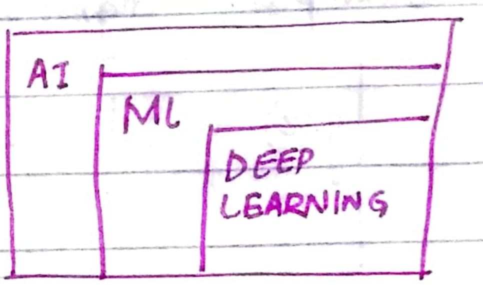
MIT : Introduction to deep learning 6.S191

→ Tensorflow Implementation



AI : Any technique that enables computers to mimic human behavior.

ML: Ability to learn without beeing explicitly programmed.

Deep learning (DL) : Extract raw patterns directly from data.

Core Idea of deep learning : Can we learn underlying patterns directly from data ?

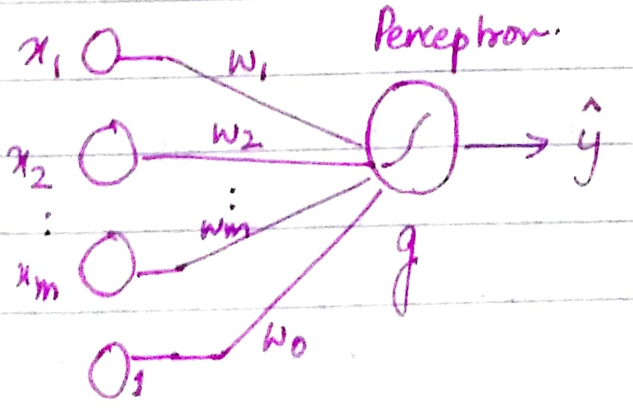
Why Now ? → Data available (collected 4 stored)

→ Availability of computing power

→ Improved software tools (Tensorflow)

Perceptron : The structural building block of deep learning.

(Forward propagation)



output

$$\hat{y} = \underbrace{g}_{\text{Non linear activation function}} \underbrace{\left(w_0 + \sum_{i=1}^{m} x_i w_i \right)}_{\text{linear combi of weights 4 inputs}}$$

'g': Non linear activation functions

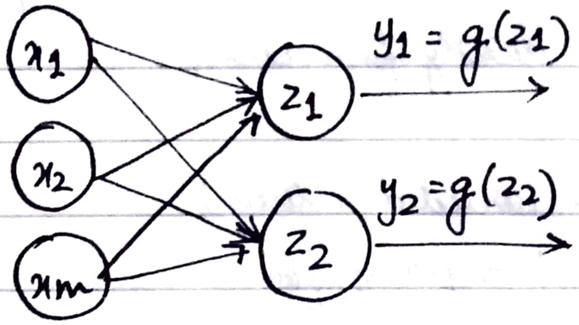
Examples: Sigmoid, tanh and relu.

$\boxed{g(z) = \frac{1}{1+e^{-z}}}$ $\boxed{g(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}}$ $\boxed{g(z) = \max(0,z)}$

SIGMOID TANH RELU.

: purpose is to introduce non-linearity into the network.

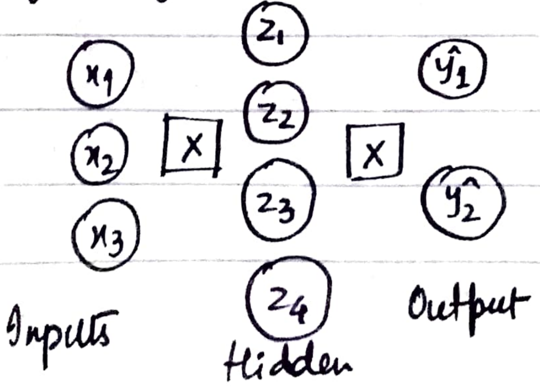
★: <u>Multi Output Perception</u> (2 outputs)



where $z_j = w_{0_j} + \sum_{i=1}^{m} x_i w_{i_j}$

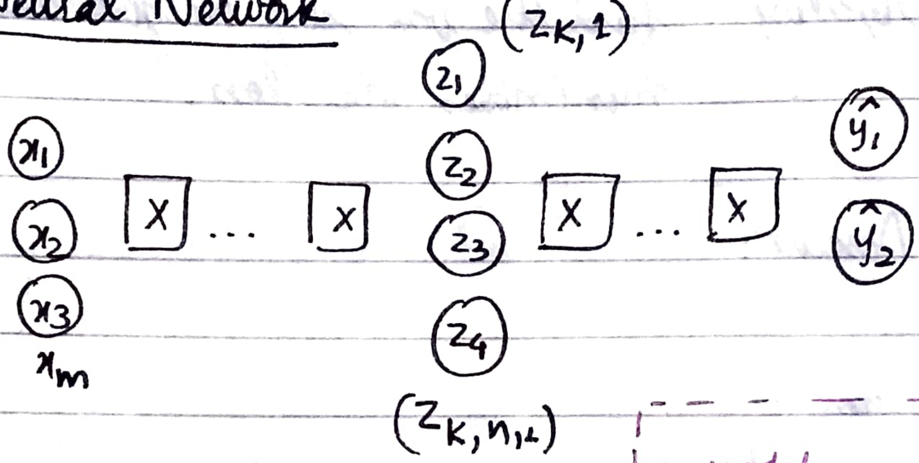
Dense layers: all inputs densely connected to all outputs.

★ <u>Single Layer Neural Network</u> (1 hidden layer)



```
import tensorflow as tf
model = tf.keras. Sequential([
    tf.keras.layers.Dense(n),
    tf.keras. layers Dense(2)
])
```

Deep Neural Network



```
model... ([
   tf.keras.layers.Dense(n1),
   "    "     " .Dense(n2),
   :
   tf.  "     " .Dense(2).
])
```

☆ Applying Neural Networks

eg: Will I pass this class.
x_1 : # of lectures attended
x_2 : # of hours spent in final project.

Training: using the objective function / Loss function as

$$\mathcal{L}\left(\underbrace{f(x^{(i)};W)}_{\text{predicted}},\ \underbrace{y^{(i)}}_{\text{Actual}}\right)$$

for all instances in data:, total loss : average loss across instances

$$J(W) = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}\left(f\left(x^{(i)};W\right), y^{(i)}\right)$$

★ Training Objective : find the best set of weights that minimises total loss.

