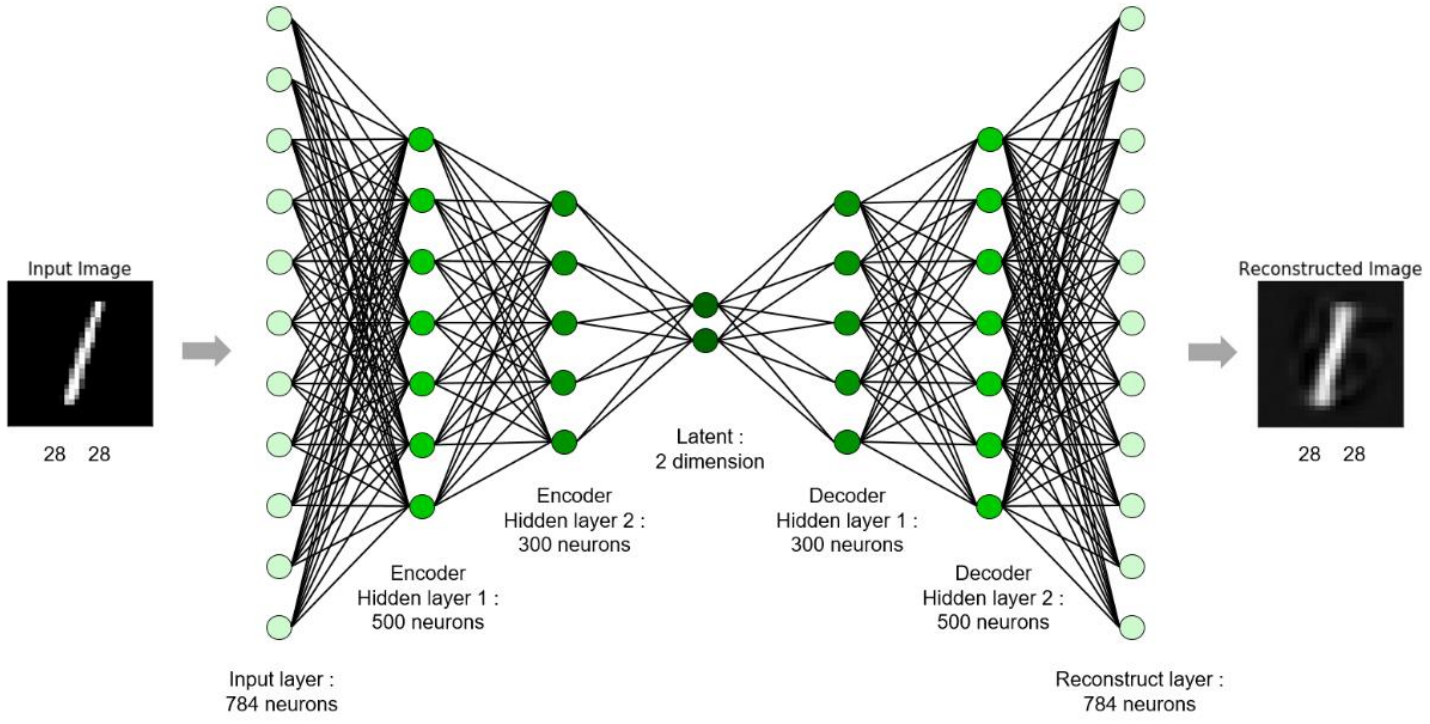
AUTOENCODER AND GAN

Shingchern D. You

Autoencoder

- Autoencoder: Input = output during training(credit: http://i-systems.github.io/HSE545/machine%20learning%20all/KIMM/06_KIMM_Autoencoder.html)

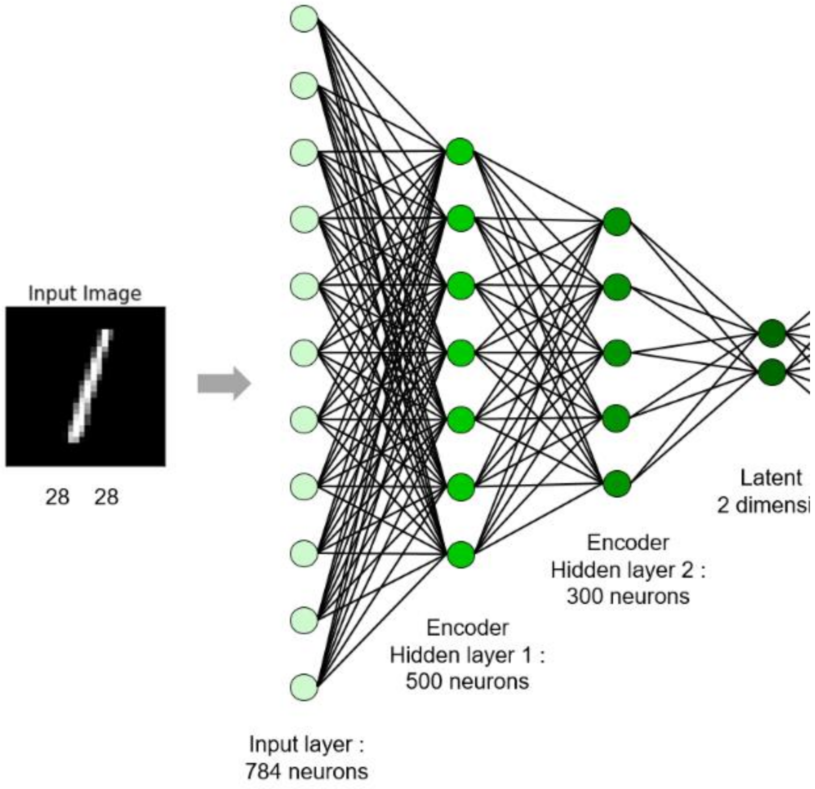

Input Image
28 28
Input layer : 784 neurons
Encoder Hidden layer 1 : 500 neurons
Encoder Hidden layer 2 : 300 neurons
Latent : 2 dimension
Decoder Hidden layer 1 : 300 neurons
Decoder Hidden layer 2 : 500 neurons
Reconstruct layer : 784 neurons
Reconstructed Image
28 28

Autoencoder

- If input = output during training, what is autoencoder used for
- Intuitive application: **Non-linear dimensionality reduction** (ref: https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798)
- Some important properties for this application
  - Data-specific
  - Lossy
  - Unsupervised learning (no labels needed)

