TOPICS IN DEEP LEARNING

Unit-4
Generative Adversarial Networks

Neural Network and Deep Learning

Introduction

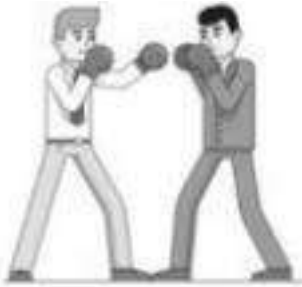
A generative adversarial network (GAN) is a class of machine learning frameworks designed by **Ian Goodfellow and his colleagues** in June 2014.

Neural Network and Deep Learning
What is GAN?

Generative

Generates data
(Creates fake data)

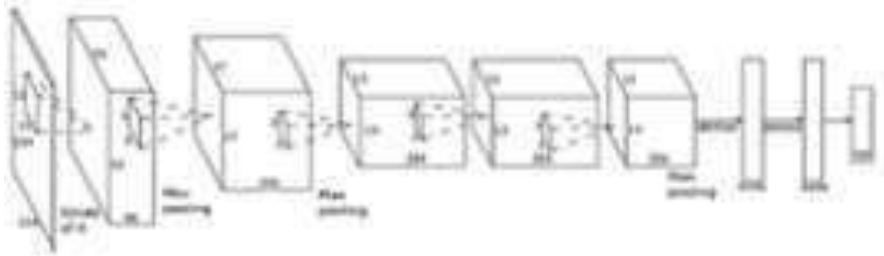
Adversarial



Generator and **discriminator**,
each competing to win.

Generator trying to fake
and
Discriminator, trying not to
be fooled.

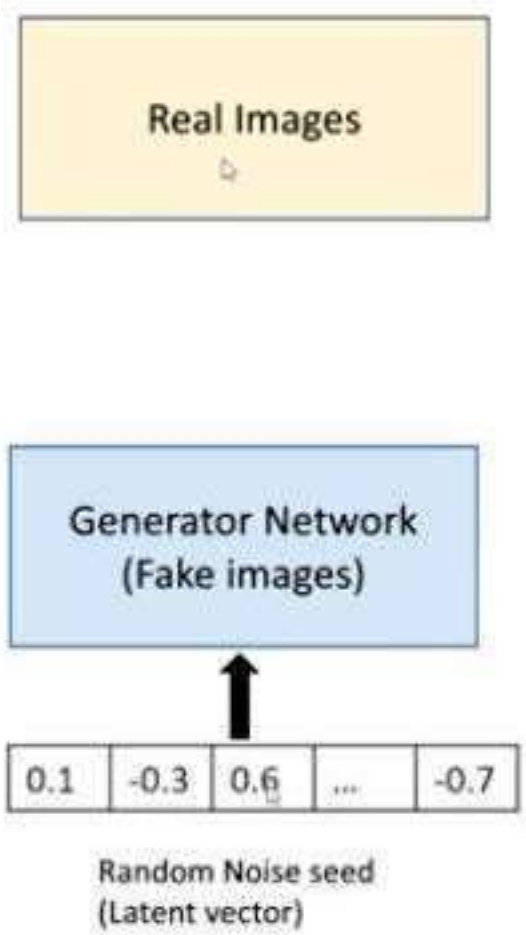
Networks



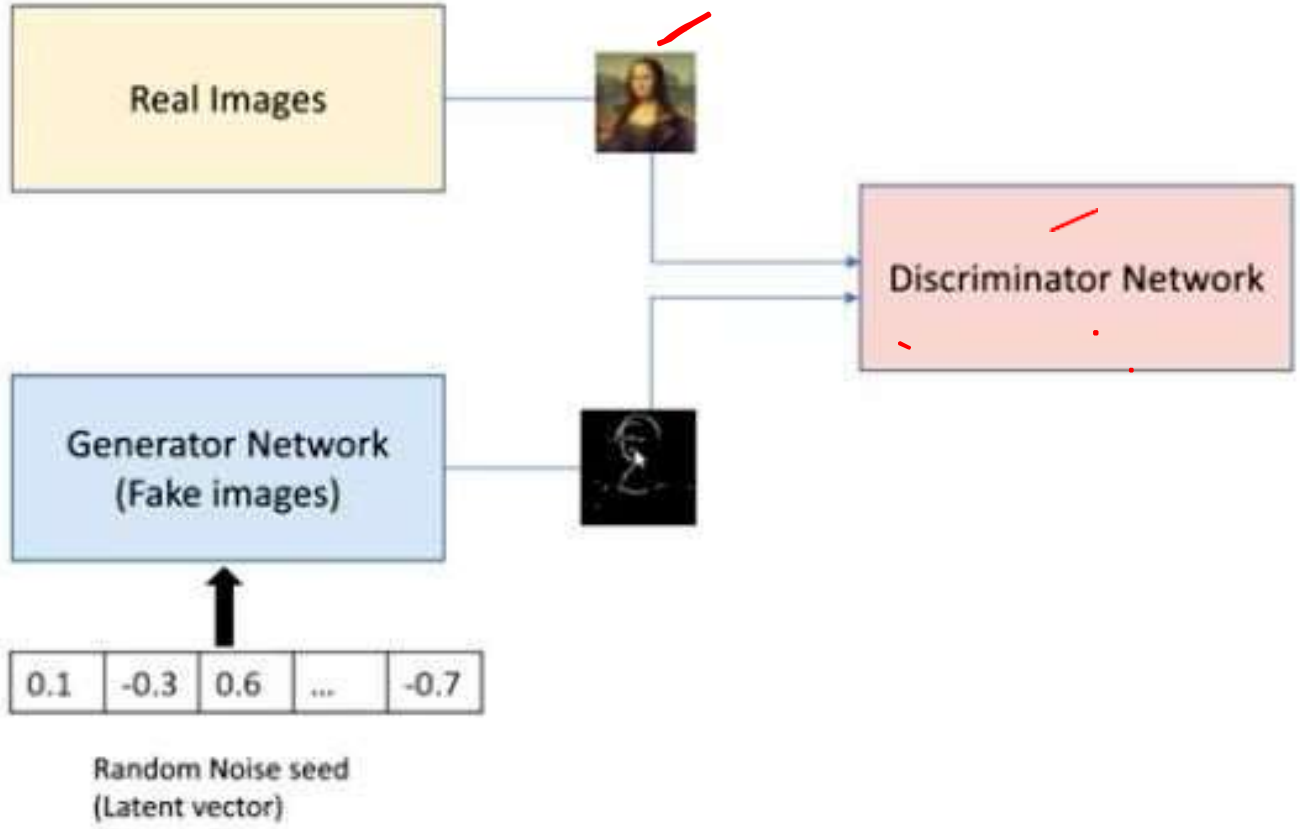
Deep Convolutional

Or fully connected netwrok

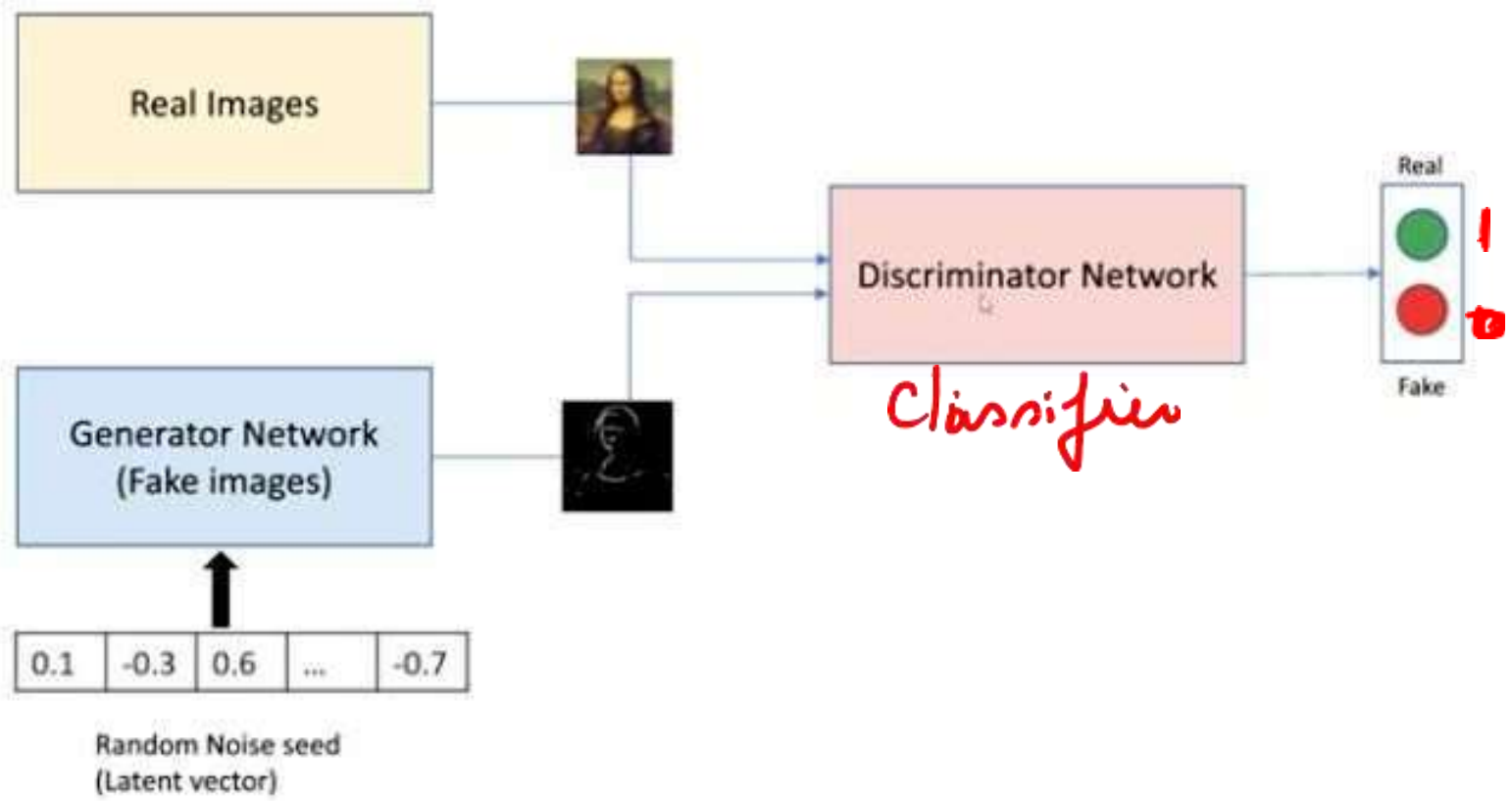
Neural Network and Deep Learning
What is GAN?



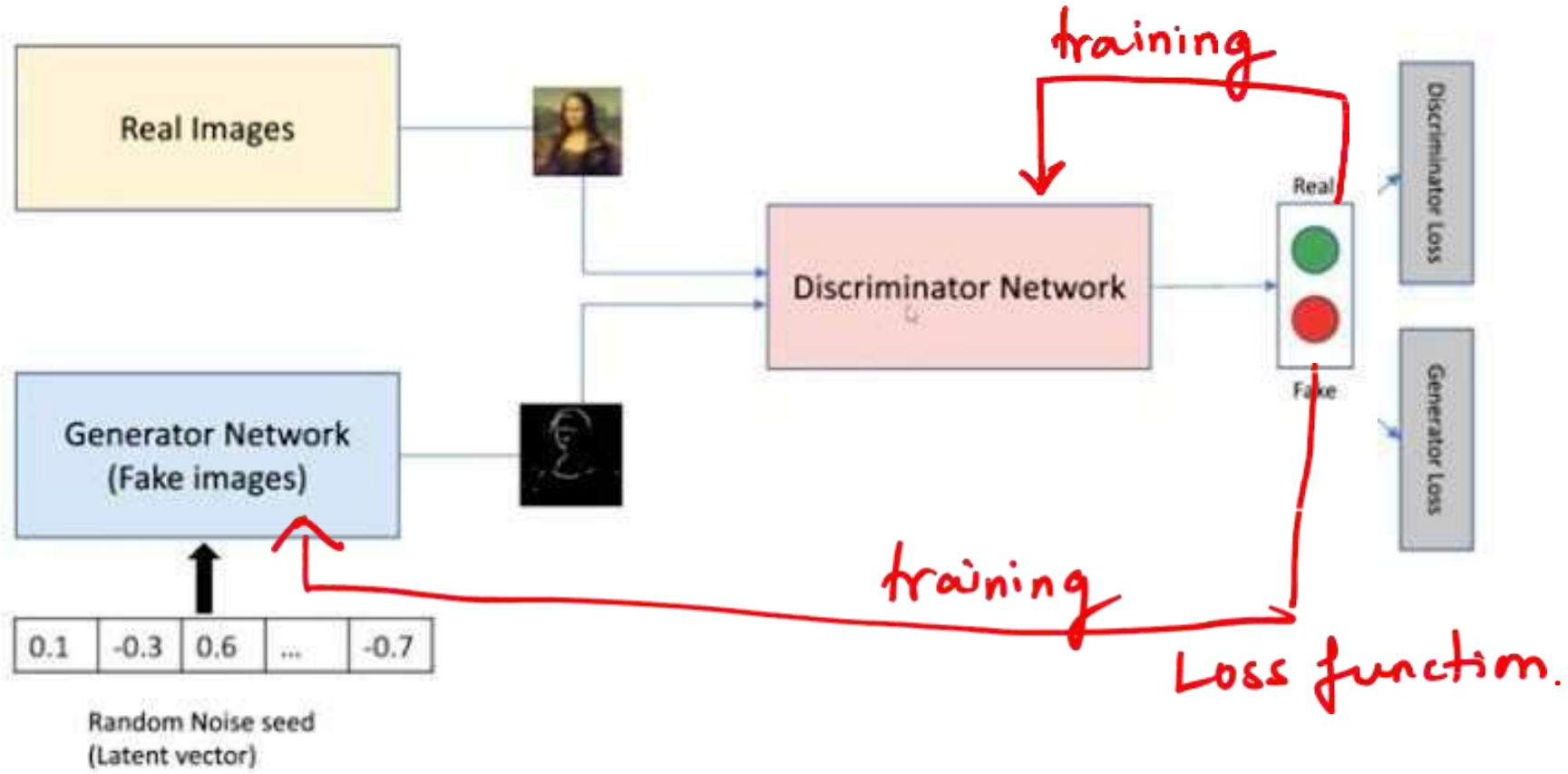
Neural Network and Deep Learning
What is GAN?



Neural Network and Deep Learning
What is GAN?



Neural Network and Deep Learning
What is GAN?



Neural Network and Deep Learning
5 Steps of GAN

1. Define GAN architecture based on the application
2. Train discriminator to distinguish real Vs fake data
3. Train the generator to fake data that can fool the discriminator
4. Continue discriminator and generator training for multiple epochs
5. Save generator model to create new, realistic data.

Note : When training the discriminator, hold the generator values constant; when training the generator, hold the discriminator constant.
Each should train against a static adversary.

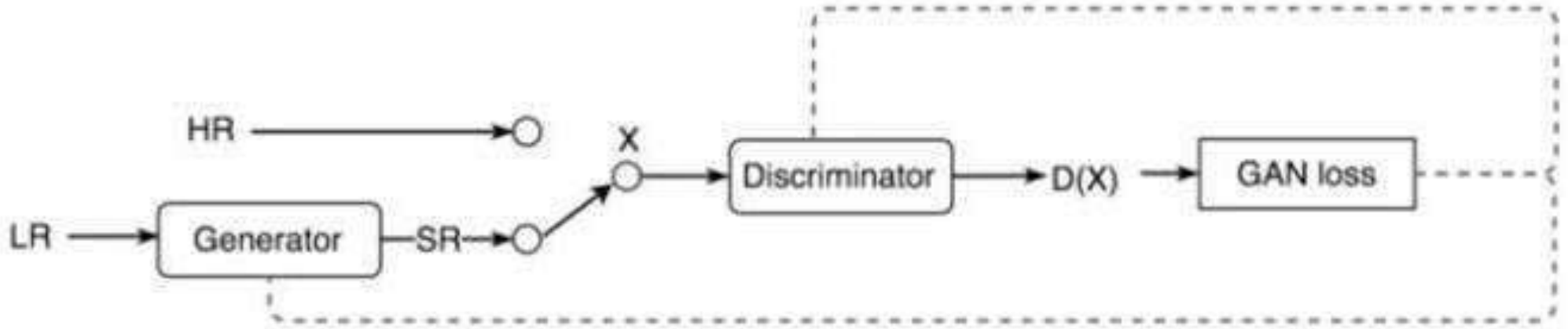
Neural Network and Deep Learning

Applications of GAN

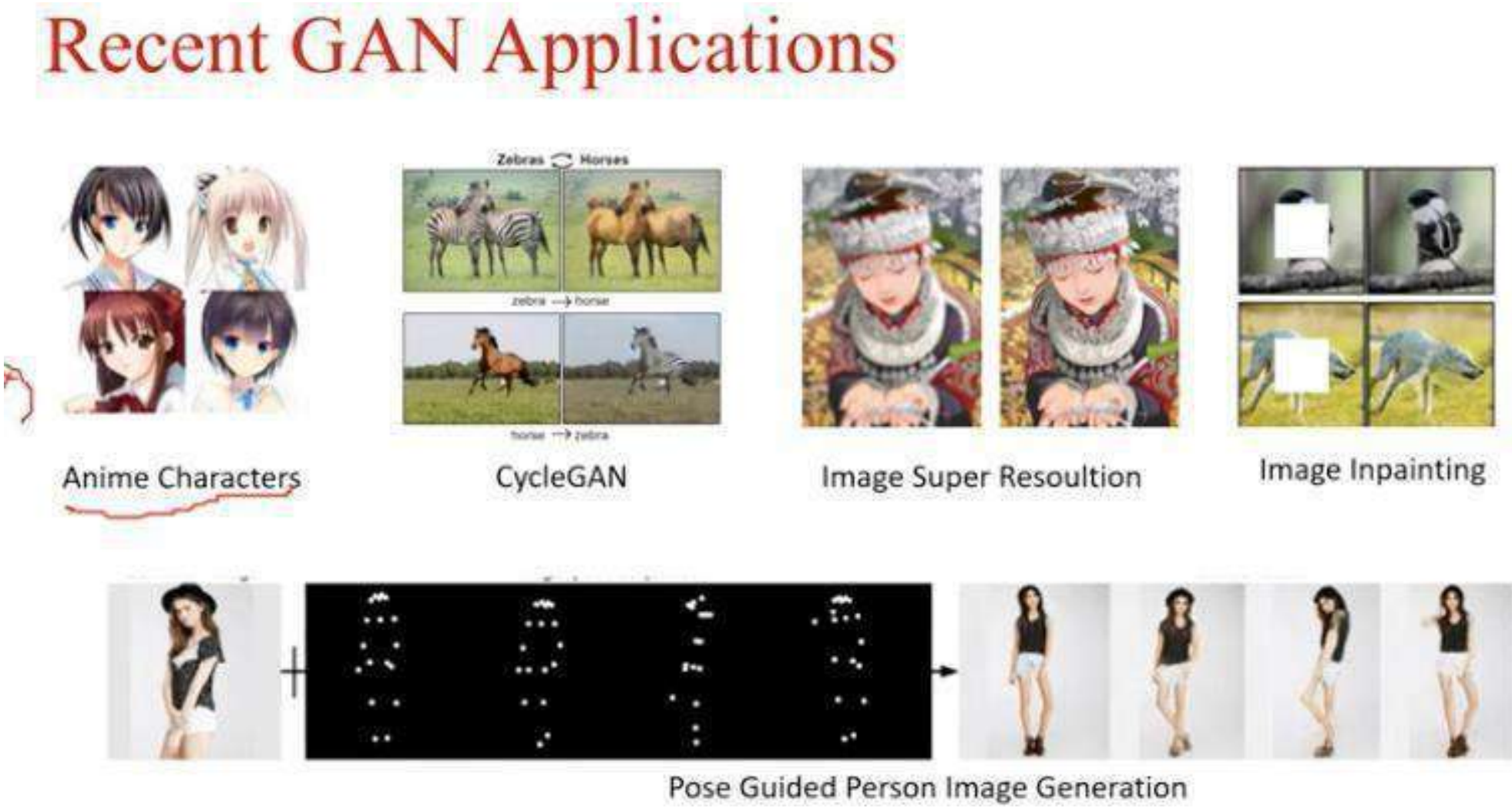
1. Generate fake data for augumenting other machine learning algorithms
2. Generate faces (https://this-person-does-not-exist.com/en)
3. Image to image translation
4. Text to image translation
5. Super resolution

GANs for Image Super-resolution: SRGAN

- Generator: gets low-resolution image as input and super-resolution image as output
- Discriminator: aims to differentiate between high-resolution image vs super-resolution image



Neural Network and Deep Learning
Applications of GAN



Neural Network and Deep Learning
Applications of GAN

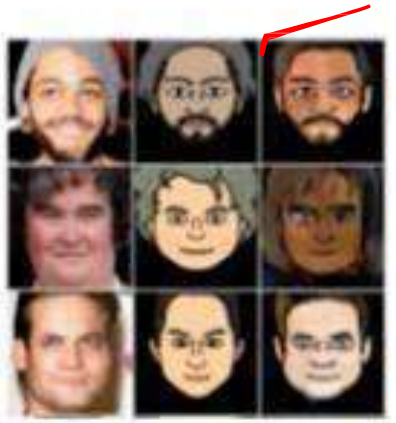
Recent GAN Applications

Fake Celebrities

Text to Image

Pix2Pix

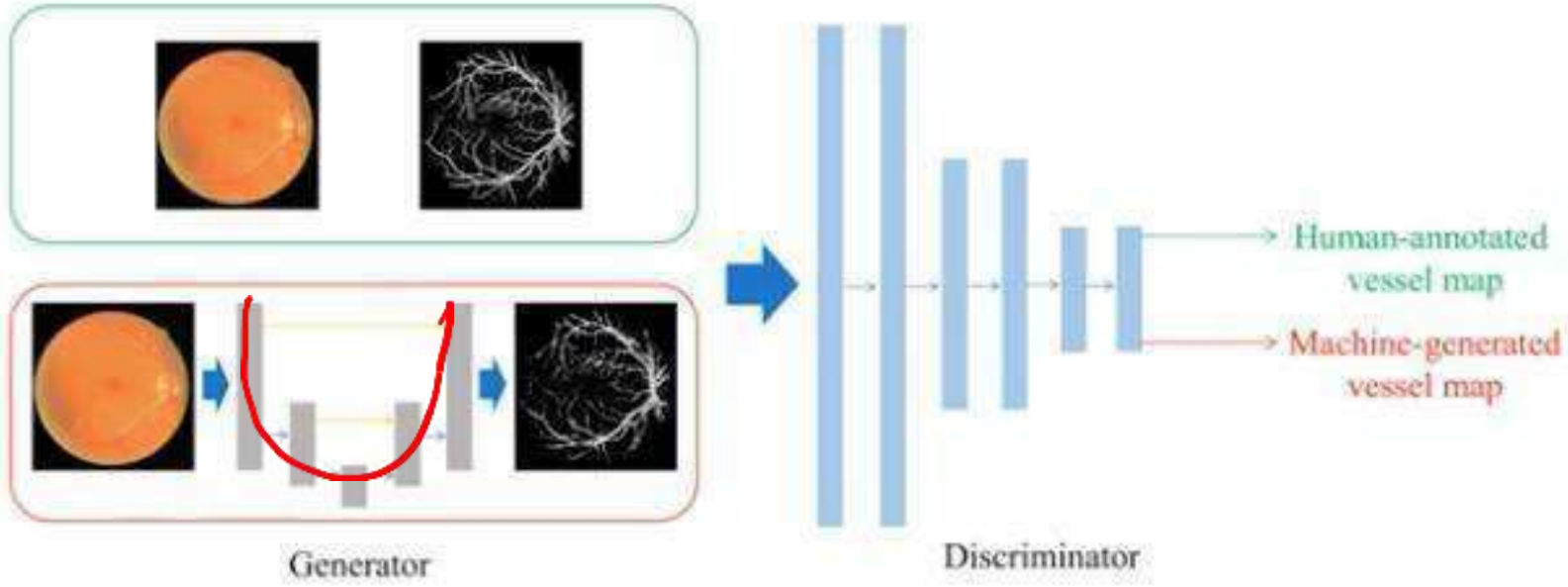
Next Frame Prediction



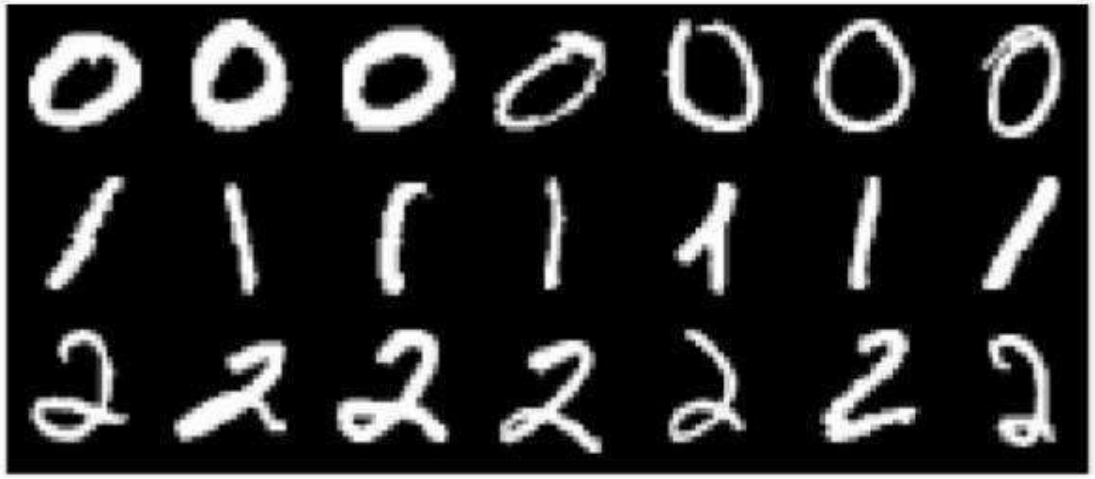
Emoji Generator

GANs for Image Segmentation

- Recently GANs are highly applied for medical image segmentation
- Generator is a segmentation network where discriminator will predict whether it is real segmentation or generated from segmentation