★ Gradient Descent :

Algorithm :

```
1: Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence
3.     Compute gradient, $\frac{\partial J(w)}{\partial \xi(w)}$
4.     Update weights (in the direction of downward
       slope and update weights)
           $W \leftarrow w - \eta \frac{\partial J(w)}{\partial \xi W}$
5. Return Weights
```

eta : learning rate

★ Computing Gradients : Using Backpropagation

$$\begin{tikzcd} x \arrow[r, "w_1"] & z_1 \arrow[r, "w_2"] & \hat{y}_1 \arrow[r] & J(w) \end{tikzcd}$$

Computing gradient : How much a differential change in w_2 affects the loss.
w.r.t w_2

$$\frac{\partial J(w)}{\partial w_2} = \frac{\partial \hat{y}_1}{\partial w_2} \times \frac{\partial J(w)}{\partial \hat{y}_1}$$

Similary, computing gradient w.r.t w_1.

$$\frac{\partial J(w)}{\partial w_1} = \frac{\partial J(w)}{\partial \hat{y}_1} * \frac{\partial \hat{y}_1}{\partial z_1} * \frac{\partial z_1}{\partial w_1}$$

★ Setting the learning rate:

- decides how fast the algorithm converges
- too low value: can get stuck in local minima
- too large value: can overshoot the minima & diverge.
- setting $\boxed{\text{eta}}$ value
  - → fixed: trial and error
  - → adaptive - lr changes with landscape.
    - SGD - Vanilla GD
    - Adagrad
    - Adam
    - RMSProp.
    - Adadelta

Stochastic Gradient Descent

→ Gradient descent is computationally expensive. Usually a random point is picked & gradient is computed for that iter.

another random point is selected in the next iteration and the gradient is computed again. But it is very noisy. Taking a middle ground - Batch computation.

$$\frac{\partial J(W)}{\partial W} = \frac{1}{\boxed{B}} \sum_{k=1}^{B} \frac{\partial J_k(W)}{\partial W}$$

↓
size of the batch

→ Batch computations helps by
- giving ability to set larger learning rates
- smoother convergence.
- parallelize computation.

☆ Overfitting :

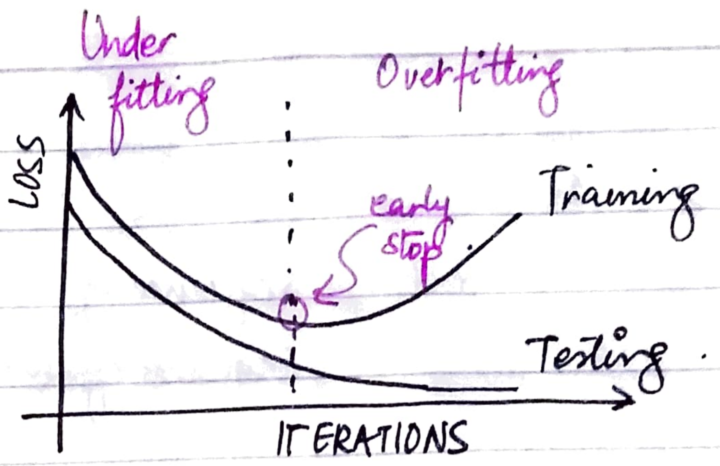
Contain overfitting using regularization.

1. Dropout: Randomly setting some activations to '0'.

tf.keras.layers.Dropout (p=0.5)

- drops 50% of activations in layer (sampled)
- does not memorize data.

2. Early Stopping :



LECTURE: 2 : DEEP SEQUENCE MODELLING

1: Audio
2: Text : Sequence of Characters
sequence of words.

Example: Sequence modelling and predicting the next word given a set of words.

Problem 1: Sentences can be of variable lengths.
Idea 1: Used fixed window (last two words → predict next).
Problem 2: Contextual info from prev. sentence or longer sentences would not be captured
Idea 2: Use bag of words representation (complex set of vocabulary).
Problem 3: Counts don't preserve order:

The food was good, not bad @ all
The food was bad, not good @ all

Same sentences (length 4 words) - diff contextual meaning.
Idea 3: Use really long fixed window
Problem 4: No parameter sharing. A phrase that starts at the start of sentence could not be distingued or would

be considered a different meaning if that phrase starts coming at the end of sentence when parameters applied to a feed forward NN.

eg: this morning took the cat. (1)

. . . . . this morning (2)

(this morning : means the same in two sentences but could be misrepresented as something else by feed forward neural network)

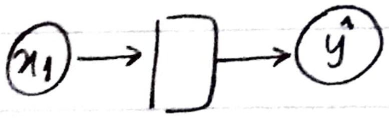
Sequence Modelling Design Criteria

Develop sequence models that

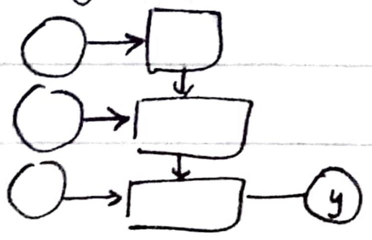
1. can handle variable length sequences
2. Can Track long term dependencies
3. Maintain information about the order.
4. Share parameter across the sequence.

RNN - Recurrent Neural Network.

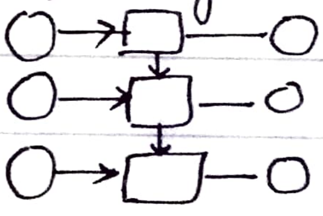
Vanilla NN



Many to One (Sentiment)



Many to Many (Music)



Define state of the cell (Recurrent cell).

