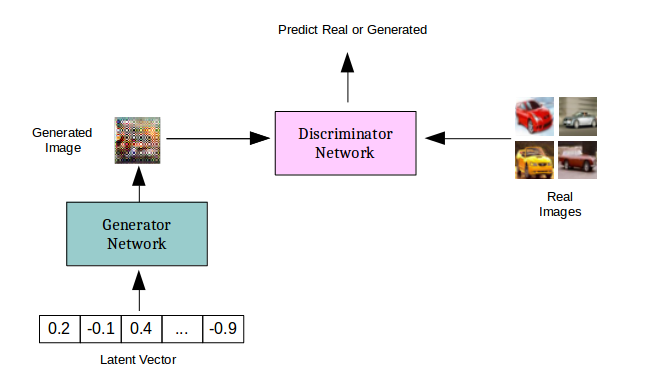
**GANs(Generative Adverserial Networks)**

**Part-1**

1. **Building a Basic GAN**
2. **Implementing a DCGAN**(with convolutional layers)
3. **WGAN(wasserstein GAN)-** overcome the shortcomings of traditional BCE loss function
4. **Controllable and conditional generation:** By modifying the noise vectors fed to the generator

* Discriminative models are used in classification problems. They determine a category Y given features X( **P(Y|X)).**Generative models on the other hand take in **noise(zeta)** and a **class Y** as inputs and comes up with features X(**P(X|Y)**
* The noise is important to make sure we don’t generate the same image/feature everytime.
* The most widely used generative models are variational autoencoders(VAE) and GANs.
* The GANs are comprised of generator and discriminator. The generator takes in the noise signal as input and generates images whereas the discriminator tries to distinguish between the real and generated images. These two factions compete against each other and learn from each other. Ultimately there’ll be a stage when the discriminator is no longer needed to generate images .
* We feed the true images to the discriminator and train it to distinguish the real ones from the fake ones(generated ones). Based on this, it assigns scores to the outputs produced by the generator. In simple words, **the generator’s goal is to fool the discriminator whereas the discriminator’s role is to distinguish between real and fake.**

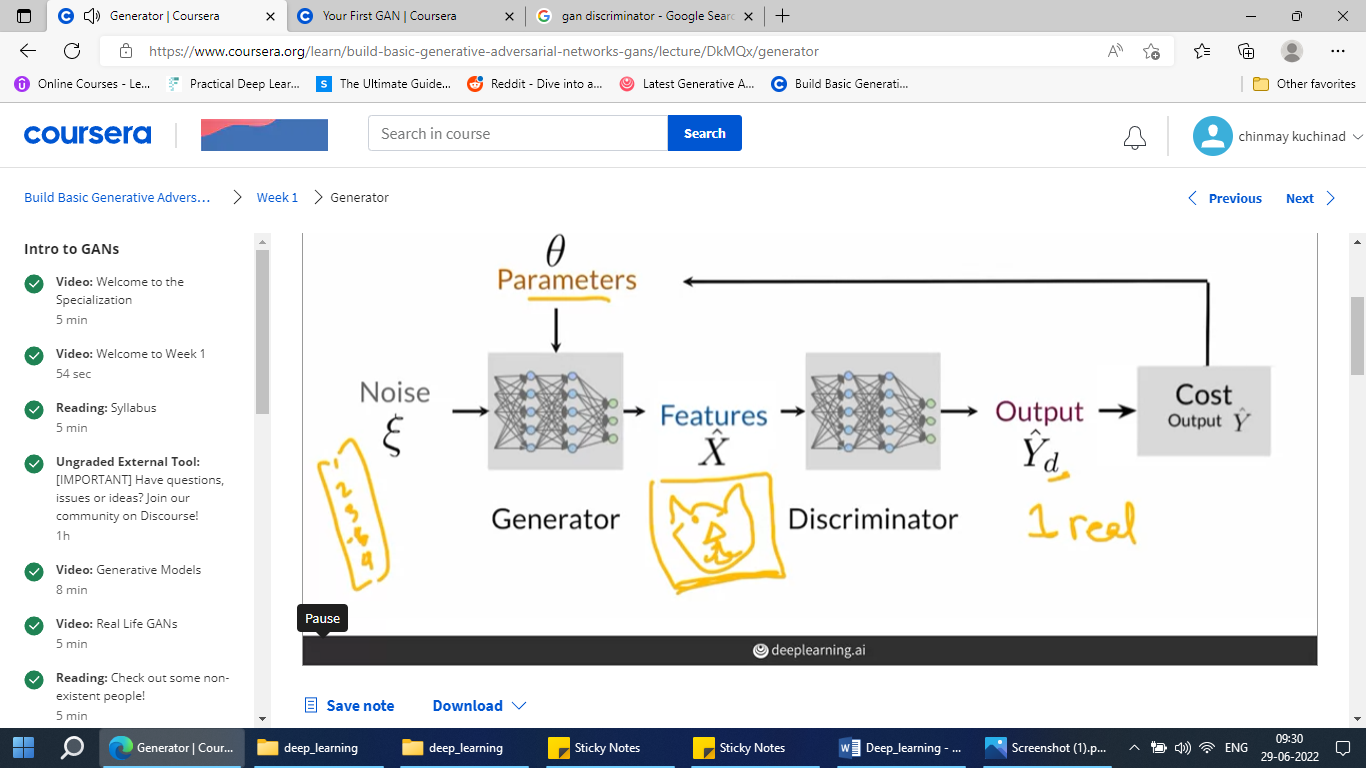


**Discriminator**

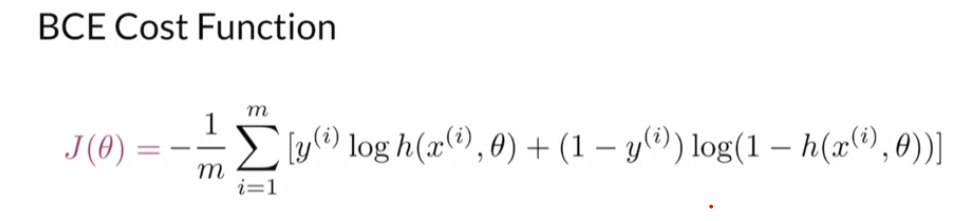
* It is a classifier which learns the probability of an input image belonging to class Y (real or fake) given the features X.
* Typically, we have a neural network which takes in the images as its inputs and calculates the probability of the image being real or fake based on the features its extracts from the input.
* The probabilities are the feedback to the generator which use it to improve their performance.

**Generator:**

* The generates take noise vector as input to generate fake data. It learns the probabilities of features X given the class Y.
* As the images are generated, they are fed to the discriminator which like we saw earlier will compute the probability of the image being real or fake. This is taken as feedback to tweak the generator parameters **theta.** This process is repeated until the generator gets close to mimicking the actual image as closely as possible.
* When it comes to the likelihood of certain images getting generated, it is worth noting that the more commonly occurring features (breeds of animals can be a specific example) have a higher probability of getting generated.



**BCE cost function(Binary cross entropy function):**



**Intuition:** It is close to zero when the labels and the predictions are similar. Approaches infinity when the label and prediction are different.

**Parameters:**

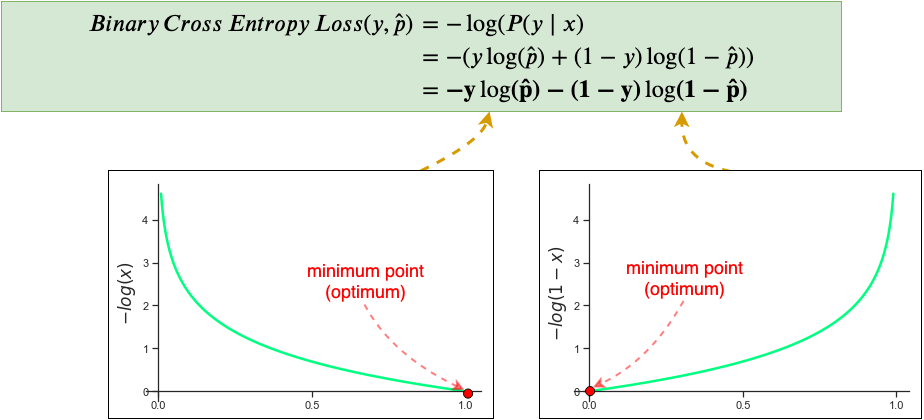
M – number of examples in batch. Used to obtain the average loss of the whole batch

x - features passed in through the predictions

y – True class labels

h – predictions that are made

* The first part(y\*log) is relevant only when the true class label is 1(real) but is mistakenly predicted as fake(0) by the model. In this case, the term approaches infinity.
* Similarly, the second term becomes relevant only when the class labels are 0 and the predictions are real(1).