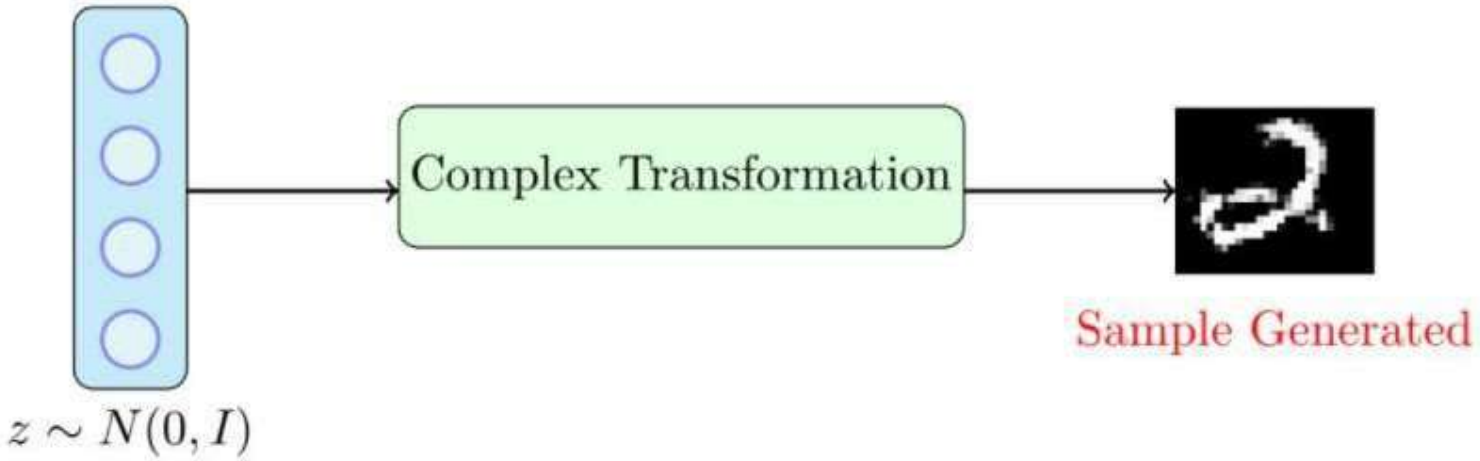


Neural Network and Deep Learning
Introduction



- As usual we are given some training data (say, MNIST images) which obviously comes from some underlying distribution
- Our goal is to generate more images from this distribution (*i.e.*, create images which look similar to the images from the training data)
- In other words, we want to sample from a complex high dimensional distribution which is intractable (recall RBMs, VAEs and AR models deal with this intractability in their own way)

Neural Network and Deep Learning
Introduction(GAN completely by passes the previous slide)



- GANs take a different approach to this problem where the idea is to sample from a simple tractable distribution (say, $z \sim N(0, I)$) and then learn a complex transformation from this to the training distribution
- In other words, we will take a $z \sim N(0, I)$, learn to make a series of complex transformations on it so that the output looks as if it came from our training distribution

Neural Network and Deep Learning

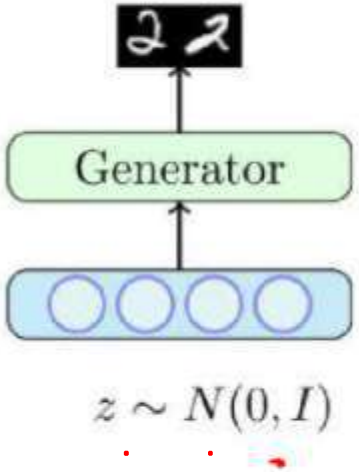
Introduction

- What can we use for such a complex transformation? A Neural Network
- How do you train such a neural network? Using a two player game

Neural Network and Deep Learning

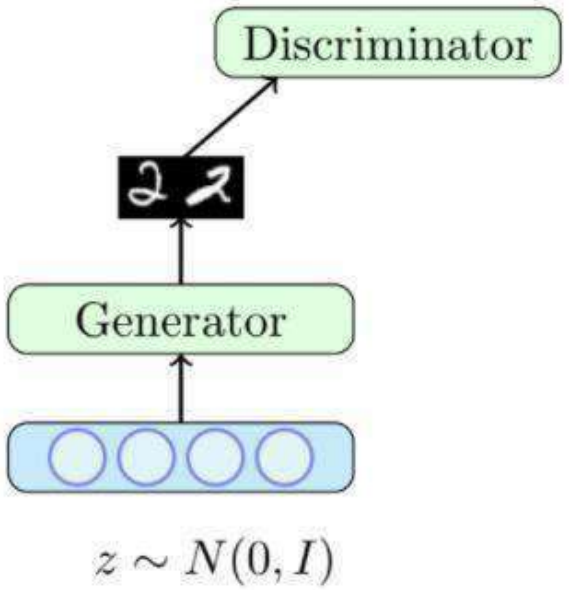
Generative Adversarial Network

- What can we use for such a complex transformation? A Neural Network
- How do you train such a neural network? Using a two player game
- There are two players in the game: a generator



Neural Network and Deep Learning

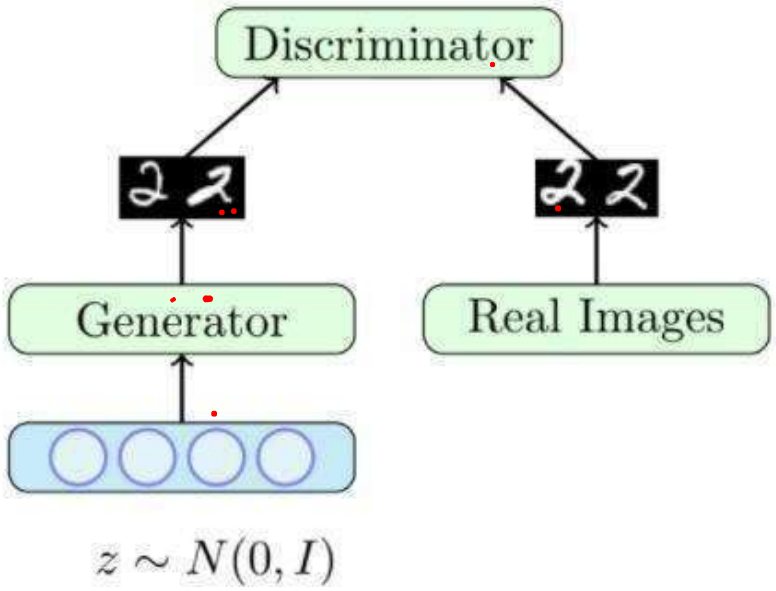
Generative Adversarial Network



- What can we use for such a complex transformation? A Neural Network
- How do you train such a neural network? Using a two player game
- There are two players in the game: a generator and a discriminator

Neural Network and Deep Learning

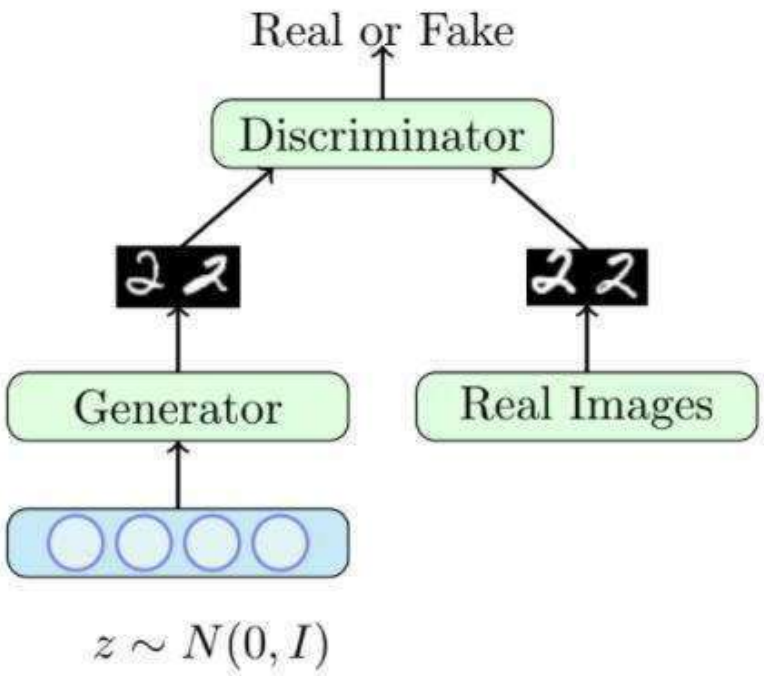
Generative Adversarial Network



- What can we use for such a complex transformation? A Neural Network
- How do you train such a neural network? Using a two player game
- There are two players in the game: a generator and a discriminator
- The job of the generator is to produce images which look so natural that the discriminator thinks that the images came from the real data distribution

Neural Network and Deep Learning

Generative Adversarial Network

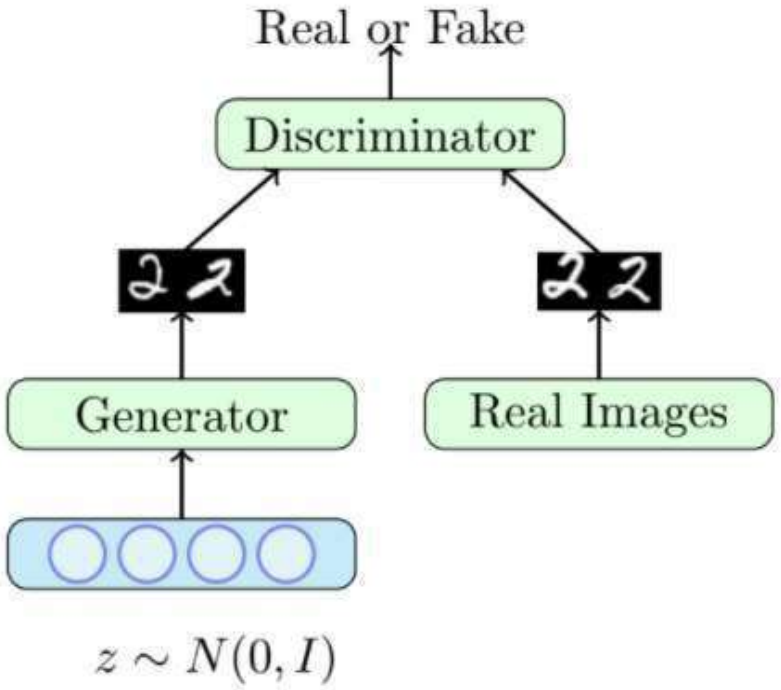


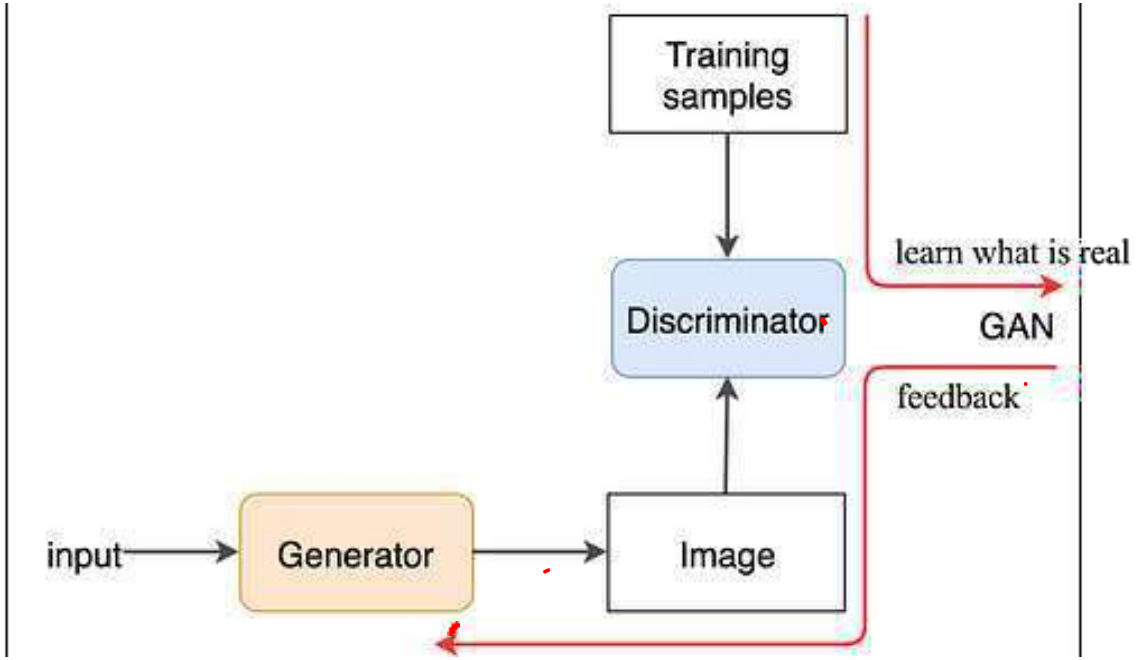
- What can we use for such a complex transformation? A Neural Network
- How do you train such a neural network? Using a two player game
- There are two players in the game: a generator and a discriminator
- The job of the generator is to produce images which look so natural that the discriminator thinks that the images came from the real data distribution
- The job of the discriminator is to get better and better at distinguishing between true images and generated (fake) images

Neural Network and Deep Learning

How to learn this?

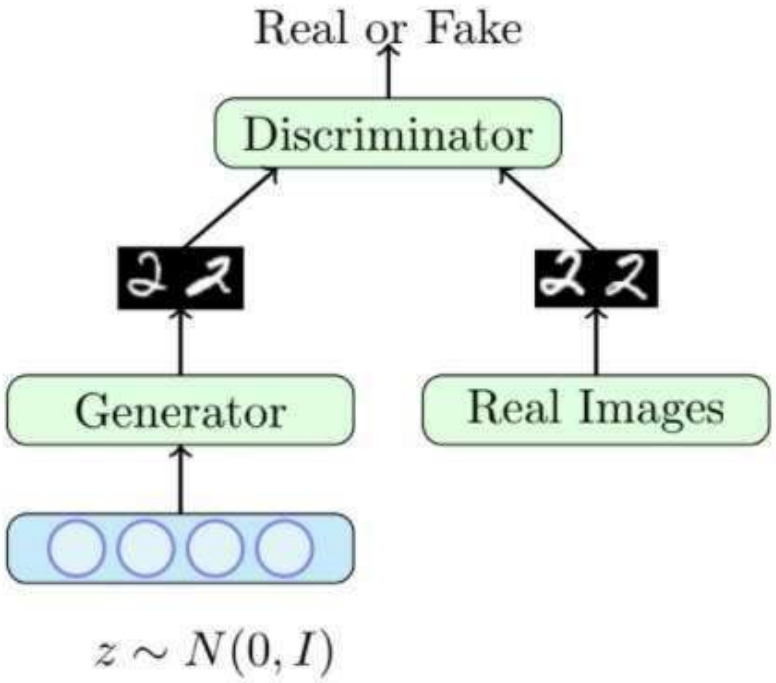
- So let's look at the full picture





Neural Network and Deep Learning

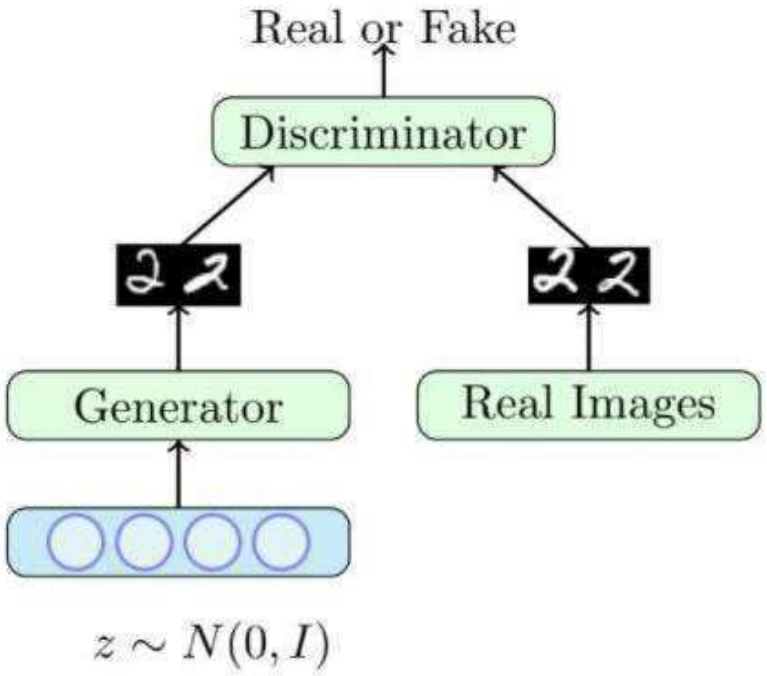
Generator and Discriminator



- So let's look at the full picture
- Let G_ϕ be the generator and D_θ be the discriminator (ϕ and θ are the parameters of G and D, respectively)

Neural Network and Deep Learning

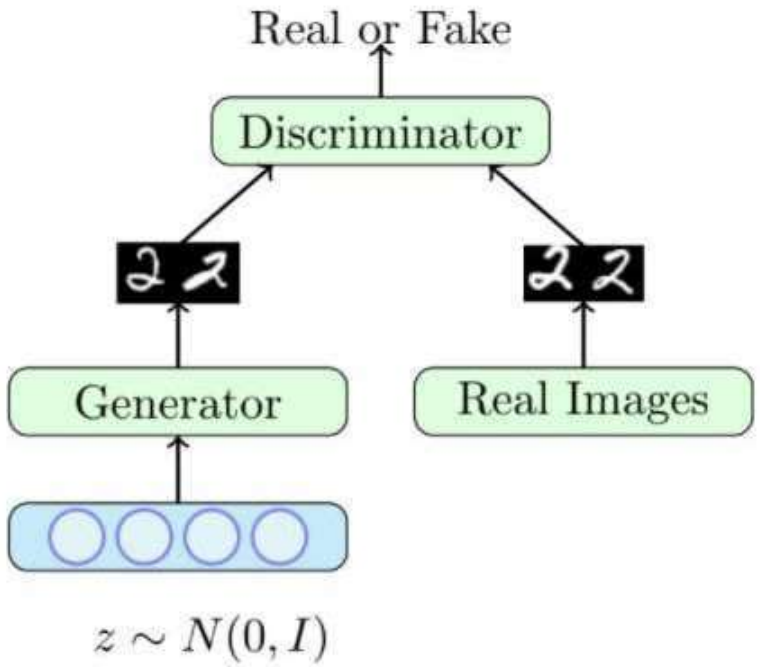
Generative Adversarial Network



- So let's look at the full picture
- Let G_ϕ be the generator and D_θ be the discriminator (ϕ and θ are the parameters of G and D, respectively)
- We have a neural network based generator which takes as input a noise vector $z \sim N(0, I)$ and produces $G_\phi(z) = X$

Neural Network and Deep Learning

Generative Adversarial Network

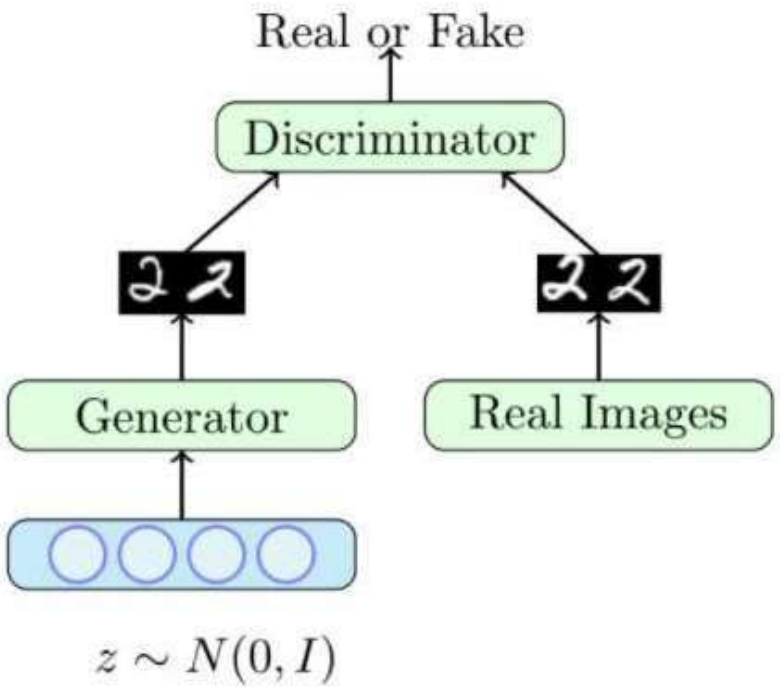


- So let's look at the full picture
- Let G_ϕ be the generator and D_θ be the discriminator (ϕ and θ are the parameters of G and D, respectively)
- We have a neural network based generator which takes as input a noise vector $z \sim N(0, I)$ and produces $G_\phi(z) = X$
- We have a neural network based discriminator which could take as input a real X or a generated $X = G_\phi(z)$ and classify the input as real/fake

Neural Network and Deep Learning

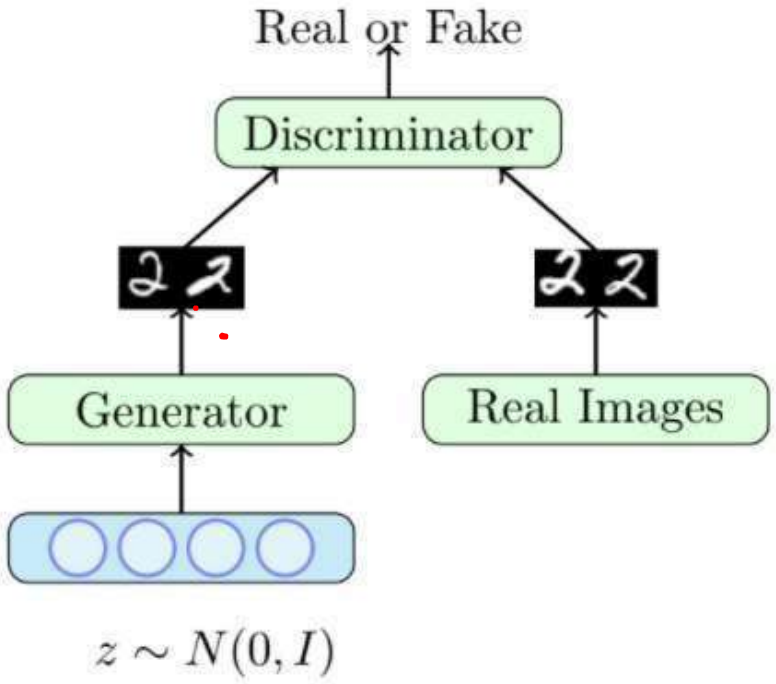
GAN- Objective function

- What should be the objective function of the overall network?



Neural Network and Deep Learning

GAN- Objective function

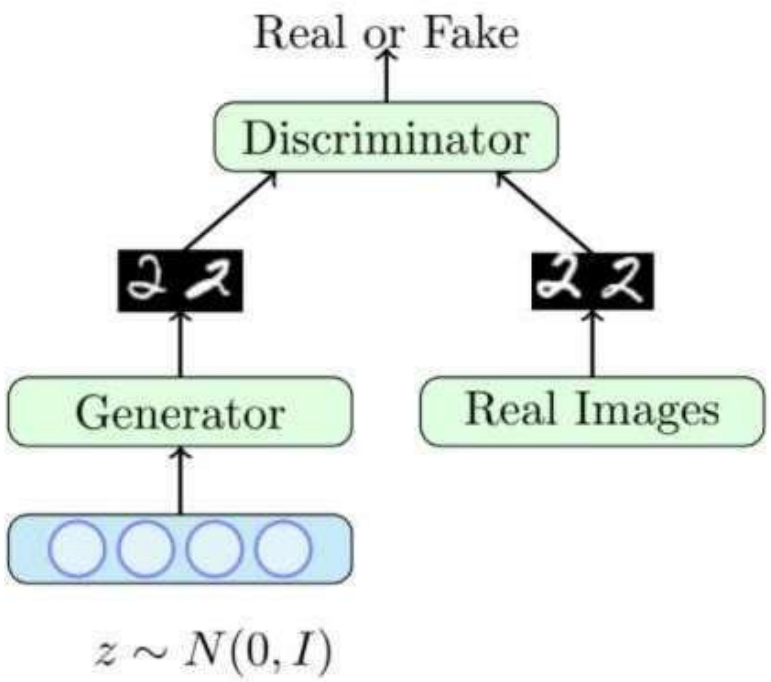


- What should be the objective function of the overall network?
- Let's look at the objective function of the generator first

- The discriminator outputs a value ***D(x)*** indicating the chance that *x* is a real image.
- Our objective is to maximize the chance to recognize real images as real and generated images as fake. i.e. the maximum likelihood of the observed data.
- To measure the loss, we use cross-entropy as in most Deep Learning: *p log(q).*
- For real image, *p* (the true label for real images) equals to 1. For generated images, we reverse the label (i.e. one minus label*)*. So the objective becomes:

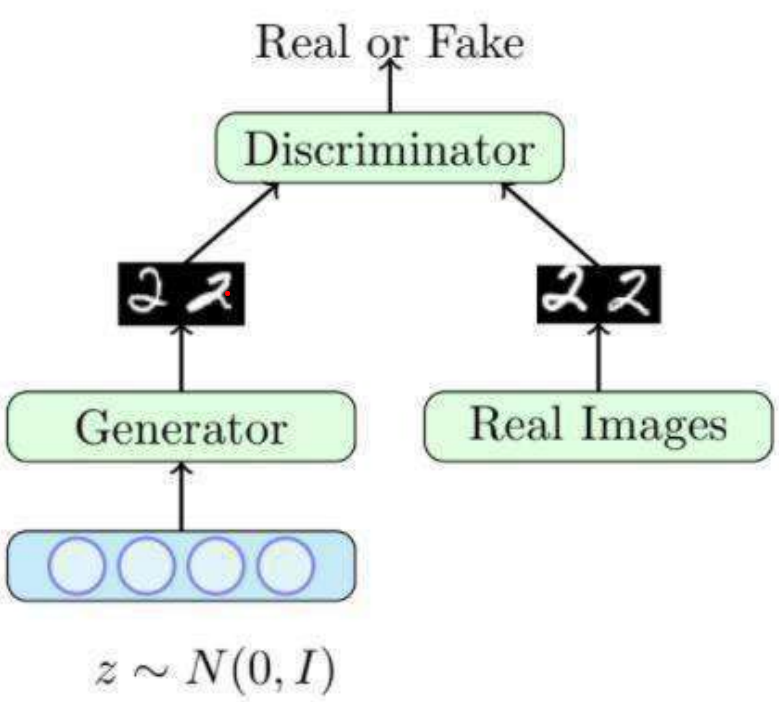
Neural Network and Deep Learning

Objective of Generator



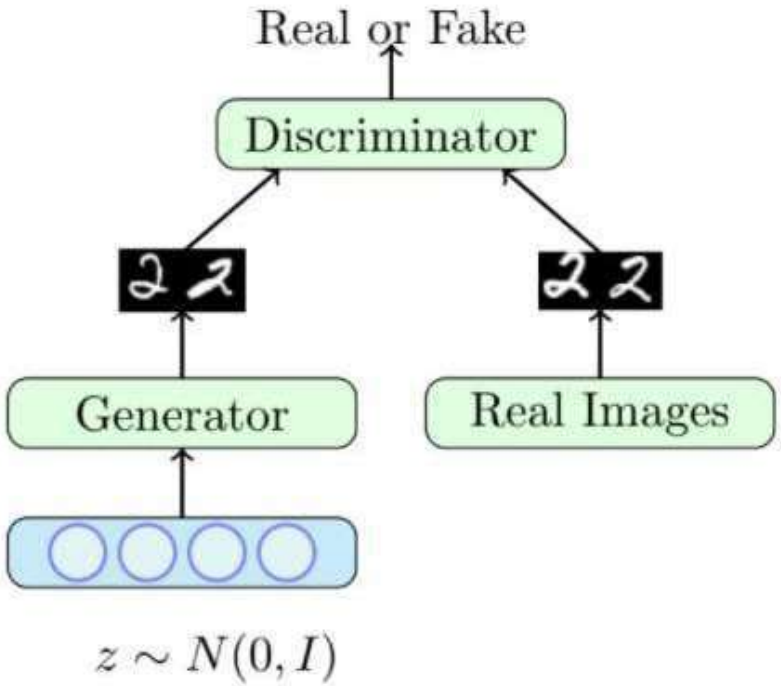
- What should be the objective function of the overall network?
- Let's look at the objective function of the generator first
- Given an image generated by the generator as $G_\phi(z)$ the discriminator assigns a score $D_\theta(G_\phi(z))$ to it
- This score will be between 0 and 1 and will tell us the probability of the image being real or fake