$$h_t = f_W(h_{t-1}, x_t)$$

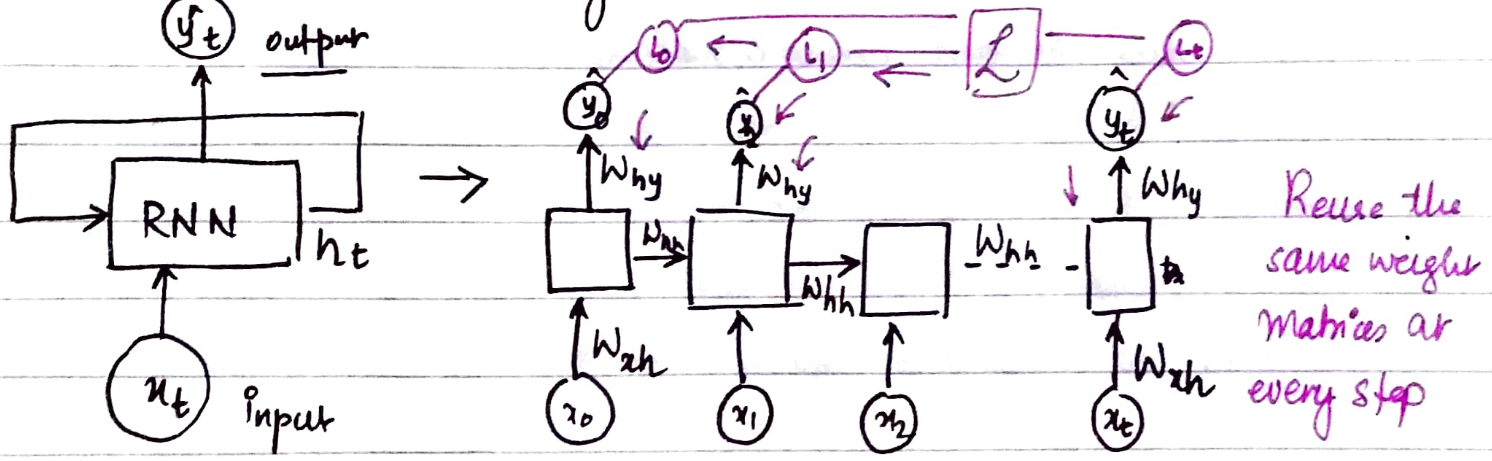
h_t ↓ State at time 't'

h_{t-1} ↓ state @ prev timestep

x_t → inputs @ 't'

f_W - parameterised by W

- Same weight matrices across



Total loss $= L_0 + L_1 + \ldots L_t.$

→ forward pass

← Backward pass.

Given

Input vector → x_t

update hidden state → $h_t = \tanh\left(W_{hh}^T h_{t-1} + W_{xh}^T x_t\right)$

output vector → $\hat{y}_t = W_{hy}^T h_t$

Back Propagation Through Time (BPTT)

→ Gradient computation @
  - each time step
  - across time steps.

→ Repeated multiplication of gradients 4 ~~[illegible]~~ with weight matrix and multiple gradient computations.

→ Problems while computing Gradient

1) Exploding Gradients - many gradients become more than 1,
   Solved by Gradient clipping, ~~or sca~~ i.e scaling back gradients to smaller values.

2) Vanishing Gradients - gradient values become exceedingly smaller. 3 approaches to solve this problem.
   → Activation function
   → intelligent weight initialization
   → Designing robust network architecture.

→ Why is Vanishing Gradient a Problem?

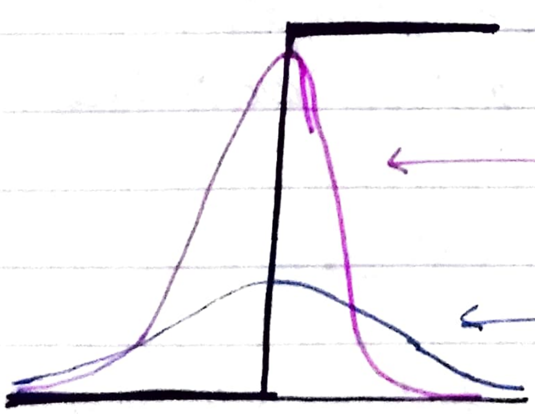
→ Vanishing gradients cause problems with long-term dependencies.

→ Computing gradients requires multiplying repeatedly with the weight matrices.

→ When we multiply many small numbers together
- leads to gradient values becoming very small.
- the error is not propagated back to the network
- leads to capture more short-term signals and thus bias towards recent values seen by the network.
- this is not always a problem, but could be a problem where reference or content from earlier in the sentence is required to make a prediction.

Solving vanishing gradients problem

# Trick 1 :



(Relu prevent f' from shrinking gradient when $x > 0$)

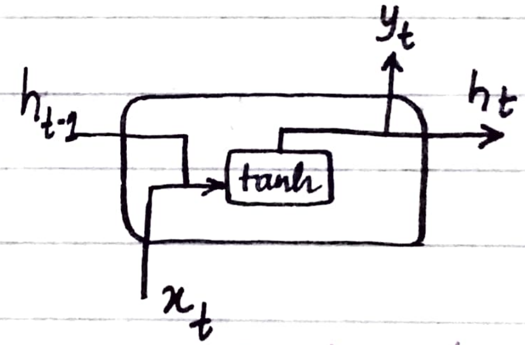
Trick #2 : 1) Initialize weights to identity matrix.
2) Initialize biases to zero.

Trick #3 : Gated Cells : More Robust

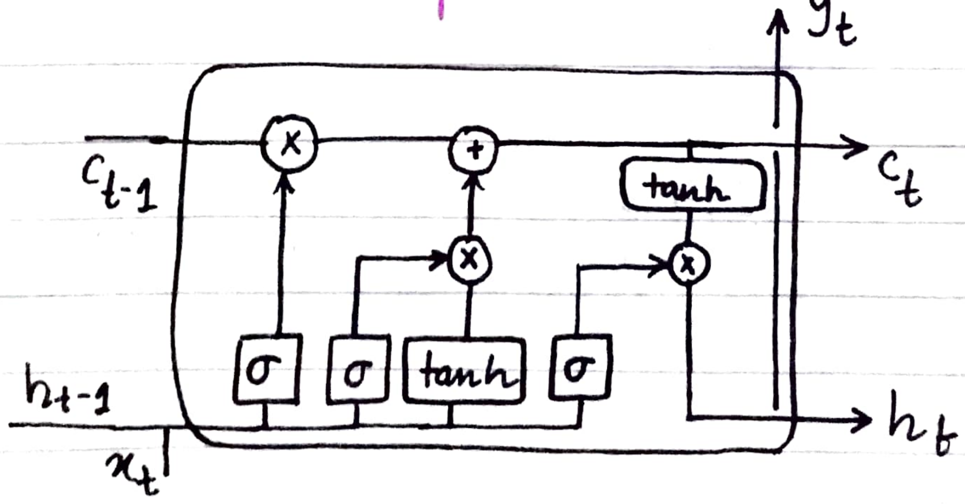
→ using a more complex NN with gates to control which information is passed through.
eg: LSTM, GRU

☆ Long Short Term Memory (LSTM) Networks

Standard RNN Cell



LSTM : modules contain computation blocks to control flow of info.



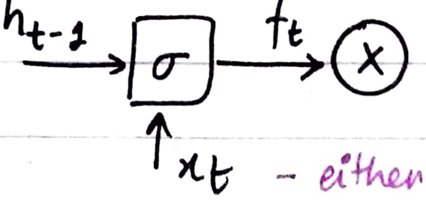
→ Information is added or removed through gates.