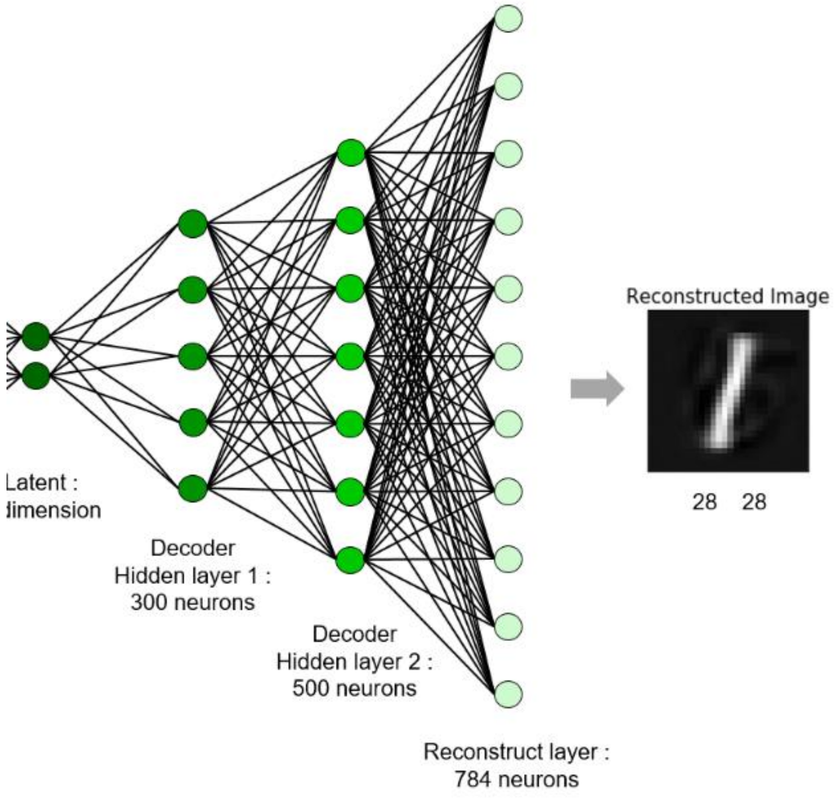
Autoencoder

- Encoder



Autoencoder

- Decoder



Autoencoder

- Internal representation is called code or latent vector
- Some hyper-parameters needed
  - Code size
  - Number of layers
  - Number of nodes in each layer
  - Loss function
- Recall the overfitting problem may also occur in the autoencoder
- Be careful about hyper-parameter setting

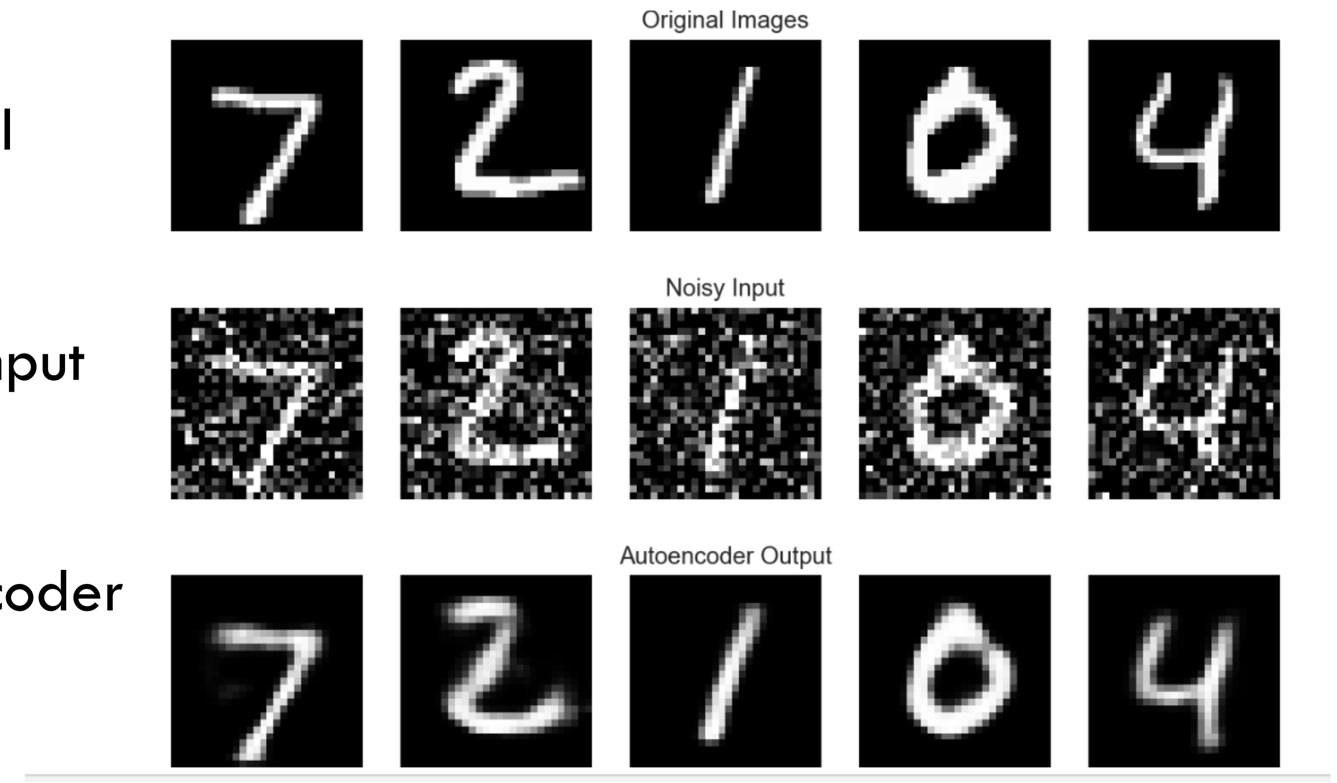
Autoencoder for de-noising

- (Source: https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798)