**Training:**

**Discriminator**

* The discriminator looks at real and fake images over time, makes guesses, and gets feedback on whether its guess was right or wrong. So, while the training the discriminator, we provide it with the labels too.
* Over time, it learns to discern real from fake better, but note that since the generator is also learning, the fake images get more realistic and harder to discern. This cat and mouse game enables both models to learn in tandem.

**Generator:**

* The discriminator looks at real and fake images over time, makes guesses, and gets feedback on whether its guess was right or wrong.
* Over time, it learns to discern real from fake better, but note that since the generator is also learning, the fake images get more realistic and harder to discern. This cat and mouse game enables both models to learn in tandem.

Through the course of this training, the generator constantly receives feedback over the features being generated via backpropagation and improves itself.

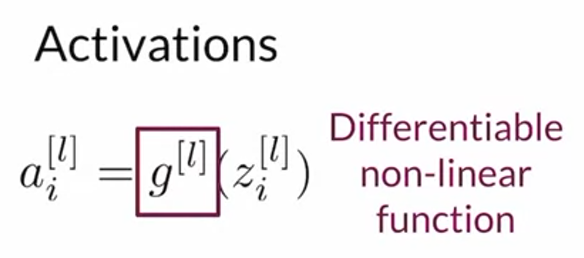
**Since these are trained in an alternative fashion, we have to make sure they are at similar skill levels**. In case we have a superior discriminator, all the images generated by the generator will be labelled as fake which will hinder the training process. Whereas in case of a superior generator, all the generated images will be labelled as real by the discriminator since it will be unable to make out the difference.

**Note:** Unlike in earlier image classification methods, we use deconvolutional neural networks where we start out with the features (noise signal) and arrive at the image.

**Applications**

* Image generation
* Image modification
* Super resolution (enhance the quality of a image)
* Photo realistic images
* Face aging

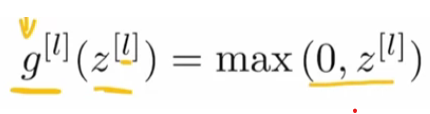
**Activation functions**



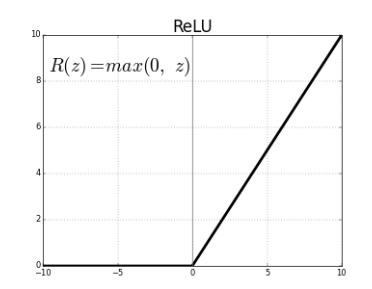
* We need them to differential because we intend to find the gradients during backpropagation.
* Non-linearity is preferred so that complex features can be computed.

**Common activation functions**

1. **ReLU(rectified linear unit)**



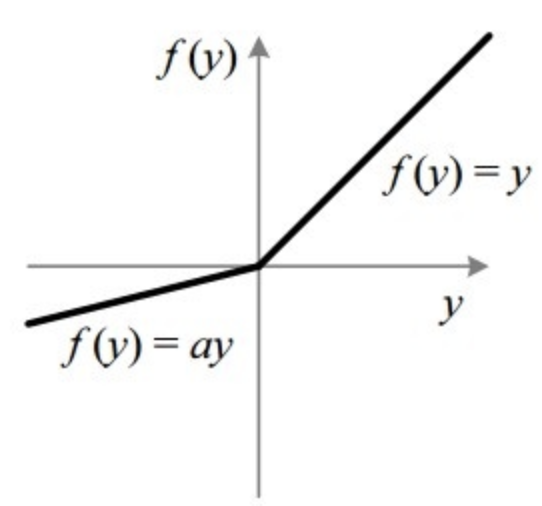
Basically, the negative values will be set to 0.



There is however an issue with this. For the negative values where the curve is flat, the gradient stays zero which hampers the training of few nodes which stagnate over time. This in turn adversely affects the training of the entire network. This is known as the **dying ReLU problem.**

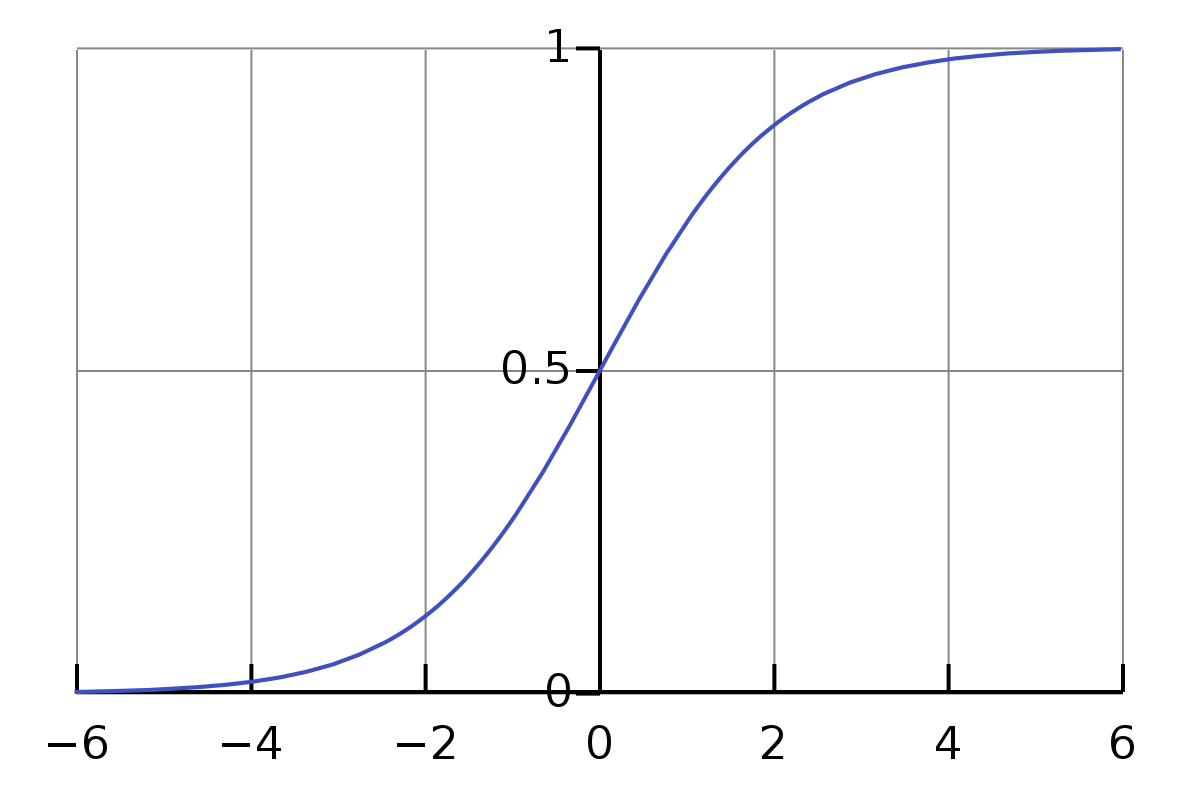
To overcome this issue, we go for the next time of activation function . The Leaky ReLU

1. **Leaky ReLU**



* For the positive values , the functioning is same as that of the ReLU function, but for the negative values, instead of driving the value to zero, the values are scaled with a negative slope(typically 0.1). This solves the dying ReLU problem to a large extent.

1. **Sigmoid function**

****

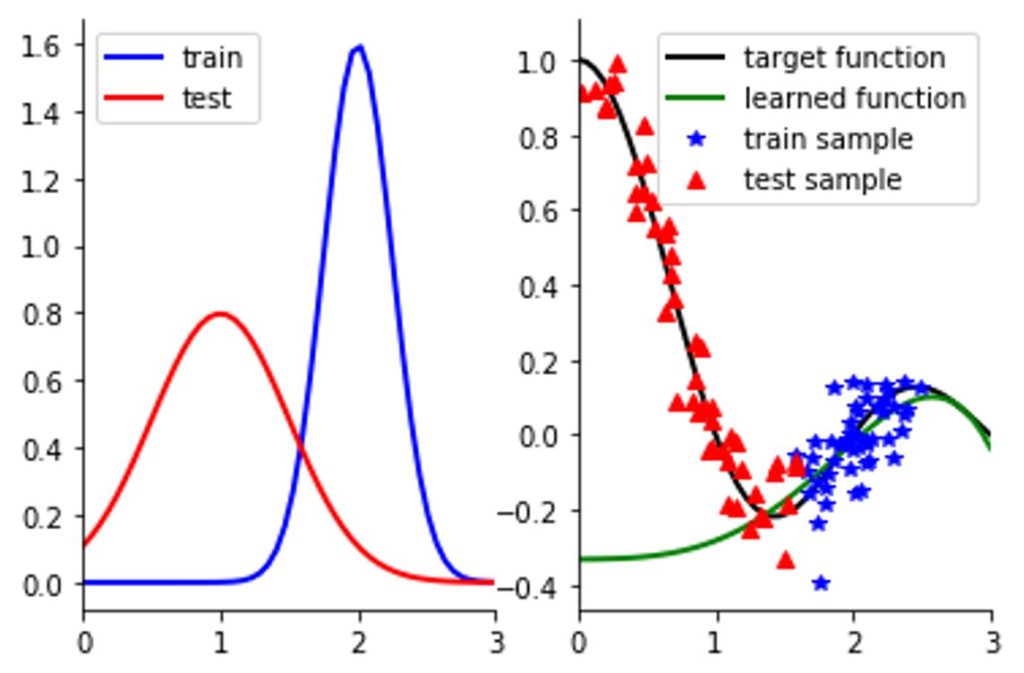
* Outputs a value between 0 and 1 and hence used in the output layer of binary classification models.