How do LSTMs Work?

1) Forget - irrelevant information of previous state
2) Store - relevant information by performing computation.
3) Update - selectively update cell state values using 1) + 2)
4) Output - control what to send in the next time step.

1) Forgetting:

function of prev. state + current input

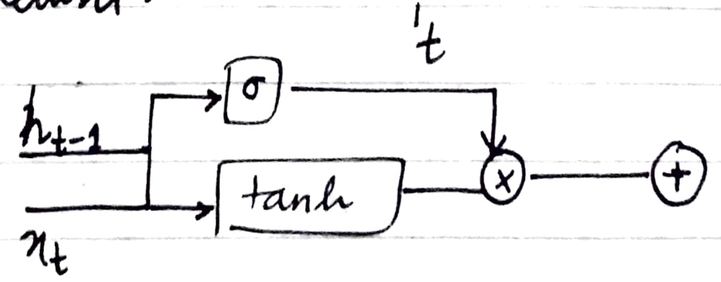


Ⓐ old useful info.

- either throw all info
- or keep all info

{using sigmoid fn.}

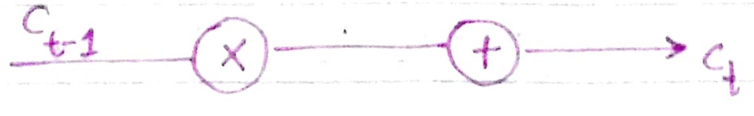
2) ~~B~~ Store: Decide what part of new information is relevant.



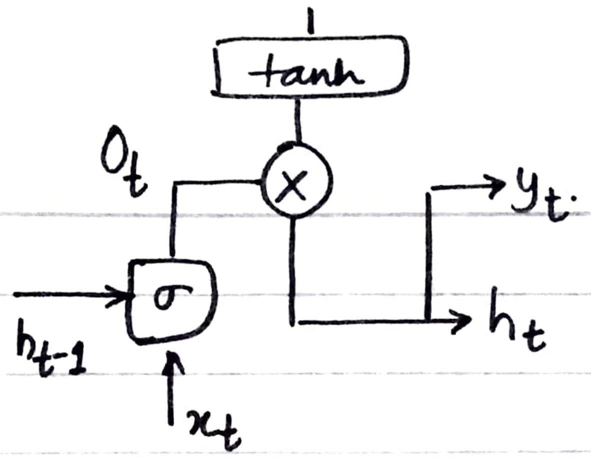
Ⓑ new useful info.

3) Update:

using old + new info.



4) <u>Output</u>:
decide what
to send out.



Key thing about LSTMs is that they use gating mechanisms to create and update internal state 'C' which leads to <u>unitterupted gradient flow</u>

LSTM Key Concepts

→ Maintain seperate cell state than what is outputted.
→ Uses gates to control flow of info
  → Forget, Store, update & output.
→ Backpropagation through time for uninterupted gradient flow.

RNN Applications
→ Music generation
→ Sentiment classification of tweets / text.
→ Machine translation - (one language to another).
  - Backbone of google translate RNN.
  - Attention Mechanisms to make avail all memory $\overset{\text{states}}{\wedge}$ steps in all time steps.

→ Self driving cars.

→ Environmental modelling - predicting wind flow directions.

→

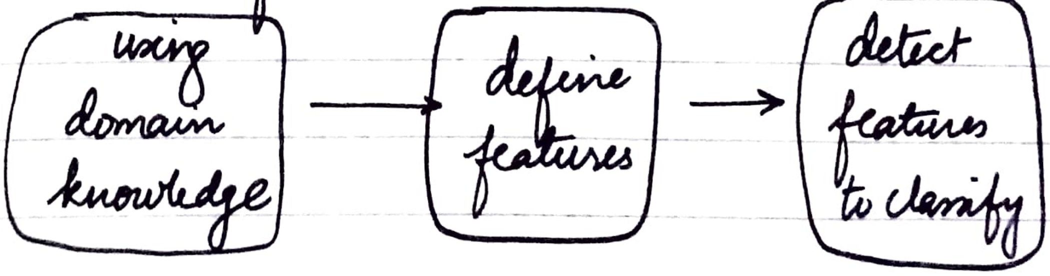
LECTURE 3 : CONVOLUTIONAL NEURAL NETWORKS

Computer Vision applications transformed by deep learning.
- → facial recognition and detection.
  - emotions, features detection.
- → Medicine, biology and healthcare
- → Self driving cars.

★ How does computers see?
- Images as two-dim. numbers. - Grayscale
- RGB - 3 X 2Dim Matrices - stacked

High level features

→ Manual feature extraction.



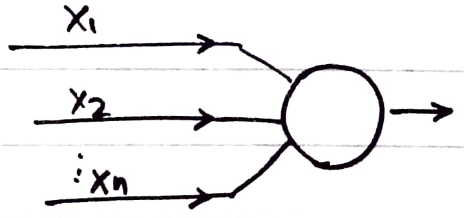
- this is difficult in practice, since to classify an image, the pipeline should be agnostic of variations in features. These variations could be deformity, brightness, viewpoint, illumination, background 4 in-class variations.

→ learning feature representation

- learn hierarchy of features directly from the data to make a representation of what makes up the final class label.
- the hierarchy of features could include
  - low level features - darkspots, edges.
  - mid level " - eyes, ears, nose
  - high level " - facial structure.

★ How to learn features?

Input
2D Image → Vector of pixel values



- Spatial information lost
- Many parameters.

→ Preserving spatial structure using 'convolution'

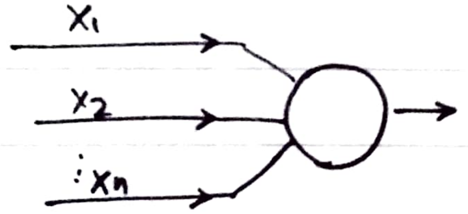
Instead of passing entire 2D image as array of numbers, pass patches of input and connect to neurons in hidden layer. using sliding window.

→ learning feature representation

- learn hierarchy of features directly from the data to make a representation of what makes up the final class label.
- the hierarchy of features could include
  - low level features - darkspots, edges.
  - mid level " - eyes, ears, nose
  - high level " - facial structure.

★ How to learn features?

Input
2D → Vector of
Image   pixel values



- Spatial information lost
- Many parameters.