Original

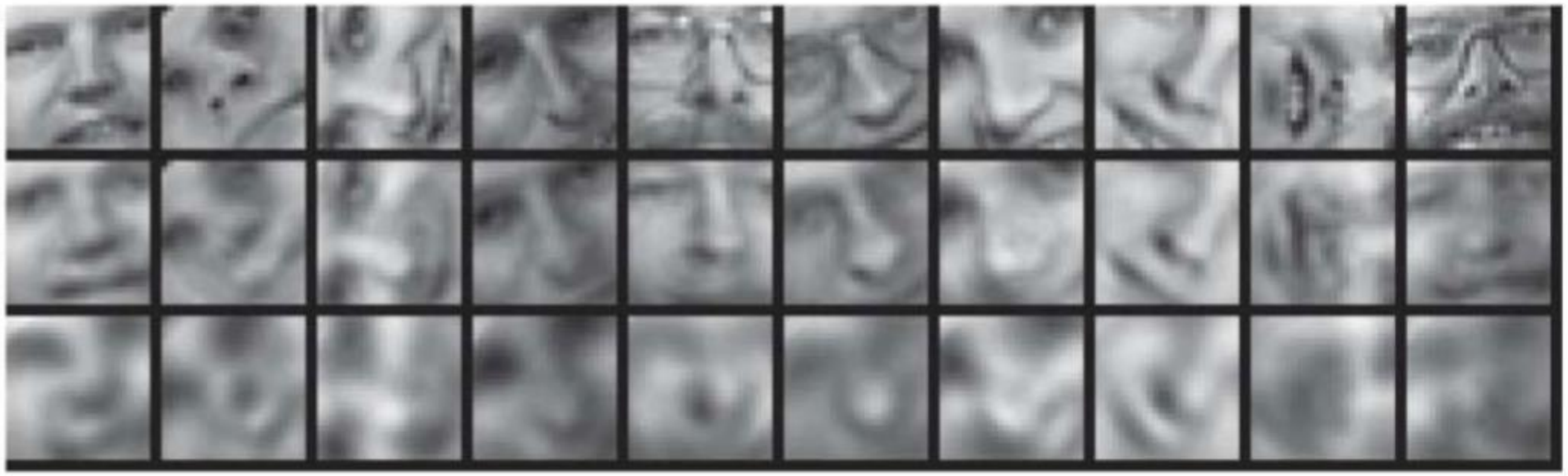
Noisy input

Autoencoder Output



Autoencoder for dim reduction

- From Hinton's paper (Reducing the Dimensionality of Data with Neural Networks): Top original, middle: from autoencoder, bottom from PCA



Autoencoder for classification

- Method I (conventional way)
  - Train one model (or one model per class)
  - Use the internal codes as features to SVM or another neural network
- Method II (we found it sometimes better)
  - Train one model per class
  - Present the test sample to each autoencoder
  - Assign the test to the class with smallest MSE between autoencoder input and output

Variations of autoencoder

- De-noising autoencoder (traditional one, given previously)
- Sparse autoencoder
- Variational autoencoder (VAE)

Sparse autoencoder

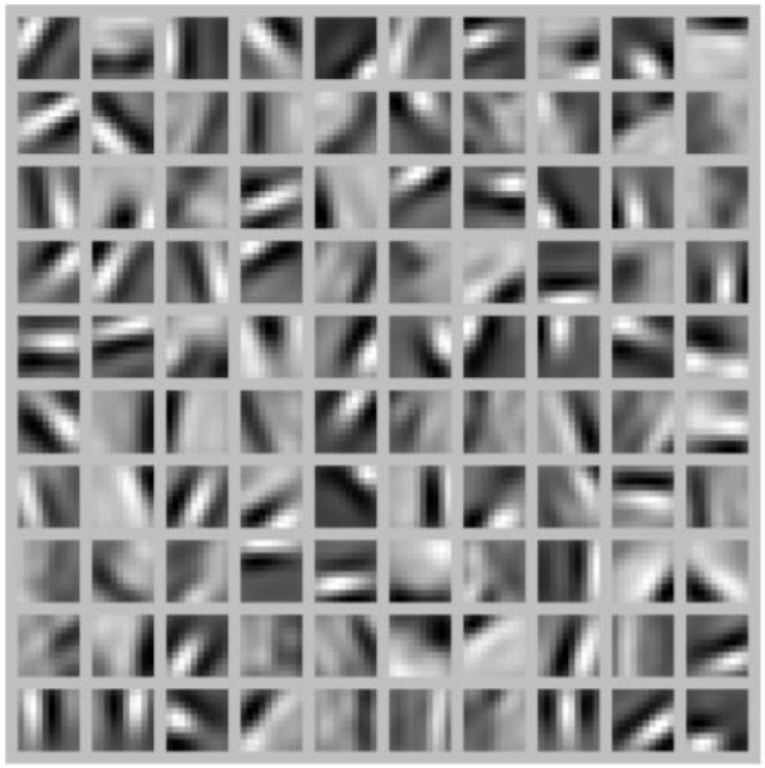
- More hidden nodes than input (and output)
- Why?
- When training an autoencoder, the hidden nodes in the middle layer would fire (activate) too frequently, for most training samples
- In sparse autoencoder, middle-layer nodes are activated infrequently so the nodes only activate for a small fraction of the training examples

Sparse autoencoder

- One recent sparse autoencoder is called k-sparse autoencoder
- Only k nodes with largest activation functions get weights updated during backprop
- With sparsity, conceptually a small portion of nodes would be highly selective to a specific type of feature

Sparse autoencoder

- Detected feature for a group of 100 hidden nodes
  (from https://web.stanford.edu/class/cs294a/sparseAutoencoder.pdf)



Variational autoencoder

- Unlike a classifier, Variational autoencoder is a powerful generative model
- Want to generate variations on available data, not just in a random way, but in a desired, specific direction
- This could also be done with de-nosing autoencoder
- So why need new type of autoeconder

Variational autoencoder

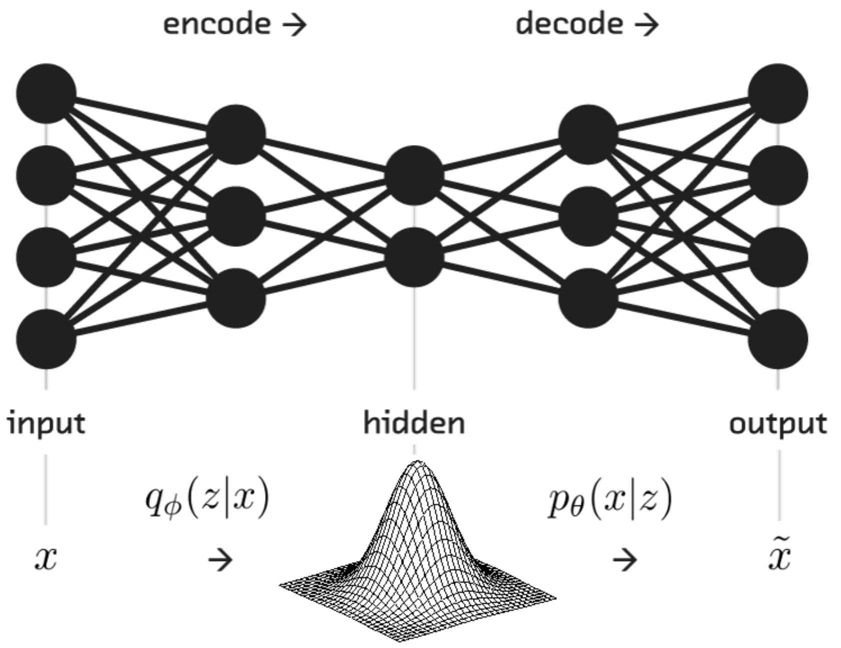
- The fundamental problem with autoencoders, for generation, is that the latent space they convert their inputs to and where their encoded vectors lie, may not be continuous, or allow **easy interpolation** --- from: https://towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf
- What if we insert a code in the discontinuous region
- Autoencoder produces nonrealistic images