1. **Tanh function**

* Difference is that it outputs a value between -1 and 1. Distinctive feature is that the original sign of the inputs are preserved.
* Similar to sigmoid function, even this faces the vanishing gradient and saturation problem.

**Batch Normalization**

* If the inputs being fed to the neural network aren’t on a similar scale(comparable terms), it becomes difficult for the neural network to go ahead with its learning process.
* **Covariate shift:** This occurs when there is a distribution of input data shifts between the training environment and test environment. Our model will be trained on a particular distribution of data and when it encounters something different, it will be unable to identify the input features thus adversely affecting the model performance. Normalization helps us overcome this.



* **Internal covariate shift:** We define Internal Covariate Shift as the change in the distribution of network activations due to the change in network parameters during training.
  + In neural networks, the output of the first layer feeds into the second layer, the output of the second layer feeds into the third, and so on. When the parameters of a layer change, so does the distribution of inputs to subsequent layers.
  + These shifts in input distributions can be problematic for neural networks, especially deep neural networks that could have a large number of layers.
* **Batch normalization during training and testing**
  + During testing, the batch normalization processes data batch-by-batch. It takes batch size as an input.
  + The running statistics are obtained over the batch (mean and standard deviation) and used to obtain normalized z values.
  + These normalized values are then scaled and shifted using parameters which are learned during the training process. As a result of these factors, our distribution need not have values between 0 and 1 and hence gives us greater control over the input distribution
  + During testing, the process remains the same except that instead of running statistics we use fixed values for mean and variance calculated over the entire distribution which are fixed values.

**Convolution**

* Reduces the size of the image while maintaining the key features.
* It helps us identify features in an image via filters. The filter is moved over the image in predefined strides to obtain the features.
* Stacking up multiple features can help us identify complex features.

**Padding and Stride**

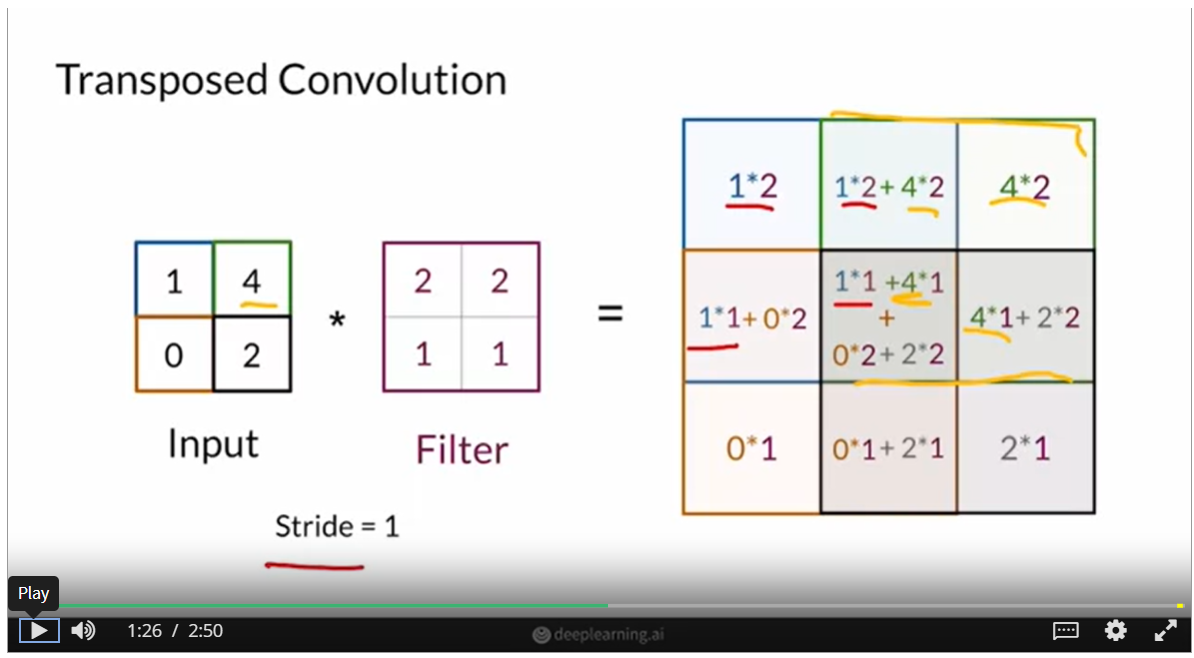
* The steps in which we move the filter over the image is called **strides.** There is a trade-off between the stride size(speed of computation) and the amount of image covered.
* Typically, it is noticed that the features at the center the image will get higher emphasis than the ones at the edges(because they are visited more by the feature detector matrix). In order to avoid this we come up with **padding.** This is analogous to placing this image matrix inside a frame of zeroes. By doing so , all the features within the image will get equal emphasis and the only the frames(filles with zeros) will be ignored.

****

**Pooling and Upsampling**

* The pooling method, which was discussed earlier during the CNN implementation is an important step which makes the computations easier and more efficient.
* Up-sampling is the opposite of pooling. We intend to rebuild the image from the pooling samples. This isn’t an easy task.
* One of the methods we use is Nearest neighbors method. Neither of these methods have any learnable parameters.

**Transposed convolutions**



* Contrary to convolution, here the filter is multiplied with every element of the input as it moves in a predefined stride. To deep it, in convolution the filter elements are multiplied with the corresponding image pixels and the sum of their products (a single pixel value) goes into the output. Here however, upon multiplication with the input pixel value, a matrix is obtained at every stride a matrix is placed in the output. (and the coinciding elements are summed up as shown in the picture)
* It is noticed that the pixels at the edges are affected by just one input whereas the ones at the center are affected by multiple/all inputs. This leads to an issue called the **checkerboard issue.** Basically, when you zoom into these images, you notice a checkerboard pattern instead of smooth pixels.

**more details -** [Deconvolution and Checkerboard Artifacts (distill.pub)](https://distill.pub/2016/deconv-checkerboard/)

Up next, we look into the issues faced by the binary cross entropy loss which we used as our cost function.

**Mode Collapse**

* In statistics mode of a distribution is its most frequently occurring element. Hence these can be labelled as the peaks in the distribution of features.
* This is typical for real life datasets. Considering the MNIST(digit classification) dataset for example, upon plotting the features we might notice that the distribution might be multi-modal with a mode for each number.(this mode will typically be the most frequently occurring feature, in case of handwritten digits consider a distribution of handwritten digit 7. The mode will be the which are easily distinguishable as 7)
* The problem arises when upon training, the generator learns to fool the discriminator by getting stuck in one mode/ producing the same feature repeatedly. So in case our hand written digit classifier has learned to distinguish between all the number except say 1 and 7, the generator will now repeatedly produce only these 2 elements in order to continue having an upper hand over the discriminator.
* The discriminator will eventually learn to differentiate the generator's fakes when this happens and outskill it, ending the model's learning.

**Problem with BCE loss**

****

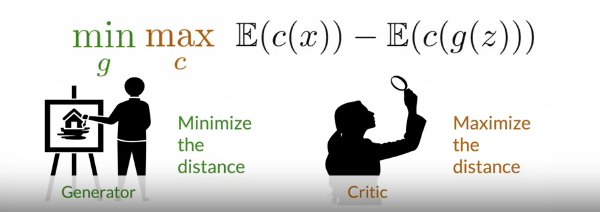
* Looking at the BCE cost function, the generator wants to maximise this which would mean that the discrimator is performing poorly ,whereas the discriminator wants to minimise the function which implies it is getting the real/fake judgements right. This is called as the minmax function.
* The primary objective of the GANs however is to make the generated and real distributions look similar which is done by minimising the cost function.
* Since the role of the discriminator is to output a single value classifying the image as real/fake whereas the generator does most of the heavylifting generating the image, the discriminator is expected to perform much better than the generator. But initially, since both these are untrained, the difference in performance isn’t that much.
* But as the training goes on and the discriminator improves too much, the feedback it provides to the generator becomes less informative and the function approximated by the BCE Loss will contain flat regions.
* This leads to the **vanishing gradient problem.**

**Earth Mover’s distance**

* In order to account for the deficiancies of BCE loss, we move on to an alternative – Earth mover’s distance.
* Given the real and generate distributions, this measures how different they are based on the distance between them as well as the amount to be moved to make the two distributions resemble each other.
* The problem with discriminators in BCE is that, when the distributions are very different, it starts outputting extreme value which aren’t very helpful for the generator to improve upon.Since this function has no ceiling, it continues to grow and doesn’t have any flat regions.