→ Preserving spatial structure using 'convolution'

Instead of passing entire 2D image as array of numbers, pass patches of input and connect to neurons in hidden layer. using sliding window.

→ Weighting the patch to detect particular features.

<u>Feature Extraction by applying filters</u>.

→ apply a set of weights to extract local features.
→ use multiple filters to extract different features.
→ spatially share parameters of each feature.

The Convolution Operation

Computing a convolution of 5×5 image using 3X3 filter.

$$\begin{array}{|c|c|c|c|c|}\hline 1_1 & {}^{1}1_0 & {}^{0}1_1 & {}^{1}0 & 0 \\ \hline 0_0 & {}^{0}1_1 & {}^{1}1_0 & {}^{0}1 & 0 \\ \hline 0_1 & {}^{1}0_0 & {}^{0}1_1 & {}^{1}1 & 1 \\ \hline 0 & 0 & 0 & 1 & 1 \\ \hline 0 & 1 & 1 & 0 & 0 \\ \hline \end{array} \otimes \begin{array}{|c|c|c|}\hline 1 & 0 & 1 \\ \hline 0 & 1 & 0 \\ \hline 1 & 0 & 1 \\ \hline \end{array} = \begin{array}{|c|c|c|}\hline 4 & 3 & 4 \\ \hline 2 & 4 & 3 \\ \hline 2 & 3 & 4 \\ \hline \end{array}$$

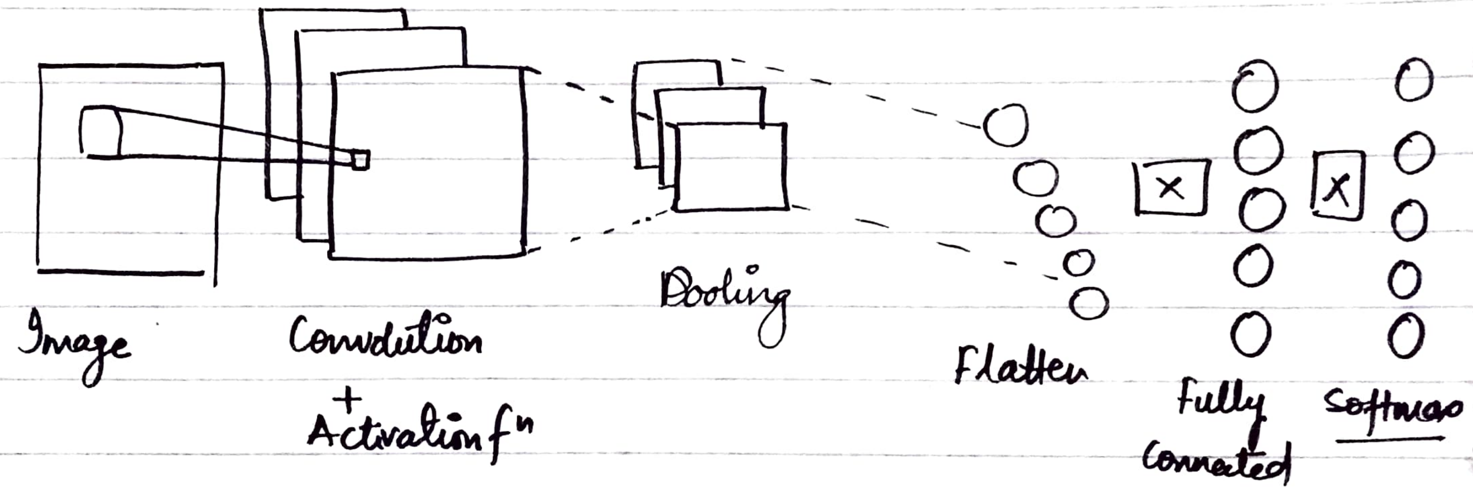
Image   filter   computing a weighted sum by element-wise operation.

The diagonal gives max. output at the diagonals. This means that this filter gives maximum overlap on this image along this diagonal.

→ changing the weights in the filter can give you a different output feature map.

★ Convolutional Neural Networks

CNN for image classification.



3 broad steps to classification.

→ Convolution - applying filters to extract features
- using multiple ~~filters~~ filters to extract a hierarchy of features

→ Non Linearity - apply non-linearity

→ Pooling - Downsampling the learned features
- reduces the dimentionality
- makes pipeline spatially invariant.

1) <u>Convolution</u> : - Take inputs from a patch
- compute weighted sum
- apply bias.

} For a neuron in hidden layer.

→ Apply a window of weights.
→ compute linear combinations.
→ Activating with non-linear function.

layer dimensions - $\underbrace{h \times w}_{\text{Spatial dim of image}} \times d \rightarrow$ depth (# of filters)

<u>stride</u> - filter step size
<u>Receptive field</u> - locations in input image that a node in path connected to.

2) <u>Non linearity</u> - ReLU $\quad g(z) = \max(0, z)$.
- pixel-by-pixel operation that replaces all negative values by zero.
- applied after every convolution operation.

3) <u>Pooling</u>:

$$\begin{array}{|c|c|c|c|} \hline 1 & 1 & 2 & 4 \\ \hline 5 & 6 & 7 & 8 \\ \hline 3 & 2 & 1 & 0 \\ \hline 1 & 2 & 3 & 4 \\ \hline \end{array} \Longrightarrow \begin{array}{|c|c|} \hline 6 & 8 \\ \hline 3 & 4 \\ \hline \end{array}$$

maxpool = 2X2 filters.

The image classification pipeline consists of two parts

= $\underbrace{\text{Feature learning}}_{\text{can be plugged into tasks like}}$ + classification task.

1) <u>Image detection</u>
2) <u>Semantic segmentation</u> - identifying class of pixel (whether pixel belongs to foreground, background, object or different objects).
3) <u>End-to-end robotic control</u> - self driving car.

LECTURE 4: DEEP GENERATIVE MODELS

Generative modelling : unsupervised learning

Goal : Take as input training samples from some distribution and learn a model that represents that distribution.

learn Model $P_{model}(x)$ similar to $P_{data}(x)$

1. Density Estimation - learning prob dist
2. Sample Generation - generate new samples of data from learned training data.

Application to
→ generate representative datasets for debiasing models.
→ outlier detection - generative models for outlier detection. train on outlier generated to improve model performance on rare circumstances.