Variational autoencoder

- VAE is designed to have continuous functions in code space
- We know Gaussian are continuous functions
- VAE uses mean and variance vectors as the codes

Variational autoencoder

- An illustration of VAE (source: http://blog.fastforwardlabs.com/2016/08/22/under-the-hood-of-the-variational-autoencoder-in.html)



Variational autoencoder

- Some detailed math about VAE is skipped (reference: https://wiseodd.github.io/techblog/2016/12/10/variational-autoencoder/)
- With some (terrible) math, we have the loss function:

$$\mathcal{L}(\phi, \theta; x) = \mathbb{E}_{z \sim q_\phi(z|x)}[log(p_\theta(x|z))] - \mathcal{D}_{KL}(q_\phi(z|x)||p_\theta(z))$$

- In plain English, loss = reconstruction loss - KL loss for each data in minibatch

Variational autoencoder

- For reconstruction loss, we can use cross-entropy if the image is binary
- Recall in this case, we have actual output and desired output for each pixel (remember desired output is input)
- The KL loss can be computed as (alternative source: https://stats.stackexchange.com/questions/7440/kl-divergence-between-two-univariate-gaussians)

$$\frac{1}{2}\sum_{j=1}^{n}\left(\mu_j^2 + \sigma_j^2 - \log(\sigma_j^2) - 1\right)$$

Variational autoencoder

- Recall that we have mean vector and variance vector as the internal representation (latent layer)
- KL is computed based on the values at those nodes
- The summation in previous slide is over all latent nodes
- In short, the loss function is to ensure
  - Low distortion between desired and actual outputs
  - Each latent component is as close to $N(0,1)$ as possible

Variational autoencoder

- Show off some examples (all synthetic)

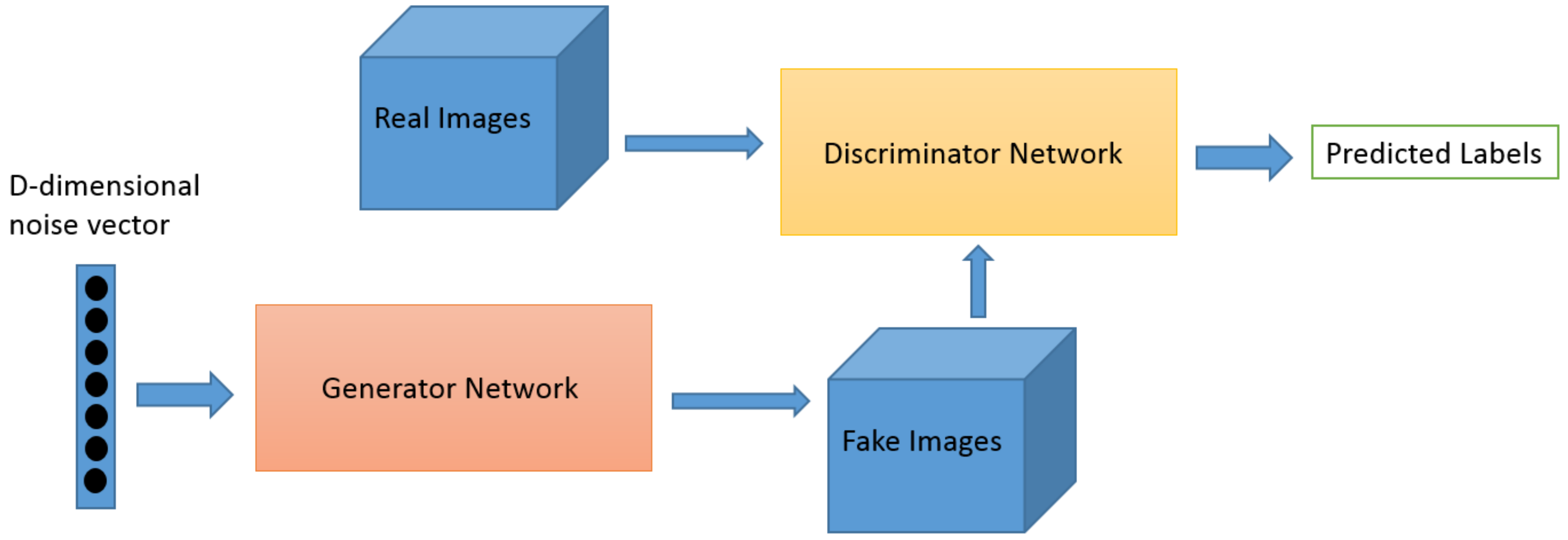
Source: https://github.com/yzwxx/vae-celebA

GAN

- GAN = Generative Adversarial Network
- According to Yann LeCun, “adversarial training is the coolest thing since sliced bread.” Sliced bread certainly never created this much excitement within the deep learning community. Generative adversarial networks—or GANs, for short—have dramatically sharpened the possibility of AI-generated content, and have drawn active research efforts since they were first described by Ian Goodfellow et al. in 2014.

GAN architecture

- Like VAE, GAN is also used for generation
- Picture from https://www.oreilly.com/learning/generative-adversarial-networks-for-beginners