**Wasserstein Loss**

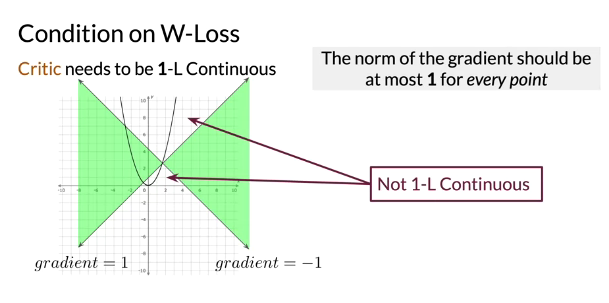
* Loss function making use of earth mover’s distance.

****

* The image here is the expression for the same. **c** is the critic(which the W-loss equivalent of discriminator). The cost function calculates the difference in performance when the critic is fed the real images **x**  vs when it is fed the generated inputs **g(z).**
* The generator wants to minimize this cost so that the generated images resemble the real ones as much as possible whereas the critic want to maximise this.
* Key difference from the BCE loss is that there is no log function in the expression and hence no compulsion for the output values to be bounded between 0-1. This is also a reason why we no longer call it a discriminator(since there are no classifications as such) but call it the critic instead.
* The unbounded output also means that the loss curve does not flatten out when the distributions are markedly different and hence the vanishing gradient issue is overcome.

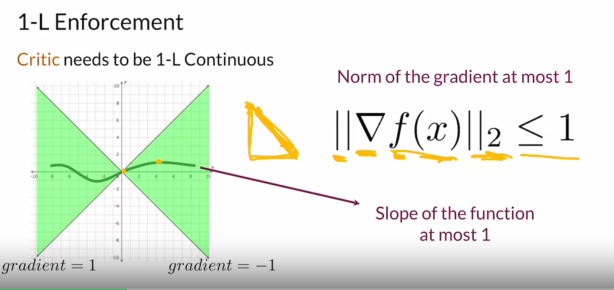
**Condition on Wasserstein critic**

* For a function to be eligible as a Wasserstein critic it must be 1-lipschitz continous , ie norm of gradient should be at most 1 at every point.

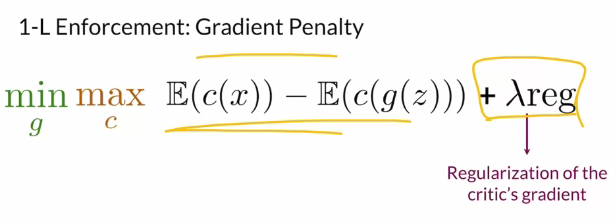


* In the plot above, at a particular point, lines with slopes 1 and -1 are drawn. For the function to be 1-Lipschitz continuous , it must grow in the region marked green. It is clearly seen that this function is not 1-L continuous.
* This condition must be satisfied along every point on the curve. This is key because it lends stability to the training process by ensuring that the variations will be bounded as the GAN learns.

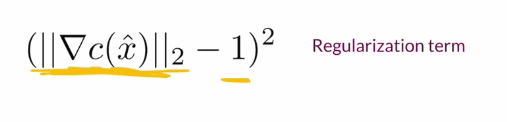
**1-Lipschitz Continuity enforcement**

****

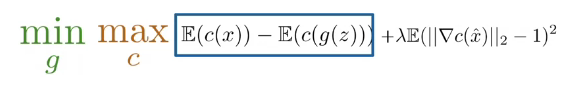
* **Weight clipping** forces the weights of the critic to a fixed inetrval. That is once we carry out gradient descent to update weights , we clip the critic’s weights . The downside is that this may limit the critic’s ability to learn and hence affect the model performance.
* **Gradient penalty** is another alternative. This involves the addition of a regularization term which penalizes the gradient when the norm is greater than 1.

****

Since checking the critic’s gradient at each possible point of the feature space is virtually impossible, you can approximate this by using interpolated images. This interpolation is obtained using a random real image and a generated one and combining them using a factor epislon.The gradient of the norm of the critic prediction over this image must be less than 1.



**Updated loss function**



* The first term in this loss function accounts for the earth mover’s distance and makes the model less prone to the vanishing gradient and mode collapse. The regularization term tries to make the loss function 1L continuous which makes it continuous and differentiable.

[From GAN to WGAN | Lil'Log (lilianweng.github.io)](https://lilianweng.github.io/posts/2017-08-20-gan/)

Protein GAN - [C1W3: ProteinGAN (Optional) - Colaboratory (google.com)](https://colab.research.google.com/github/https-deeplearning-ai/GANs-Public/blob/master/ProteinGAN.ipynb#scrollTo=q04P9icA8xIK)

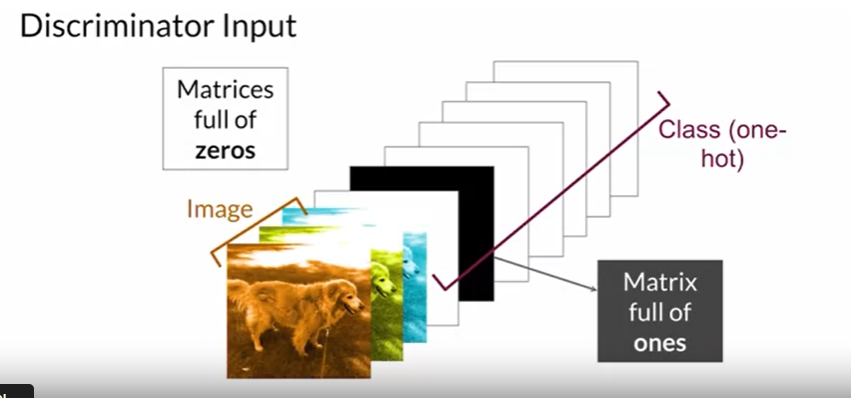
**Conditional Generation**

[Step by Step Implementation of Conditional Generative Adversarial Networks | by Neeraj Varshney | Analytics Vidhya | Medium](https://medium.com/analytics-vidhya/step-by-step-implementation-of-conditional-generative-adversarial-networks-54e4b47497d6)

* Allows us to control the type of output that is being produced. This means that examples can be generated for the selected class.
* For this we need the training dataset to be labelled.

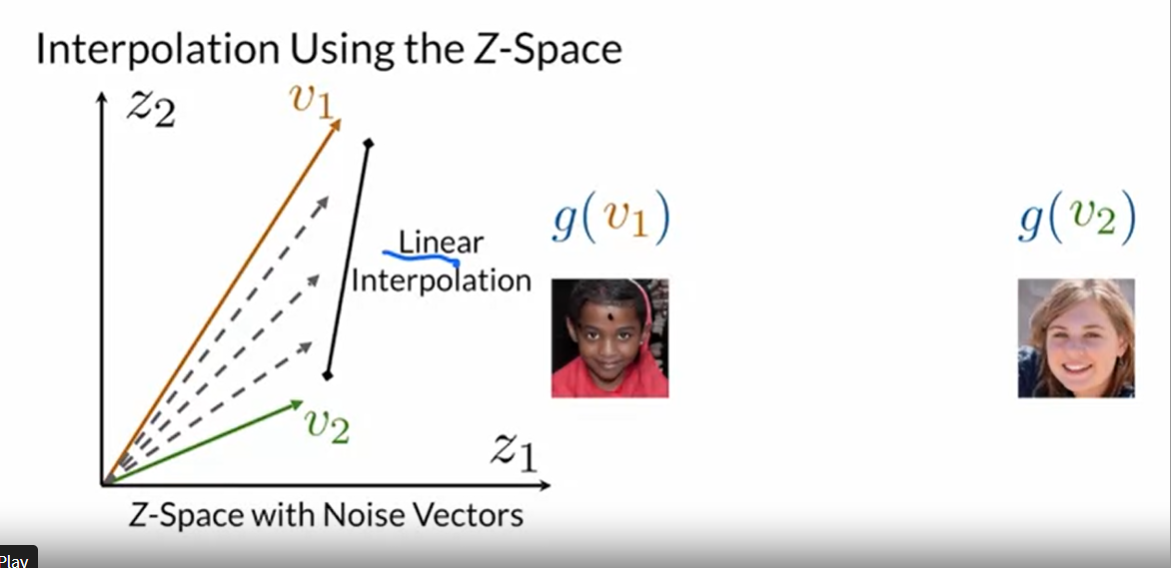
**Inputs**

* Typically, generators need a noise vector as an input to generate random examples. In case of conditional generation, we need to pass the class information as well for which we use the class vector (one-hot encoded). (input size will be **z\_dim(size of noise vector) + number\_of\_classes)**
* The input to the generator is a concatenated vector of noise and the class vector.
* Discriminators take in the images as well as the class information as its inputs. The image is fed in through 3 channels (RGB) whereas the one hot encoder vector as passed as channels too. (size will be **Number\_of\_channels + number\_of\_classes)**



**Controllable Generation**

* Allows us to control specific features in the image being generated. This is done by tweaking the input noise vector z.
* It doesn’t need training dataset to be labelled. While conditional generation leverages labels during training, controllable generation controls what features you want in the output examples after the model has been trained.
* The way this is done by finding vectors corresponding to different output features in the z-space(noise vector space).