Two classes of Modes: Latent Variable Models
→ GAN - General Adverserial Networks.
→ Autoencoders and Variational Autoencoders (VAEs)

Latent Variable vs Observed Variables

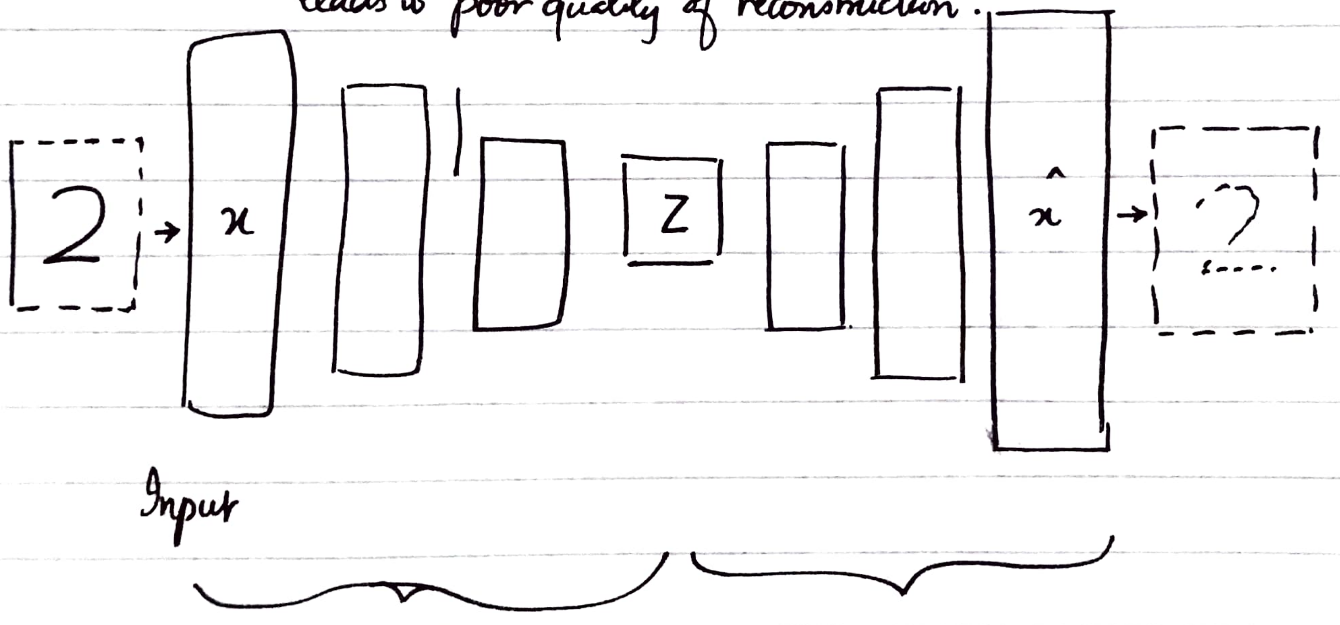
↓

variables that are ~~not~~ true explanotory factors that are / ~~can be only~~ ~~not to~~ cannot be observed in the data · only inferred through a model from other obs. variables

Goal of Generative modelling - is to learn these latent variables from only obsewed data.

★ **AUTOENCODERS** - Simple generative model.

-> autoencoding is a form of compression. lower dimensions leads to poor quality of reconstruction.



Encodes Information in Input into lowdimensional output 'z'.

Train CNN/Fullyconnected layer to predict/reconstruct training data.

$$\underbrace{\mathcal{L}(x-\hat{x})}_{\text{Reconstruction loss}} = \|x-\hat{x}\|^2. \qquad \text{(MSE).}$$

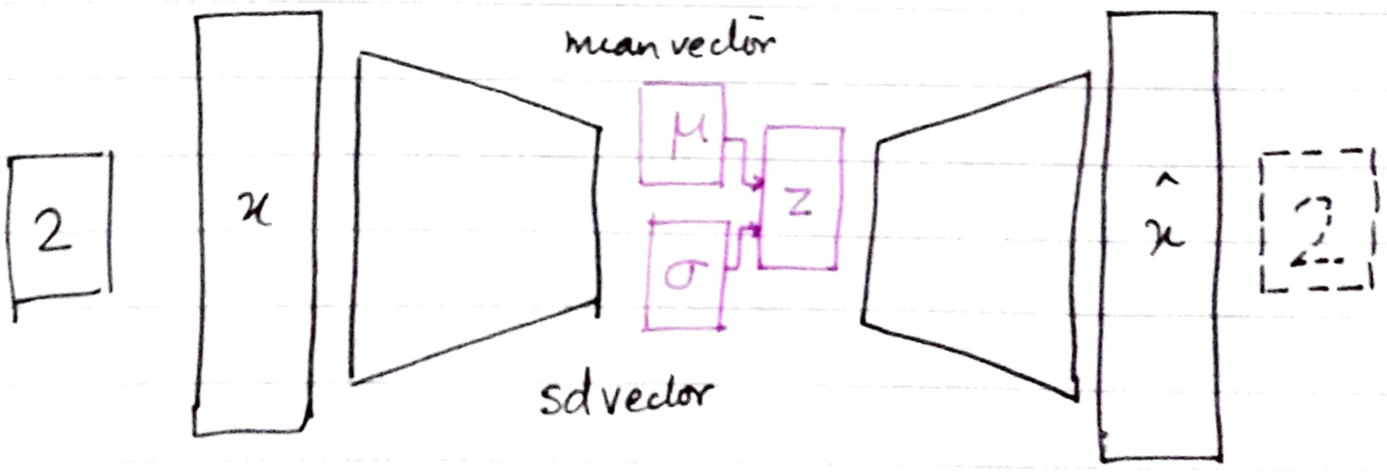
In Summary, Autoencoders

→ use <u>bottleneck hidden layer</u> to compress input and learn latent representation.

→ Reconstruction loss. forces latent representations to encode max. possible info about the data.

<u>Variational Autoencoders</u> (VAEs).

- replaced bottleneck layer (deterministic) → stochastic sampling.
- for generating similar data.
- Instead of learning latent variable 'z', the model learns mean and std. dev for each variable, that parameterizes a prob. distribution for each of those latent variables.



Encoder: computes prob. dist ~~of latent~~ p_ϕ of the latent space 'z' given an input 'x'.

- $p_\phi(z|x)$

Decoder: Computing new prob dist q_θ of x given latent variables 'z'.

- $q_\theta(x|z)$

$$\mathcal{L}(\phi,\theta,x) = (\text{reconstruction loss}) + (\text{regularization term}).$$

$$\downarrow \qquad\qquad \downarrow$$

same as before $\|x-\hat{x}\|^2$

$$D\left(\underbrace{p_\phi(z|x)}_{\text{Inferred latent dist}} \,\|\, p(z)\right)$$

- Some initial hypothesis of what 'z' looks like
- to help network not overfit.

