference between controllable generation and conditional GAN

Controllable	Conditional
Examples with the **features that you want**	Examples from *the classes you want*
Training dataset **doesn't need to be labeled**	Training dataset *needs to be labeled*
Manipulate the *z* vector input	*Append a class vector* to the input

Adversarial Variational Auto-encoder (AVAE)

References

Adversarial Variational Auto-encoder

Last update 05.07.2021