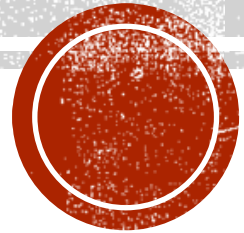
2. FUNDAMENTALS OF DEEP NETWORKS

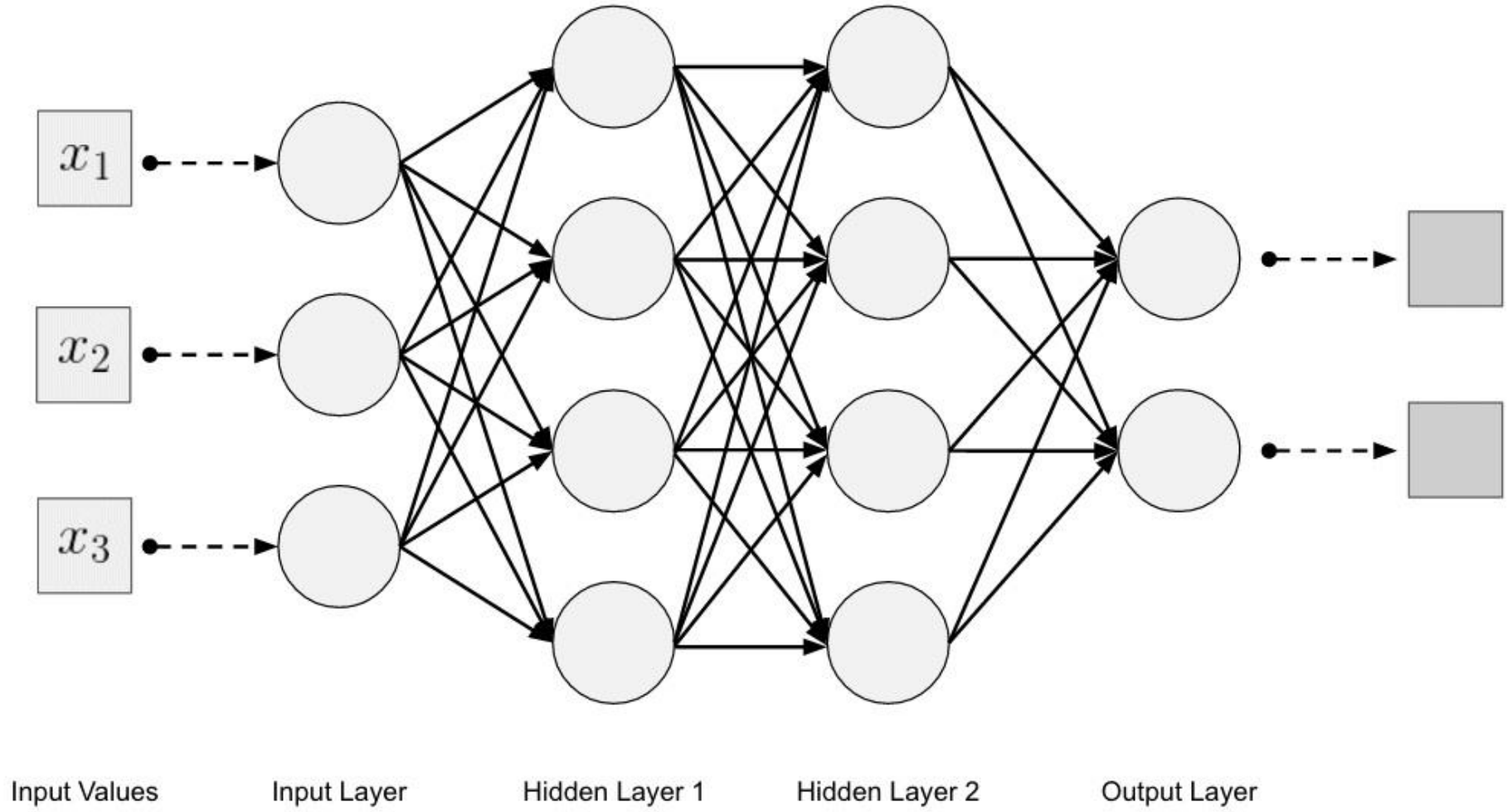


NEURAL NETWORKS

The behavior of neural networks is shaped by its network architecture

- Number of neurons
- Number of layers
- Types of connections between layers

MULTILAYER NEURAL NETWORK TOPOLOGY



Multilayer neural network topology

MULTILAYER NEURAL NETWORK TOPOLOGY

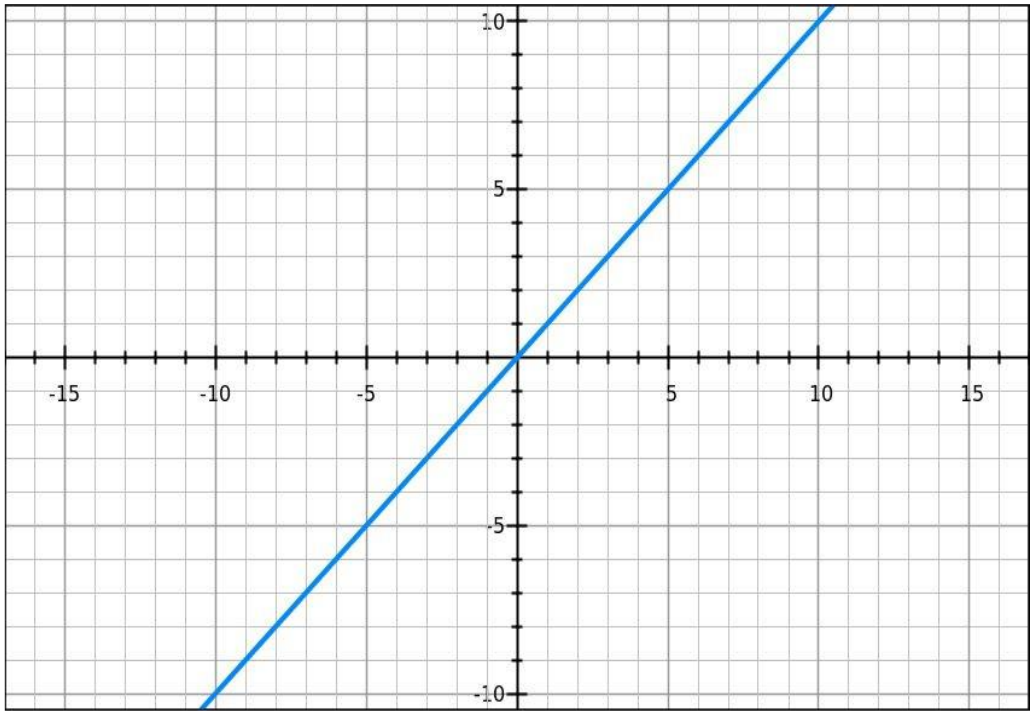
- *Connection weights*-coefficients that scale (amplify or minimize) the input signal to a given neuron in the network
- *Biases*-
  - Scalar values added to the input to ensure that at least few nodes per layer are activated regardless of signal strength.
  - Biases allow learning to happen by giving the network action in the event of low signal.
  - Allow the network to try new interpretations or behaviors.
  - Notated as *b*, and, like weights, biases are modified throughout the learning process
- *Activation functions*-The functions that govern the artificial neuron's behavior