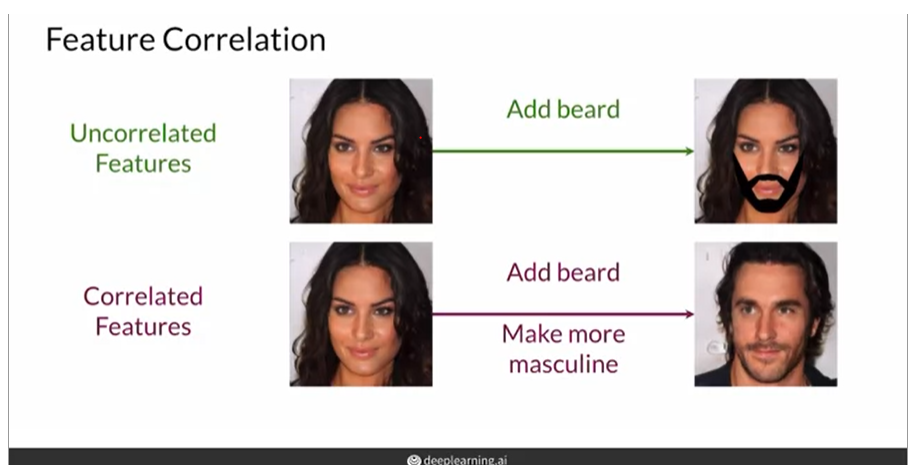
* As seen in the image, the vector v1 produces the image g(v1) whereas v2 produces g(v2). To explore the intermediate images, one can explore the vector values between v1 and v2 through linear interpolation.



* To control the output features, we need to find the directions in the z-space.

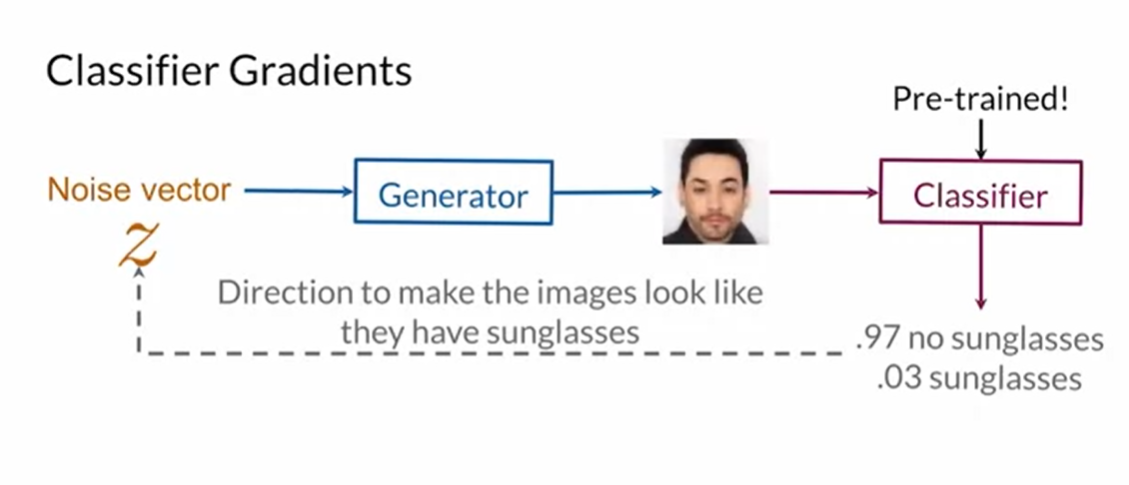
**Challenges**

* **Feature correlation:** Sometimes when features are correlated, we might unknowingly modify one feature while trying toe modify the other. For example,



* **Z-Space entanglement :** It becomes difficult to control single output features.

**Classifier Gradients**



* In this method, the generated image is passed through a pre-trained classifier and the results are used to obtain the noise vector (note- the weights of the generator aren’t modified).
* The noise vector is modified until the desired feature emerges. Like for example in the image shown above, let us say person with sunglasses is the desired feature.
* So, the noise vector is penalized for every image which is classified as no sunglasses until with get a generated image with classifier.
* Summary: Pre-trained classifiers can be used to find directions in the Z-space associated with features in the output of GANs. Remember that to find those directions, you need to modify the noise vectors without changing the generator.

**Disentanglement**

* In case of a disentangled feature space, every element in the noise vector would correspond to a particular feature in the output .(think of a one-to-one mapping in functions)
* Due to the indirect influence they have on the output ultimately obtained, the noise vectors are called the latent factors of variation.
* Encouraging disentanglement:
  + **Supervised method**: In conditional generation, we embed the class vector (one-hot vector) to the noise input. Here however the class data can be embedded within the noise data itself. But for continuous data it might become tedious.
  + **Unsupervised method**: BY introducing a regularization term which associates some error to each feature .

**Part 2 – Building better Generative Adversarial Networks**

**Evaluating GANs**

* Evaluating the performance of GANs is not straight forward as in classifiers since there is no ground truth to compare with.
* Thus we use different parameters: **Fidelity(**quality of images produced), **Diversity(**variety of images produced). We need a right balance by quantifying these two properties.

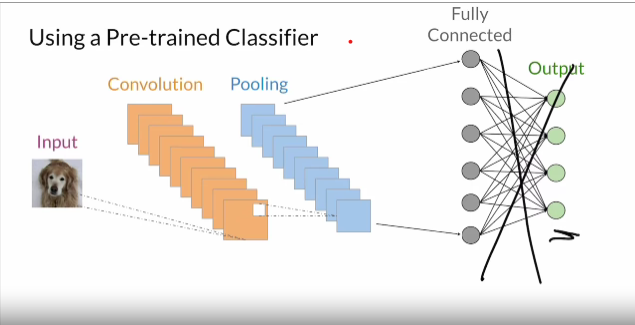
**Comparing Images**

Although fidelity and diversity are crucial while measuring the performance of our GAN, it becomes difficult to measure them while comparing images. Hence we move onto the following features.

1. **Pixel distance :**  This method involves the computation of absolute difference between pixel values in the real and generated image. Under ideal circumstances, the sum of these differences add up to 0 implying that the images are identical.
   * However, this method is not reliable .**Sometimes imperceptible differences might have large absolute difference due to the pixel values** (say for example a single column of pixels have been swapped or are different).Hence we move onto a more robust measure.
2. **Feature distance :** Self explanatory. We first extract the features from the images and then compare them.
   * SInce this method involves higher level semantics, it is less sensitive to shift. This method is common outside GANs too.

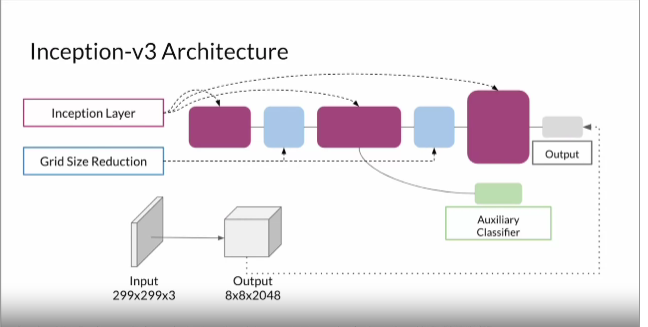
**Feature Extraction**

* For the purpose of feature extraction, we use pretrained classifiers(ones that have been typically trained on large datasets). The weights of these classifiers have features encoded within them.
* It must be noted that we are not interested in the final output of the classifier. **We cut up the network instead and use the intermediate layers which have useful information within them**. Typically, the max pooling layers are used mostly since they have fine grained feature data imbibed into them.
* However it is not a rule of the thumb to go for the outermost layer. We can choose any suitable inner layer based on our need. It is important to note that as we move deeper into the network., features get primitive(as we move towards input layer). The first few layers may only help us in simple tasks like edge detection.



* It is common practice to use classifiers trained on the image net dataset. Using features from this classifier is referred to as embedding.