GAN architecture

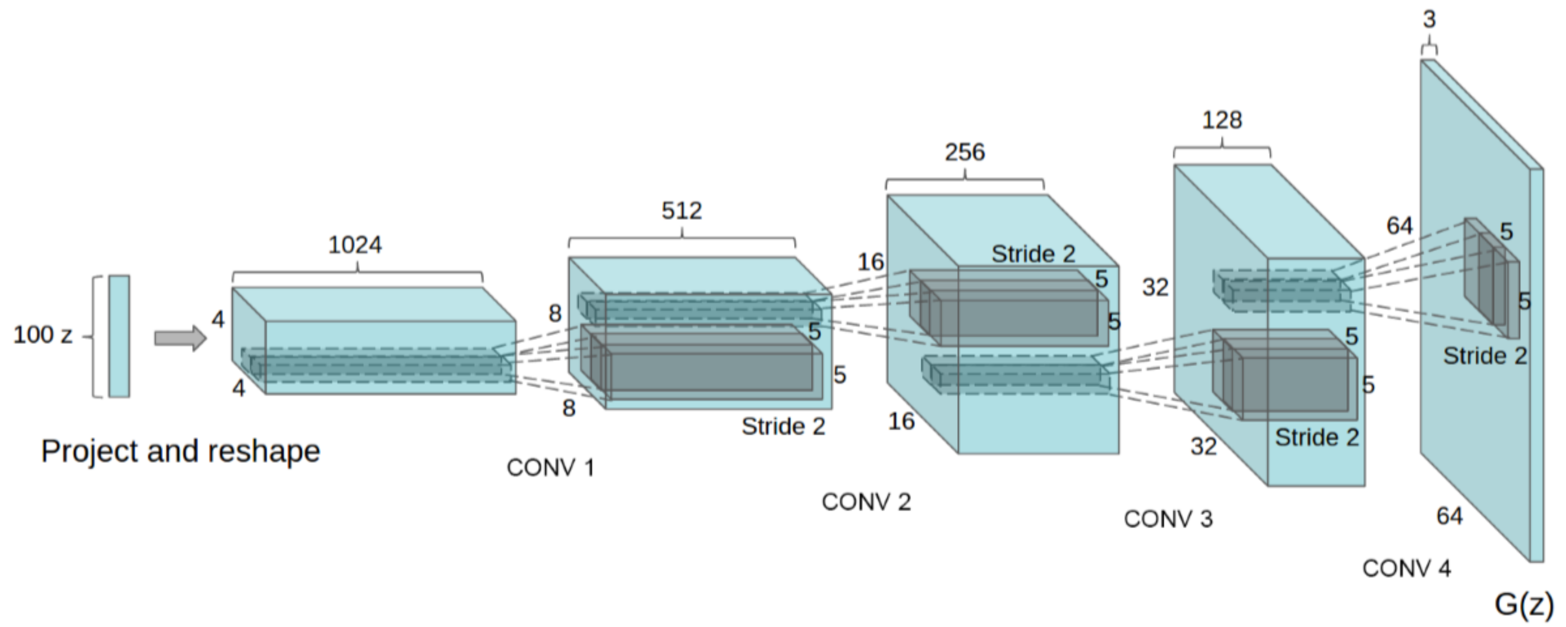
- Discriminator is a classifier that determines whether a given image is a real image or a fake (produced by generator) by using CNN (or DCNN)
- The generator takes random input values to produce (fake) images through a deconvolutional neural network
- Actually, the term “deconvolution” should be understood as a type of up-sampling because it is not really doing deconvolution

GAN architecture

- The weights of generator and discriminator are trained by backprop
- Note that there are many variations of GANs and many different theories talking about insights and loss functions of GANs
- One variation of GAN involves the use of autoencoder
- We are unable to cover too much, so we choose to introduce DCGAN (deep convolutional GAN)

GAN architecture

- Generator structure (from https://arxiv.org/abs/1511.06434)



Generator

- Input is a vector of 100 random numbers (usually uniformly distributed)
- 100 -> 4 through full connection
- Totally 1024 images with size of 4x4
- The size of the “image” grows from 4x4 to 8x8, to finally 64x64
- How could this be done? A kind of up-sampling (in the context of signal processing)
- Many people don’t like pooling so they use stride 2

Generator

- Several confusing terms
  - Transposed convolution with stride 2
  - subpixel convolution with stride (1/2)
  - Deconvolution
- We explain the concepts following the paper: https://arxiv.org/abs/1609.07009

Up-sampling

- 1-D convolution with stride = 2 (gray cell = 0)

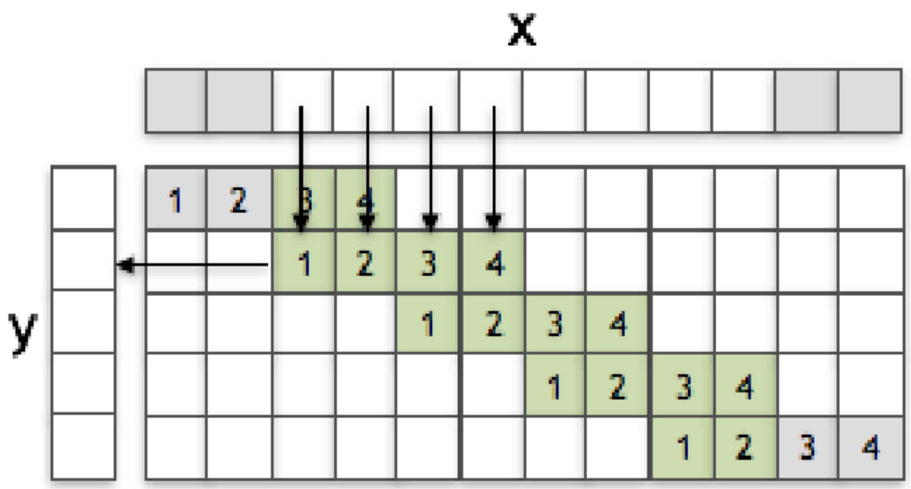
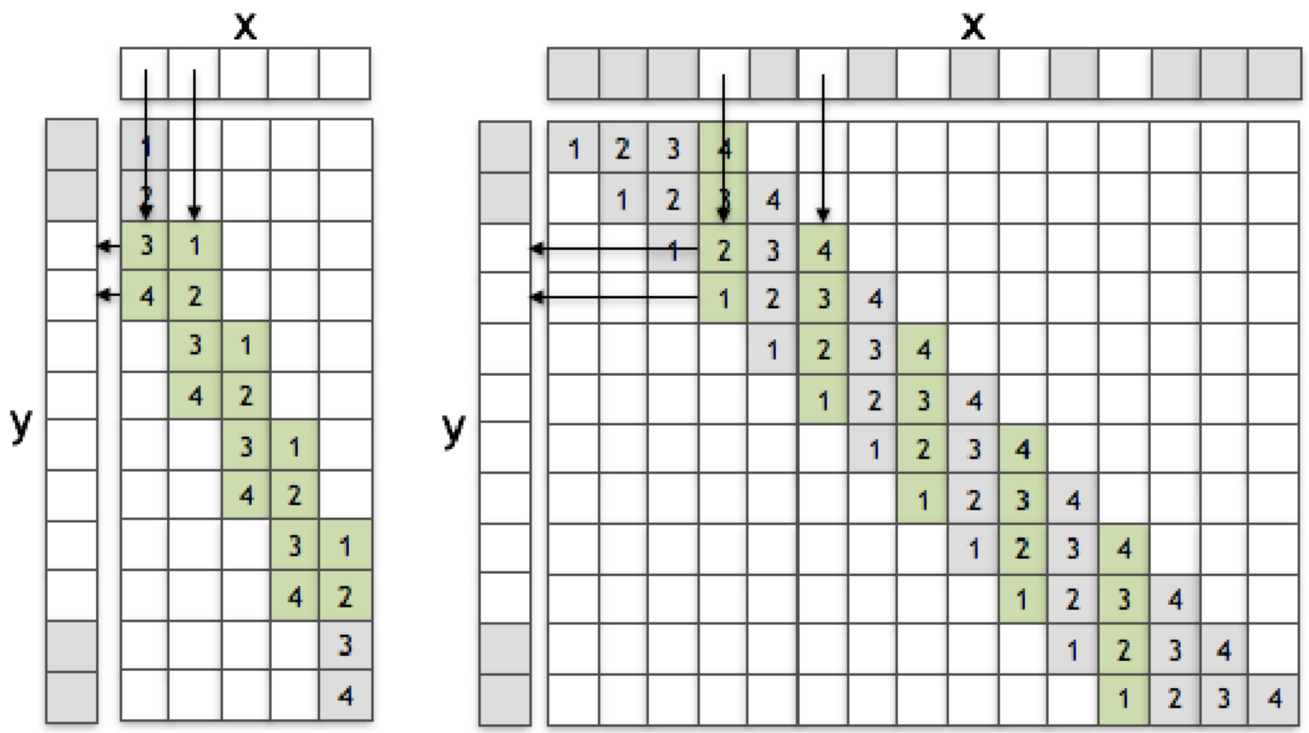