ACTIVATION FUNCTIONS

Activation functions-

- Propagate the output of one layer's nodes forward to the next layer (up to and including the output layer).
- a scalar-to-scalar function, yielding the neuron's activation.
- Used for hidden neurons in a neural network to introduce nonlinearity into the network's modeling capabilities.
- Many activation functions belong to a logistic class of transforms that (when graphed) resemble an S.
- This class of function is called *sigmoidal*.
  - The sigmoid family of functions contains several variations, one of which is known as the Sigmoid function.

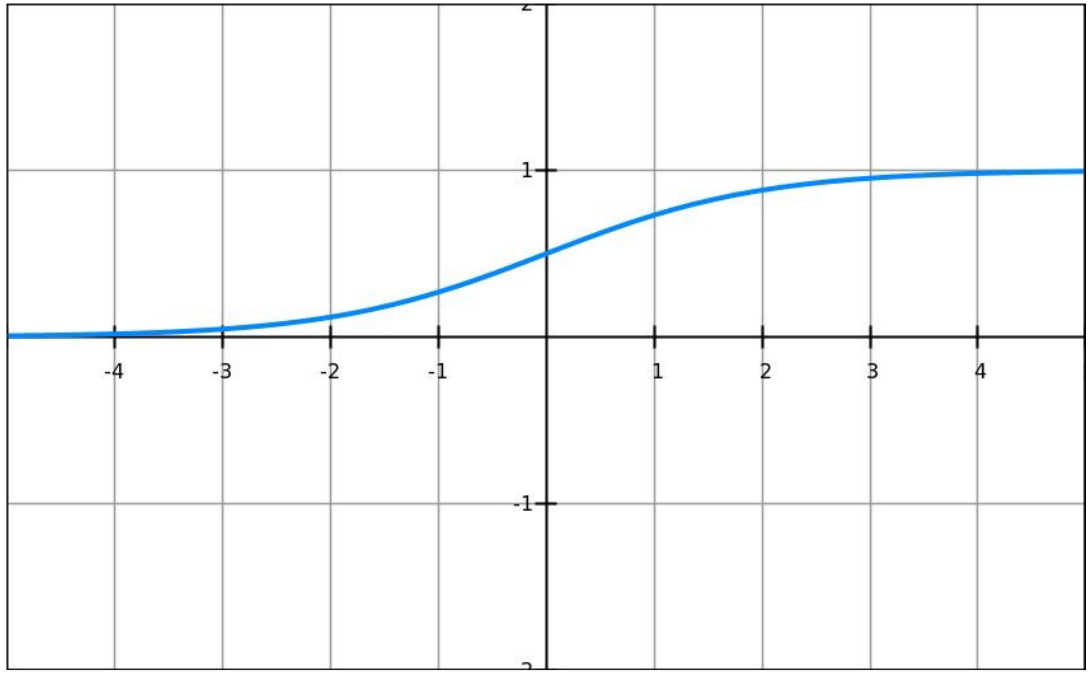
ACTIVATION FUNCTIONS-LINEAR



- A linear transform is basically the identity function,
- f(x) = W * x
- Dependent variable has a direct, proportional relationship with the independent variable.
- It means the function passes the signal through unchanged.
- Used in the input layer of neural networks