**Inception-v3 and Embeddings**

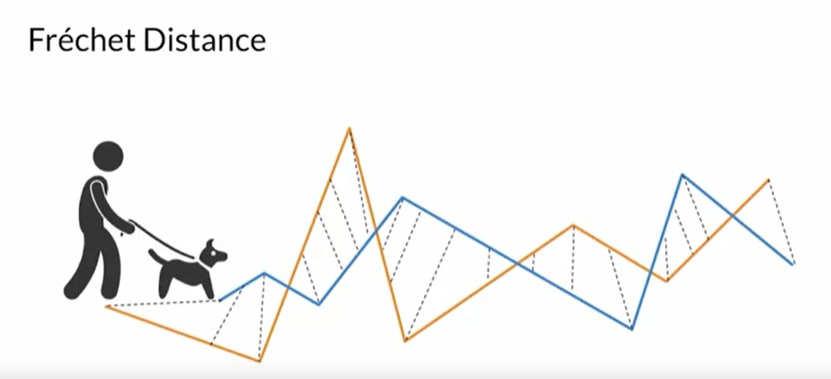
****

* **Obtain Embeddings :** By compressing the essential information within the fed image(output is of dimension 2048), the inception-v3 network allows operation using fewer dimensions per image. Then we obtain feature vectors corresponding to the features detected in the image. It must be noted that similar vectors will have similar embedding values.
* **Comparing embeddings** will get us the feature distance. The methods for this will be discussed next.

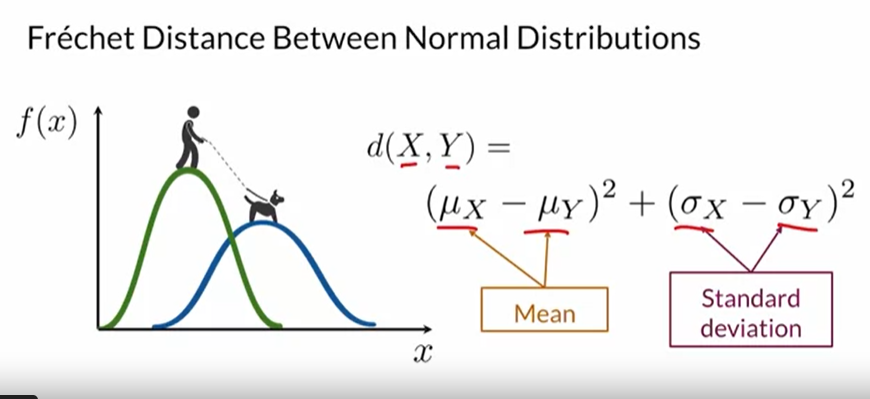
**Frechet Inception Distance(FID)**

[Fréchet Inception Distance | Neal Jean](https://nealjean.com/ml/frechet-inception-distance/)

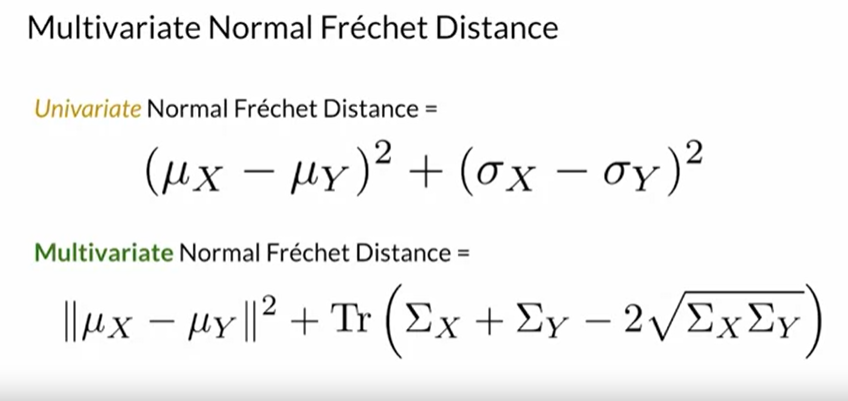
* For intuition, consider a man walking his dog with a leash round its collar. They both will walk along different curves at their own speeds. Frechet Distance involves calculating the minimum leash length that would be needed so that the dog and the man and walk almost in sync from beginning to the end.

****

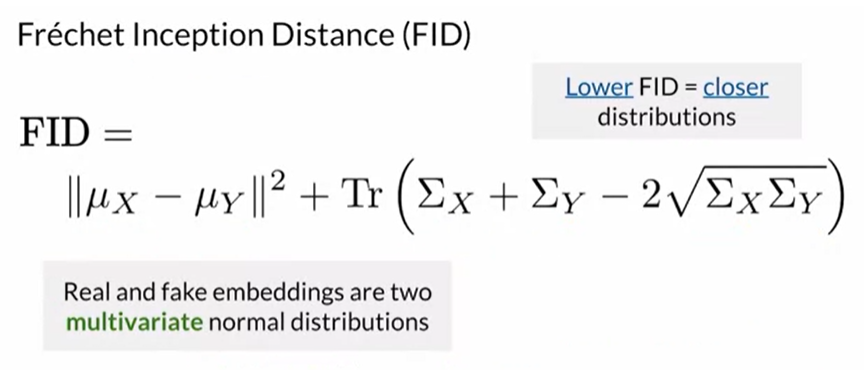
* This intuition can be expanded to obtain the distance between two normal distributions. It is noticed that the distances further away from each other are penalized (the squared term).

****

* In most real-life use cases, we go for multivariate normal distributions which facilitate the modelling of more complex distributions. The features are better represented here.
* When we move to multivariate distributions we move away from variance and start using covariance (how the features affect one another, including themselves). We will often encounter **covariance matrices**. If the diagonal elements are non-zero whereas the non-diagonal elements are zero, we can say that the features are independent. However, in case the non-diagonal are non-zero, it means that the dimensions co-vary. Negative value means negative correlation (inverse proportion).
* Upon viewing the frechet distance formulae for univariate and multivariate distributions, it is observed that they are essentially the same.

****

**Tr** means trace (sum of diagonal elements)

****

* In the formula above, **x** stands for real embeddings while y stands for fake embeddings. FID looks at the mean and the covariance matrices of the real and fake multivariate normal distributions and calculates how far apart those statistics are from each other.
* In short, smaller the fid values closer the real and fake distributions are. Although there is no fixed range of values considered **ideal,** it is better to be in the vicinity of zero.

**Shortcomings of FID**

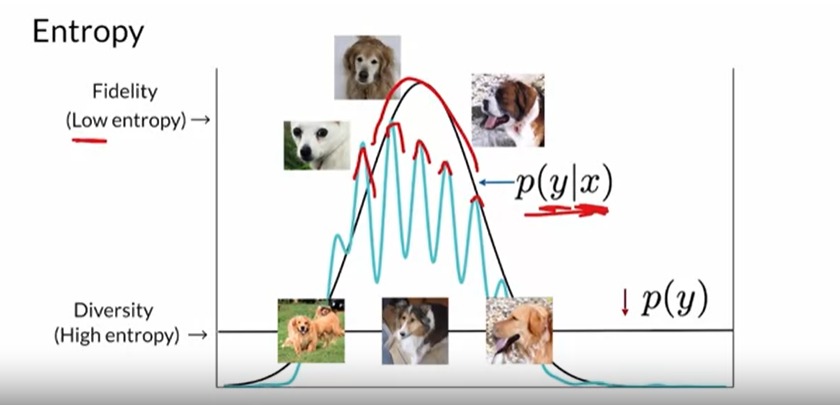
* They need large sample size of input data. Although result must be independent of size, it is seen that the FID value is lower for examples trained on a larger sample size.
* Pre-trained inception model might not capture all features.
* The process is slow to run. It only used limited stats (only mean and covariance).