Objective of Generator



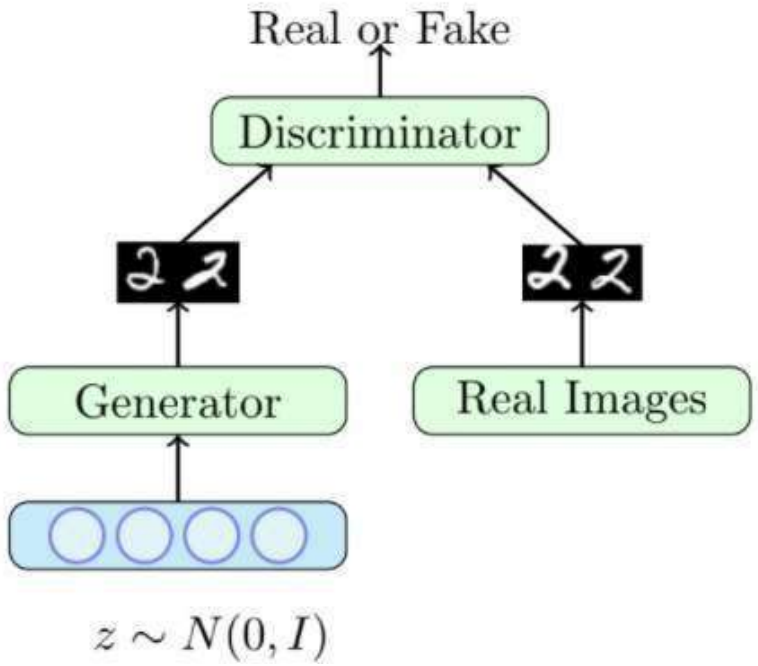
- What should be the objective function of the overall network?
- Let's look at the objective function of the generator first
- Given an image generated by the generator as $G_\phi(z)$ the discriminator assigns a score $D_\theta(G_\phi(z))$ to it
- This score will be between 0 and 1 and will tell us the probability of the image being real or fake
- For a given z, the generator would want to maximize $\log D_\theta(G_\phi(z))$ (log likelihood) or minimize $\log(1 - D_\theta(G_\phi(z)))$

Objective of Generator



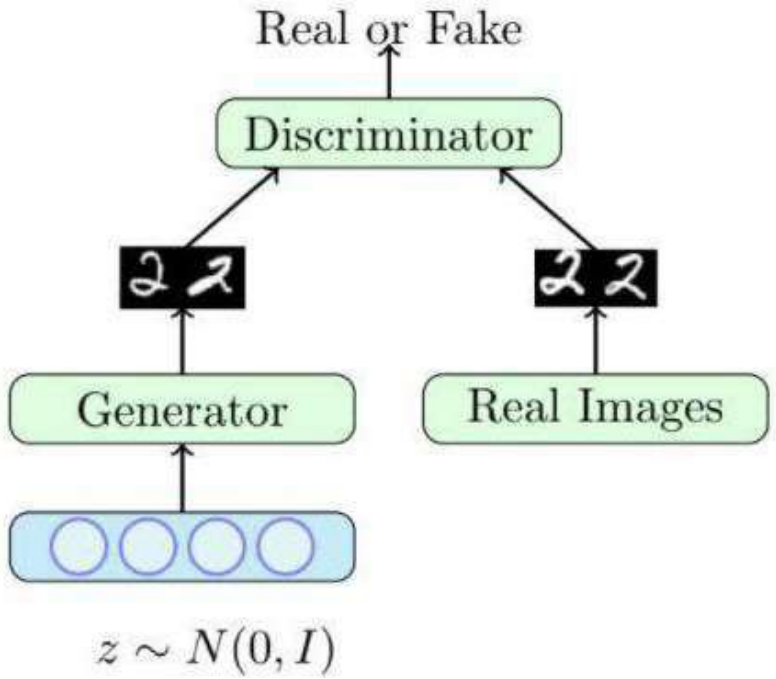
- This is just for a single z and the generator would like to do this for all possible values of z,

Objective of Discriminator



- Now let's look at the discriminator
- The task of the discriminator is to assign a high score to real images and a low score to fake images

Objective of Discriminator



- Now let's look at the discriminator
- The task of the discriminator is to assign a high score to real images and a low score to fake images
- And it should do this for all possible real images and all possible fake images

GAN: Architecture

- We will now look at one of the popular neural networks used for the generator and discriminator (Deep Convolutional GANs)
- For discriminator, any CNN based classifier with 1 class (real) at the output can be used (e.g. VGG, ResNet, etc.)

GAN: Architecture

- We will now look at one of the popular neural networks used for the generator and discriminator (Deep Convolutional GANs)
- For discriminator, any CNN based classifier with 1 class (real) at the output can be used (e.g. VGG, ResNet, etc.)

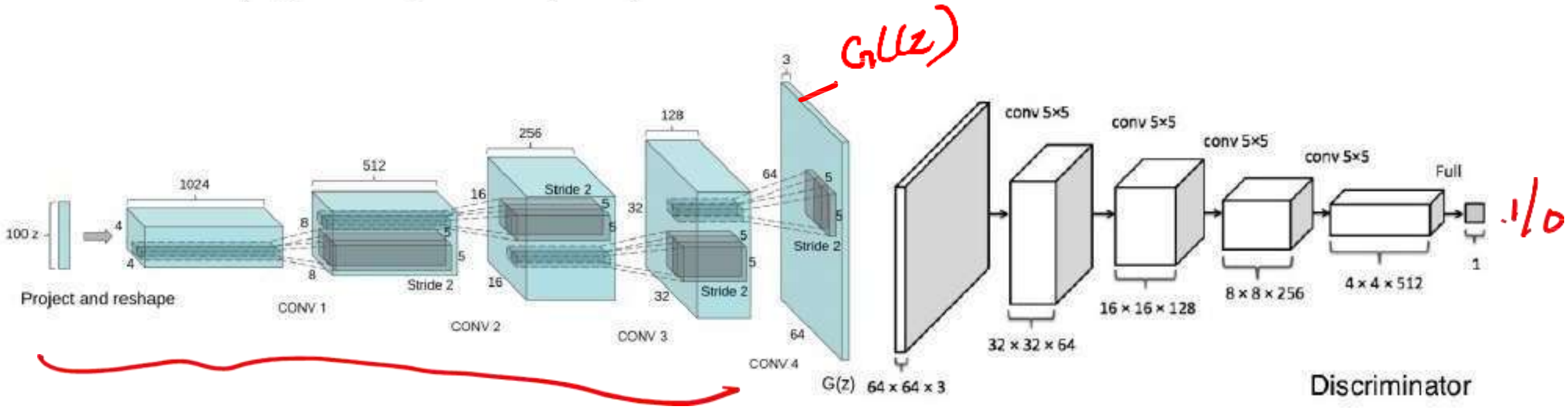


Figure: Generator (Redford et al 2015) (left) and discriminator (Yeh et al 2016) (right) used in DCGAN

GAN: Architecture

Architecture guidelines for stable Deep Convolutional GANs

- 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.

UE20CS342-TDL-GANs

References

1. https://www.youtube.com/watch?v=1ju4qmdtRdY
2. http://www.cse.iitm.ac.in/~miteshk/CS7015/Slides/Teaching/pdf/Lecture23.pdf
3. https://jonathan-hui.medium.com/gan-whats-generative-adversarial-networks-and-its-application-f39ed278ef09

TOPICS IN DEEP LEARNING

Deep Convolutional Generative Adversarial Networks

Introduction - Recollection of GAN

- GAN is Generative Adversarial Network having two neural networks: Generator and Discriminator that are pitted against each other and are simultaneously trained by an adversarial process.
- **Generator**: learns to **generate plausible data that is very similar to the training data**. Data generated from the Generator should be indistinguishable from the real data.
- **Discriminator**: the key objective is to **distinguish between the generator's fake data from the real data** and is a simple classification network.
- A classic analogy used is that the Generator is a forger who wants to create fake art while the discriminator is an investigator/cop whose job is to catch fake art. Discriminator penalizes the Generator for generating fake data when failing to fool the Discriminator. The Discriminator's classification loss helps update the Generator's weights through backpropagation to get better at generating data close to real data.
- During training, the *generator* progressively becomes better at creating images that look real, while the *discriminator* becomes better at telling them apart. The process reaches equilibrium when the *discriminator* can no longer distinguish real images from fakes.

Introduction - Recollection of GAN

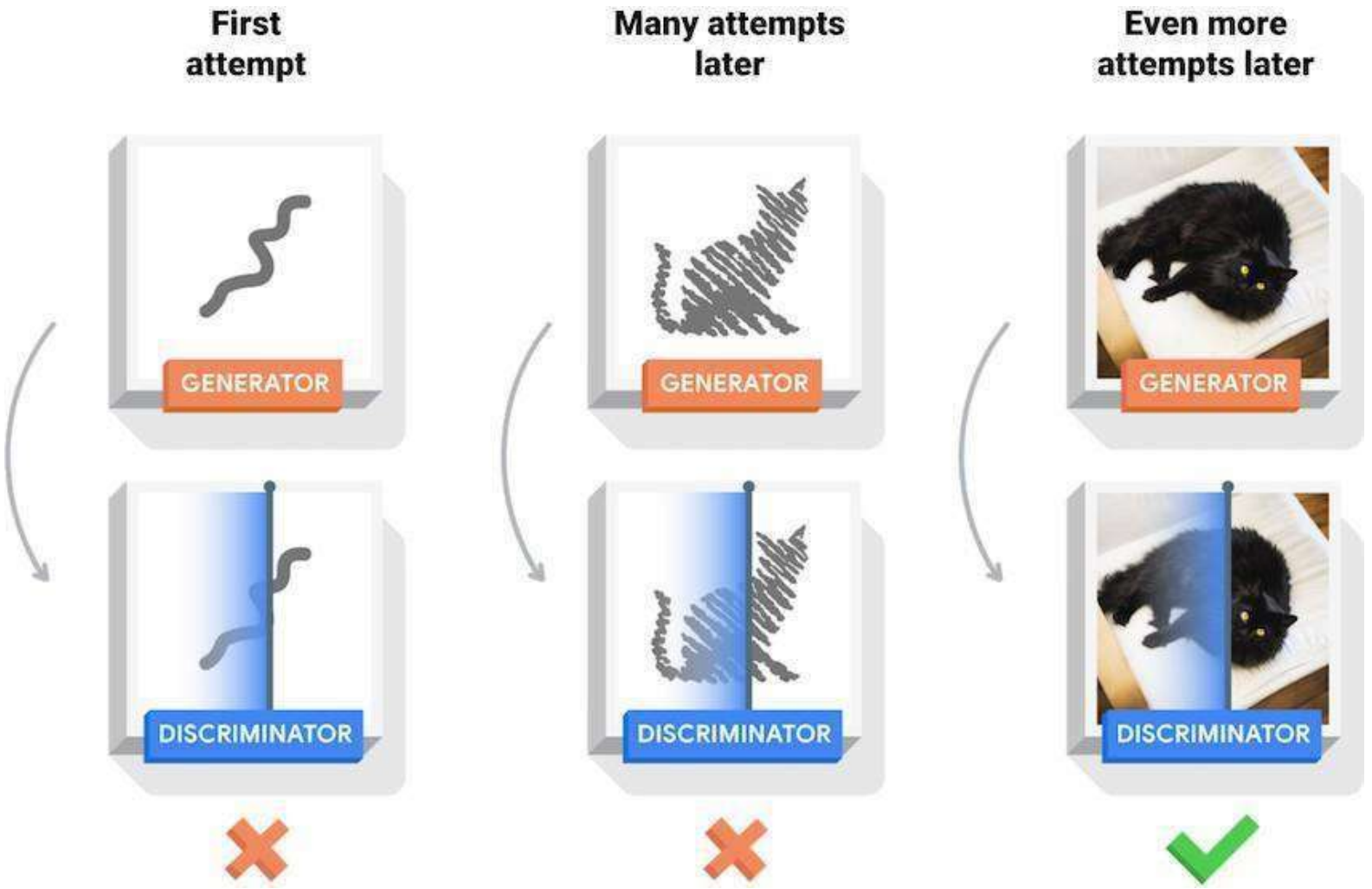
GENERATOR
"The Artist"
A neural network trying to create pictures of cats that look real.

DISCRIMINATOR
"The Art Critic"
A neural network examining cat pictures to determine if they're real or fake.

Thousands of real-world images labeled "CAT"

UE20CS342-TDL-GANs

Introduction - Recollection of GAN

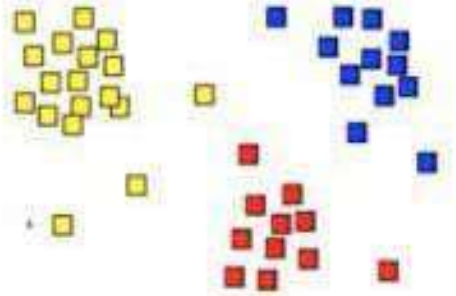


Key differences/additions - DCGAN

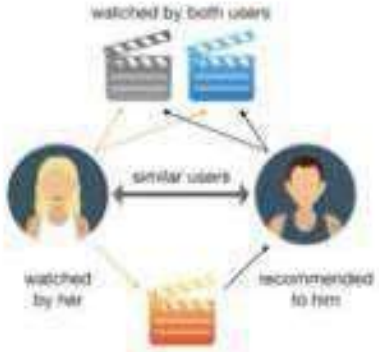
- Here is the summary of DCGAN:
  - Replace all max pooling with convolutional stride
  - Use transposed convolution for upsampling.
  - Eliminate fully connected layers.
  - Use Batch normalization except for the output layer for the generator and the input layer of the discriminator.
  - Use ReLU in the generator except for the output which uses tanh.
  - Both Generator and Discriminator do not use a Max pooling.
  - The Generator uses Leaky Relu as the activation function for all layers except the output, and the Discriminator uses Leaky Relu for all layers.

Autoencoder

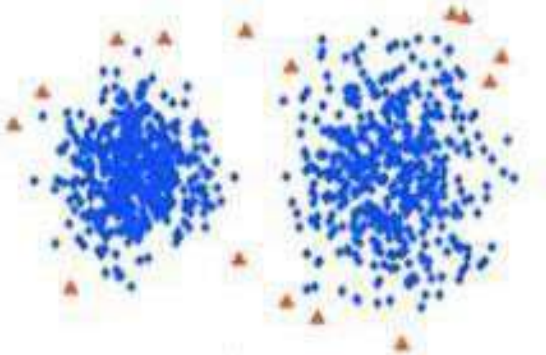
Applications of Unsupervised Learning



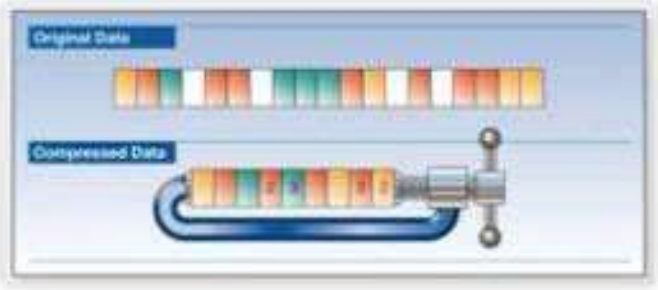
Clustering



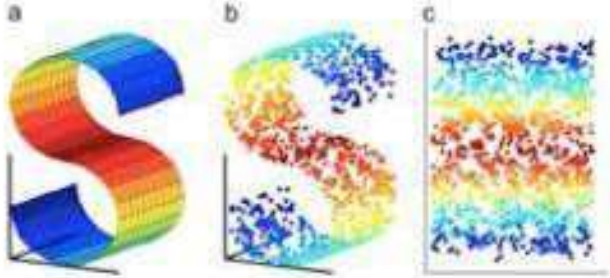
Recommender System



Anomaly Detection



Data Compression



Dimensionality Reduction

Data Generation

Topics in Deep Learning

What is Autoencoder ?

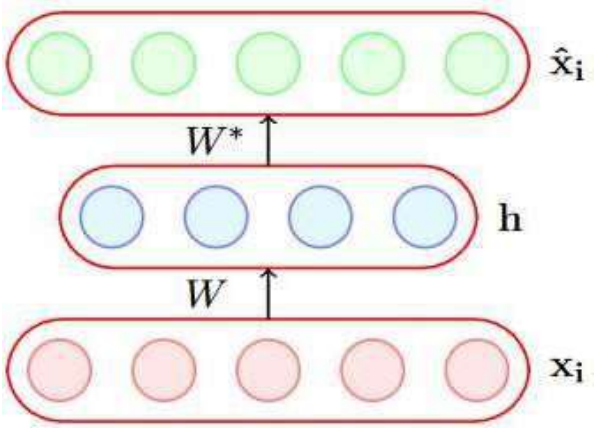
An autoencoder is a type of neural network that is commonly used for unsupervised learning, data compression, and feature extraction.

It consists of two parts: an encoder that maps the input data into a lower-dimensional space, and a decoder that reconstructs the original input data from the encoded representation.

The primary objective of an autoencoder is to learn a compressed representation of the input data that captures its most salient features. This compressed representation, also known as the latent space, can be used for a variety of tasks such as data compression, denoising, image generation, and anomaly detection.

One of the key advantages of autoencoders is that they can learn useful features from raw data without the need for explicit labels or supervision. This makes them particularly useful in cases where labeled data is scarce or expensive to obtain.

Additionally, autoencoders can be used to pretrain deep neural networks, which can improve the performance of supervised learning tasks.



$$\mathbf{h} = g(W\mathbf{x_i} + \mathbf{b})$$
$$\hat{\mathbf{x}}_\mathbf{i} = f(W^*\mathbf{h} + \mathbf{c})$$

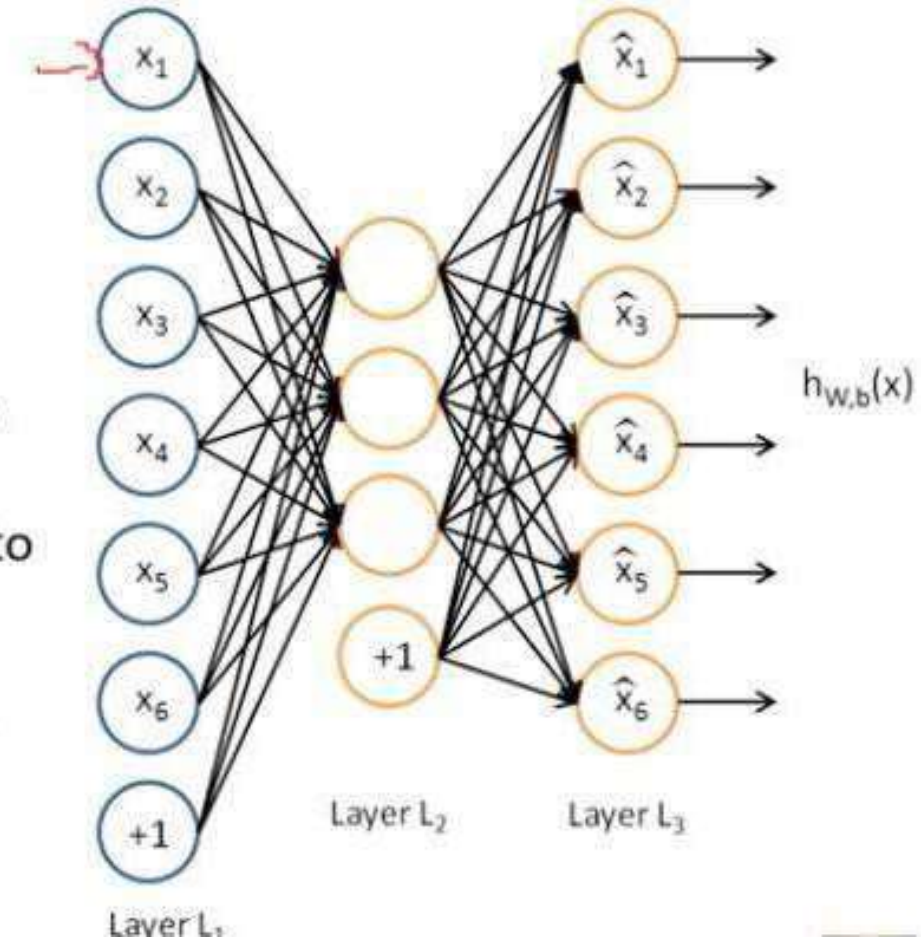
Autoencoder

Autoencoder

An Autoencoder is a feedforward neural network that learns to predict the input itself in the output.

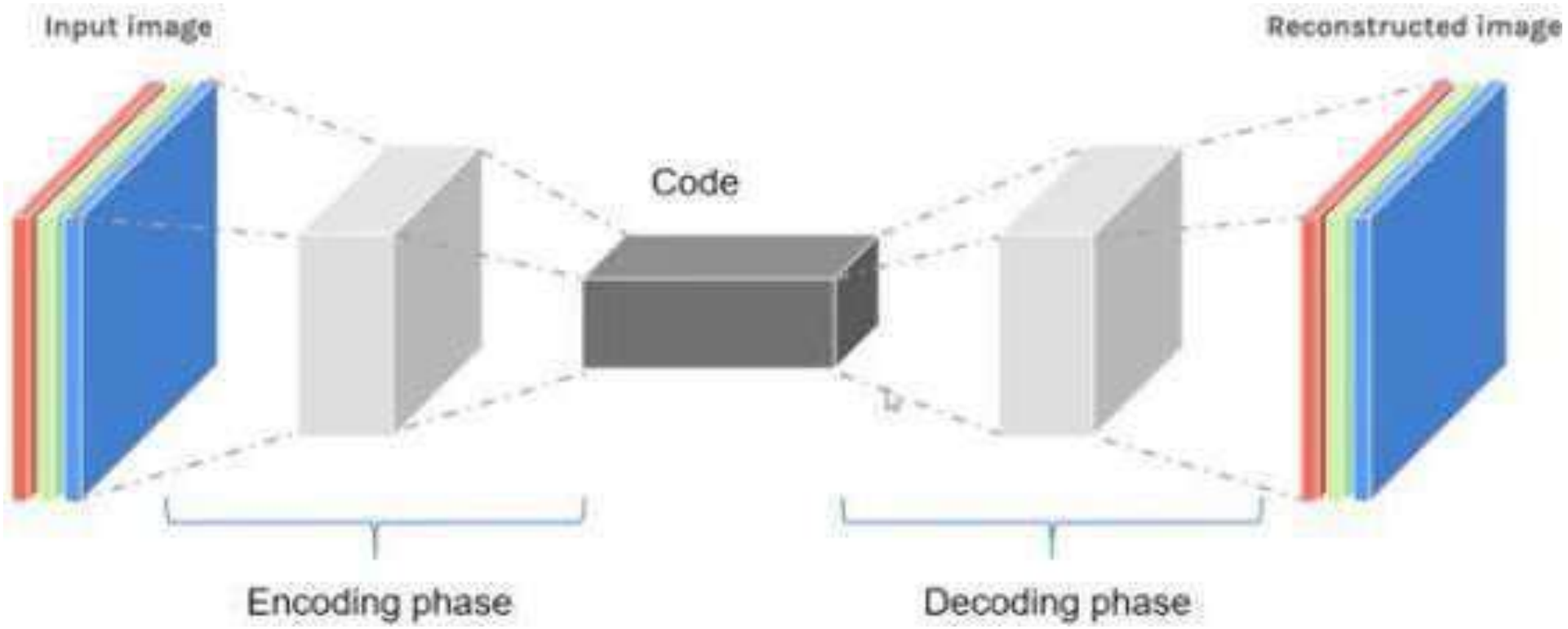
$$y^{(i)} = x^{(i)}$$

- The input-to-hidden part corresponds to an **encoder**
- The hidden-to-output part corresponds to a **decoder**
- Different from RBM which is not a feedforward network but a probabilistic model
- Many cool applications are based on encoder-decoder frameworks



Topics in Deep Learning

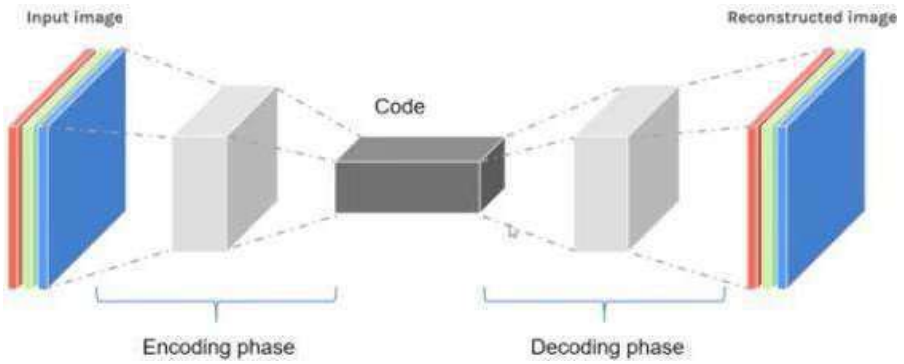
Autoencoder



Topics in Deep Learning

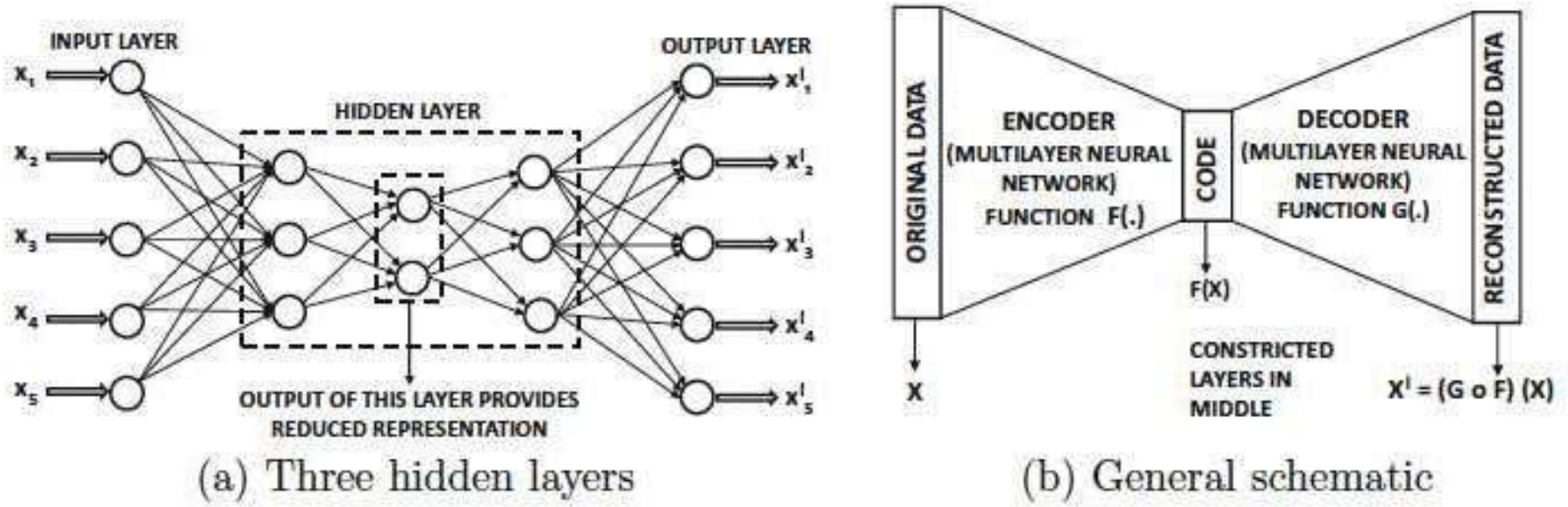
Autoencoder

- The basic idea of an autoencoder is to have an output layer with the same dimensionality as the inputs.
- The idea is to try to reconstruct each dimension exactly by passing it through the network.
- An autoencoder replicates the data from the input to the output, and is therefore sometimes referred to as a replicator neural network.
- Although reconstructing the data might seem like a trivial matter by simply copying the data forward from one layer to another, this is not possible when the number of units in the middle are constricted.
- In other words, the number of units in each middle layer is typically fewer than that in the input (or output).
- As a result, these units hold a reduced representation of the data, and the final layer can no longer reconstruct the data exactly. (Reconstruction Loss)
- Therefore, this type of reconstruction is inherently lossy.