Priors on the latent distribution

$$D\left(p_\phi(z|x) \,\|\, p(z)\right)$$

Inferred latent distribution $\leftarrow$ $p_\phi(z|x)$

$p(z)$ $\rightarrow$ Fixed prior on latent distribution

Common choice of prior - Gaussian distribution.
$\mu = 0 ; \sigma = 1$.

= <u>KL Divergence</u> when gaussian (with $\mu=0, \sigma=1$)

$$-\frac{1}{2} \sum_{j=0}^{k-1} (\sigma_j + \mu_j^2 - 1 - \log \sigma_j)$$

'Basically learning latent variables with a known distribution?'.

{<u>Machine learning mastery</u>}

- <u>Kullback-leibler divergence</u> - a measure of how one probability distribution is different from a second one.
- also called relative entropy
- can be used for feature selection (information gain).
- or cross entropy used as loss function.
- P and Q. score for divergence $\neq$ Q and P score.

• $KL(P \| Q)$
$\uparrow$
'<u>divergence</u>' of P from Q

sum $\left(p[i] * \log_2 \left(\frac{p_i}{q_i} \right) \right)$ for all values in p.

available in <u>scipy</u> library. use <u>rel_entr()</u> instead of <u>kl_div()</u>

Jenson -Shannon Divergence

→ JS divergence is symmetrical i.e

$$JS(P \| Q) == JS(Q \| P)$$

→ $JS(P \| Q) = \frac{1}{2} KL(P \| M) + \frac{1}{2} KL(Q \| M)$

$$M = \frac{1}{2}(P+Q)$$

→ More useful measure as it provides more smooth and normalized version of KL.

→ Score = 0 (Identical distributions)
= 1 (Maximally different)
when using log base 2.

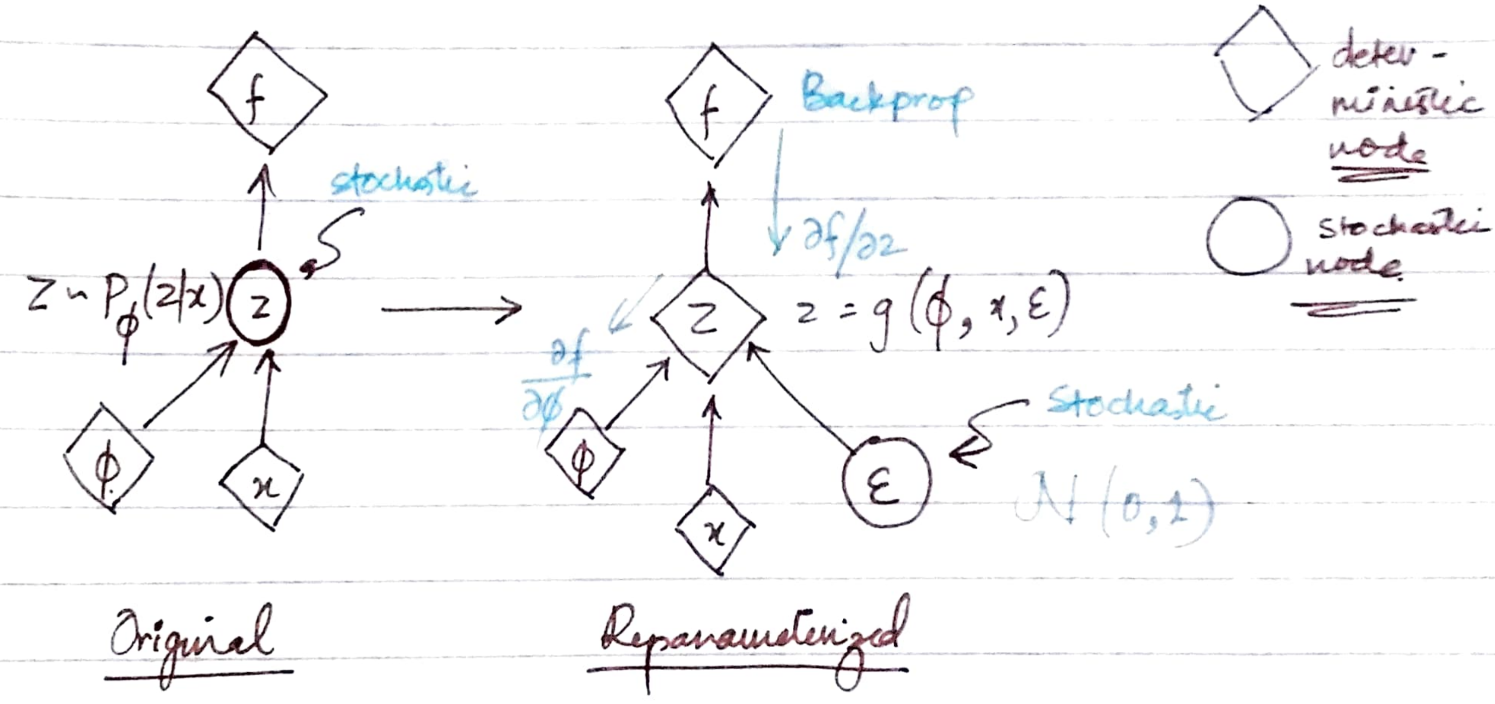
Training VAEs using Backpropagation.

- z ← is a result of stochastic sampling.
- problem : cannot compute gradient over non-deterministic nodes.
- Solution : Use stochasticity as an input to a determinant node and compute gradient.

☆ Reparameterize the sampling layer

$\boxed{z = \mu + \sigma \odot \varepsilon}$ instead of $\boxed{z \sim \mathcal{N}(\mu, \sigma^2)}$

$\varepsilon \sim \mathcal{N}(0,1)$



☆ What do these latent variables mean?