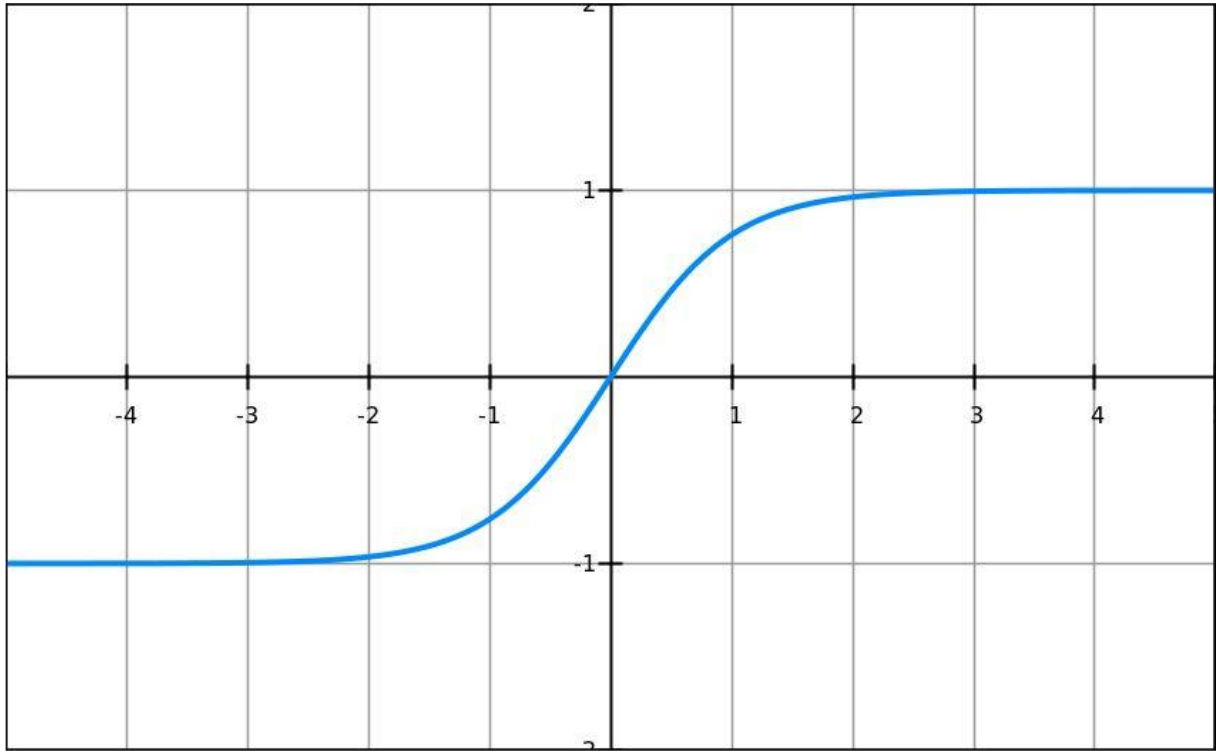
Linear activation function

ACTIVATION FUNCTIONS-SIGMOID



- Sigmoids can reduce extreme values or outliers in data without removing them.
- A sigmoid activation function outputs an independent probability for each class.
- A sigmoid function is a machine that converts independent variables of near infinite range into simple probabilities between 0 and 1, and most of its output will be very close to 0 or 1.

ACTIVATION FUNCTIONS-TANH



- Hyperbolic trignometric function
- Just as the tangent represents a ratio between the opposite and adjacent sides of a right triangle, tanh represents the ratio of the hyperbolic sine to the hyperbolic cosine:
- tanh(x) = sinh(x) / cosh(x).
- normalized range of tanh is –1 to 1.
- Advantage -it can deal more easily with negative numbers

ACTIVATION FUNCTIONS-HARD TANH

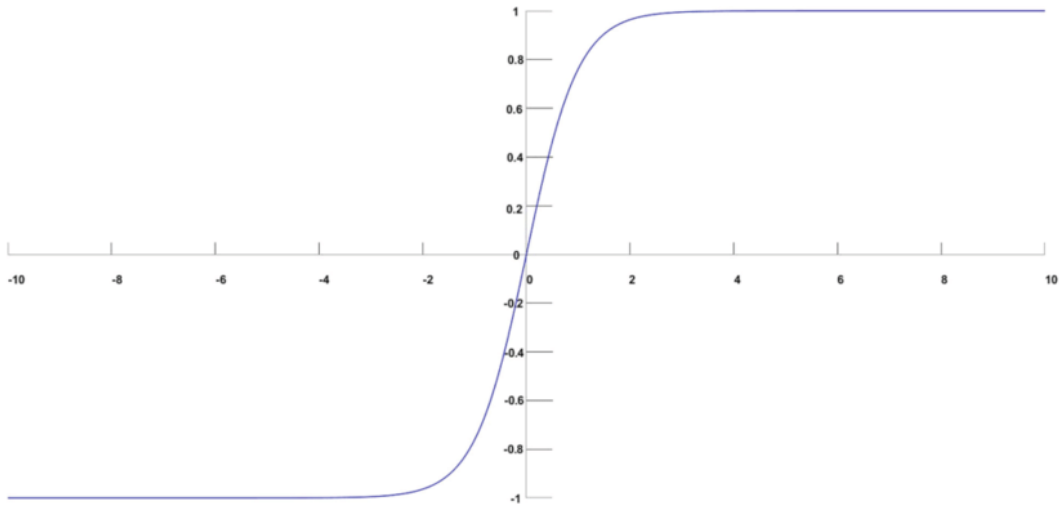
Hard Tanh-

- Similar to tanh, hard tanh simply applies hard caps to the normalized range.
- Anything more than 1 is made into 1, and anything less than –1 is made into –1.
- This allows for a more robust activation function that allows for a limited decision boundary.