Topics in Deep Learning

The Basic Autoencoder

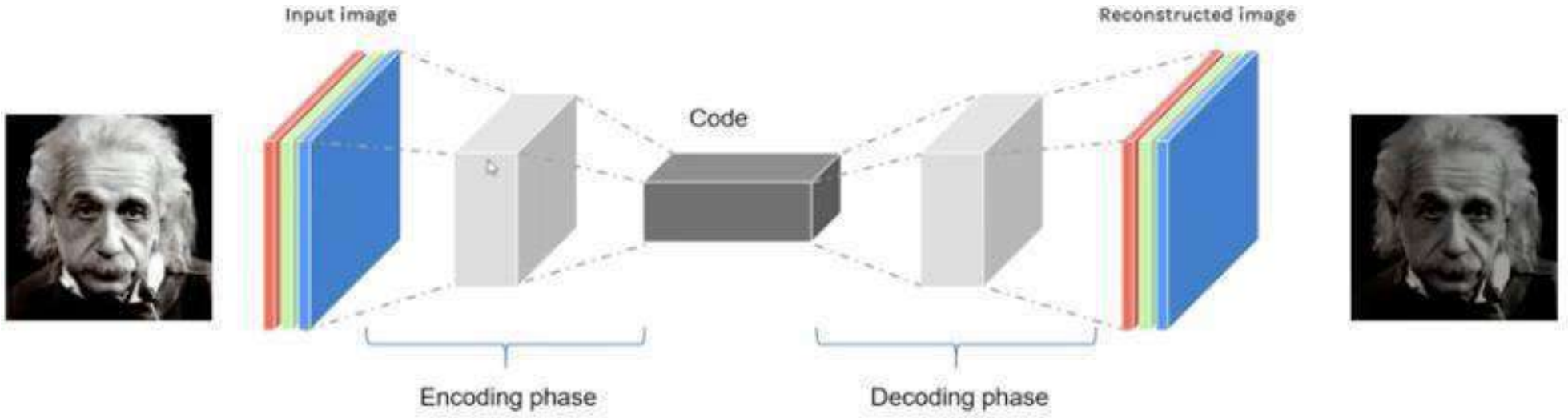


(a) Three hidden layers (b) General schematic

Figure 2.6: The basic schematic of the autoencoder

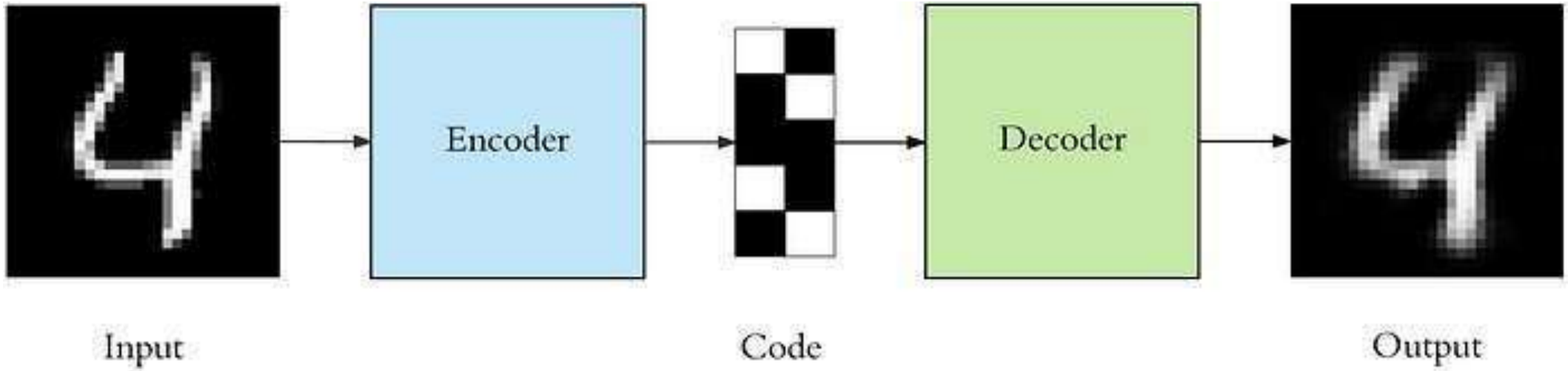
Topics in Deep Learning

Autoencoder- Example1- Image generation



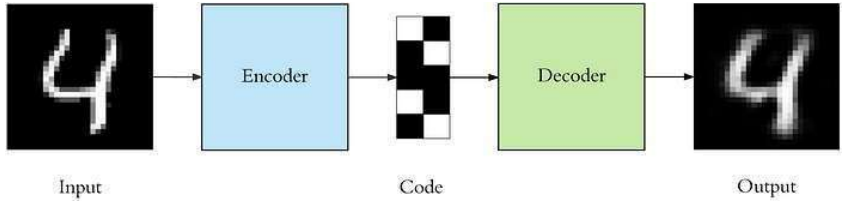
Topics in Deep Learning

Autoencoder- Example2



Topics in Deep Learning

Autoencoder- Properties



Autoencoders are mainly a dimensionality reduction (or compression) algorithm with a couple of important properties:

1. Data-specific: Autoencoders are only able to meaningfully compress data similar to what they have been trained on. Since they learn features specific for the given training data, they are different than a standard data compression algorithm like gzip. So we can't expect an autoencoder trained on handwritten digits to compress landscape photos.

2. Lossy: The output of the autoencoder will not be exactly the same as the input, it will be a close but degraded representation. If you want lossless compression they are not the way to go.

3. Unsupervised: To train an autoencoder we don't need to do anything fancy, just throw the raw input data at it. Autoencoders are considered an *unsupervised* learning technique since they don't need explicit labels to train on. But to be more precise they are *self-supervised* because they generate their own labels from the training data.

Topics in Deep Learning

How to Train a Autoencoder ?

The encoder takes an input vector x and maps it to a compressed representation z, also known as the latent space, using an encoder function f(x):

z = f(x)

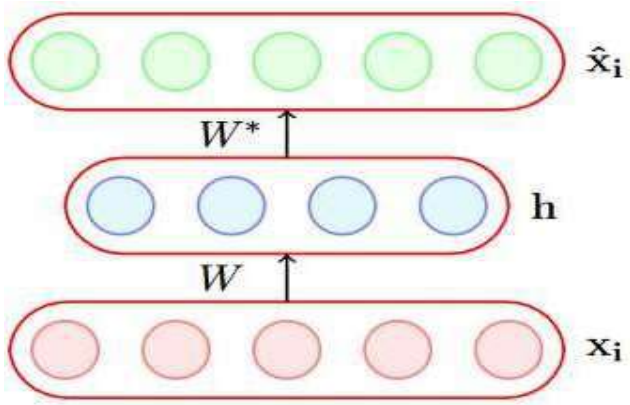
The decoder then takes this compressed representation z and maps it back to the original input vector x, using a decoder function g(z):

x' = g(z)

The goal of training an autoencoder is to minimize the difference between the original input vector x and the reconstructed vector x'. This is typically done by minimizing the mean squared error between the two vectors:

L(x, x') = ||x - x'||^2

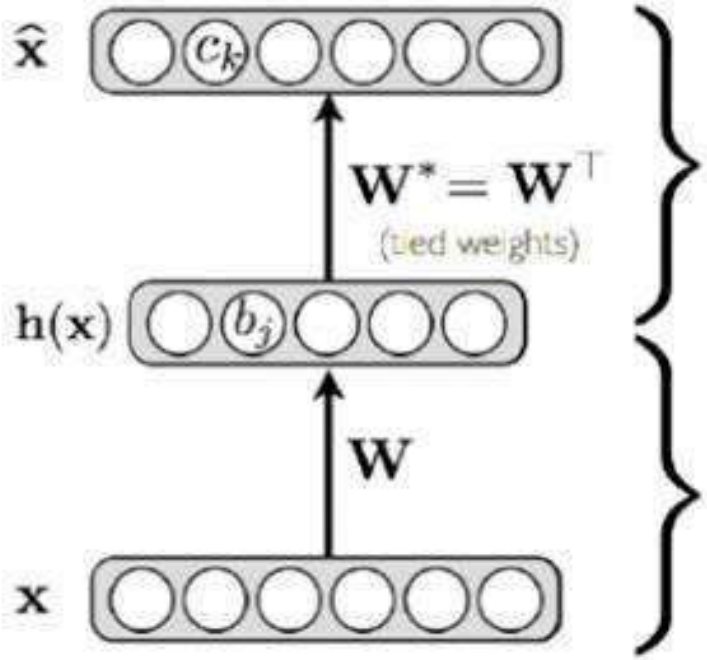
where ||.|| represents the Euclidean norm.



$$\mathbf{h} = g(W\mathbf{x}_i + \mathbf{b})$$
$$\hat{\mathbf{x}}_i = f(W^*\mathbf{h} + \mathbf{c})$$

How to Train a Autoencoder ?

- Feed-forward neural network trained to reproduce its input at the output layer



Decoder

$$\hat{\mathbf{x}} = o(\hat{\mathbf{a}}(\mathbf{x})) = \underbrace{\mathrm{sigm}(\mathbf{c} + \mathbf{W}^*\mathbf{h}(\mathbf{x}))}_{\text{For binary units}}$$

Encoder

$$\mathbf{h}(\mathbf{x}) = g(\mathbf{a}(\mathbf{x})) = \mathrm{sigm}(\mathbf{b} + \mathbf{W}\mathbf{x})$$

Topics in Deep Learning

Autoencoder- Tied Weights

- An autoencoder with tied weights has decoder weights that are the transpose of the encoder weights; this is a form of parameter sharing, which reduces the number of parameters of the model.

- Advantages of tying weights include increased training speed and reduced risk of overfitting, while yielding comparable performance than without weight tying in many cases (Li et al. (2019)).

- It is therefore a common practice to tie weights when building a symmetrical autoencoder.

Topics in Deep Learning

How to Train a Autoencoder ?

You need to set 4 hyperparameters before *training* an autoencoder:

1. **Code size:** The code size or the size of the bottleneck is the most important hyperparameter used to tune the autoencoder. The bottleneck size decides how much the data has to be compressed. This can also act as a regularisation term.
2. **Number of layers:** Like all neural networks, an important hyperparameter to tune autoencoders is the depth of the encoder and the decoder. While a higher depth increases model complexity, a lower depth is faster to process.
3. **Number of nodes per layer:** The number of nodes per layer defines the weights we use per layer. Typically, the number of nodes decreases with each subsequent layer in the autoencoder as the input to each of these layers becomes smaller across the layers.
4. **Reconstruction Loss:** The loss function we use to train the autoencoder is highly dependent on the type of input and output we want the autoencoder to adapt to. If we are working with image data, the most popular loss functions for reconstruction are MSE Loss and L1 Loss. In case the inputs and outputs are within the range [0,1], as in MNIST, we can also make use of Binary Cross Entropy as the reconstruction loss.

Autoencoder- Loss Function

- Loss function for binary inputs

$$l(f(\mathbf{x})) = -\sum_k \left(x_k \log(\widehat{x}_k) + (1 - x_k) \log(1 - \widehat{x}_k)\right)$$

  - Cross-entropy error function (reconstruction loss) $f(\mathbf{x}) \equiv \widehat{\mathbf{x}}$

- Loss function for real-valued inputs

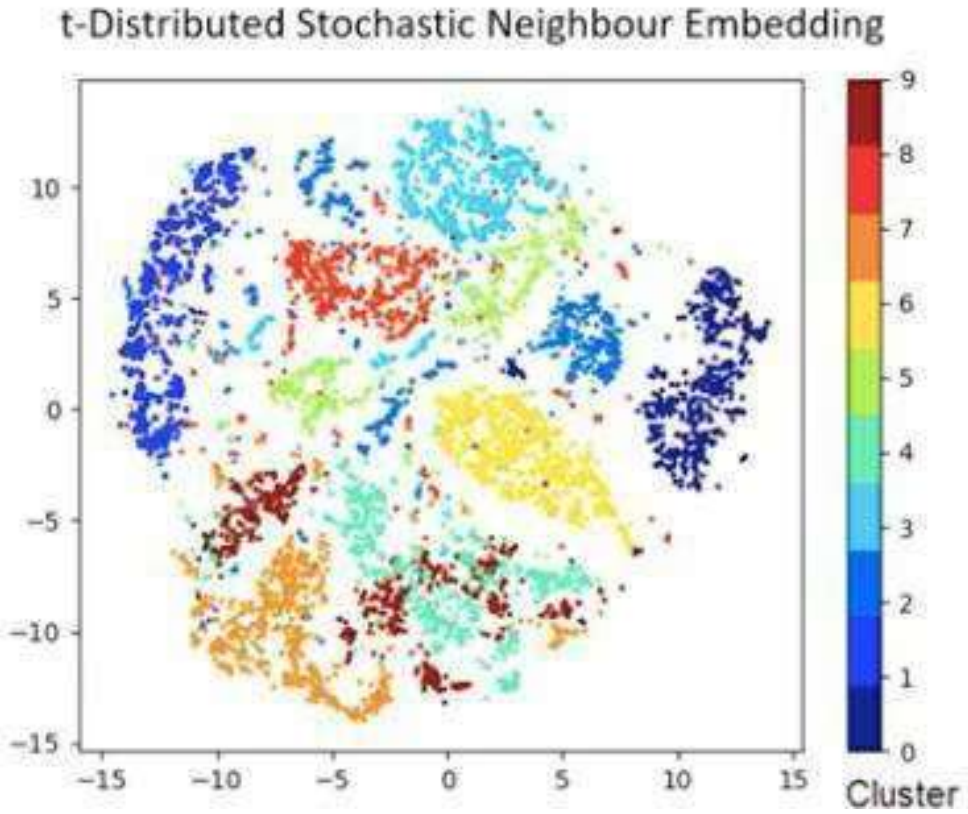
$$l(f(\mathbf{x})) = \frac{1}{2} \sum_k (\widehat{x}_k - x_k)^2$$

  - sum of squared differences (reconstruction loss)
  - we use a linear activation function at the output

Autoencoder- Application-1

AE for Dimensionality Reduction

- **Visualizing high-dimensional data** is challenging.
- **t-SNE** is the most commonly used method but struggles with large number of dimensions (typically above 32).
- Autoencoders are used as a pre-processing step to reduce the dimensionality, and this compressed representation is used by t-SNE to visualize the data in 2D space.



T-SNE visualization for clustering on MNIST dataset

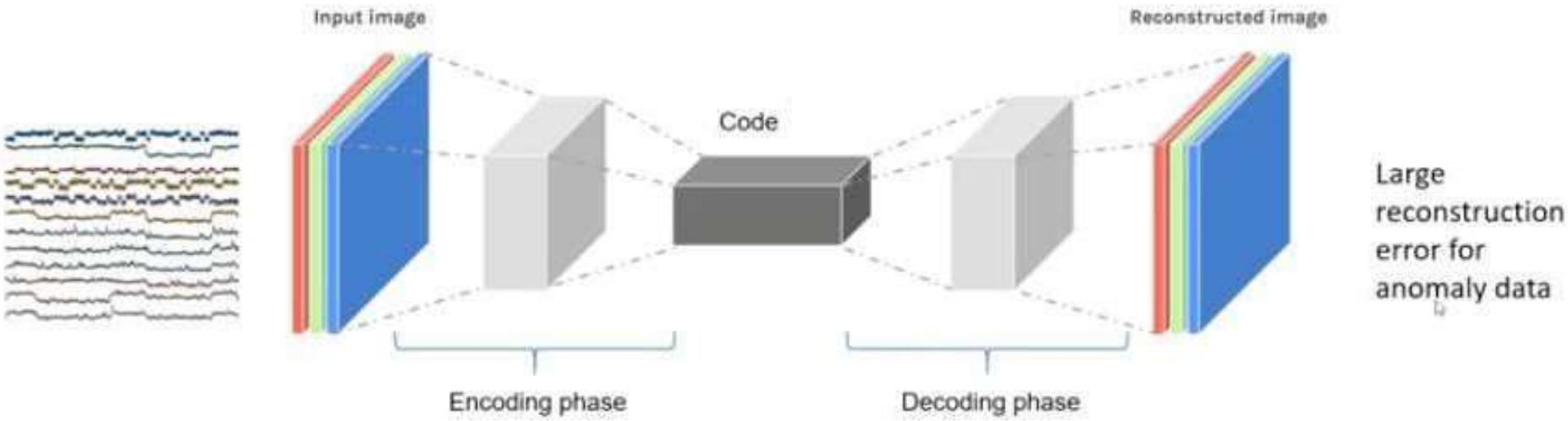
MNIST -28*28

Topics in Deep Learning

Autoencoder- Application 2

Autoencoder Applications

Anomaly detection



Large reconstruction error for anomaly data

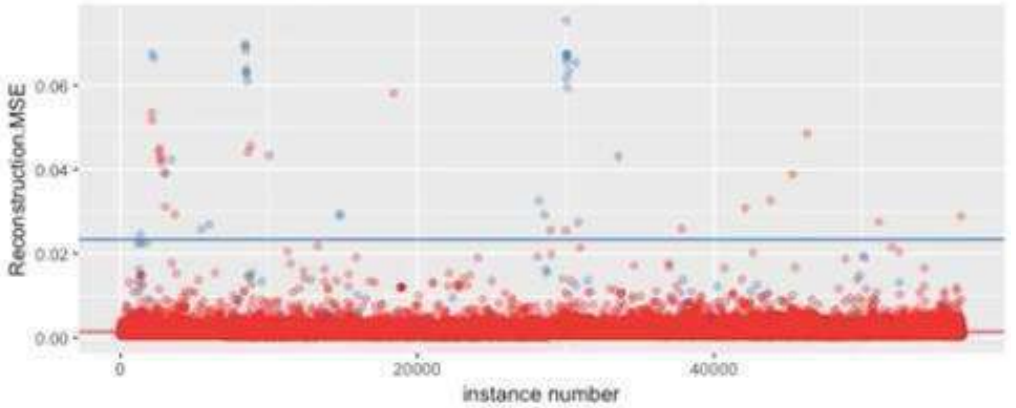
Topics in Deep Learning

Autoencoder- Application 2

AE for Anomaly Detection

- AE can be used for anomaly detection such as Credit Card Frauds
- **Features:** such as time, amount, etc can be learned using an AE
- **Training**: AE will aim to minimize the reconstruction error of training data
- **Testing:** Transactions with higher reconstruction error will be labelled as Credit Card Frauds based on some threshold

Note: there is no perfect classification into fraud and non-fraud cases but the mean MSE is definitely higher for fraudulent transactions than for regular ones



Reconstruction error rate is high for anomalous transactions

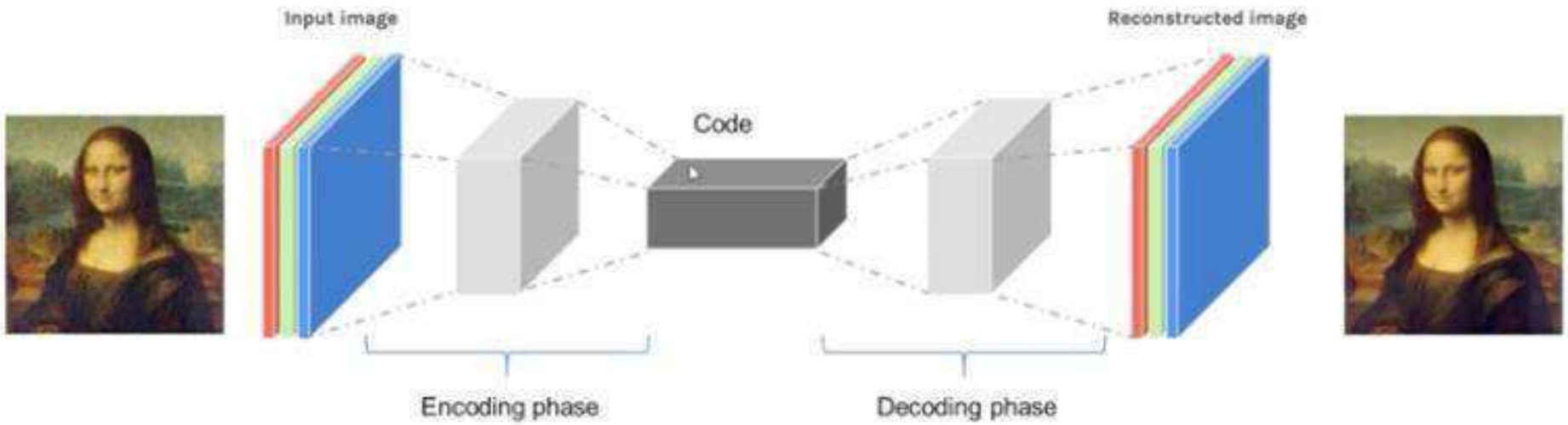
Blue –Anomaly
Red- Non Anomaly

Topics in Deep Learning

Autoencoder- Application 3

Autoencoder Applications

Noise Reduction

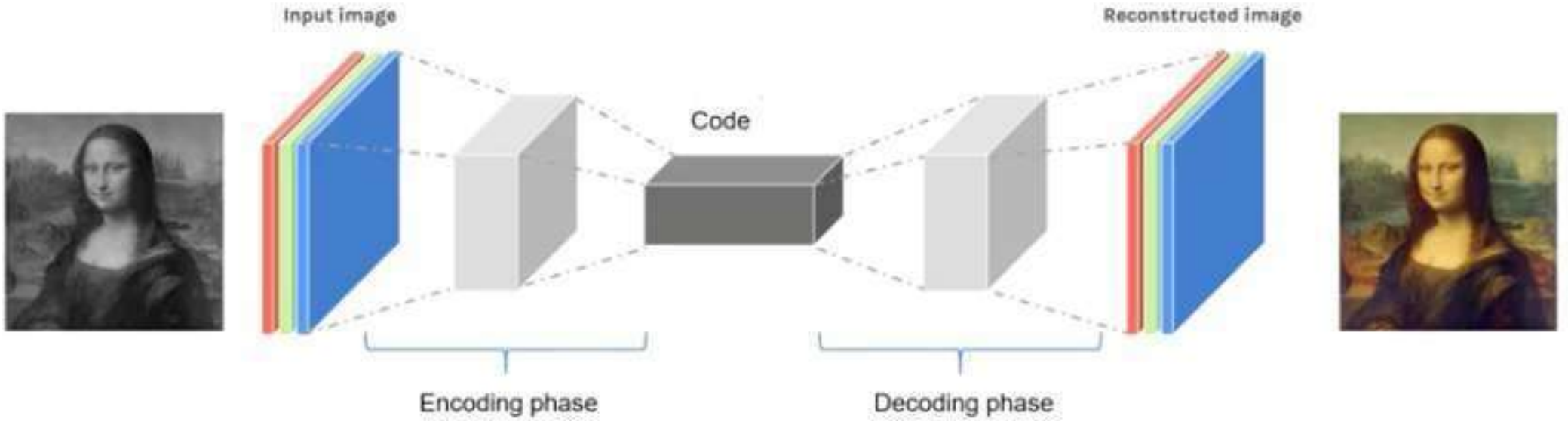


Topics in Deep Learning

Autoencoder- Application 4

Autoencoder Applications

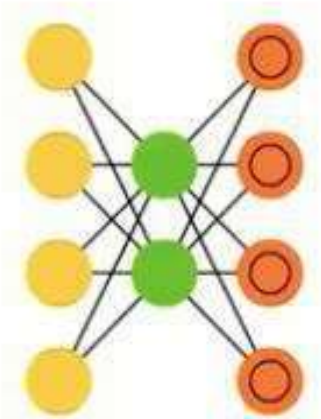
Image colorization



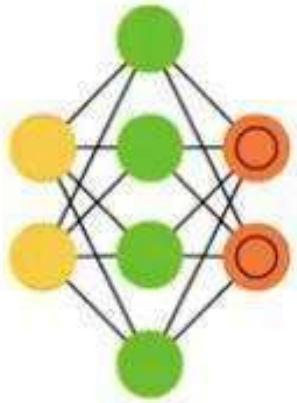
Topics in Deep Learning

Autoencoder- Types

1. Under Complete
2. Over Complete



Undercomplete Autoencoder

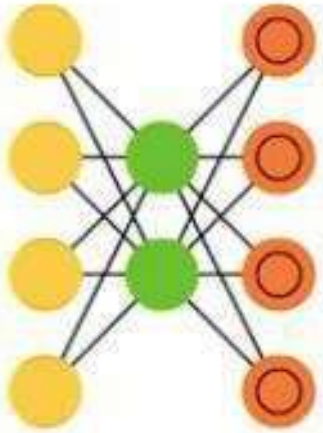


Overcomplete Autoencoder

- What are the differences?
- Where they can be applied?