Figure 2: Convolution with stride 2 in 1D

Up-sampling

- Stride = 2 in (a) and stride = 0.5 in (b)



(a) (b)

Up-sampling

- Consider the output y(3) (index from 1,2,3, …)
  - (a) produces x(1) * w(3) + x(2) * w(1)
  - (b) produces x(1) * w(2) + x(2) * w(4)
- It turns out if we do change of variables, both can produce same output sequence (equivalent)

Up-sampling

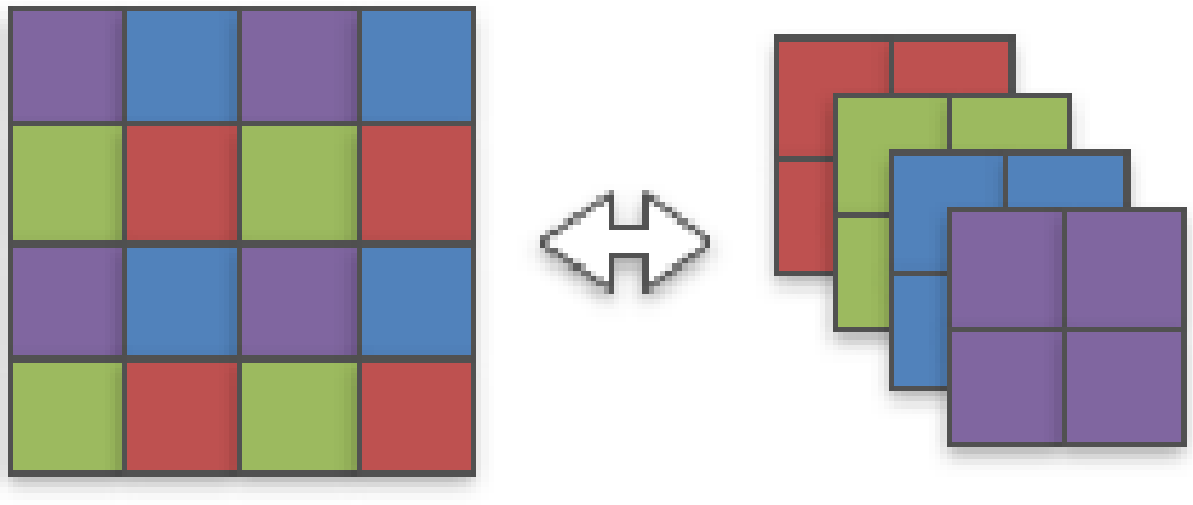
- While sub-sampling with stride (1/2) is easy to understand, its implementation is not efficient (lots of multiplication with zero)
- In signal processing, we can do efficient up-sampling via polyphase decomposition
- We can use the same trick here

Up-sampling

- Steps:
  - Decompose the kernel (window)
  - Perform convolution on each subkernel
  - Reconstruct the image by interleaving

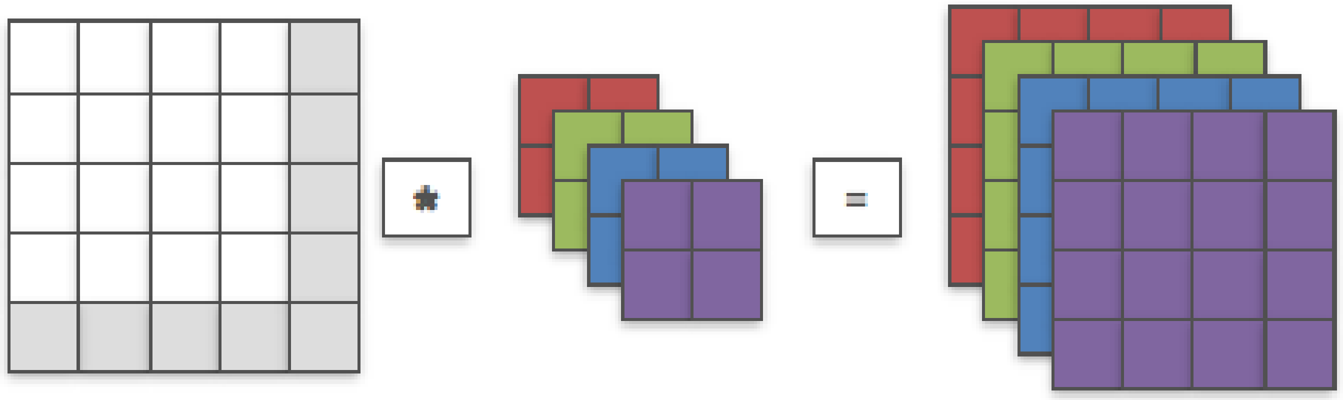
Up-sampling

- Illustration of step 1 (notice the down-sampling)