ACTIVATION FUNCTIONS-SOFTMAX

Softmax-

- Also known as -softargmax or normalized exponential function
- Allows us to express our inputs as a discrete probability distribution
- a generalization of logistic regression in as much as it can be applied to continuous data (rather than classifying binary) and can contain multiple decision boundaries.
- It handles multinomial labeling systems.
- Generally used at the output layer of a classifier
- Defined as follows:
  - for each value in input vector, the Softmax value is the exponent of the individual input divided by a sum of the exponents of all the inputs.
- Returns the probability distribution over mutually exclusive output classes.



$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

σ = softmax

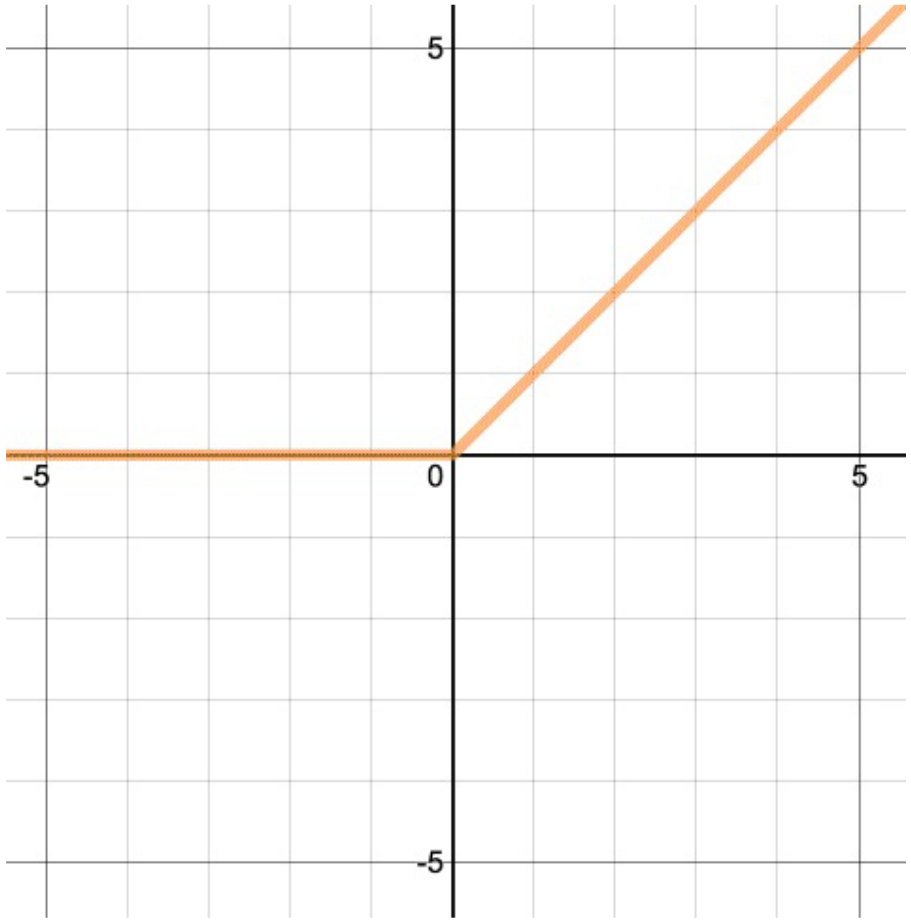
$\vec{z}$ = input vector

e^{z_i} = standard exponential function for input vector

K = number of classes in the multi-class classifier

e^{z_j} = standard exponential function for output vector

ACTIVATION FUNCTIONS-RECTIFIED LINEAR

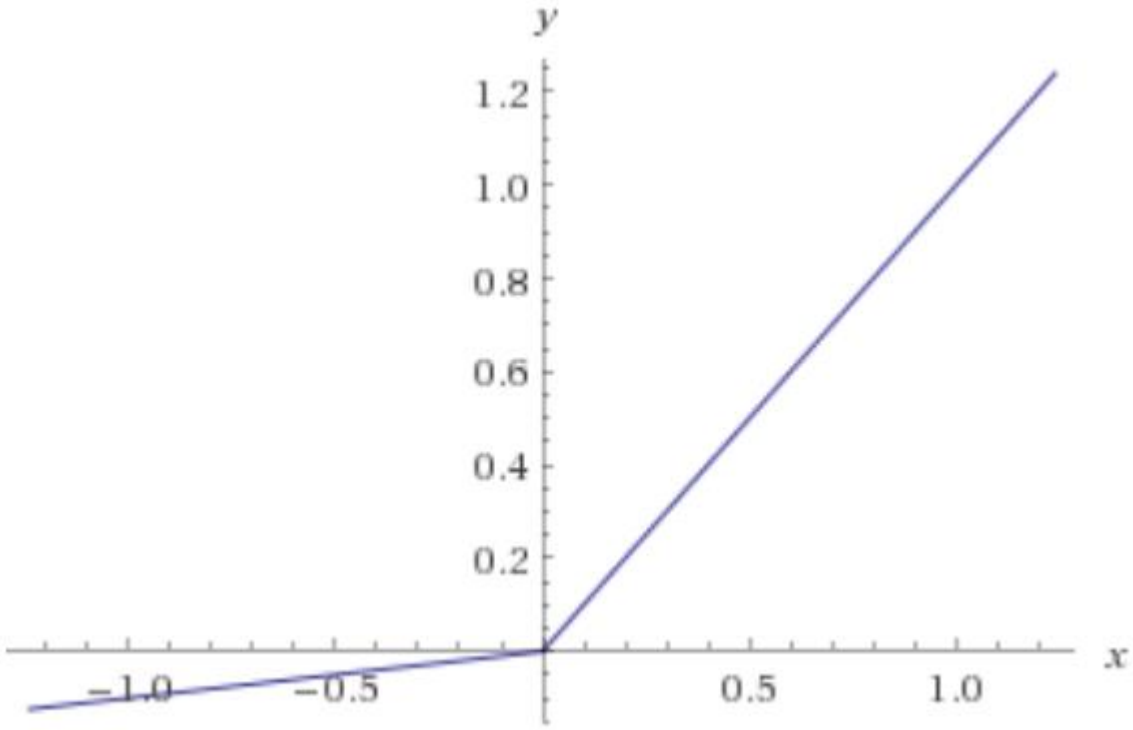


- More interesting transform that activates a node only if the input is above a certain quantity.