Topics in Deep Learning

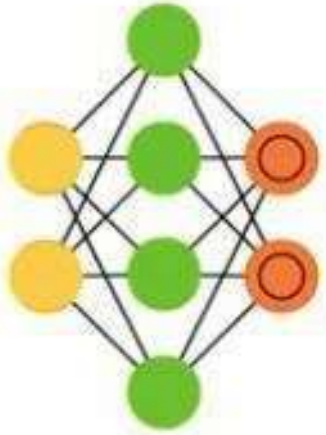
Autoencoder- Types



Undercomplete Autoencoder

- Used for compressing data into low dimension
- Objective is to minimize the reconstruction error

Feature Reduction



Overcomplete Autoencoder

- Used for representing data with more features via high dimension (useful for supervised learning tasks)
- Caution: Sometimes memorizes data since reconstruction error is low... **Solution: Add noise**

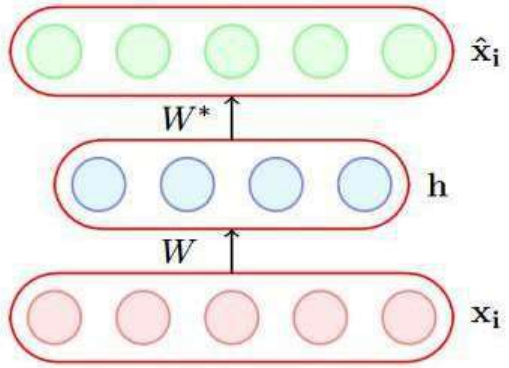
Feature Enhancement

Adding Noise is a kind of regularization-
DenoisingAutoencoder

Topics in Deep Learning

Autoencoder- Under Complete

- An autoencoder is said to be undercomplete if the dimensionality of the latent space is smaller than the dimensionality of the input space.
- In other words, the encoder is forced to learn a compressed representation of the input data that captures only the most essential features.
- This can help prevent overfitting and encourage the autoencoder to learn a more general representation of the input data.
- However, an undercomplete autoencoder may not be able to capture all the essential features of the input data.



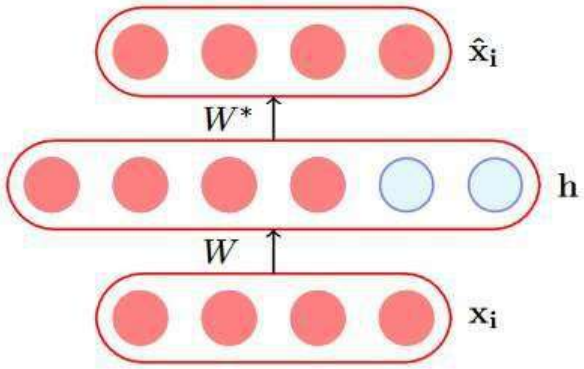
$$\mathbf{h} = g(W\mathbf{x_i} + \mathbf{b})$$
$$\hat{\mathbf{x}}_\mathbf{i} = f(W^*\mathbf{h} + \mathbf{c})$$

An autoencoder where $\dim(\mathbf{h}) < \dim(\mathbf{x_i})$ is called an under complete autoencoder

Topics in Deep Learning

Autoencoder- Overcomplete

- An autoencoder is said to be overcomplete if the dimensionality of the latent space (i.e., the number of neurons in the encoder's output layer) is greater than the dimensionality of the input space.
- In other words, there are more hidden units in the encoder than necessary to capture all the essential features of the input data.
- This means that an overcomplete autoencoder can learn multiple compressed representations of the input data, each capturing a different aspect of the data.
- However, this can also lead to overfitting, where the autoencoder learns to simply copy the input data to the latent space without capturing meaningful features.



$$\mathbf{h} = g(W\mathbf{x_i} + \mathbf{b})$$
$$\hat{\mathbf{x}}_\mathbf{i} = f(W^*\mathbf{h} + \mathbf{c})$$

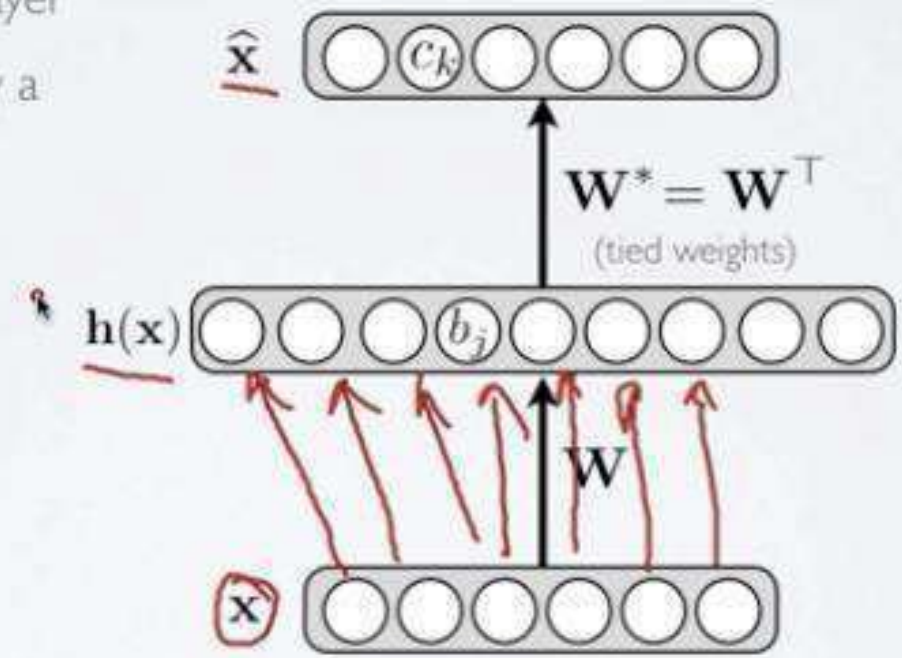
An autoencoder where $\dim(\mathbf{h}) \geq \dim(\mathbf{x_i})$ is called an over complete autoencoder

Topics in Deep Learning

Autoencoder- Overcomplete

Topics: overcomplete representation

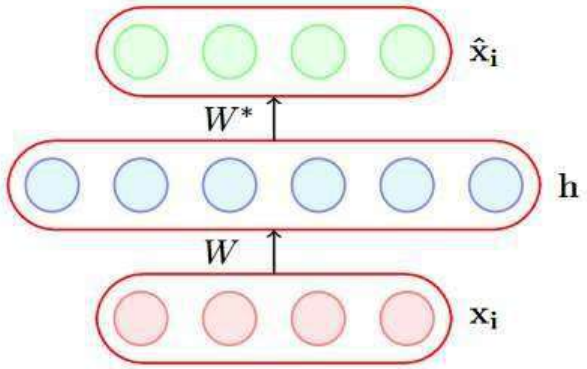
- Hidden layer is overcomplete if greater than the input layer
  - no compression in hidden layer
  - each hidden unit could copy a different input component
- No guarantee that the hidden units will extract meaningful structure



Regularisation in overcomplete autoencoders

- While poor generalization could happen even in undercomplete autoencoders it is an even more serious problem for overcomplete auto encoders
- The simplest solution is to add a L2-regularization term to the objective function

$$\min_{\theta,w,w^*,\mathrm{b,c}} \frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{n}(\hat{x}_{ij}-x_{ij})^2+\lambda\|\theta\|^2$$



- Another trick is to tie the weights of the encoder and decoder i.e., $W^* = W^T$
  - This effectively reduces the capacity of Autoencoder and acts as a regularizer

Types of Autoencoders

Topics in Deep Learning

Types of Autoencoders

1. **Vanilla Autoencoders**: The most basic type of autoencoders are vanilla autoencoders. Vanilla autoencoders are composed of an input layer, an output layer and one hidden layer in between. The hidden layer has fewer nodes than the input layer, which forces the autoencoders to compress the input data.

2. Sparse Autoencoders: Sparse autoencoders are similar to vanilla autoencoders, but they include an additional regularization term that encourages the model to use only a small subset of the input nodes. This results in a more compact representation of the input data.

3. Denoising Autoencoders: Denoising autoencoders are similar to vanilla autoencoders, but they are trained using corrupted input data. This forces the autoencoders to learn the structure of the input data and disregard the corruptions.

4. Contractive Autoencoders: Contractive autoencoders are similar to sparse autoencoders, but they include an additional regularization term that encourages the model to learn a sparse, but also robust representation of the input data.

Topics in Deep Learning

Types of Autoencoders

5. Convolutional Autoencoders: Autoencoders that use convolutional neural networks (CNNs) to reduce the input dimensionality

.

6. Variational Autoencoders: Autoencoders that encode input data as a set of latent variables that are randomly sampled from a specific distribution.

7. Generative Adversarial Autoencoders: Autoencoders that are trained in an adversarial manner to generate new data.

Topics in Deep Learning

Variants in Autoencoders

Regularized Auto encoders

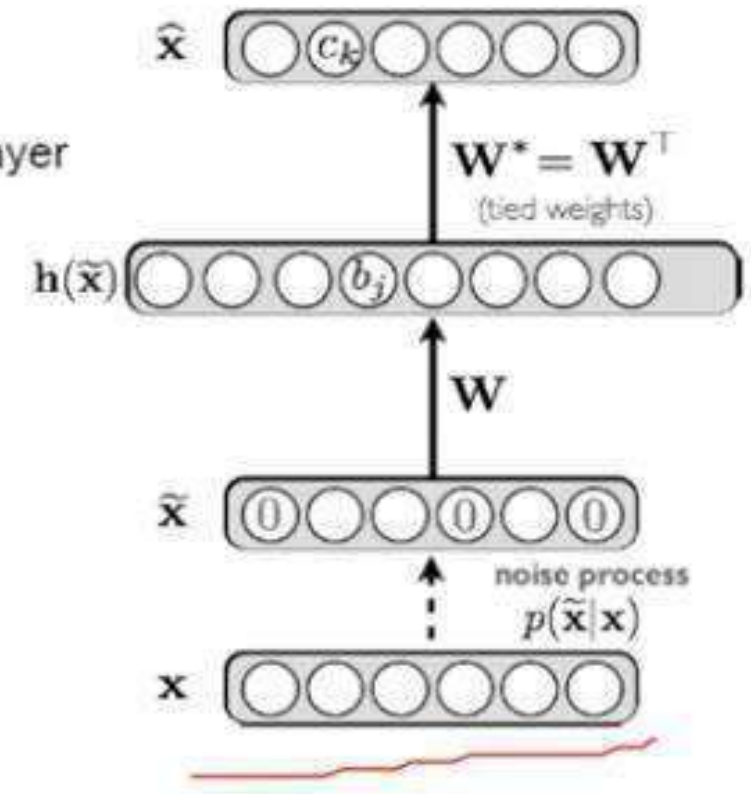
1. Denoising Autoencoders

2. Sparse Autoencoders
3. Variational Autoencoders

Topics in Deep Learning

Denoising Autoencoders

- Idea: representation should be robust to introduction of noise:
  - random assignment of subset of inputs to 0, with probability v
  - Similar to dropouts on the input layer
  - Gaussian additive noise



- Reconstruction $\hat{\mathbf{x}}$ computed from the corrupted input $\tilde{\mathbf{x}}$

- Loss function compares $\hat{\mathbf{x}}$ reconstruction with the noiseless input $\mathbf{x}$

You shoud get 1,2,3,4,5,6

These nodes will learn how to remove the noise and learn data

1.002, 2.009, 3.0067 …..

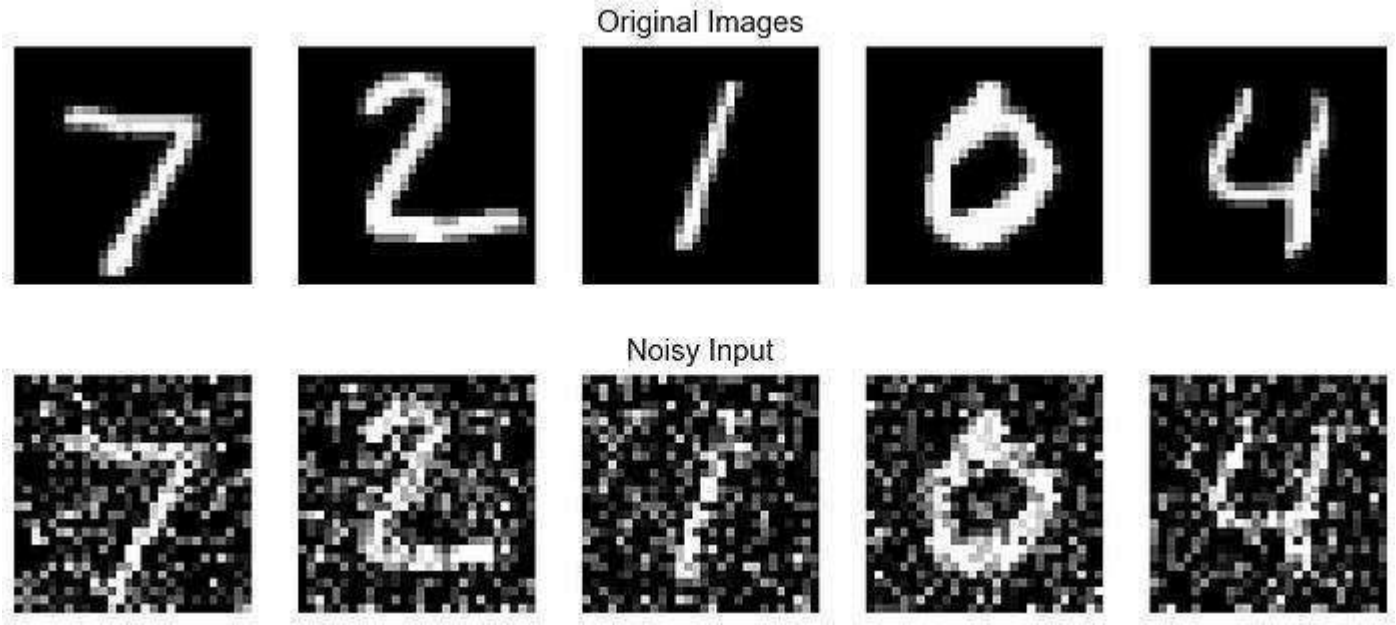
1, 2,3, 4,5, 6

Topics in Deep Learning

Denoising Autoencoders

- Keeping the code layer small forced our autoencoder to learn an intelligent representation of the data. There is another way to force the autoencoder to learn useful features, which is adding random noise to its inputs and making it recover the original noise-free data.

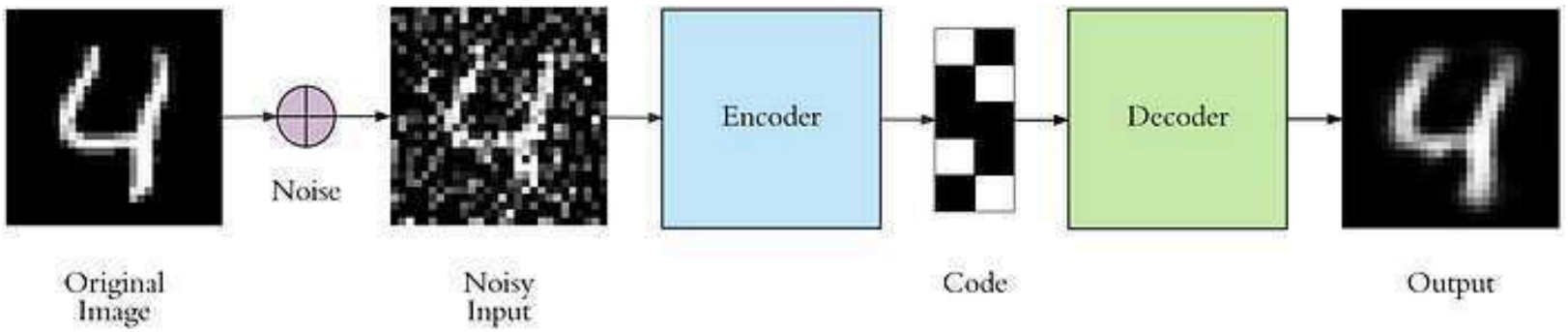
- This way the autoencoder can't simply copy the input to its output because the input also contains random noise. We are asking it to subtract the noise and produce the underlying meaningful data. This is called a *denoising autoencoder*.



Topics in Deep Learning

Denoising Autoencoders

Add random Gaussian noise to them and the noisy data becomes the input to the autoencoder.
The autoencoder doesn't see the original image at all.
But then we expect the autoencoder to regenerate the noise-free original image.



Topics in Deep Learning

Visualising Autoencoders - % of Noise added

More corruption, filters are learned well.
Too much corruption can lead to lower construction

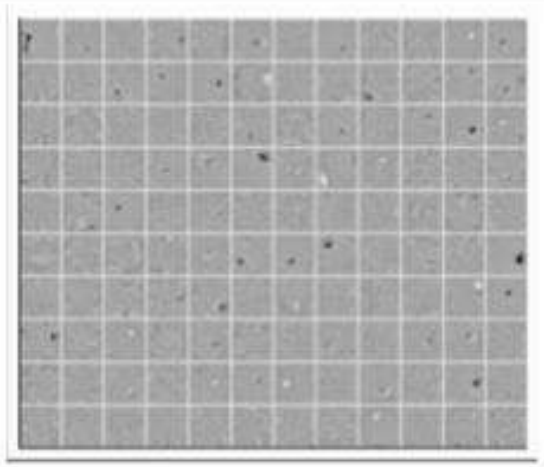


Figure: Vanilla AE (No noise)

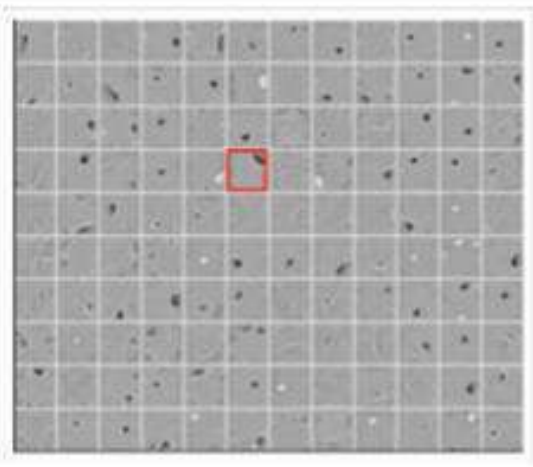


Figure: 25% Denoising AE (q=0.25)

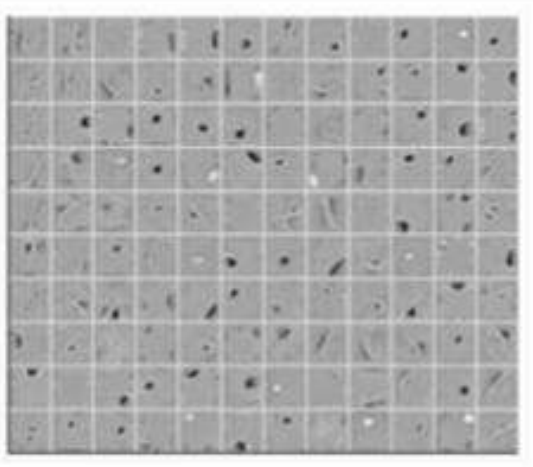


Figure: 50% Denoising AE (q=0.5)

- The vanilla AE does not learn many meaningful patterns
- The hidden neurons of the denoising AEs seem to act like pen-stroke detectors (for example, in the highlighted neuron the black region is a stroke that you would expect in a ’0’ or a ’2’ or a ’3’ or a ’8’ or a ’9’)
- As the noise increases the filters become more wide because the neuron has to rely on more adjacent pixels to feel confident about a stroke

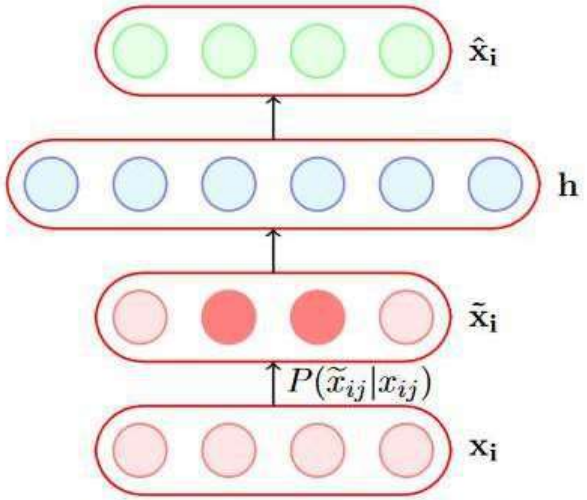
Topics in Deep Learning

Denoising Autoencoders

- A denoising autoencoder is a type of autoencoder that is designed to remove noise from input data. It does this by corrupting the input data with some form of noise (e.g., Gaussian noise or dropout) and then training the autoencoder to reconstruct the original, noise-free data.
- A denoising encoder simply corrupts the input data using a probabilistic process (P (˜ xij |xij)) before feeding it to the network
- A simple P could be

$$P(\tilde{x}_{ij} = 0|x_{ij}) = q$$
$$P(\tilde{x}_{ij} = x_{ij}|x_{ij}) = 1 - q$$

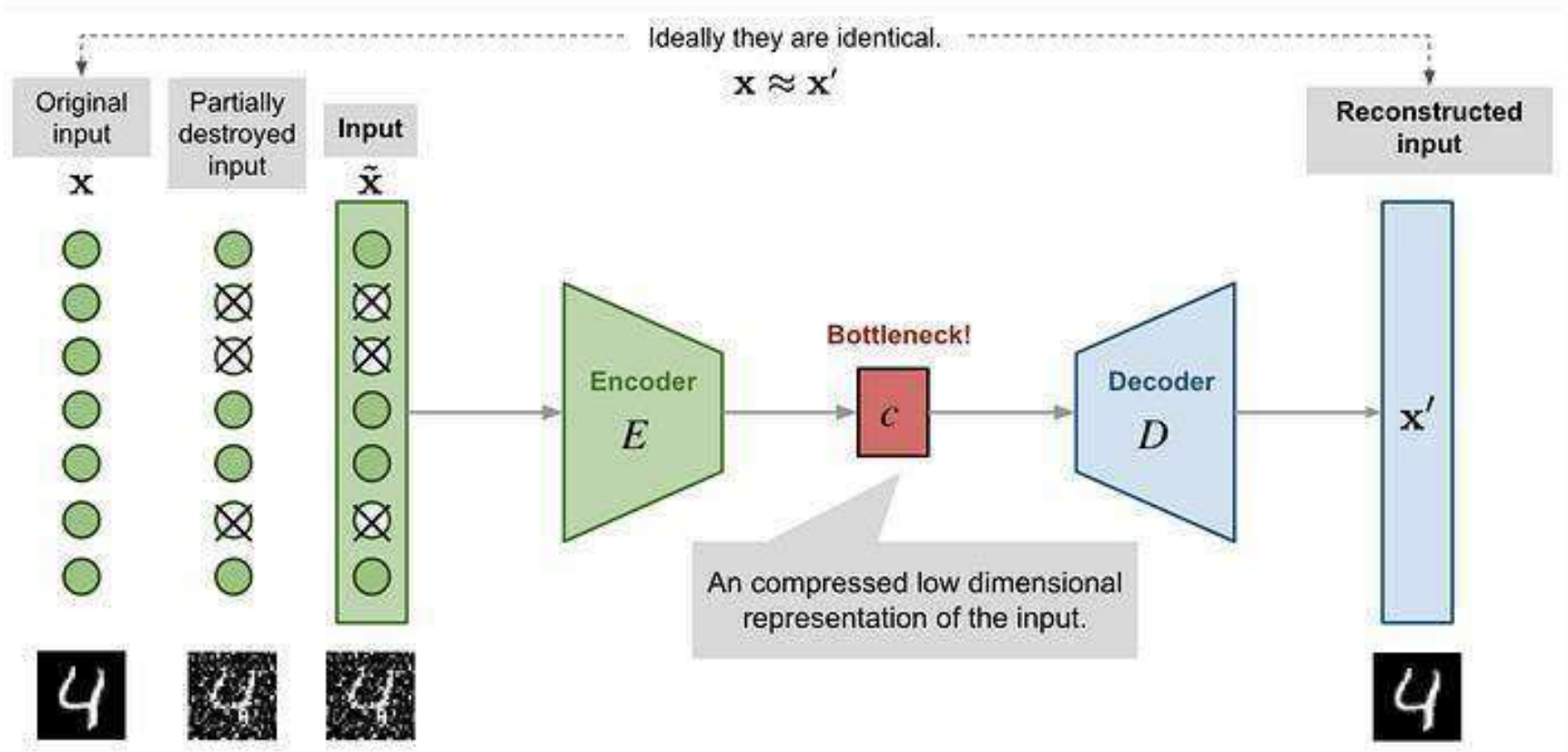
- Why noise?
  - This helps because the objective is still to reconstruct the original (uncorrupted) xi
  - It no longer makes sense for the model to copy the corrupted˜ xi into h(˜ xi) and then into ˆxi (the objective function will not be minimized by doing so)
  - Instead the model will now have to capture the characteristics of the data correctly.



For example, it will have to learn to reconstruct a corrupted x_{ij} correctly by relying on its interactions with other elements of $\mathbf{x}_i$

Topics in Deep Learning

Example of Denoising Autoencoders



Topics in Deep Learning

Uses:Denoising Autoencoders

- De-noising autoencoders are useful when dealing with data that is corrupted. Therefore, the main application of such autoencoders is to reconstruct corrupted data.
- The inputs to the autoencoder are corrupted training records, and the outputs are the uncorrupted data records.

- As a result, the autoencoder learns to recognize the fact that the input is corrupted, and the true representation of the input needs to be reconstructed.

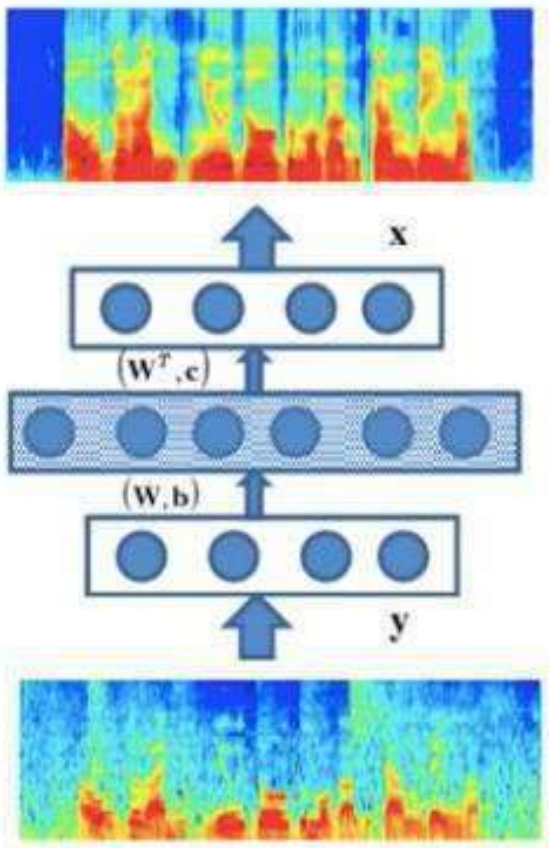
- Therefore, even if there is corruption in the test data (as a result of application-specific reasons), the approach is able to reconstruct clean versions of the test data.