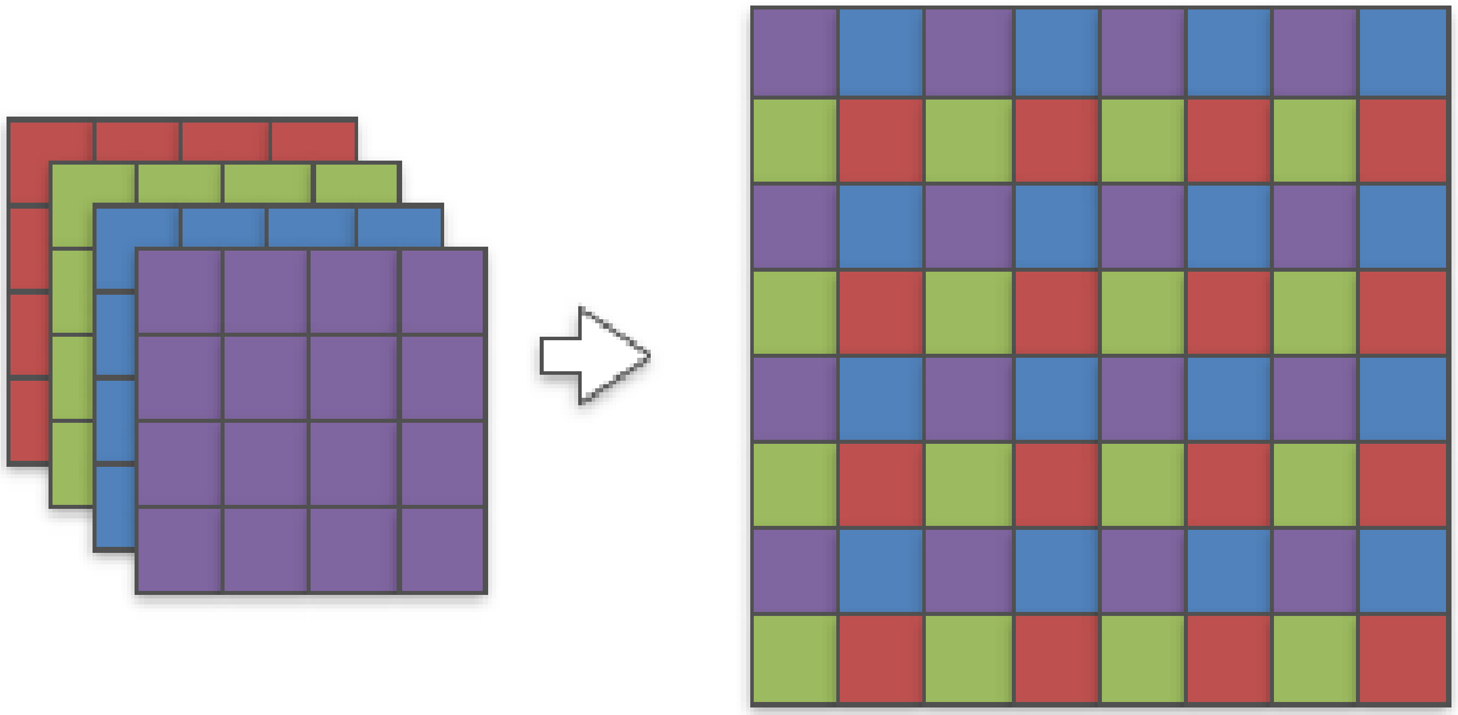
Up-sampling

- Step 2: convolution



Up-sampling

- Step 3: Interleaving (notice positions of each color are same as down-sampling)

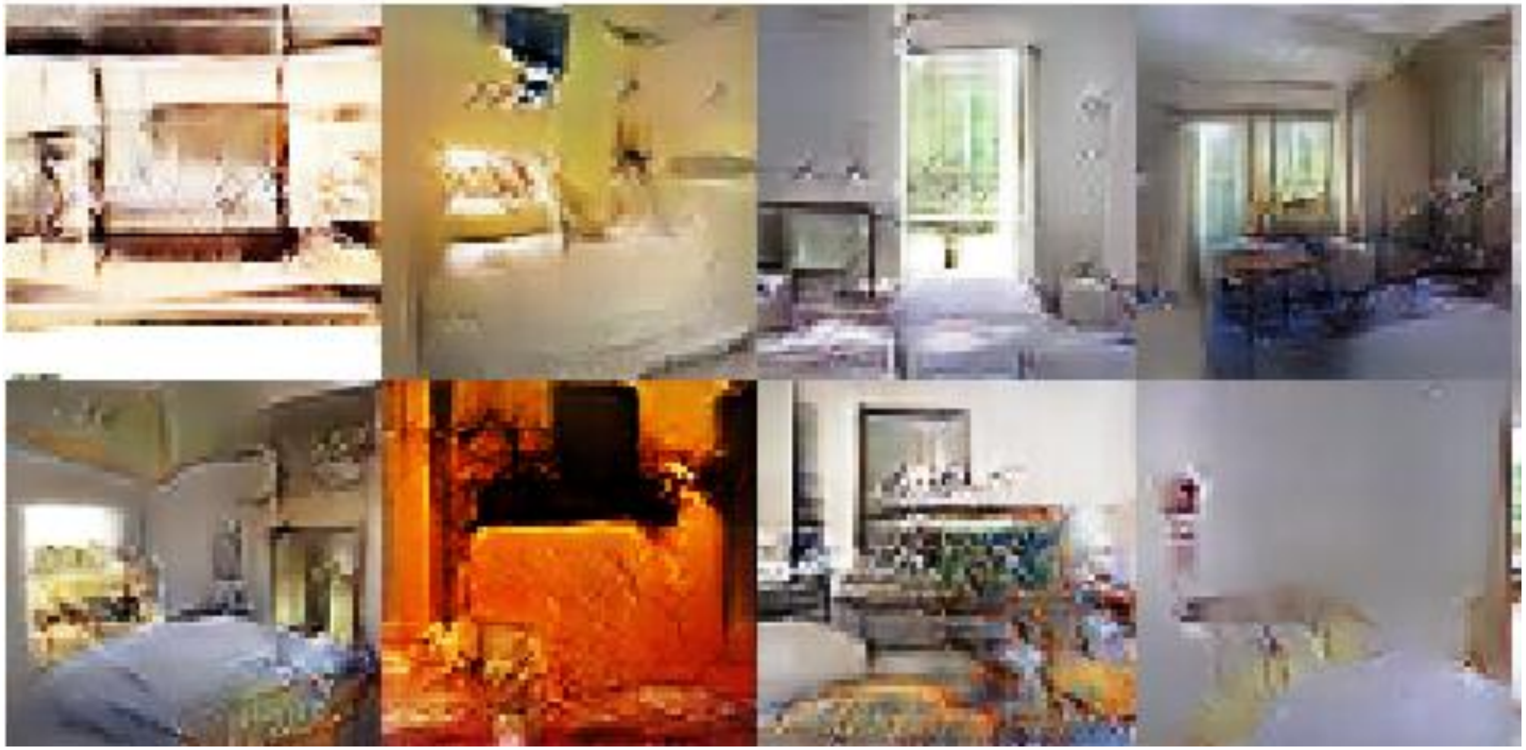


Discriminator network

- How about discrimination network
- One simple model is using the reverse order of generator network
- As this part is not difficult to understand, we omit the details

Generated pictures

- Generated bedrooms from the paper



Training GAN

- Since GAN has two networks, how to train it
- Conceptual training procedure
  - Set the discriminator trainable
  - Train the discriminator with the real images and fake images (from generator)
  - Set the discriminator non-trainable
  - Train generator with latent (random) samples as input to produce an image with high discriminator score (like a real one)

Loss function in discriminator

- Source: https://danieltakeshi.github.io/2017/03/05/understanding-generative-adversarial-networks/
- Starting point: cross entropy
$$H = -(d \log_2 y + (1-d) \log_2 (1-y))$$
- To be similar to existing literature, we let desired output be y, the input be x, and the discriminator output be D(x). Thus,
$$H((x_1, y_1), D) = \\ -(y_1 \log_2 D(x_1) + (1-y_1) \log_2 \left(1 - D(x_1)\right))$$