- While the input is below zero, the output is zero, but when the input rises above a certain threshold, it has a linear relationship with the dependent variable $f(x) = \max(0, x)$
- ReLU activation functions have shown to train better in practice than sigmoid activation functions
- Compared to the sigmoid and tanh activation functions, the ReLU activation function does not suffer from vanishing gradient issues

ACTIVATION FUNCTIONS-LEAKY RELU

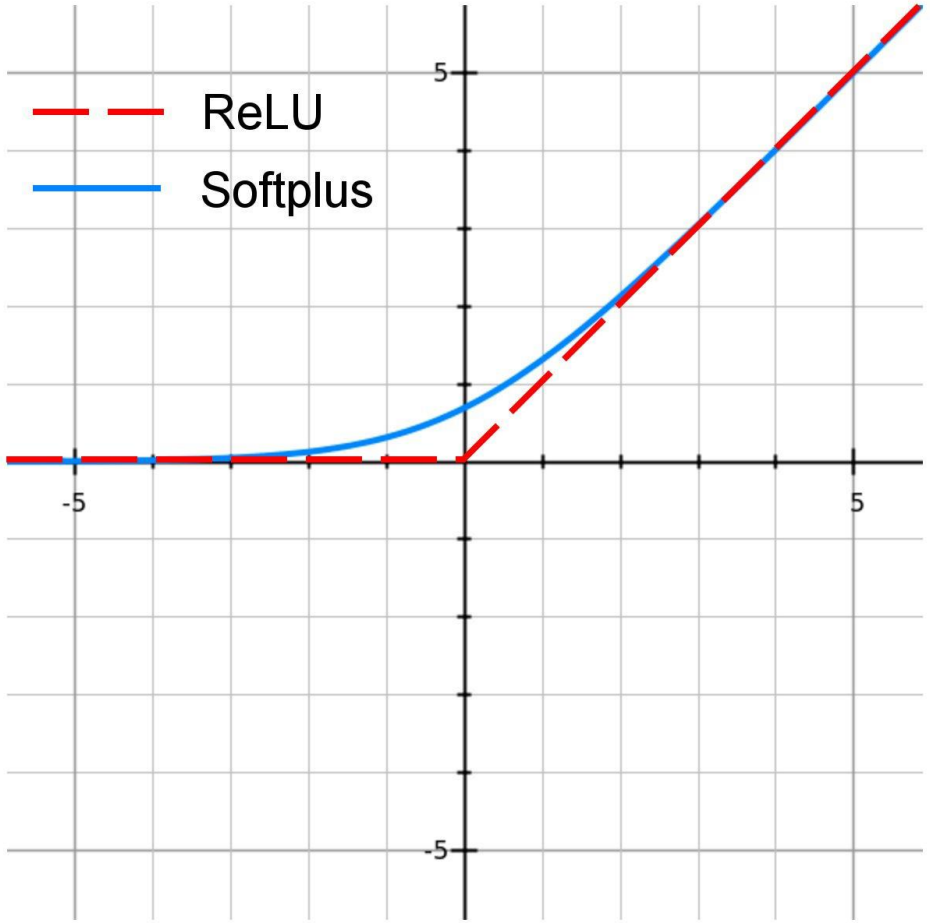
- A strategy to mitigate the "dying ReLU"
- As opposed to having the function being zero when x < 0, the leaky ReLU will instead have a small negative slope (e.g., "around 0.01").
- Some success has been seen in practice with this ReLU variation, but results are not always consistent
- The equation is :

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ 0.01x & \text{otherwise} \end{cases}$$

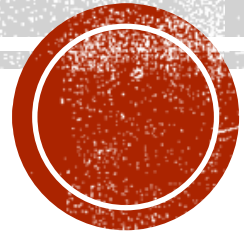


ACTIVATION FUNCTIONS-SOFTPLUS

- This activation function is the "smooth version of the ReLU,"
- Compare this plot to the ReLU - shows that the softplus activation function (f(x) = ln[1 + exp(x)]) has a similar shape to the ReLU.