- Note that the noise in the training data is explicitly added, whereas that in the test data is already present as a result of various application-specific reasons.

Applications of Autoencoders

DAE for Speech Enhancement

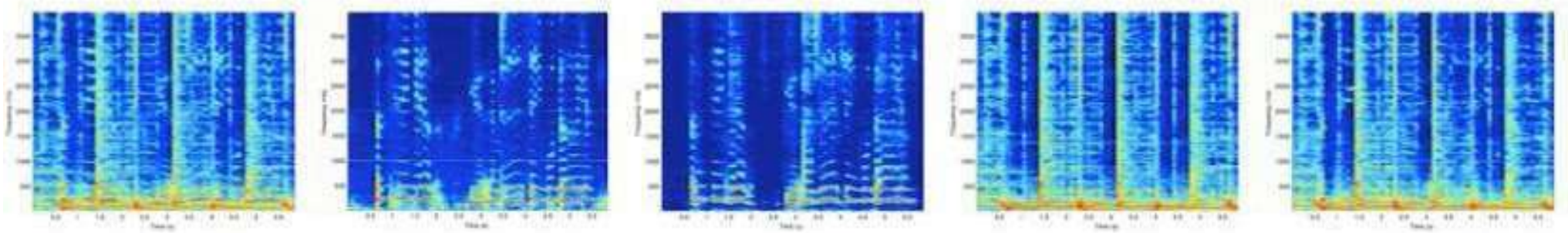
- **Features:** Raw audio or frequency features such as STFT, MFCC, etc.
- **Training:** DAE will get noisy audio as input and aim to construct noiseless audio as output
- **Testing:** Noisy audio input will be processed where DAE will loose the noises in the latent representation and clean audio will come as output in decoder.
- **Data:** Noisy and noiseless versions of same audio
- What if you have only noiseless audio?
  - Solution: Add some noise through augmentation and generate
- What if you have only noisy audio?
  - Solution: Manual/semi automated cleaning of audio



DAE based Speech Enhancer

DAE for Music Voice Separation

- Music can be model as noise where voice can be modelled as target
- Or voice can be modelled as noise where music can be predicted as target
- Recent Example: MaD TwinNet for music voice separation



(a) Mixutre (b) Clean vocal (c) Recovered vocal (d) Clean music (e) Recovered music

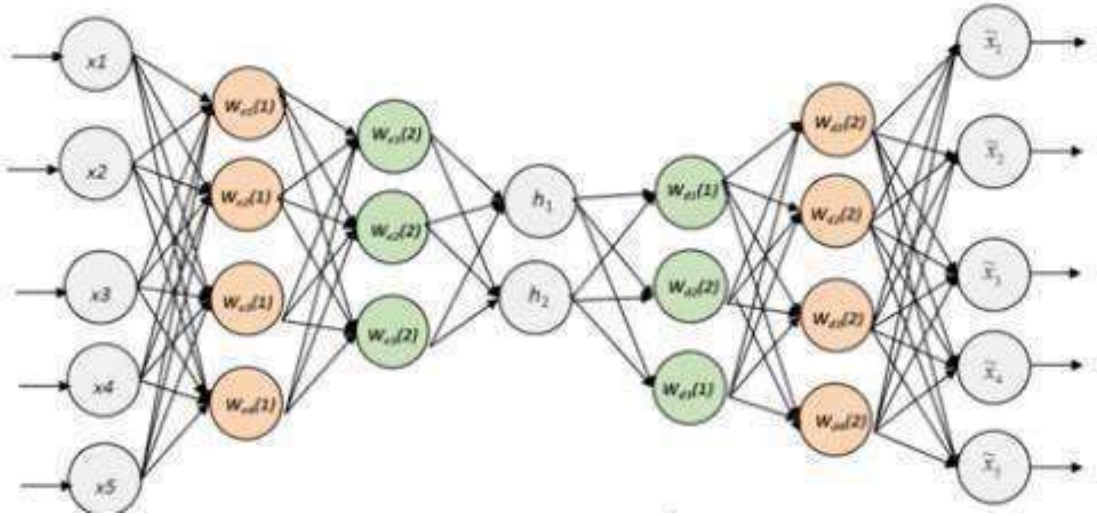
Output of DAE that separates voice from music

Topics in Deep Learning

Types of Autoencoders- Deep Autoencoders

Deep Autoencoder

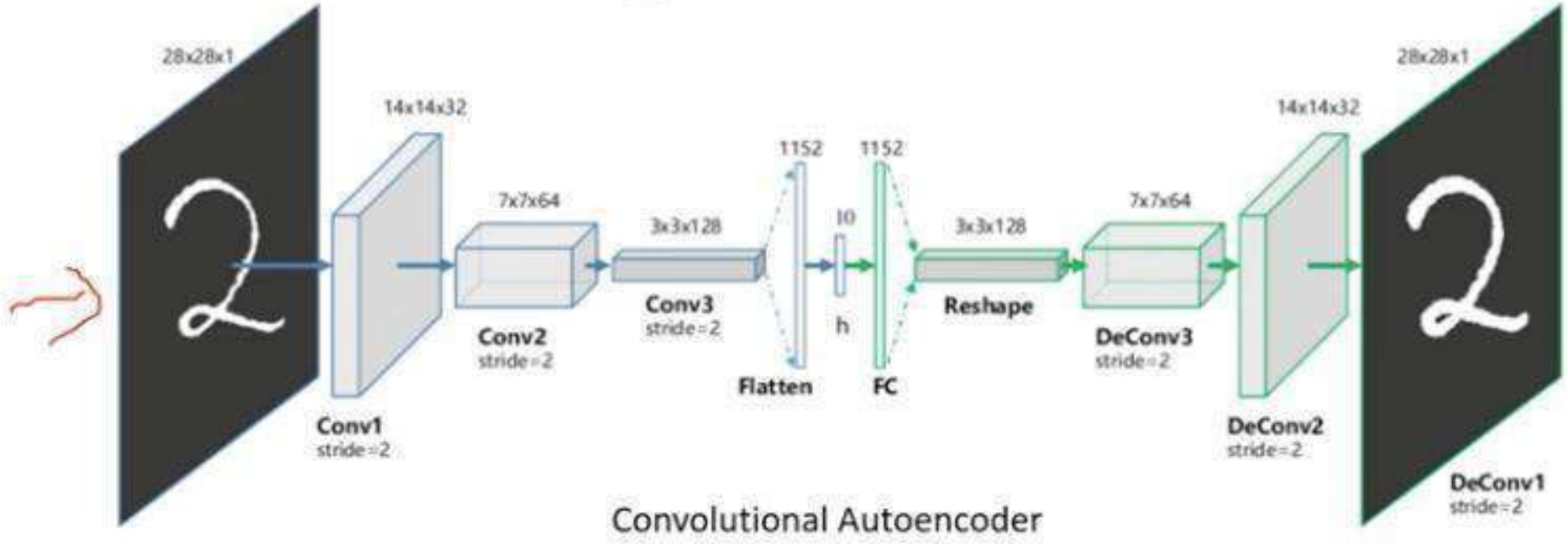
- Deep Neural Network with multilayer encoder and decoder can be trained
- Any number of layers and their nodes can be chosen but usually symmetry is preferred
- Output activation depends upon input values X



Types of Autoencoders- Convolutional Autoencoders

Convolutional Autoencoder

- ConvAE comprises encoder with convolution pooling and decoder with deconvolution and up sampling.
- Referred as Hourglass Model



Convolutional Autoencoder

Acknowledgements & References

- http://www.cse.iitm.ac.in/~miteshk/CS7015/Slides/Handout/Lecture21.pdf’
- http://www.cse.iitm.ac.in/~miteshk/CS7015/Slides/Handout/Lecture7.pdf
- https://medium.com/@lmayrandprovencher/building-an-autoencoder-with-tied-weights-in-keras-c4a559c529a2
- https://medium.com/@syoya/what-happens-in-sparse-autencoder-b9a5a69da5c6

Acknowledgements & References

- http://www.cse.iitm.ac.in/~miteshk/CS7015/Slides/Handout/Lecture21.pdf’
- http://www.cse.iitm.ac.in/~miteshk/CS7015/Slides/Handout/Lecture7.pdf
- https://medium.com/@lmayrandprovencher/building-an-autoencoder-with-tied-weights-in-keras-c4a559c529a2
- https://medium.com/@syoya/what-happens-in-sparse-autencoder-b9a5a69da5c6
- https://www.jeremyjordan.me/variational-autoencoders/
- https://towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf

Graph Neural Network - GNN

Topics in Deep Learning

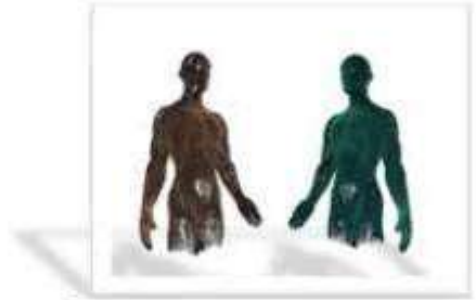
Motivation

Graph Data is everywhere

Medicine / Pharmacy

Recommender Systems

Social Networks



3D Games / Meshes

Motivation

Imagine you are working for a social media company that wants to recommend new friends to its users.

- We can use other methods of deep learning for social media recommendation,
- One issue with traditional neural networks is that they are designed to process fixed-size data, such as images or text. However, social networks are inherently variable in size and structure, which makes it difficult to use traditional neural networks
- Another issue with traditional neural networks is that they may not be able to effectively capture the relational information that is present in the social network.
- Furthermore, social network graphs are usually sparse, meaning that there are many missing connections between nodes.

Motivation

One way to do this is to analyze the social network graph, which is a graph where each node represents a user, and each edge represents a connection between two users (for example, if they are friends or follow each other).

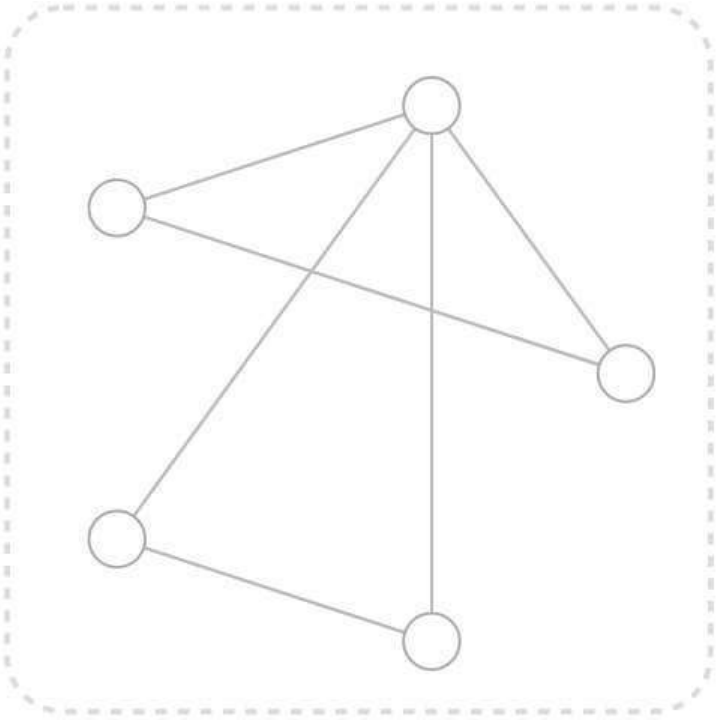
- To recommend new friends, you need to analyze the graph and identify users who are similar to each other based on their connections. However, this is not a straightforward task because the graph is large and complex, and the connections between users are constantly changing.
- This is where Graph Neural Networks (GNNs) come in. GNNs are a type of machine learning model that can be used to analyze graph-structured data like social network graphs. They can identify patterns in the graph and make predictions based on those patterns.

For example, a GNN can learn to predict which users are likely to be friends based on their shared connections in the graph. It can also learn to recommend new friends to users based on their similarities to other users in the graph.

Topics in Deep Learning

Graph Neural Network - GNN

To start, let’s establish what a graph is. A graph represents the relations (edges) between a collection of entities (nodes).



V Vertex (or node) attributes

e.g., node identity, number of neighbors

E Edge (or link) attributes and directions

e.g., edge identity, edge weight

U Global (or master node) attributes

e.g., number of nodes, longest path

Topics in Deep Learning

GNN

- To further describe each node, edge or the entire graph, we can store information in each of these pieces of the graph.

- We can additionally specialize graphs by associating directionality to edges (directed, undirected).

Topics in Deep Learning

Graphs and where to find them (some examples)

Images as graphs

- Another way to think of images is as graphs with regular structure, where each pixel represents a node and is connected via an edge to adjacent pixels. Each non-border pixel has exactly 8 neighbors, and the information stored at each node is a 3-dimensional vector representing the RGB value of the pixel.

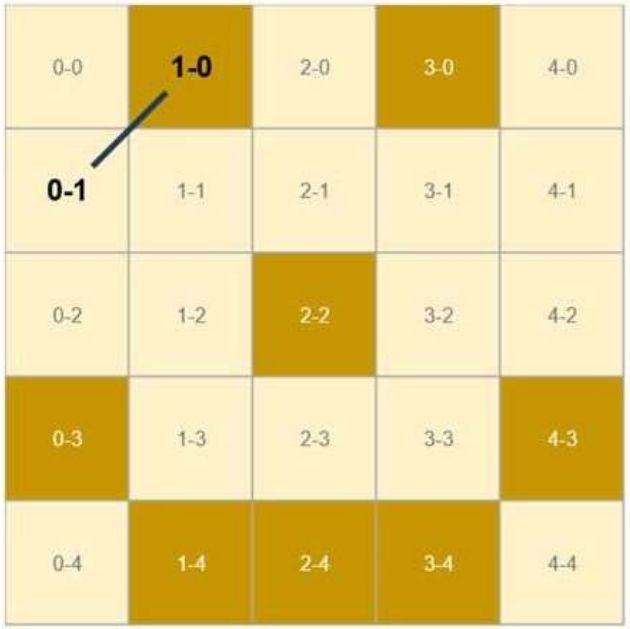
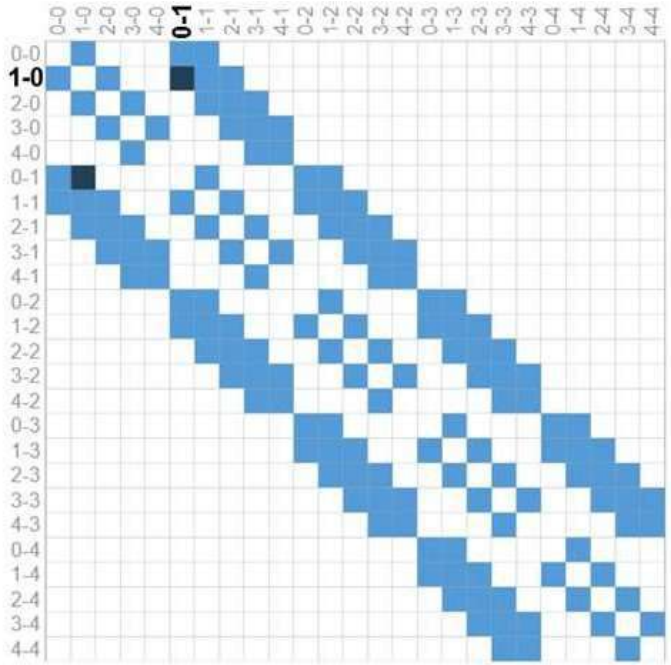
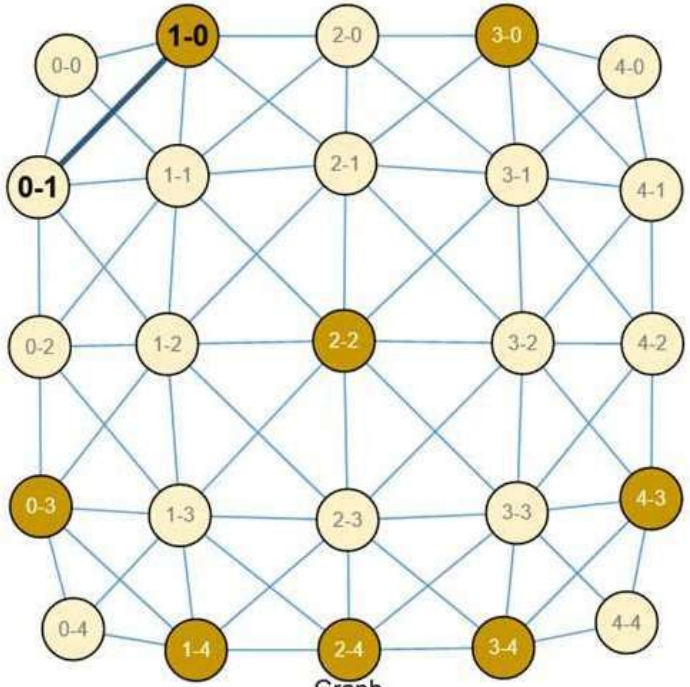


Image Pixels



Adjacency Matrix



Graph

Topics in Deep Learning

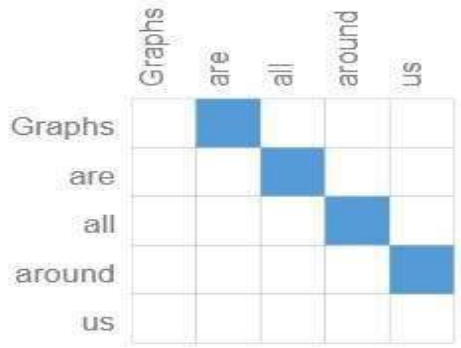
Graphs and where to find them (some examples)

https://distill.pub/2021/gnn-intro/

Graphs and where to find them (some examples)

Text as graphs

- We can digitize text by associating indices to each character, word, or token, and representing text as a sequence of these indices. This creates a simple directed graph, where each character or index is a node and is connected via an edge to the node that follows it.

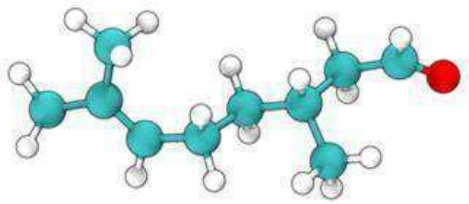


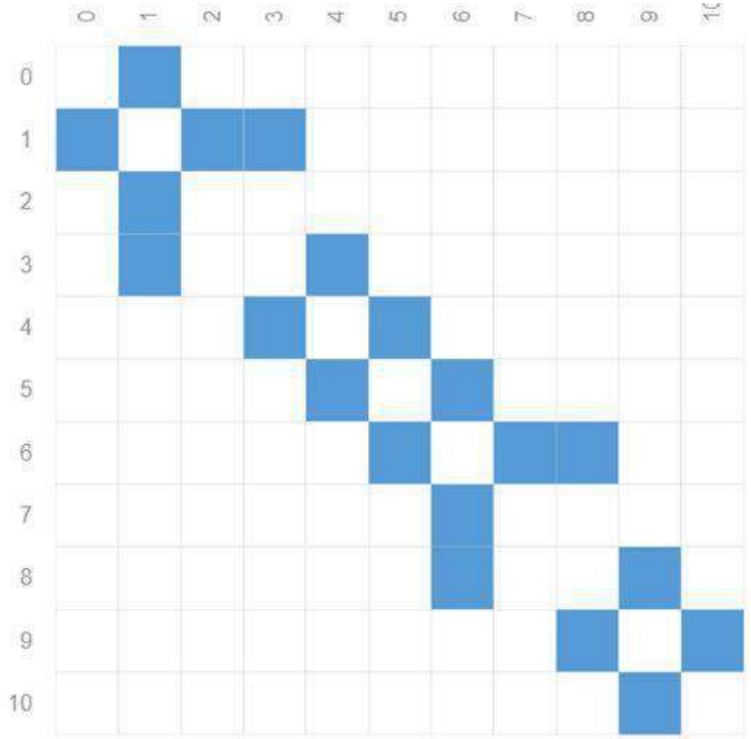
Topics in Deep Learning

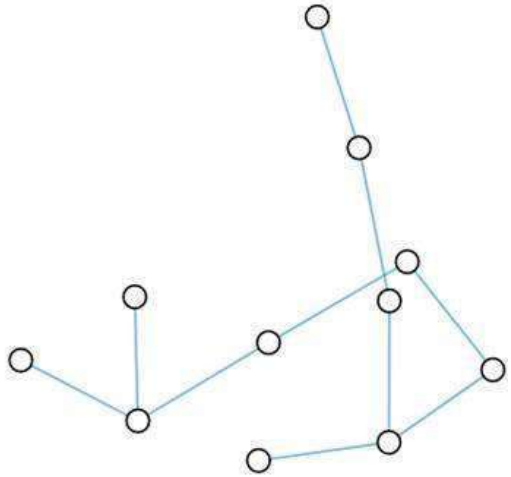
Graphs and where to find them (some examples)

Molecules as graphs

- It’s a very convenient and common abstraction to describe this 3D object as a graph, where nodes are atoms and edges are covalent bonds







(Left) 3d representation of the Citronellal molecule (Center) Adjacency matrix of the bonds in the molecule (Right) Graph representation of the molecule.

Topics in Deep Learning

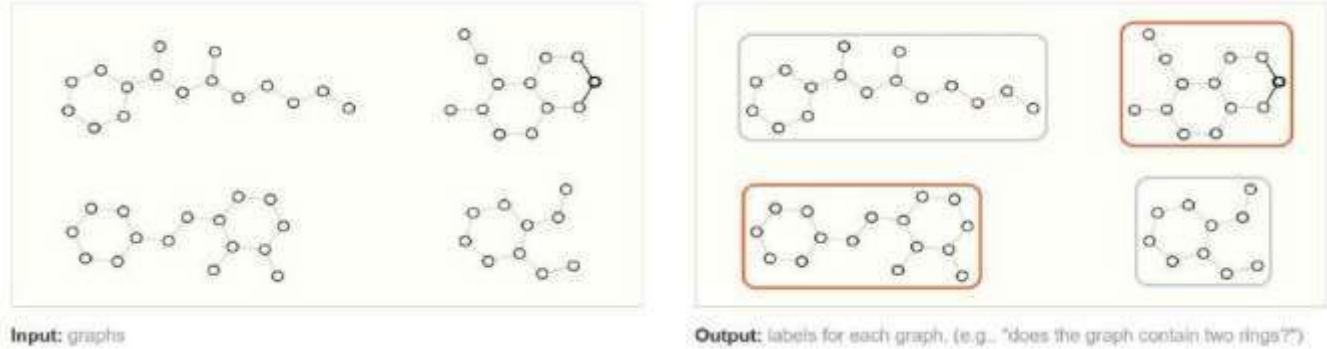
Graphs tasks

There are three general types of prediction tasks on graphs: graph-level, node-level, and edge-level.

- In a **graph-level task**, we predict a single property for a whole graph.
- For **a node-level task**, we predict some property for each node in a graph.
- For **an edge-level task**, we want to predict the property or presence of edges in a graph.

Graph level

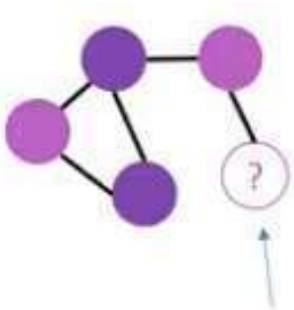
In a graph-level task, our goal is to predict the property of an entire graph. For example, for a molecule represented as a graph, we might want to predict what the molecule smells like, or whether it will bind to a receptor implicated in a disease.



Topics in Deep Learning

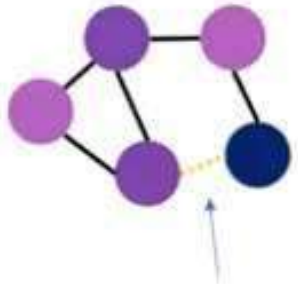
Three general types of prediction tasks on graphs: graph-level, node-level, and edge-level.

Node-level predictions



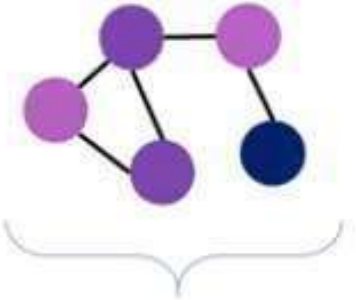
Does this person smoke? (unlabeled node)

Edge-level predictions (Link prediction)



Next Netflix video?

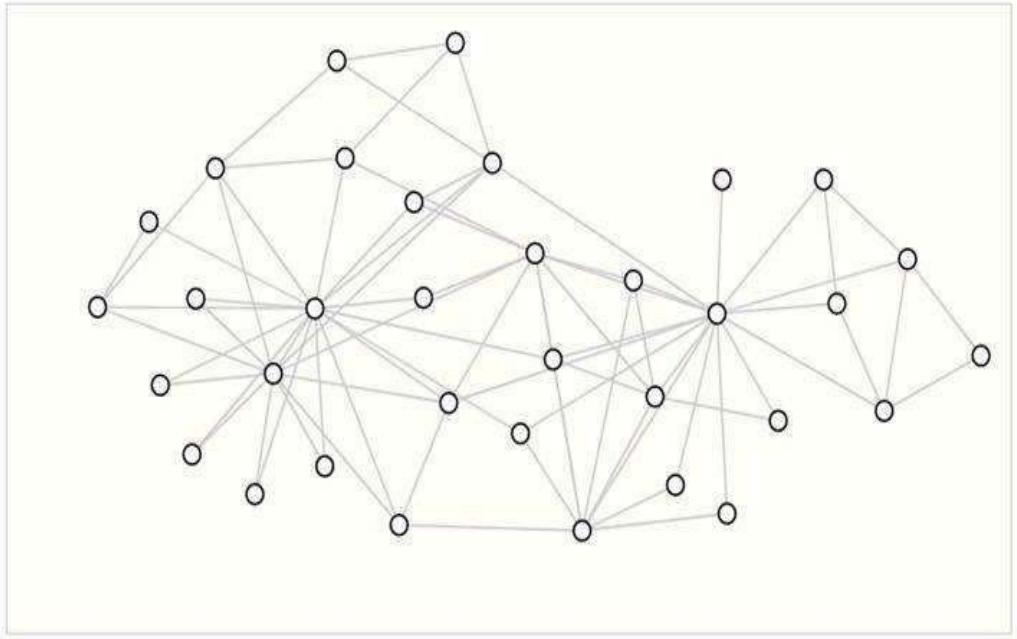
Graph-level predictions

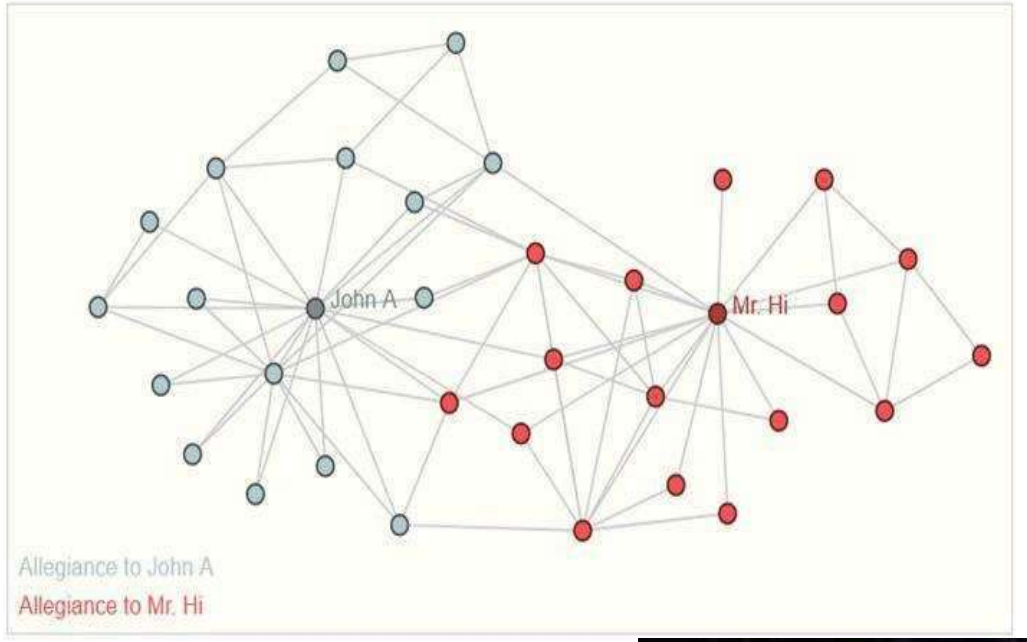


Is this molecule a suitable drug?

