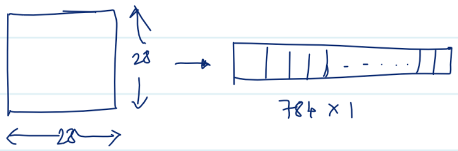
Generative Adversarial Networks (GAN) :-

→ Popular since 2018.

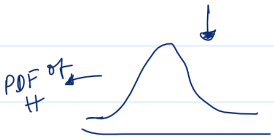
→ task:- Given MNIST images $[D = \{0,1,2,\dots,9\}]$, create / generate new MNIST images $D' = \{0,1,2,\dots,9\}$ that are similar to images in D but not the same.



Simpler task:- Given $D = \{$set of heights of students in a class$\}$, generate $D' = \{h_1', h_2', h_3' \dots\}$

very similar

$D = \{h_1, h_2, \dots, h_n\}$



→ $PDF = P_\theta(H)$. Assume it's $N(150, 30)$

↳ could be normal / uniform / log normal etc., can be determined using QQ plot / KS-testing.

$\underbrace{\underbrace{\text{normal}}_{\mu,\sigma} / \underbrace{\text{uniform}}_{a,b} / \underbrace{\text{log normal}}_{\mu,\sigma} \dots}_{\theta}$

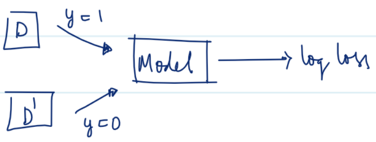
② Once the distribution has been found, we can generate random samples from the distribution = D'

D' & D have the same distribution

∴ D & D' are "similar" (probabilistically)

③ Measure how similar D & D' are. We could use something like KL-Divergence, JS-dist etc.,

or we could do something like this (Model Based Similarity test)



if model is able to seperate well ⇒ D & D' are different

ie, model is able to distinguish b/w them. Vice-versa.

G → Generating data set

A → D & D' are "adversaries". They are compared with one another.

N → We use Deep Neural Networks instead of prob models generally.

→ Heights : Univariate, distribution is easy.

MNIST : Multivariate with 784 variables. Complex distributions. KL div and JS-dist won't scale as well. This is why models are preferred.

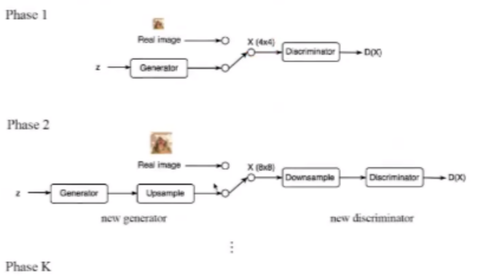
→ GANs are extremely hard to train and involve lots of hyperparameter tuning. There are hacks to make them work.

→ Pixel CNN, Pixel RNN :- Generating data based on each pixel. Hard to generate new samples from.

→ Variation AutoEncoders :- Do not work as well as GANs today.

→ There are task specific GANS.

Progressive GANs :- to create fake images

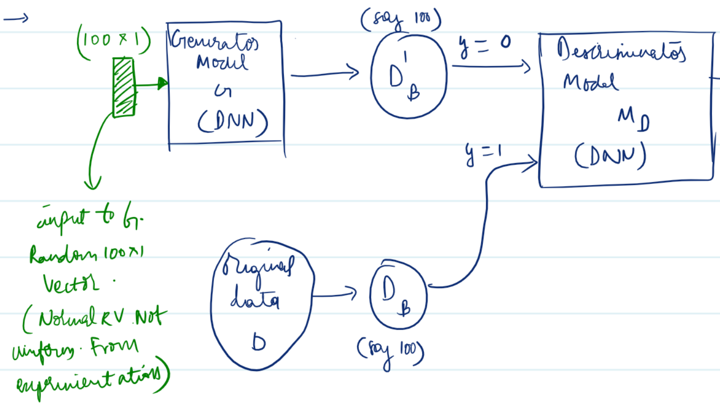


slowly increase quality

→ SRGAN to deblur an image:

→ Most of the intuition behind GANs come from game theory.

→



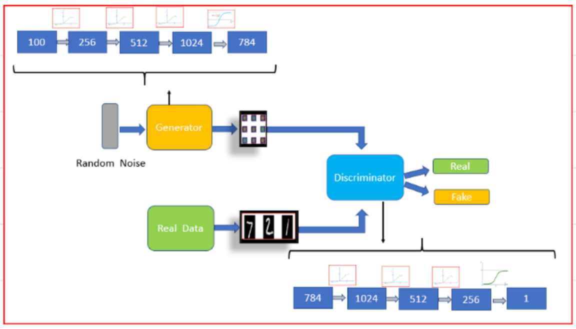
→ maximize log loss (then $D' \simeq D$)

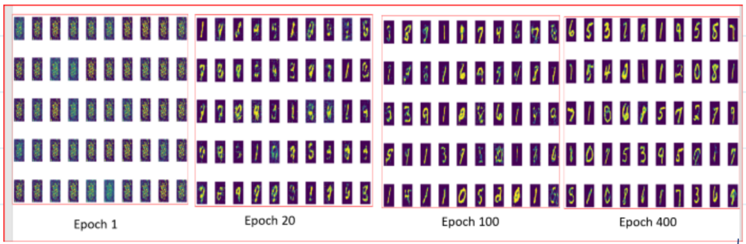
2 stages :-

① Fix M_D & train G -
Initially i/p to G is random & the weights are also random. ⇒ image generated is also very random.
⇒ model will discriminate easily.

② Fix G & train M_D. We do Gradient Ascent to maximise log loss. Errors is sent back to G & weights are updated.

→ This is similar to SVM SMO. When 2 vars are to be minimized, we fix one, optimize the other & then fix that & optimize prev one. Can be done in Keras by using model_name.trainable = False. This freezes the model.





initially everything is random.

By the end, looks indistinguishable.

→ After lots of experimentation, [-1, 1] normalization aka tanh showed better results than [0, 1] norm aka sigmoid.
∴ tanh activation is preferred.

→ DCGAN (Deep Convolutional GAN) uses CNN's for generator & discriminator.

→ SGD for discriminator & Adam for generator best optimizers.

→ Use dropouts in G in both train & test phase.