LOSS FUNCTIONS



LOSS FUNCTIONS

- Loss Function Notation-
  - "N" - number of samples (set of inputs with corresponding outcomes) that have been gathered.
  - "P" - number of input features gathered
  - "M" - the number of output features that have been observed.
  - (X,Y) to denote the input and output data collected
  - N such pairs where the input is a collection of P values and the output Y is a collection of M values.
  - Denote the i^{th} pair in the dataset as Xi and Yi .
  - We will use $\hat{Y}$ to denote the output of the neural net.
  - $\hat{Y}$ - network's guess at Y and therefore it will also have M features.

LOSS FUNCTIONS

- Use the notation $h(X_i) = \hat{Y}_i$ to denote the neural network transforming the input to give the output .
- When referring to *j th* output feature, use it as a subscript firmly linking the notation to a matrix where the rows are different data points and the columns are the different unique features. Thus $y_{i,j}$ refers to the j^{th} feature observed in the i^{th} sample collected.
- Represent the loss function by *L(W,b)*.

$$h_{w,b}(X) = \hat{Y}$$

LOSS FUNCTIONS

- Loss Functions for Regression-
  - Mean squared error loss

$$L(W, b) = \frac{1}{N}\sum_{i=1}^{N}\left(\hat{Y}_i - Y_i\right)^2$$

$$L(W, b) = \frac{1}{2N}\sum_{i=1}^{N}\sum_{j=1}^{M}\left(\hat{y}_{ij} - y_{ij}\right)^2$$

$$L(W, b) = \frac{1}{2N}\sum_{i=1}^{N}\sum_{j=1}^{M}\left(\hat{y}_{ij} - y_{ij}\right)^2$$

LOSS FUNCTIONS

- Loss Functions for Regression-
  - Mean absolute error loss

$$L(W, b) = \frac{1}{2N} \sum_{i=1}^{N} \sum_{j=1}^{M} | \hat{y}_{ij} - y_{ij} |$$

  - Mean squared log error loss

$$L(W, b) = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{M} (\log \hat{y}_{ij} - \log y_{ij})^2$$

  - Mean absolute percentage error loss

$$L(W, b) = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{M} \frac{100 \times | \hat{y}_{ij} - y_{ij} |}{y_{ij}}$$

LOSS FUNCTIONS

- Loss Functions for Classification-
  - Hinge loss- most commonly used loss function when the network must be optimized for a hard classification
  - Also seen in a class of models called maximum-margin classification models (e.g., support vector machines)
  - equation for hinge loss when data points must be categorized as –1 or 1

$$L(W, b) = \frac{1}{N}\sum_{i=1}^{N} max(0, 1 - y_{ij} \times \hat{y}_{ij})$$

  - Mostly used for binary classifications

LOSS FUNCTIONS

- Loss Functions for Classification-
  - Logistic loss- used when probabilities are of greater interest than hard classifications.

$$P(y_i = 1 \mid X_i; \mathbf{W}, \mathbf{b}) = h_{\mathbf{W},\mathbf{b}}(X_i)$$

$$P(y_i = 0 \mid X_i; \mathbf{W}, \mathbf{b}) = 1 - h_{\mathbf{W},\mathbf{b}}(X_i)$$

$$P(y_i \mid X_i; \mathbf{W}, \mathbf{b}) = (h_{\mathbf{W},\mathbf{b}}(X_i))^{y_i} \times (1 - h_{\mathbf{W},\mathbf{b}}(X_i))^{1-y_i}$$

$$L(W, b) = \prod_{i=1}^{N} \hat{y}_i^{y_i} \times (1 - \hat{y}_i)^{1-y_i}$$

LOSS FUNCTIONS

- Loss Functions for Classification-
  - Negative log likelihood

$$L(W, b) = -\sum_{i=1}^{N} y_i \times \log \hat{y}_i + (1 - y_i) \times \log (1 - \hat{y}_i)$$

  - Extending the loss function from two classes to M classes gives us the following equation for logistic loss

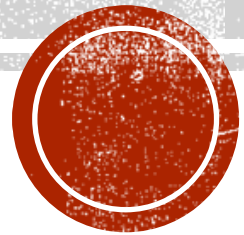
$$L(W, b) = -\sum_{i=1}^{N} \sum_{j=1}^{M} y_{i,j} \times \log \hat{y}_{i,j}$$

LOSS FUNCTIONS

- Loss Functions for Reconstruction-
- equation for KL divergence

$$D_{KL}(Y \mid\mid \hat{Y}) = -\sum_{i=1}^{N} Y_i \times \log\left(\frac{Y_i}{\hat{Y}_i}\right)$$

COMMON ARCHITECTURAL PRINCIPLES OF DEEP NETWORKS



WHAT IS DEEP LEARNING?