**Inception Score**

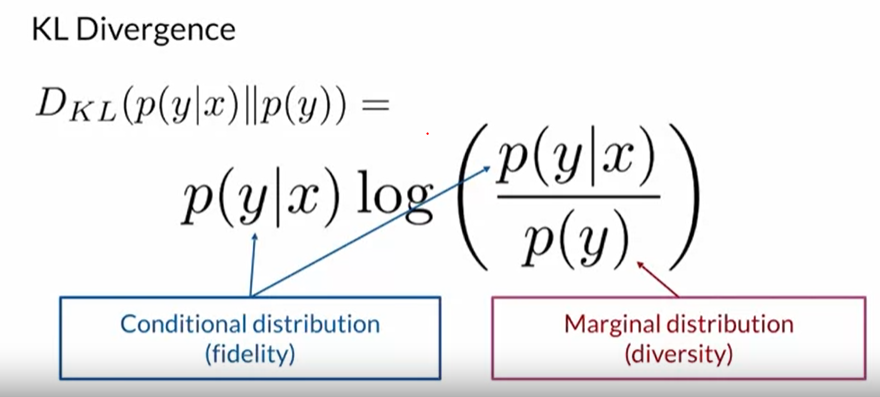
[A simple explanation of the Inception Score | by David Mack | Octavian | Medium](https://medium.com/octavian-ai/a-simple-explanation-of-the-inception-score-372dff6a8c7a)

[GAN — How to measure GAN performance? | by Jonathan Hui | Medium](https://jonathan-hui.medium.com/gan-how-to-measure-gan-performance-64b988c47732)

* This is an alternative method to calculate performance of GANs but now has been widely replaced by FID. Here a classifier is used as is (ie, we obtain class-based predictions on the GAN generated images). This serves as a measure of the **fidelity** of our system.
* The operation mentioned in the previous step is analogous to calculating p(y|x). We are trying to estimate the distribution given the input images. Ideally, we must obtain peaks at certain class outputs indicating that the GAN is able to produce specimens of a particular class. This means that our system is moving towards **low entropy** since our examples are clustered at peaks and the randomness has been reduced to an extent.
* However, when we consider **diversity,** we expect our model to be **high entropy**. We expect it to produce as many distinct classes as possible else we will have an issue of mode collapse. The curve p(y) (**marginal label distribution)** models this behavior.

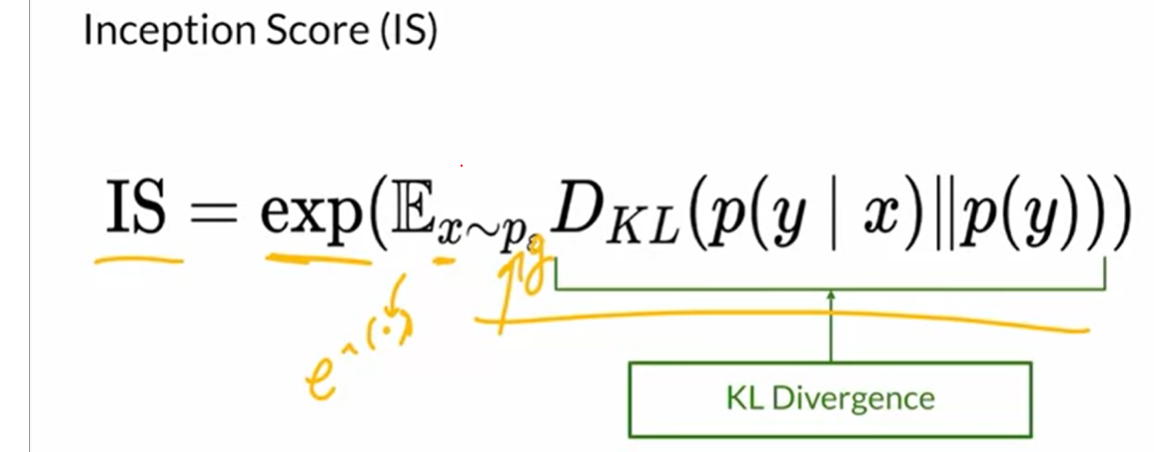


* These two features/factors can be combined using a measure called **KL divergence.** It measures the relative entropies of fidelity and diversity.



So our goal is to minimise the numerator(we want to definitively predict the class of input images being fed) while increaing the denominator.

* **Inception score formula:**



**Higher the score, better it is.**

**Shortcomings**

* Mode collapse can go undetected by inception score, because the score will still be high.
* It only looks at fake images and there is no comparison with the real images.
* It can sometimes miss useful features.

**Sampling and Truncation**

* There are considerations to be made while sampling images that are to be evaluated through FID. Our fake images are generated based on the input noise vector and this vector is usually a normal distribution.
* We can tweak fidelity and diversity by choosing where to sample (closer to the mean or away from it).

**Truncation trick**

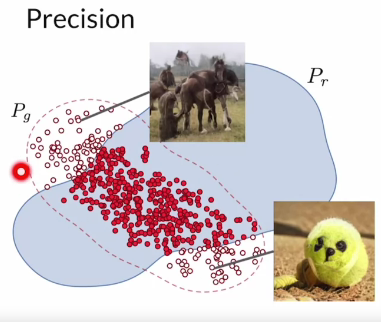
* For higher fidelity we sample more towards the mean, truncating off the tail values as well as the peaks. But the downside of doing this is that we lose the funky examples, most of our chosen images will look similar.(low diversity)
* For higher diversity, we need to sample more from the tail by having a low truncation value.
* The procedure involves tradeoff between fidelity and diversity. Empirically it has been observed that the FID score is high when either of these values are extreme. So there has to be a balance.

**Summary**

* It truncates the normal distribution that you sample your noise vector from based on a hyperparameter that determines how much of the tails to cut off or keep.
* If you want higher fidelity, you want to sample around 0 and truncate a larger part of the tails. If you want greater diversity, then you want to sample more from the tails of the distribution and have a lower truncation value.

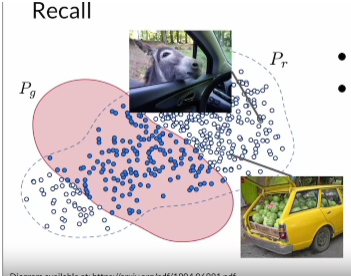
**Precision and Recall**

**Precision :** This relates to fidelity . In the graphs that follow, Pg stands for the probability distribution of the generated images whereas Pr stands for the real images



* It looks at the overlap between reals and fakes divided by the extra ones that are generated(the non overlap reds). In short ,precision = fakes that look real / all fakes.
* Truncation trick can be used to improve precision.

**Recall :** Relates to diversity



* This looks at the overlap between reals and fakes over the reals that the generator cannot model(non overlap blue)
* recall = overlap of reals / all reals. Measure of how well the generator is able to model real images

**Disadvantages of GAN**

* Lack of intrinsic evaluation metrics(to measure model performance)
* Unstable training. Model might take a long time to train. Although this is overcome to a large extent using Wasserstein’s loss.
* Density estimation is not possible. Basically, this involves predicting/modelling how often a particular feature might show up.
* The generator is not trained to be invertible. That is given the generated image, it is not that straightforward to extract the input noise vector that was fed to the model. This feature is desired when editing images specially using controllable generation.

**Variable Autoencoders**

* VAEs work with two models, an encoder and a decoder, that take a real image, find a good way of representing that image in latent space, and then reconstruct a realistic image. A GAN takes noise as input and never directly sees the real image.
* The advantage with this method is that training is more stable. We can use the decoder encoder to arrive at the input noise hence the model is invertible.
* However, the images produced are of a lower quality(fidelity takes a hit). Therefore when the goal is realistic image generation,we still prefer GANs.
* There is a recent example called VQ VAE which produces good results but it is not a pure autoencoder. They use something called autoregressive models which rely on previous pixels to predict the upcoming pixels.

**Machine Bias**

* Based on findings for COMPAS: an algorithm used to determine the future risk factor of a convict. IT was increasingly biased towards black people and often produced results that defied the ground truth.

[Machine Learning Glossary: Fairness  |  Google Developers](https://developers.google.com/machine-learning/glossary/fairness)

**Defining Fairness**

* Demographic parity : Making sure that the prediction obtained is independent of factors like ethnicity

**Ways in which bias can be introduced**

* **During training phase.** 
  + When there is no variation in the inputs being fed.(one class or race might be overrepresented).
  + If we are using labellers to label the generated images, their demographic might also introduce some bias into the system.
* **Evaluation bias :** Images can be biased to reflect ‘correctness’ in the dominant culture.(example- the side of the road on which vehicles are driven)
* **Model architecture bias :** Can be influences by the coders who designed the architecture or optimized the code

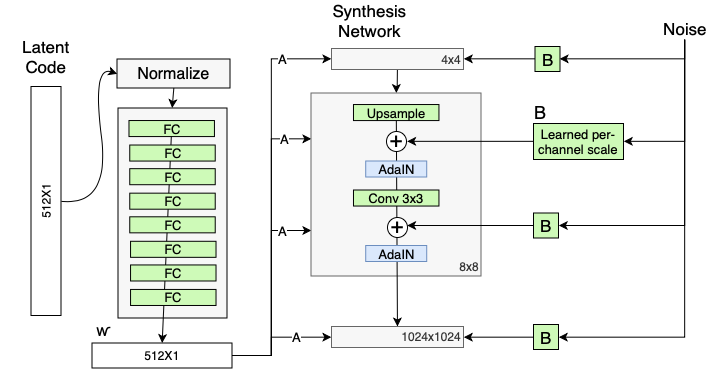
**Main improvements**

* **Stable Training techniques:** One of overcome mode collapse.
  + As seen earlier, mode collapse can be a major issue when it comes to training GANs. One approach to overcome this is to use standard deviation across a batch of generated images to encourage diversity. While the disciminator may not be able to pick up on mode collapse based on individual images, standard deviation will help them to pick up the pattern(ie, low standard deviation would mean that the same outputs are being duplicated).
  + Enforcing 1-lipschitz continuity. As seen in Wasserstein loss function
  + Using moving average of weights for generator across different iterations.
* **Capacity** – using high resolution images for training and better hardware
* **Diversity** - have larger datasets with more variety of images