Topics in Deep Learning

Graphs tasks- Node level tasks





Input: graph with unlabled nodes

Output: graph node labels

Topics in Deep Learning

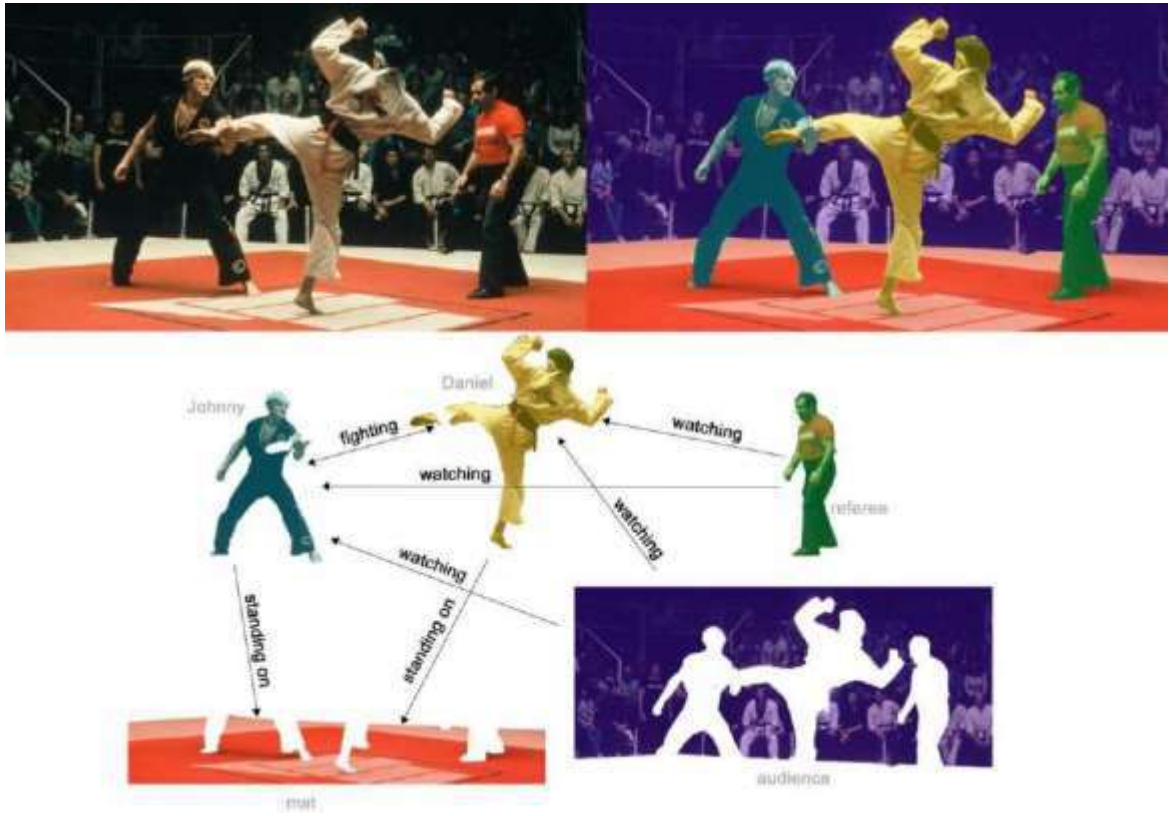
Graphs tasks- Edge level tasks

One example of edge-level inference is in image scene understanding.

Beyond identifying objects in an image, deep learning models can be used to predict the relationship between them.

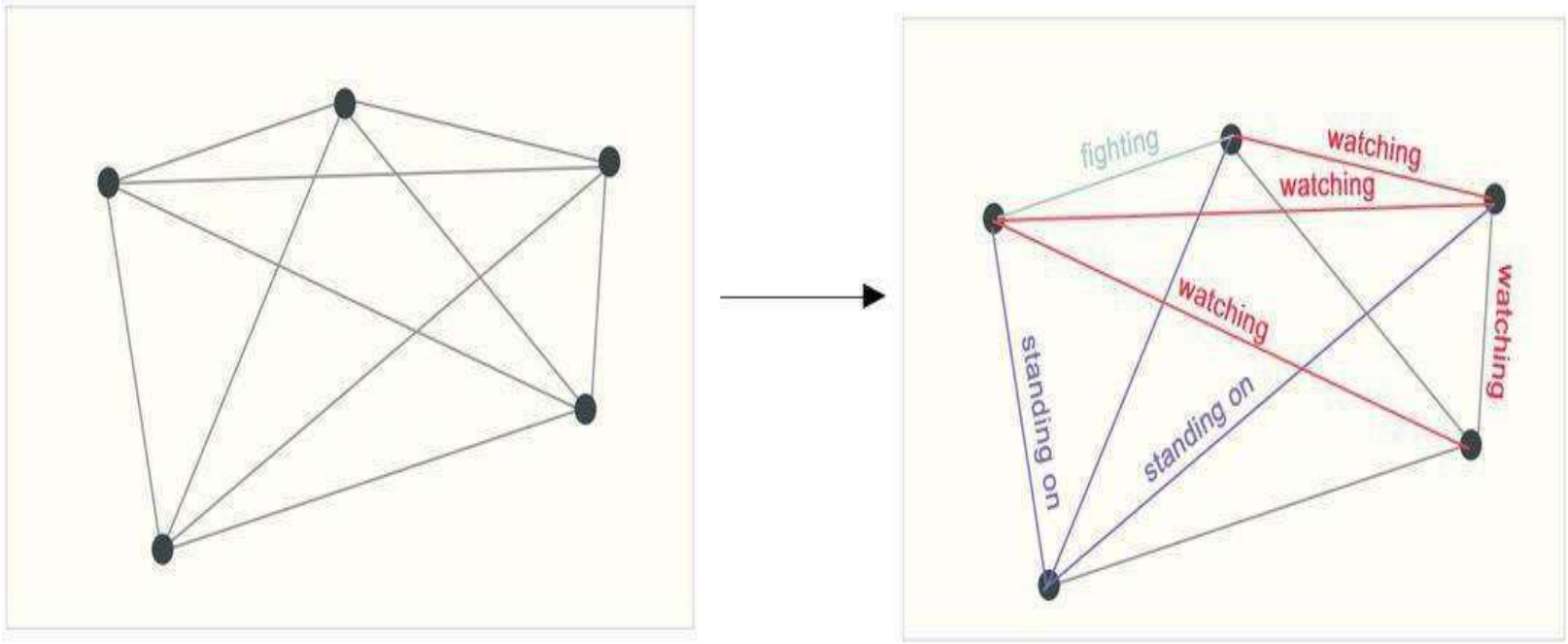
We can phrase this as an edge-level classification: given nodes that represent the objects in the image, we wish to predict which of these nodes share an edge or what the value of that edge is.

If we wish to discover connections between entities, we could consider the graph fully connected and based on their predicted value prune edges to arrive at a sparse graph.



Topics in Deep Learning

Graphs tasks- Edge level tasks



Input: fully connected graph, unlabeled edges

Output: labels for edges

On the left we have an initial graph built from the previous visual scene. On the right is a possible edge-labeling of this graph when some connections were pruned based on the model's output.

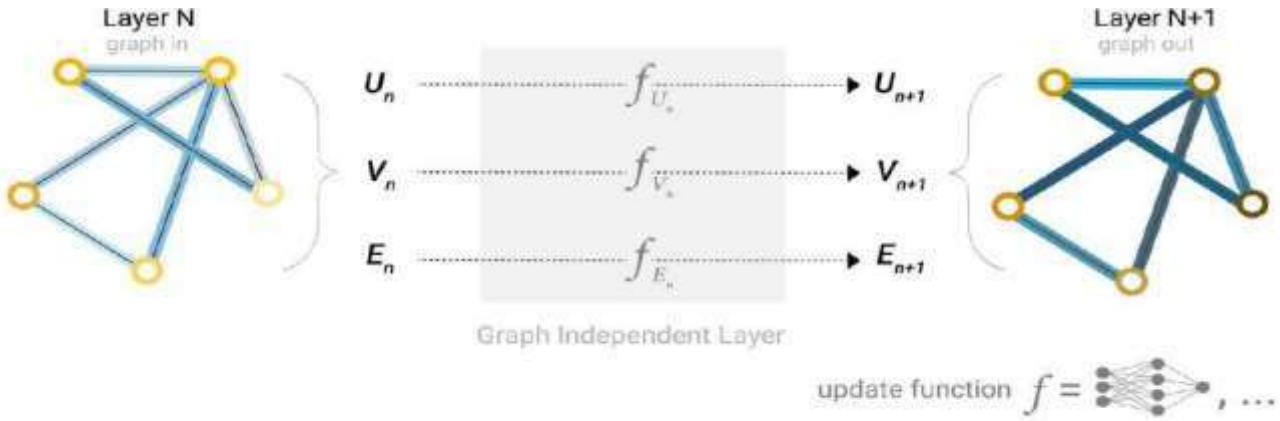
Topics in Deep Learning

GNN

A GNN is an optimizable transformation on all attributes of the graph (nodes, edges, global-context) that preserves graph symmetries (permutation invariances)

- GNNs adopt a “graph-in, graph-out” architecture meaning that these model types accept a graph as input, with information loaded into its nodes, edges and global-context, and progressively transform these embeddings, without changing the connectivity of the input graph.

Here is a simple GNN



A single layer of a simple GNN. A graph is the input, and each component (V,E,U) gets updated by a MLP to produce a new graph. Each function subscript indicates a separate function for a different graph attribute at the n-th layer of a GNN model.

Topics in Deep Learning

GNN - Basic overview of the GNN architecture-message passing neural network" framework

A basic GNN architecture consists of the following components:

1. **Node Embedding**: Each node in the graph is represented as a low-dimensional vector, called a node embedding. These embeddings capture the features of the node, such as its attributes, connections, or position in the graph.

2. **Message Passing**: The key idea behind GNNs is to update the node embeddings by aggregating information from their neighboring nodes. This is done through a message passing algorithm, where each node sends and receives messages to and from its neighbors, and updates its embedding based on the received messages. The messages typically consist of the embeddings of neighboring nodes and some edge-specific information.

3. **Node Aggregation**: After receiving messages from their neighbors, each node aggregates the received information into a new embedding vector, by applying some aggregation function, such as summation or averaging.

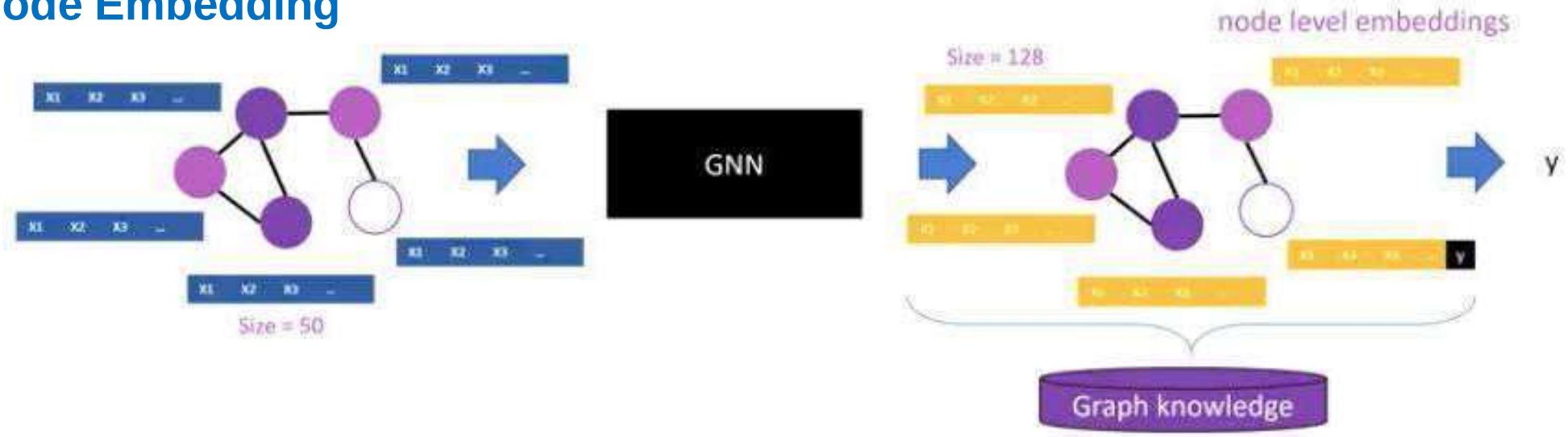
4. **Readout**: Finally, the aggregated node embeddings are used to make predictions or solve the task at hand. This is done through a readout function that maps the aggregated embeddings to the desired output.

The above steps are typically repeated for multiple iterations or layers, each time updating the node embeddings and aggregating information from a larger neighborhood of nodes.

Topics in Deep Learning

GNN- How do Graph Neural Networks work?

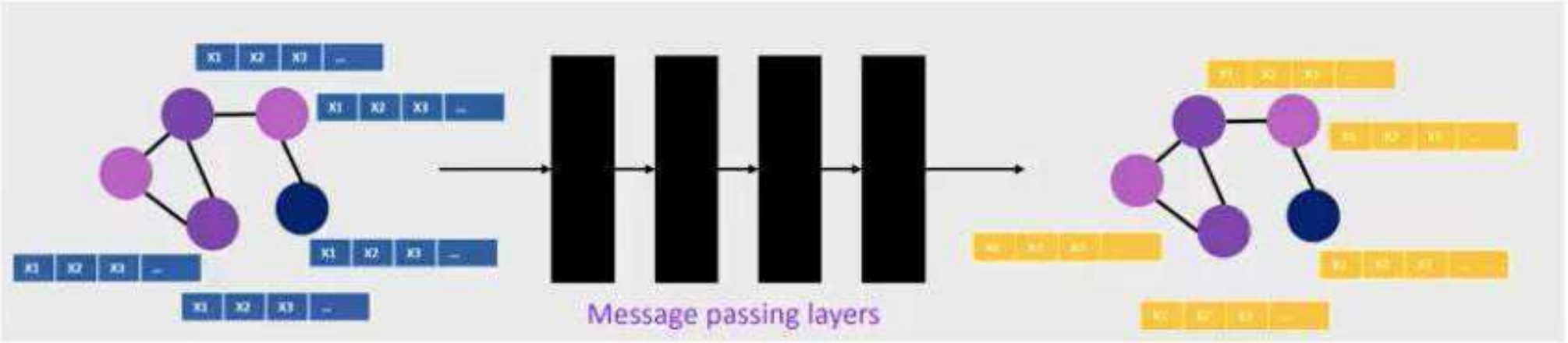
1. Node Embedding



Each node in the graph is represented as a low-dimensional vector, called a node embedding. These embeddings capture the features of the node, such as its attributes, connections, or position in the graph.

GNN- How do Graph Neural Networks work?

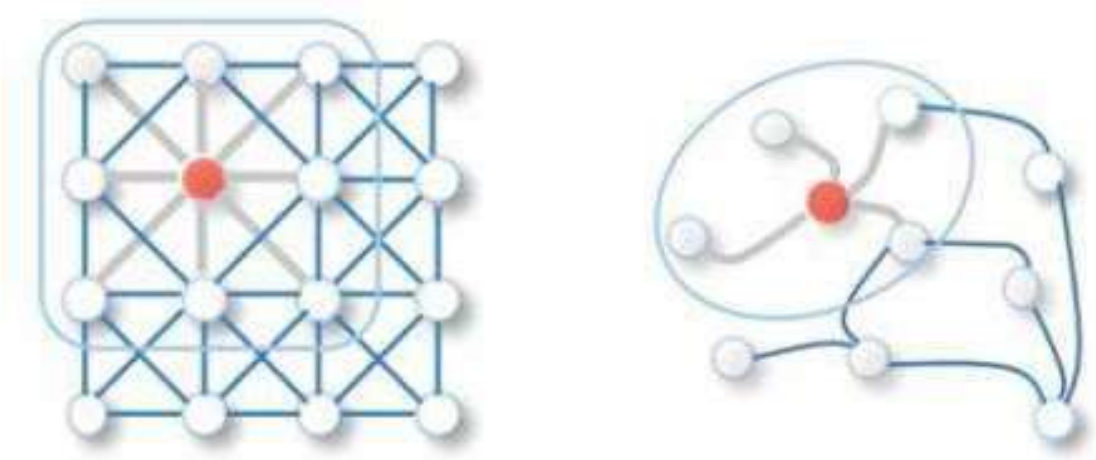
2.Message Passing:



The key idea behind GNNs is to update the node embeddings by aggregating information from their neighboring nodes. This is done through a message passing algorithm, where each node sends and receives messages to and from its neighbors, and updates its embedding based on the received messages. The messages typically consist of the embeddings of neighboring nodes and some edge-specific information.

GNN- Steps performed on every node

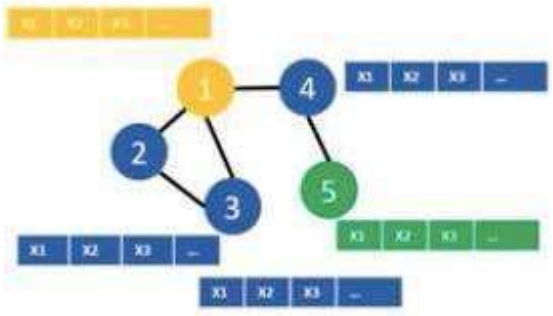
What is happening in the message passage layers?



Similar to filter in CNN

GNN- Steps performed on every node

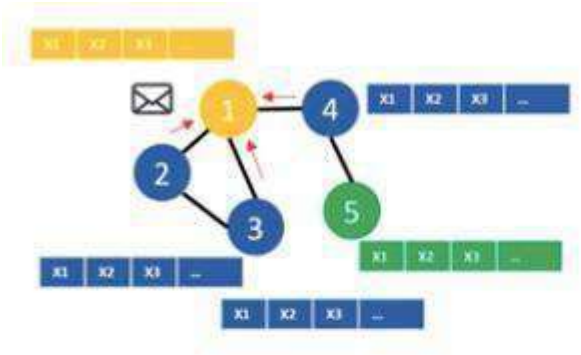
What is happening in the message passage layers?

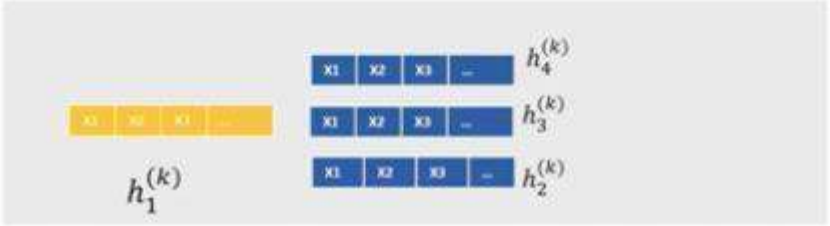


Topics in Deep Learning

GNN- Steps performed on every node

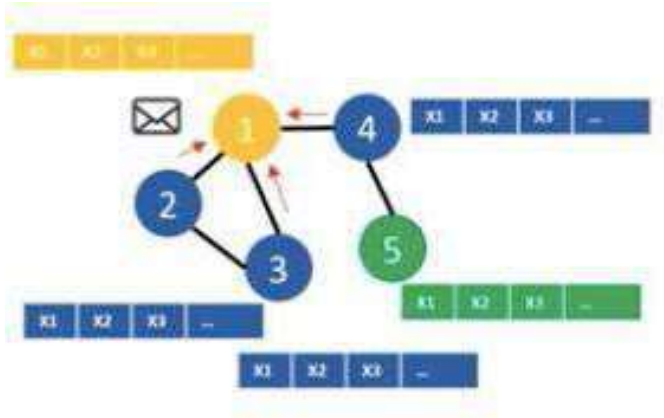
What is happening in the message passage layers?

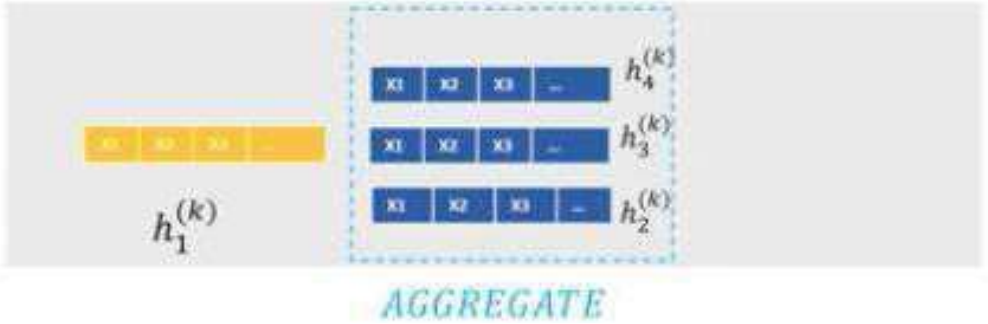




GNN- Steps performed on every node

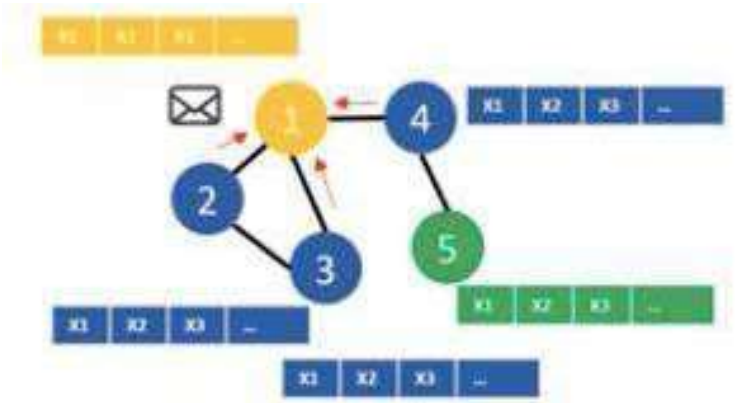
3. Aggregate

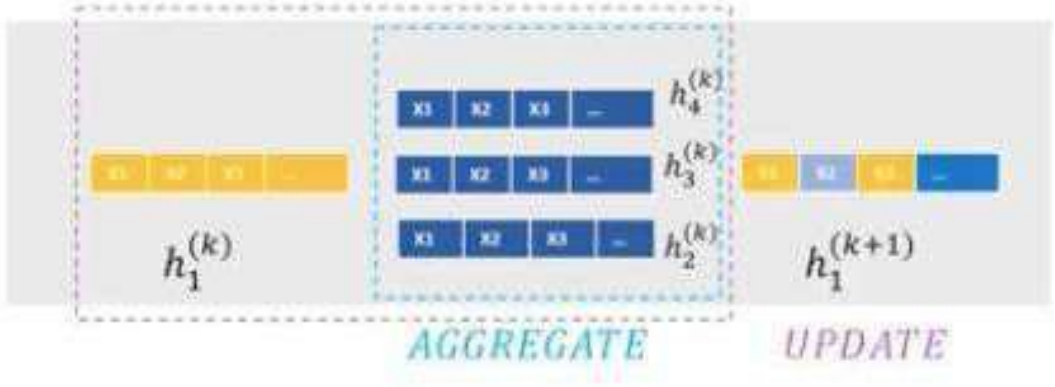




GNN- Steps performed on every node

4. Update

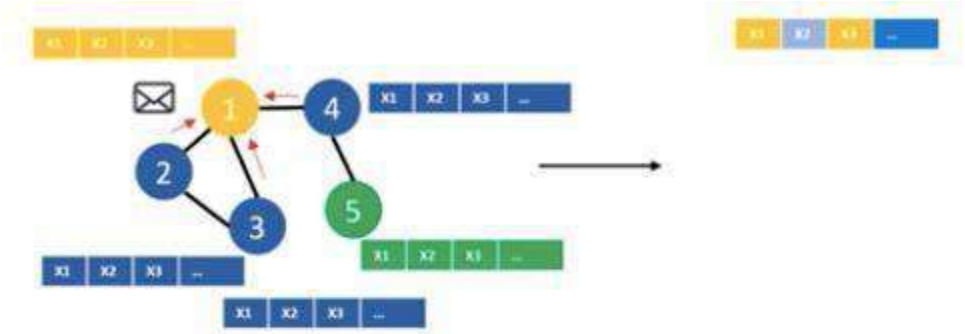


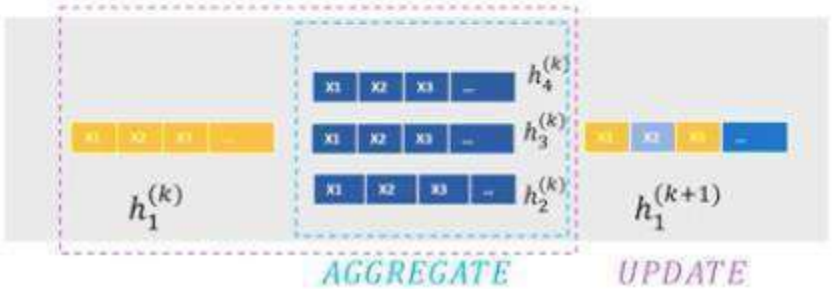


Topics in Deep Learning

GNN- Steps performed on every node

What is happening in the message passage layers?