Loss function in discriminator

- Summing over many elements, we have

$$H\left((x_1, y_1)_{i=1,\ldots,N}, D\right)$$
$$= -\sum_{i=1}^{N} y_i \log_2 D(x_i) - \sum_{i=1}^{N} (1 - y_i) \log_2\left(1 - D(x_i)\right)$$

- We know x_i is either from real images or fake images with $x_i = G(z_i)$ (z is a uniform/ Gaussian RV used to generate random input vectors
- We also assume real images have some type of distribution

Loss function in discriminator

- Summation operation is replaced with expectation operation if infinitely many samples presented
- As $P(y_i = 1) = 0.5$ and $P(y_i = 0) = 0.5$
- Finally, we have

$$H((x_i, y_i)_{i=1}^{\infty}, D) = -\frac{1}{2}\mathbb{E}_{x\sim p_{\text{data}}}\Big[\log D(x)\Big] - \frac{1}{2}\mathbb{E}_z\Big[\log(1 - D(G(z)))\Big]$$

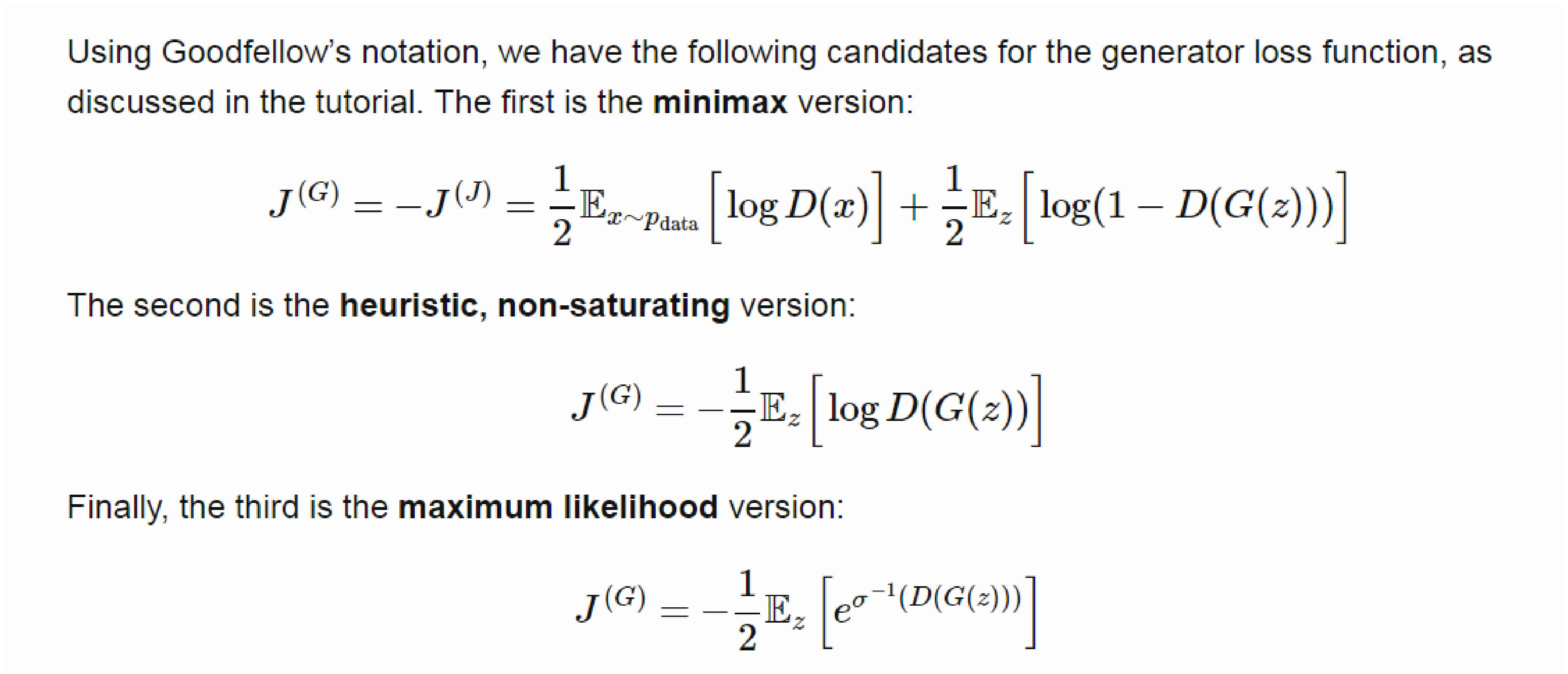
- In the equation $E_z[\cdot]$ denotes taking the expected value with respect to the RV z

Loss function in discriminator

- The above equation is the discriminator loss (cost) function given in GAN papers, denoted as $J^{(J)}$
- In terms of implementation, certainly we use summation (or even one sample in the SGD case) because we do not know the distribution of the sample dataset

Loss function in generator

- We could have some alternatives in defining the loss function of generator



Using Goodfellow's notation, we have the following candidates for the generator loss function, as discussed in the tutorial. The first is the **minimax** version:

$$J^{(G)} = -J^{(J)} = \frac{1}{2}\mathbb{E}_{x\sim p_{\text{data}}}\Big[\log D(x)\Big] + \frac{1}{2}\mathbb{E}_z\Big[\log(1 - D(G(z)))\Big]$$

The second is the **heuristic, non-saturating** version:

$$J^{(G)} = -\frac{1}{2}\mathbb{E}_z\Big[\log D(G(z))\Big]$$

Finally, the third is the **maximum likelihood** version:

$$J^{(G)} = -\frac{1}{2}\mathbb{E}_z\Big[e^{\sigma^{-1}(D(G(z)))}\Big]$$

Loss function in generator

- The minimax problem is studied in "game theory"
- Therefore, people sometimes talk about **Nash equilibrium**
- You can check Internet to know who John Nash is
- His life was the theme of a movie (A beautiful mind)

Training GAN

- So far so good, but is GAN easy to use
- Absolutely NOT! GAN is known for hard to train
- Some problems are
  - Hard to achieve Nash equilibrium
  - Vanishing gradient
  - Mode collapse (producing same output regardless the input vector)
  - Lack of a proper evaluation metric to indicate training progress

Further reading

- Because we don't have enough time, I could not cover more about GAN (also too much theoretical)
- If you are interested in GAN, you can read
  - Tutorial: https://arxiv.org/abs/1701.00160
  - Some tips on training GAN: https://github.com/soumith/ganhacks
  - More theoretical stuff: https://lilianweng.github.io/lil-log/2017/08/20/from-GAN-to-WGAN.html