**StyleGAN**

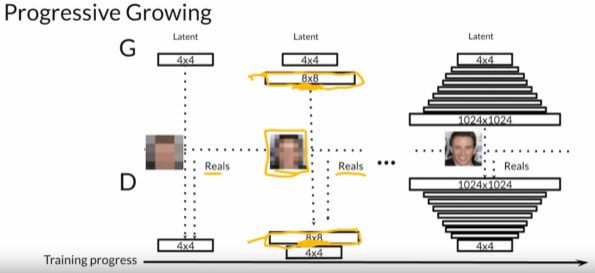
[GAN — StyleGAN & StyleGAN2. Do you know your style? Most GAN models… | by Jonathan Hui | Medium](https://jonathan-hui.medium.com/gan-stylegan-stylegan2-479bdf256299)

* The goal of styleGAN is to have higher fidelity in images being generated, have greater diversity in the images generated as well as have more control over the features in these images.
* **Style based generator :** The input noise is fed to a mapping network which produces intermediate noise which is fed to generator multiple times. In addition to this, random noise is added for generating stochastic features.
* The styles are extracted at various points from the intermediate noise **w** and fed to the styleGAN generator. In the beginning **w**  will have coarser features(ex-shape of face), and ultimately as the training continues it extracts the finer features(ex-hair colour).
* This intermediate noise is fed to the generator through a process known as **AdaIN**(adaptive instance normalization)
* **Progressive growing :** Improves image resolution as the training progresses.

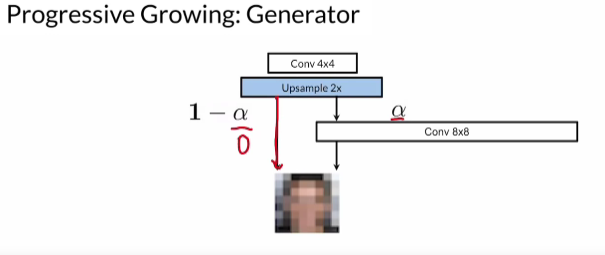


**Progressive Growing**

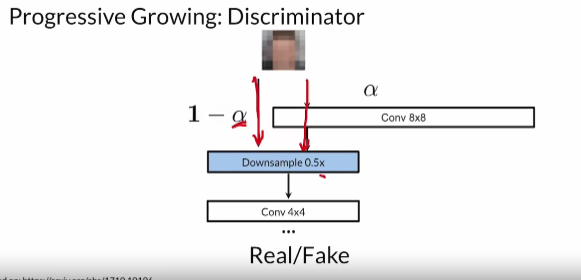
* Generating high resolution images starting with low res images.



* Generator : The doubling procedure shown in the image isn’t straightforward. Initially we rely heavily upon upsampling to generate the higher resolution picture and less on learned parameters(a split of 99%-1%) let’s say. As the training progresses, this changes and reaches a point where the model will rely almost soley upon the learned parameters to obtain the high resolution images.

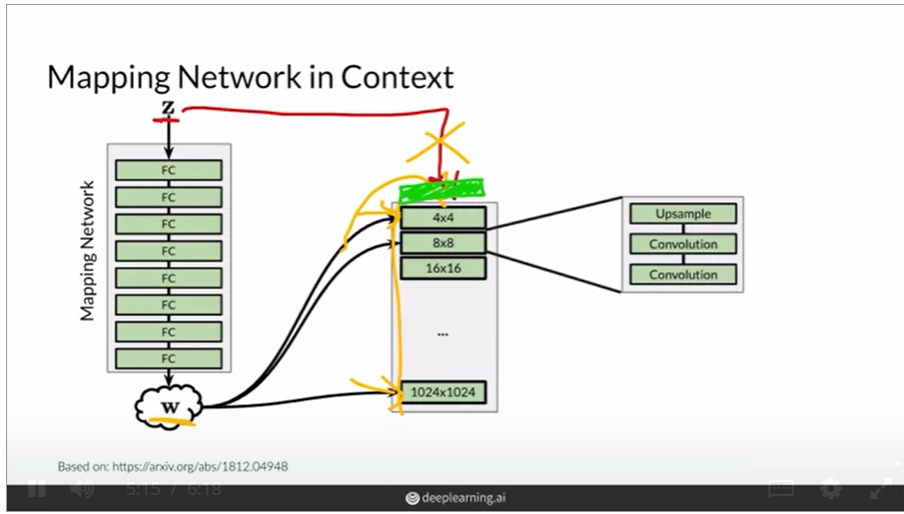


* The alpha parameter shown here will be 0 to start with and will eventually tend to 1.
* At the discriminator, a similar procedure is followed but we go for downsampling instead to obtain the output.



**Noise Mapping Network**

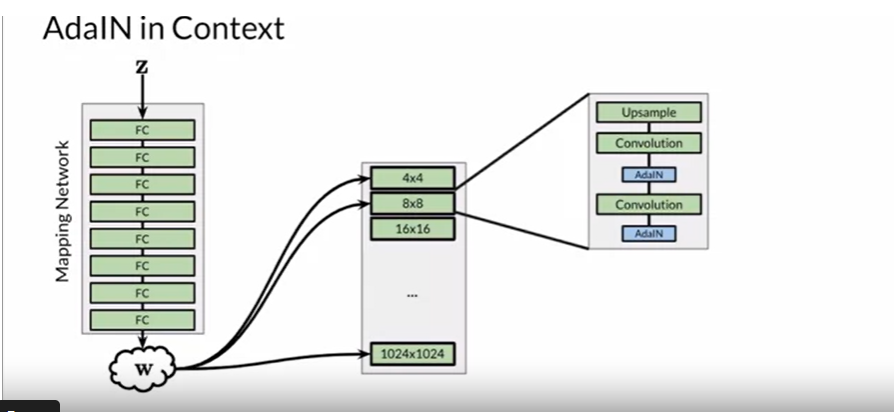
* This involves processing the noise vector into an intermediate noise vector(w) of the same dimension. This is achieved using the 8-layer Multi-layered perceptron network. Having an intermediate mapping for the noise vector (z) allows the intermediate vector (w) to be in a more disentangled space. This makes it easier to learn 1:1 mappings.
* We do this to increase disentanglement (noise vector and image features are mapped in one-to-one manner). Without mapping this can be difficult to achieve.
* The output of the mapping network(w) is fed to the progressive growing network at different junctures.



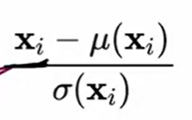
* The progressive growing network has a constant value in the beginning and subsequently updates itself based on the noise it receives from the mapping network.

**Adaptive Instance Normalization(AdaIN)**

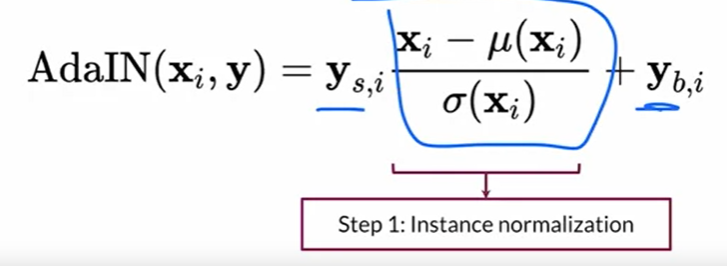
* AdaIN transfers style information onto the generated image from the intermediate noise vector w



* **Step 1:** Normalize convolution outputs using Instance Normalization. Unlike batch normalization, this only considers one instance at a time.



* Apply adaptive styles using the intermediate noise vector. The intermediate noise vector w goes through fully connected layers to obtain two parameters ys (scale) and yb (shift). These parameters then go into the AdaIN network via the following formula:



* When w is added to the earlier blocks of the PG network it will represent the coarser features and when it is added to the later later blocks it will represent the finer features.

**Style and Stochastic Variation**

* **Style mixing**: A way to achieve this is by feeding in different w values (intermediate noise) to different layers of the PG network. For example, **w2** obtained from noise vector **z2** can be fed to the first half of the PG network while **w1** obtained from **z1** can be fed to the second half of the network. It is not necessary to be a near linear split. Random noise can be fed based on the features we are interested in mixing.
  + This increases the diversity of the generated images too.
* **Stochastic variation:** We can obtain finer variations within a given image by injecting noise into the model. To achieve this , we first sample noise from the normal distribution which is then concatenated with x(the convolution output) before AdaIN.