→ Keeping all latent variables same, sample one variable from the prior distribution and tune the variables to find out what they have learnt

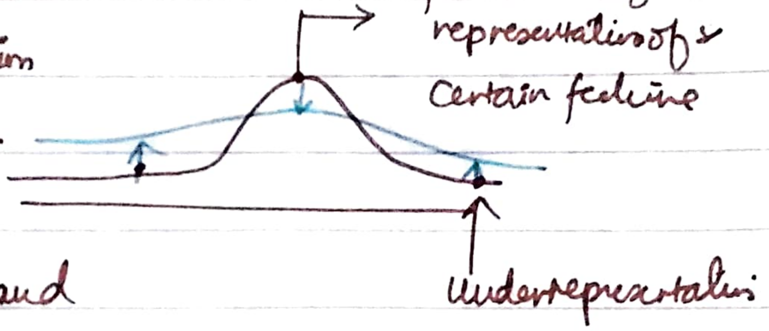
→ tuned variable (face) ⟶ Decoder ⟶ output (orientation of faces)

eg latent? - orientation

→ latent variables should be independent / uncorrelated.
called disentanglement

★ Mitigating Bias through learned latent structure

Step 1: learn latent structure
Step 2: Estimate distribution
Step 3: Adaptively Resample
Step 4: ~~to ge~~ Generate diverse set of data and fit a Fully connected layer to the data.



★ General Adversarial Network (GAN)

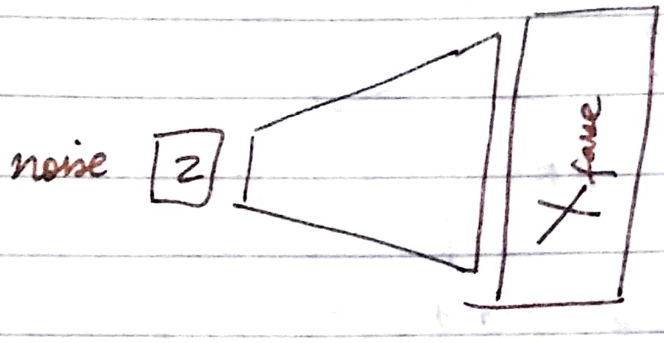
Idea: Generate new instance similar to the training data.
→ Not learn the underlying dist 4 latent var. distribution.

Problem: Cannot directly sample from complex dist

Solution: Generator network starts from random noise.
then " trained going from noise to learning a

transformation to the training distribution.

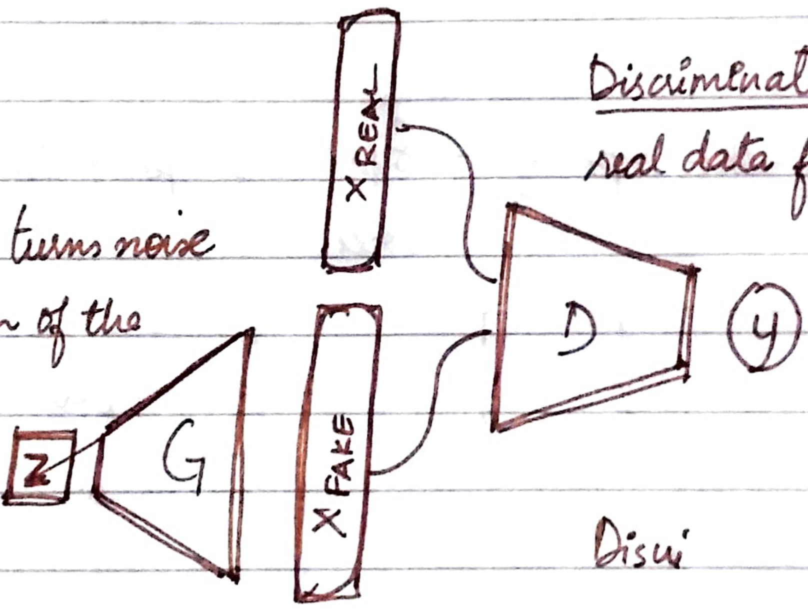
Goal:



To generate a data that is very similar to the training data.

Discriminator tries to identify real data from fakes created by generator

The generator turns noise into imitation of the data to try to trick discriminator



Disci

After training - Trained generator network to sample and create new data never seen before.

Applications → 1. Progressively growing GANs.
- low resolution starting
- incrementally add resolution during training.

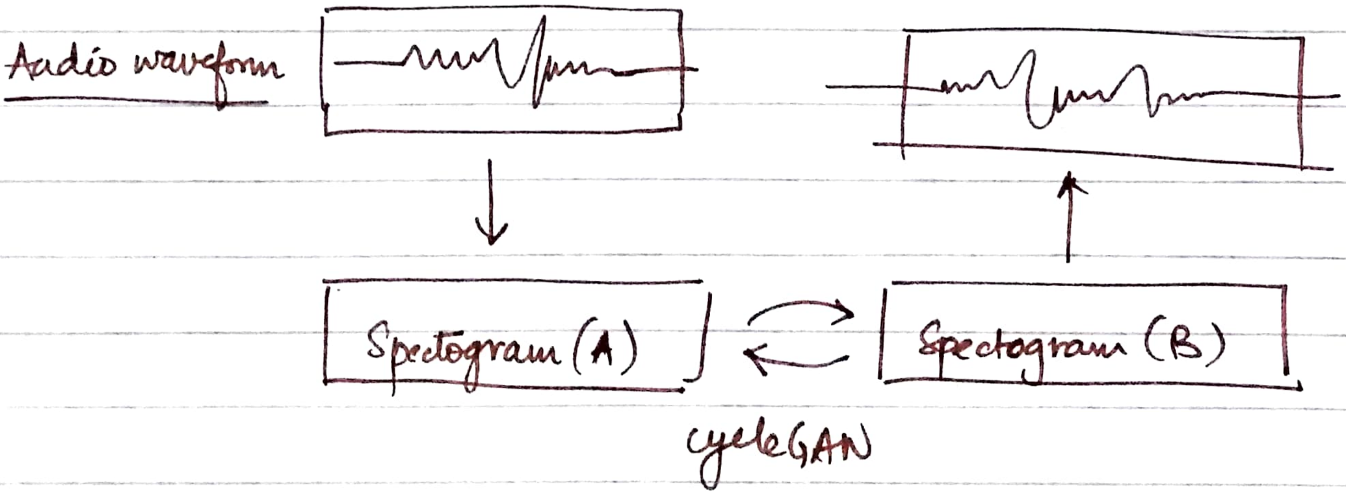
2. CycleGAN : Style Transfer

1)

$$\begin{tikzcd} D_X & & D_Y \\ \boxed{X} \arrow[u, dashed] \arrow[rr, bend left, "G"] & & \boxed{Y} \arrow[ll, bend left, "F"] \end{tikzcd}$$

learns from domain X and Y. - Two generators and discriminators Working.

2) Transfer Speech - Synthesise speech in someoneelse voice


Audio waveform
Spectogram (A)
Spectogram (B)
cycleGAN