- More neurons than previous networks
- More complex ways of connecting layers
- Automatic feature extraction

ARCHITECTURES OF DEEP NETWORKS

- Four major architectures of deep networks-
  - Unsupervised Pretrained Networks
  - Convolutional Neural Networks
  - Recurrent Neural Networks
  - Recursive Neural Networks

ADVANCES IN NETWORK ARCHITECTURE

- Advances in layer types
- Advances in neuron types
- Hybrid architectures

FROM FEATURE ENGINEERING TO AUTOMATED FEATURE LEARNING

- Feature engineering-handcrafted features → highly accurate models but take a lot of time and experience to produce.
- Feature learning-
  - Feature maps
  - CNN

Feature Engineering: This involves manually creating features for machine learning models. It can yield accurate results but requires substantial time, effort, and expertise.

Feature Learning: This introduces automated methods, such as feature maps and Convolutional Neural Networks (CNNs), to extract meaningful features from data. This reduces manual effort and enables models to adapt to complex data patterns.

FROM FEATURE ENGINEERING TO AUTOMATED FEATURE LEARNING

Generative modelling-

- Inceptionism-
  - technique in which a trained convolutional network is taken with its layers in reverse order and given an input image coupled with a prior constraint.
  - The images are modified iteratively to enhance the output in a manner that could be described as "hallucinative."
- Modeling artistic style-to learn the style of specific painters and then generate a new image in this style of arbitrary photographs
- Generative Adversarial Networks-The generative visual output of a GAN can best be described as synthesizing novel images by modeling the distributions of input data seen by the network.
- Recurrent Neural Networks- Applied in tasks where sequential data generation is required, like text or music.

COMMON ARCHITECTURAL PRINCIPLES OF DEEP NETWORKS

Core components- [Penguins Love A Lot Of Hugs]

- Parameters
- Layers
- Activation functions
- Loss functions
- Optimization methods
- Hyperparameters

COMMON ARCHITECTURAL PRINCIPLES OF DEEP NETWORKS

- Core components-
- Layers-
  - Fundamental architectural unit
  - Combinations of layers /activation function functions
  - Layer specific hyperparameters

- What Are Parameters?
  - Parameters include weights and biases in the network's layers.
  - Weights decide the strength of connections between nodes (neurons) across layers.
  - Biases allow the model to shift activation functions, enabling better flexibility in representation.

- Role in Deep Networks:
  - Parameters are initialized with random values at the start of training.
  - They are adjusted iteratively through optimization methods (like gradient descent) based on the loss function, which evaluates prediction errors.
  - Well-optimized parameters enable the network to learn patterns and generalize to unseen data.

COMMON ARCHITECTURAL PRINCIPLES OF DEEP NETWORKS

Core components-

- **Activation Functions-**
  - two main areas across all architectures:
    - Hidden layers
    - Output layers
  - Hidden layer activation functions-
    - Sigmoid
    - Tanh
    - Hard tanh
    - Rectified linear unit (ReLU) (and its variants)
  - A more continuous distribution of input data is generally best modeled with a ReLU activation function.
  - Optionally use tanh activation function (if the network isn't very deep) in the event that the ReLU did not achieve good results

COMMON ARCHITECTURAL PRINCIPLES OF DEEP NETWORKS

Core components-

Activation Functions-

- Output layer for regression-
  - what type of answer we expect our model to output.
  - For e.g. a single real-valued number as o/p from our model, then use a linear activation function.
- Output layer for binary classification-
  - Use a sigmoid output layer with a single neuron to give us a real value in the range of 0.0 to 1.0 (excluding those values) for the single class.
  - This real-valued output is typically interpreted as a probability distribution.
- Output layer for multiclass classification-
  - If we only care about the best score across these classes, use a softmax output layer with an arg-max() function to get the highest score of all the classes.
  - The softmax output layer gives us a probability distribution over all the classes.

COMMON ARCHITECTURAL PRINCIPLES OF DEEP NETWORKS

Core components-

Loss functions-

- **Regression** Used for continuous-valued predictions (e.g., predicting house prices). Example: Mean Squared Error (MSE).
- **Classification** Ideal for categorical outputs (e.g., classifying images into 'cats' and 'dogs'). Example: Cross-Entropy Loss.
- **Reconstruction-**
  - Involved in unsupervised feature extraction, ensuring the network can recreate input data.
  - In certain architectures of deep networks reconstruction loss functions help the network extract features more effectively when paired with the appropriate activation function.
  - Example - use the multiclass cross-entropy as a loss function in a layer with a softmax activation function for classification output, Mean Absolute Error (MAE).

COMMON ARCHITECTURAL PRINCIPLES OF DEEP NETWORKS

Core components-

- **Optimization Algorithms-**
  - First-order
    - calculate the Jacobian matrix.
    - The Jacobian has one partial derivative per parameter
    - The algorithm then takes one step in the direction specified by the Jacobian.
    - Calculate a gradient (Jacobian) at each step to determine which direction to go in next.
    - At each iteration, or step, algorithm try to find the next best possible direction to go, as defined by objective function.
    - For e.g- GD

$$\mathbf{H}_f = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1\,\partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1\,\partial x_n} \\ \frac{\partial^2 f}{\partial x_2\,\partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2\,\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n\,\partial x_1} & \frac{\partial^2 f}{\partial x_n\,\partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix},$$