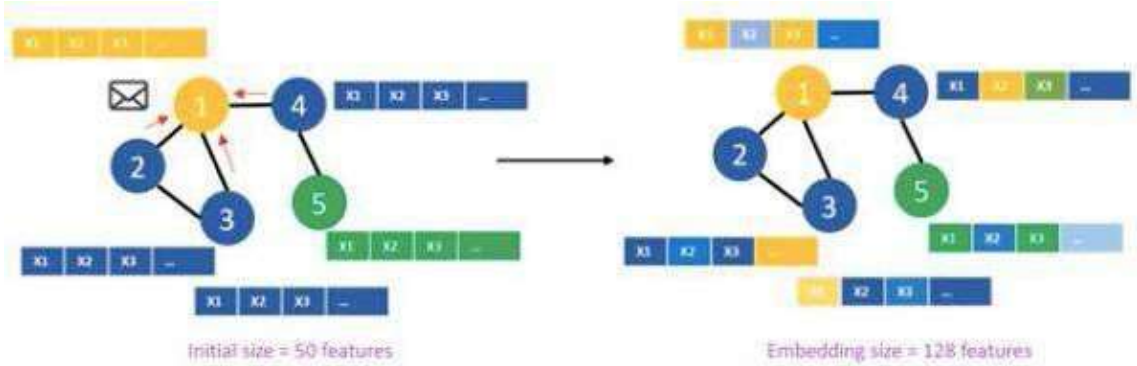


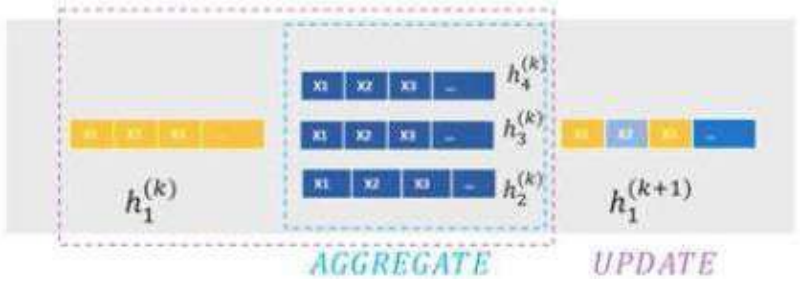


Topics in Deep Learning

GNN- Steps performed on every node

What is happening in the message passage layers?

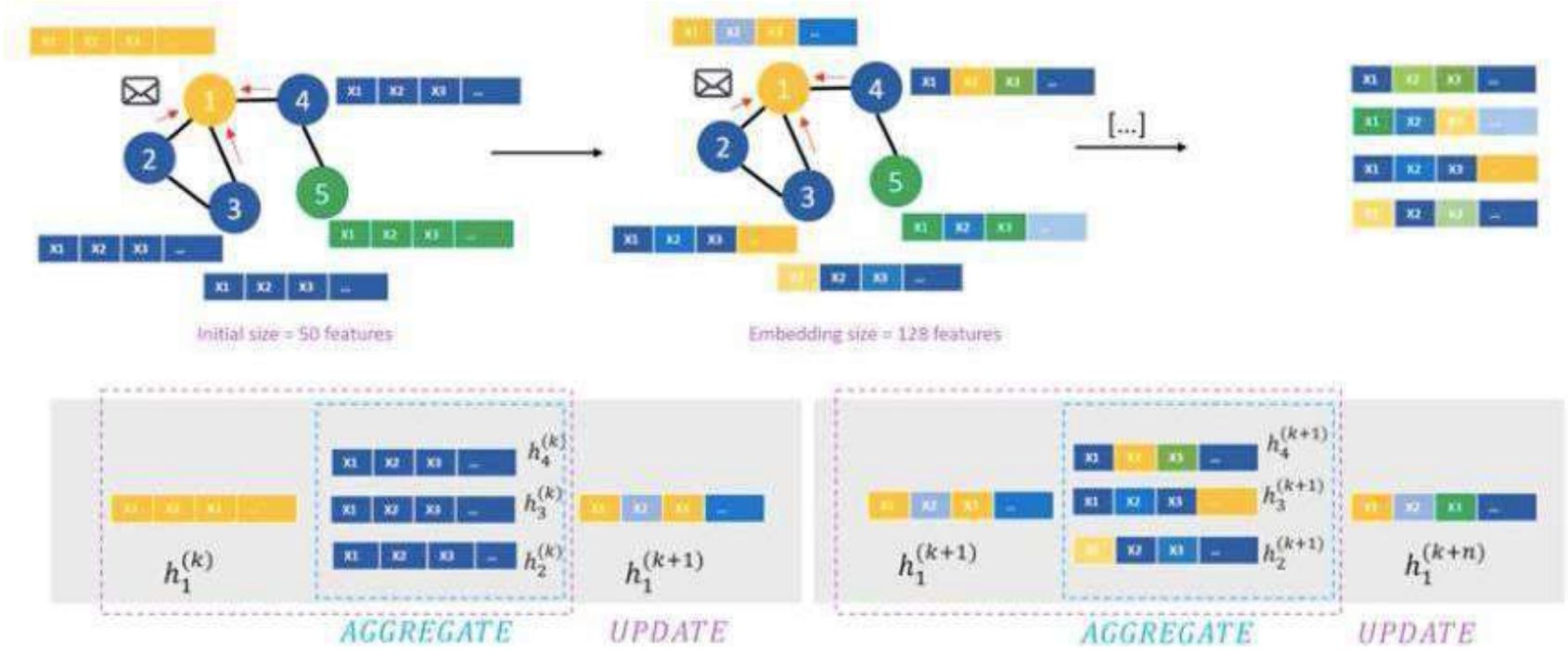




Topics in Deep Learning

GNN- Steps performed on every node

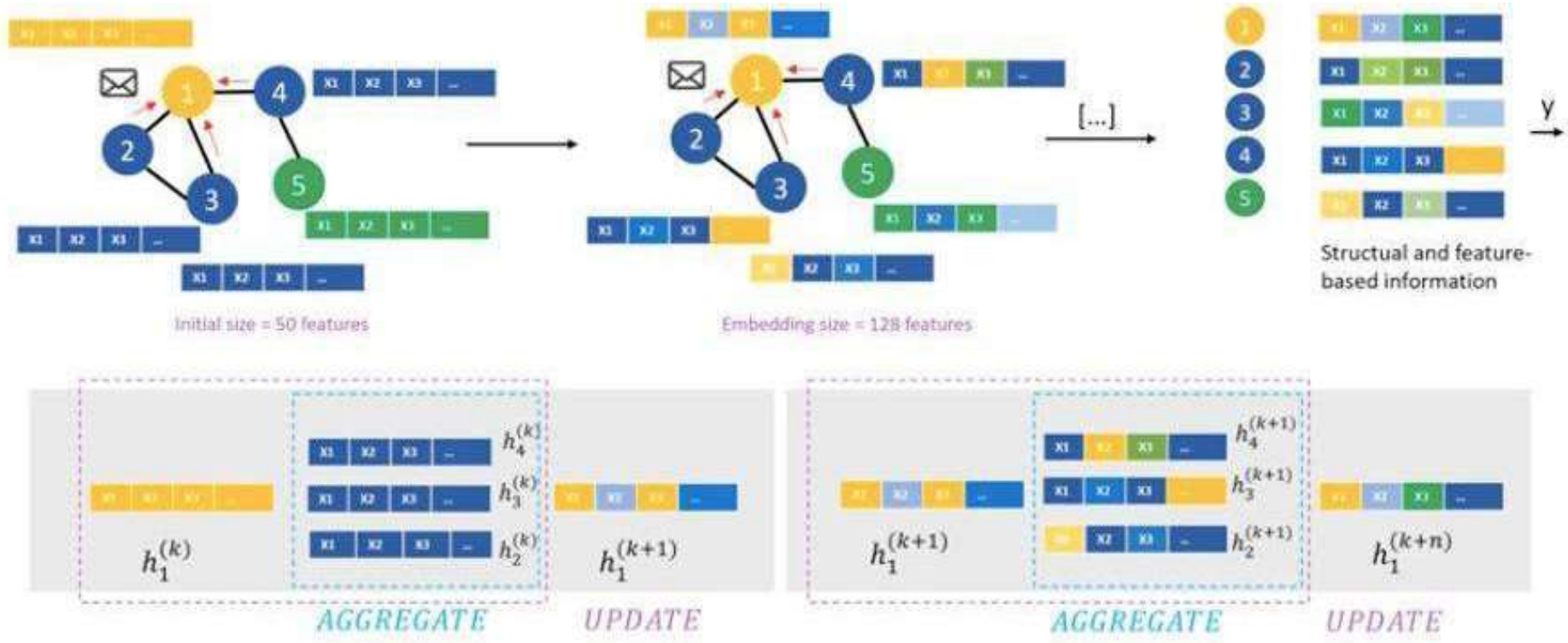
What is happening in the message passage layers?



Topics in Deep Learning

GNN- Steps performed on every node

What is happening in the message passage layers?



Topics in Deep Learning

GNN- Steps performed on every node

A single Graph Neural Network (GNN) layer has a bunch of steps that's performed on every node in the graph:

1. **Message Passing**
2. **Aggregation**
3. **Update**

Together, these form the building blocks that learn over graphs. Innovations in GDL mainly involve changes to these 3 steps.

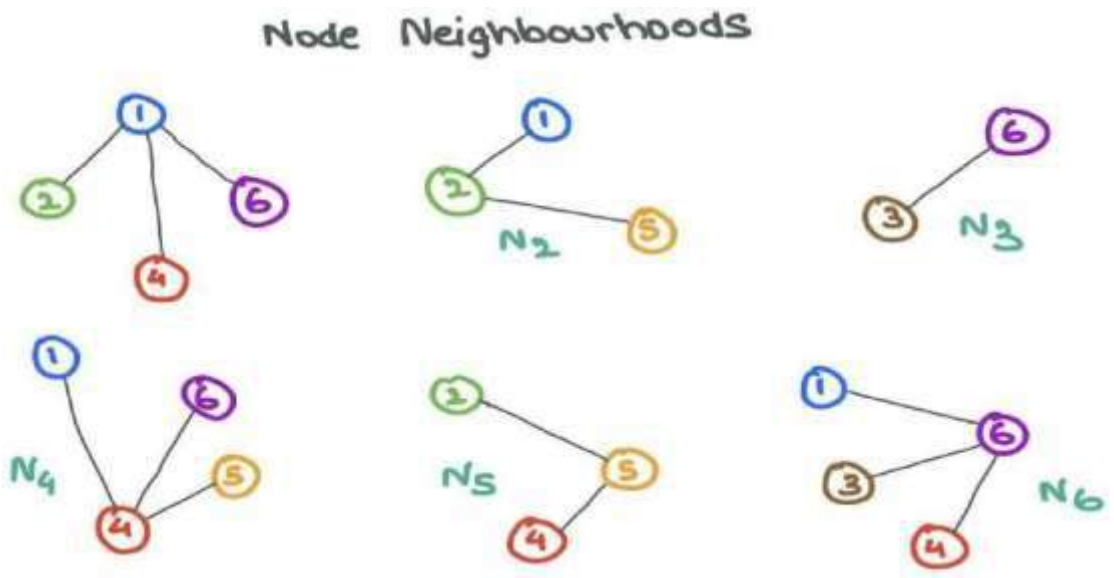
- Each node has features which can be represented using vectors in Rd. This vector is either a latent-dimensional embedding or is constructed in a way where each entry is a different property of the entity.
  - For instance, in a social media graph, a user node has the properties of age, gender, political inclination, relationship status, etc. that can be represented numerically.
  - Likewise, in a molecule graph, an atom node might have chemical properties like affinity to water, forces, energies, etc. that can also be represented numerically.
  - Formally, every node i has associated node features $x_i \in R^d$ and labels y_i

Topics in Deep Learning

GNN - Message passing

GNNs are known for their ability to learn structural information. Usually, nodes with similar features or properties are connected to each other (this is true in the social media setting). The GNN exploits this fact and learns how and why specific nodes connect to one other while some do not. To do so, the GNN looks at the Neighbourhoods of nodes.

*The **Neighbourhood** $\mathcal{N}_i$ of a node i is defined as the set of nodes j connected to i by an edge. Formally, $\mathcal{N}_i = \{j \ : \ e_{ij} \in E\}$.*

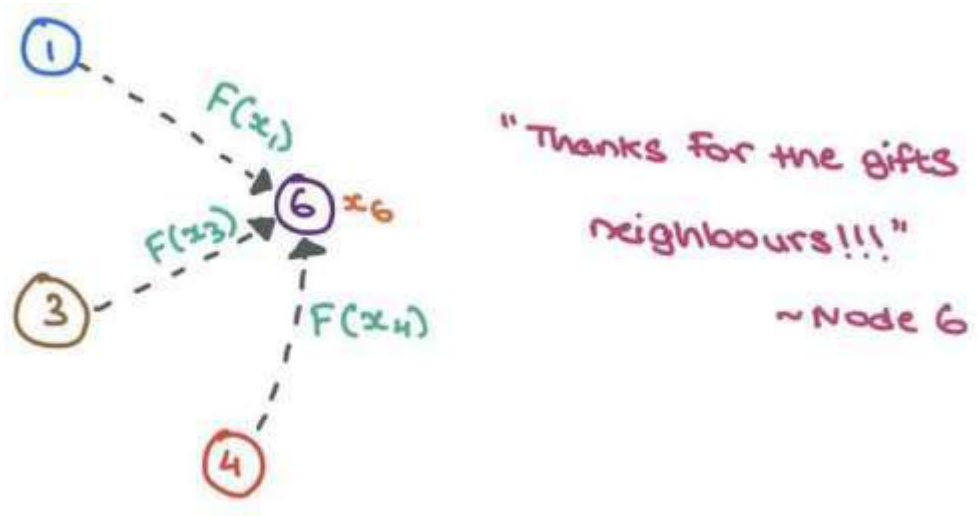


Topics in Deep Learning

GNN - Message passing

A person is shaped by the circle he is in. Similarly, a GNN can learn a lot about a node i by looking at the nodes in its neighbourhood **N_i**. To enable this sharing of information between a source node i and its neighbours j, GNNs engage in **Message Passing**.

For a GNN layer, Message Passing is defined as the process of taking node features of the neighbours, transforming them, and **"passing"** them to the source node. This process is repeated, in parallel, for all nodes in the graph. In that way, all neighbourhoods are examined by the end of this step.



Topics in Deep Learning

GNN - Message passing

Let's zoom into node 6 and examine the neighbourhood N6={1, 3, 4}. We take each of the node features x1, x3, and x4, and transform them using a function F, which can be a simple neural network (MLP or RNN) or affine transform

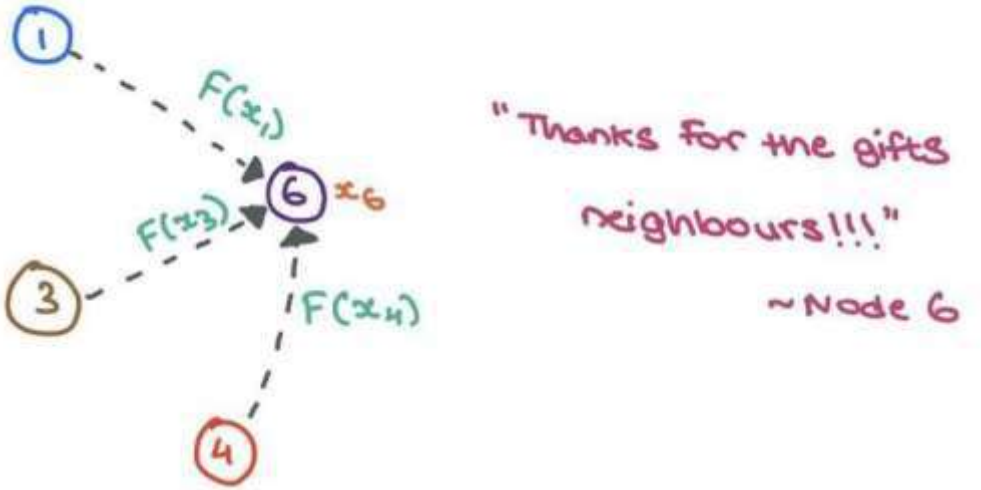
$$F(x_j) = \mathbf{W}_j \cdot x_j + b.$$

Simply put, a "message" is the transformed node feature coming in from source node.

F can be a simple affine transform or neural network.

For now, let's say $F(x_j) = \mathbf{W}_j \cdot x_j$

for mathematical convenience. I



GNN - Aggregation

Aggregation

Now that we have the transformed messages $\{F(x_1), F(x_3), F(x_4)\}$ passed to node 6, we have to aggregate ("combine") them someway. There are many things that can be done to combine them. Popular aggregation functions include,

$$\text{Sum} = \sum_{j \in \mathcal{N}_i} \mathbf{W}_j \cdot x_j \tag{1}$$

$$\text{Mean} = \frac{\sum_{j \in \mathcal{N}_i} \mathbf{W}_j \cdot x_j}{|\mathcal{N}_i|} \tag{2}$$

$$\text{Max} = \max_{j \in \mathcal{N}_i}(\{\mathbf{W}_j \cdot x_j\}) \tag{3}$$

$$\text{Min} = \min_{j \in \mathcal{N}_i}(\{\mathbf{W}_j \cdot x_j\}) \tag{4}$$

Suppose we use a function G to aggregate the neighbours' messages (either using sum, mean, max, or min). The final aggregated messages can be denoted as follows:

$$\bar{m}_i = G(\{\mathbf{W}_j \cdot x_j : j \in \mathcal{N}_i\}) \tag{5}$$

Topics in Deep Learning

GNN - Update

Using these aggregated messages, the GNN layer now has to update the source node i's features. At the end of this update step, the node should not only know about itself but its neighbours as well. This is ensured by taking the node i's feature vector and combining it with the aggregated messages. Again, a simple addition or concatenation operation takes care of this.

Using addition:

$$h_i = \sigma(K(H(x_i) + \bar{m}_i))) \tag{6}$$

where σ is an activation function (ReLU, ELU, Tanh), H is a simple neural network (MLP) or affine transform, and K is another MLP to project the added vectors into another dimension.

Using concatenation:

$$h_i = \sigma(K(H(x_i) \oplus \bar{m}_i))) \tag{7}$$

To abstract this update step further, we can think of K as some projection function that transforms the messages and source node embedding together:

$$h_i = \sigma(K(H(x_i), \bar{m}_i))) \tag{8}$$

👉 Notation-wise, the initial node features are called x_i.

After a forward pass through the first GNN layer, we call the node features h_i instead. Suppose we have more GNN layers, we can denote the node features as h_i^l where l is the current GNN layer index. Also, it's evident that $h_i^0 = x_i$ (i.e., the input to the GNN).

Topics in Deep Learning

GNN - Message passing

Putting Them Together

Now that we've gone through the Message Passing, Aggregation, and Update steps, let's put them all together to formulate a single GNN layer on a single node i:

$$h_i = \sigma(W_1 \cdot h_i + \sum_{j \in \mathcal{N}_i} \mathbf{W}_2 \cdot h_j) \tag{9}$$

Here, we use the `sum` aggregation and a simple feed-forward layer as functions F and H.

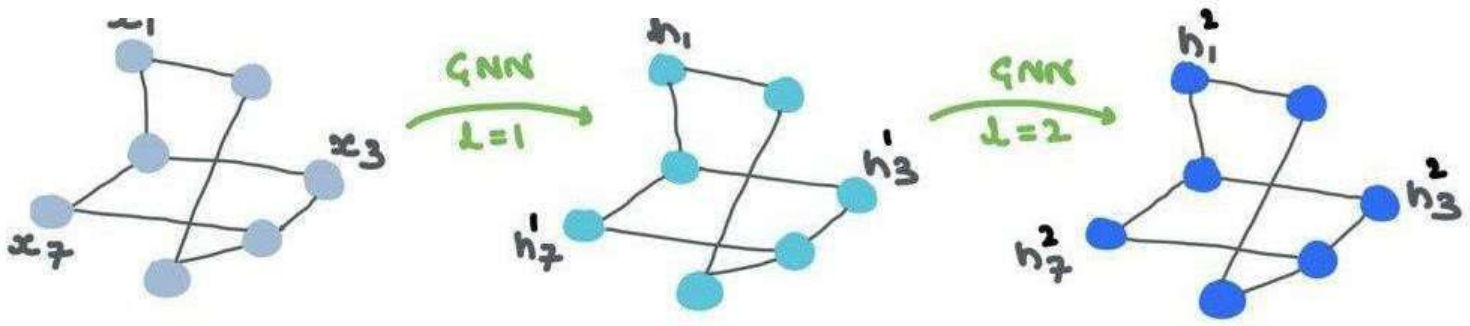
⚠ Do ensure that the dimensions of $\mathbf{W}_1$ and $\mathbf{W}_2$ commute properly with the node embeddings. If $h_i \in \mathbb{R}^d$, $\mathbf{W}_1, \mathbf{W}_2 \subseteq \mathbb{R}^{d' \times d}$ where d' is the embedding dimension.

Topics in Deep Learning

GNN - Stacking layers

Now that we've figured out how single GNN layers work, how we build a whole "network" of these layers? How does information flow between the layers and how the GNN *refine* the embeddings/representations of the nodes (and/or edges)?

1. The input to the first GNN layer is the node features $X \subseteq \mathbb{R}^{N \times d}$. The output is the intermediate node embeddings $H^1 \subseteq \mathbb{R}^{N \times d_1}$ where d_1 is the first embedding dimension. H^1 is made up of $h^1_{i:1 \to N} \in \mathbb{R}^{d_1}$.
2. H^1 is the input to the second layer. The next output is $H^2 \subseteq \mathbb{R}^{N \times d_2}$ where d_2 is the second layer's embedding dimension. Likewise, H^2 is made up of $h^2_{i:1 \to N} \in \mathbb{R}^{d_2}$.
3. After a few layers, at the output layer L, the output is $H^L \subseteq \mathbb{R}^{N \times d_L}$. Finally, H^L is made up of $h^L_{i:1 \to N} \in \mathbb{R}^{d_L}$.

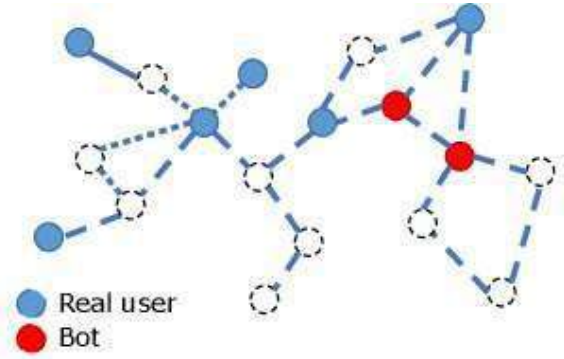


GNN – Problems GNN can solve

Node Classification

- One of the powerful applications of GNNs is adding new information to nodes or filling gaps where information is missing.
- In node classification, the task is to predict the node embedding for every node in a graph, i.e to determine the labelling of samples (represented as nodes) by looking at the labels of their neighbours
- This type of problem is usually trained in a semi-supervised way, where only part of the graph is labelled.
- Typical applications for node classification include citation networks, Reddit posts, YouTube videos, etc.

For example, say you are running a social network and you have spotted a few bot accounts. Now you want to find out if there are other bot accounts in your network. You can train a GNN to classify other users in the social network as “bot” or “not bot” based on how close their graph embeddings are to those of the known bots.



(a) Bot detection task

Topics in Deep Learning

GNN - Aplications

Graph Neural Networks (GNNs) have a wide range of applications in various domains. Here are some examples:

Social network analysis:

- GNNs can be used for various tasks in social network analysis, such as node classification, link prediction, community detection, and recommendation systems.
- For example, in node classification, the task is to predict the category or label of a node in the network based on its connections to other nodes. GNNs can be used to learn node representations that capture the structural information of the network, which can then be used to classify the nodes.
- Similarly, in link prediction, the task is to predict whether there will be a link between two nodes in the future.
- GNNs can be used to learn representations of the nodes and the links between them, which can then be used to predict whether a link will exist or not.

Topics in Deep Learning

GNN - Aplications

Graph Neural Networks (GNNs) have a wide range of applications in various domains. Here are some examples:

Chemistry and drug discovery:

- GNNs can be used for various tasks in chemistry and drug discovery, such as molecular property prediction, drug design, and chemical reaction prediction.
- For example, in molecular property prediction, the task is to predict the properties of a molecule, such as its solubility, toxicity, or bioactivity.
- GNNs can be used to learn representations of the atoms and bonds in the molecule, which can then be used to predict the molecule's properties.
- Similarly, in drug design, the task is to design new molecules that can bind to a specific target protein.
- GNNs can be used to generate new molecules by iteratively modifying existing ones and evaluating their properties using a GNN-based molecular property predictor.

GNN – Aplications- Natural language processing:

- GNNs can be used for various tasks in natural language processing, such as text classification, sentiment analysis, and machine translation.
- For example, in text classification, the task is to classify a text document into one or more predefined categories. GNNs can be used to learn representations of the words or phrases in the document, which can then be used to classify the document.
- Similarly, in sentiment analysis, the task is to determine the sentiment or emotion expressed in a text document.
- GNNs can be used to learn representations of the words or phrases and their relationships with other words or phrases in the document, which can then be used to predict the sentiment.

GNN – Aplications- Computer vision and image processing:

- GNNs can be used for various tasks in computer vision and image processing, such as image segmentation, object detection, and image captioning.
- For example, in image segmentation, the task is to partition an image into regions that correspond to different objects or parts of objects.
- GNNs can be used to propagate information between adjacent pixels or regions to refine the segmentation. Similarly, in object detection, the task is to detect the presence and location of objects in an image.
- GNNs can be used to learn representations of the objects and their relationships with other objects in the image, which can then be used to detect the objects.

GNN- Advantages

1. **Ability to handle structured data**: GNNs are designed to work with structured data such as graphs, which is a common representation for many real-world problems, such as social network analysis, recommendation systems, and molecular design.
2. **Incorporation of graph topology**: GNNs can capture the graph topology by using the node connections as a way to propagate information through the network, which makes them more powerful than traditional neural networks that treat input data as a set or a sequence.
3. **Transfer learning**: GNNs can leverage the pre-trained graph representations on one task and apply them to another related task. This can save computational resources and improve performance on the target task.
4. **Interpretable**: The message passing process in GNNs can be interpreted as passing information between nodes in a graph, which makes them more interpretable than traditional neural networks.

GNN- Disadvantages

1. **Computational complexity**: GNNs can be computationally expensive, especially for large graphs with many nodes and edges. This can make training and inference time-consuming and resource-intensive.
2. **Limited scalability**: GNNs may not be suitable for extremely large graphs due to memory limitations and computational complexity.
3. **Difficulty in handling dynamic graphs**: GNNs assume that the graph structure is fixed and known a priori. Handling dynamic graphs, where the structure of the graph changes over time, is still an active area of research.
4. **Lack of standardization**: There is currently no standard architecture or training procedure for GNNs, which can make it difficult to compare results across different studies and applications.

Acknowledgements & References

- https://distill.pub/2021/gnn-intro/

- https://rish-16.github.io/posts/gnn-math/

- https://www.youtube.com/watch?v=fOctJB4kVlM&list=PLV8yxwGOxvvoNkzPfCx2i8an--Tkt7O8Z&index=1

- https://www.youtube.com/watch?v=ABCGCf8cJOE&list=PLV8yxwGOxvvoNkzPfCx2i8an--Tkt7O8Z&index=2