COMMON ARCHITECTURAL PRINCIPLES OF DEEP NETWORKS

Core components-

- **Optimization Algorithms-**
  - Second-order-
    - Calculate or approximate the Hessian
    - Calculate the derivative of the Jacobian (i.e., the derivative of a matrix of derivatives) by approximating the Hessian.
    - Take into account interdependencies between parameters when choosing how much to modify each parameter
    - Methods-
      - Limited-memory BFGS (L-BFGS) Broyden-Fletcher-Goldfarb-Shanno
      - Conjugate gradient
      - Hessian-free

COMMON ARCHITECTURAL PRINCIPLES OF DEEP NETWORKS

Core components-

- Hyperparameters-any configuration setting free to be chosen by the user that might affect performance.
- Categories-
  - Layer size
  - Magnitude (momentum, learning rate)
  - Regularization (dropout, drop connect, L1, L2)
  - Activations (and activation function families)
  - Weight initialization strategy
  - Loss functions
  - Settings for epochs during training (mini-batch size)
  - Normalization scheme for input data (vectorization)

COMMON ARCHITECTURAL PRINCIPLES OF DEEP NETWORKS

Hypermeters-

1. Layer size

- Number of neurons in a given layer.
- For the input layer- match up to the number of features in the input vector.
- For the output layer- either be a single output neuron or a number of neurons matching the number of classes we are trying to predict.
- Challenge in hyperparameter tuning- Deciding on neuron counts for each hidden layer
- Complexity of a problem -directly correlated to how many neurons to be used in the hidden layers of networks
- More parameters -increase in the amount of effort needed to train the network.
- Large parameter counts can lead to long training times and models that struggle to find convergence.

COMMON ARCHITECTURAL PRINCIPLES OF DEEP NETWORKS

Hypermeters-

2. Magnitude hyperparameters-

- Learning rate-
  - One of the key hyperparameters in neural networks.
  - How fast we change the parameter vector as we move through search space.
  - Too high--move toward goal faster (but we might also take a step so large that we shoot right past the best answer to the problem)
  - Too small-longer time for training process to complete, make our learning algorithm inefficient.
  - Specific to the dataset and to other hyperparameters.
  - Schedule learning rates to decrease over time according to some rule

COMMON ARCHITECTURAL PRINCIPLES OF DEEP NETWORKS

Hypermeters-

2. Magnitude hyperparameters-

- Sparsity –
  - Recognizes that for some inputs only a few features are relevant

- Momentum-
  - Momentum helps the learning algorithm get out of spots in the search space where it would otherwise become stuck.
  - Momentum is to the learning rate what the learning rate is to weights, and it helps us produce better quality models

COMMON ARCHITECTURAL PRINCIPLES OF DEEP NETWORKS

Hypermeters-

2. Magnitude hyperparameters-

- **Momentum-**
  - **Nesterov's momentum**
    - Momentum is a factor between 0.0 and 1.0 that is applied to the rate of change of the weights over time.
    - Typical value for momentum between 0.9 and 0.99.
  - **AdaGrad**
    - One technique developed to help augment finding the "right" learning rate.
    - Named in reference to how it "adaptively" uses sub-gradient methods to dynamically control the learning rate of an optimization algorithm.
    - Monotonically decreasing and never increases the learning rate above whatever the base learning rate was set at initially.
    - Square root of the sum of squares of the history of gradient computations.
    - Speeds training in the beginning and slows it appropriately toward convergence, allowing for a smoother training process.

COMMON ARCHITECTURAL PRINCIPLES OF DEEP NETWORKS

Hypermeters-

2. Magnitude hyperparameters-

- **Momentum-**
  - **RMSProp-**
    - Very effective, but currently unpublished adaptive learning rate method
  - **Adam-**
    - ADAM (a more recently developed updating technique from the University of Toronto) derives learning rates from estimates of first and second moments of the gradients
  - **AdaDelta-**
    - Variant of AdaGrad that keeps only the most recent history rather than accumulating it like AdaGrad does.

COMMON ARCHITECTURAL PRINCIPLES OF DEEP NETWORKS

Regularization-

- a measure taken against overfitting

- Regularization for hyperparameters helps modify the gradient so that it doesn't step in directions that lead it to overfit
  - Dropout
  - DropConnect
  - L1 penalty
  - L2 penalty
- Regularization works by adding an extra term to the normal gradient computed.

COMMON ARCHITECTURAL PRINCIPLES OF DEEP NETWORKS

Regularization-

- *Dropout-*
  - Mechanism used to improve the training of neural networks by omitting a hidden unit.
  - Speeds training.
  - Driven by randomly dropping a neuron so that it will not contribute to the forward pass and backpropagation.
- *DropConnect*
  - Does the same thing as Dropout, but instead of choosing a hidden unit, it mutes the connection between two neurons.
- **Dropout and DropConnect mute parts of the input to each layer, such that the neural network learns other portions.**
- **Zeroing-out parts of the data causes a neural network to learn more general representations**

COMMON ARCHITECTURAL PRINCIPLES OF DEEP NETWORKS

Regularization-

- The penalty methods L1 and L2, in contrast, are a way of preventing the neural network parameter space from getting too big in one direction.
- They make large weights smaller.

- *L1-*
  - Computationally inefficient in the nonsparse case, has sparse outputs, and includes built-in feature selection.
  - Multiplies the absolute value of weights rather than their squares.
  - This function drives many weights to zero while allowing a few to grow large, making it easier to interpret the weights

COMMON ARCHITECTURAL PRINCIPLES OF DEEP NETWORKS

Regularization-

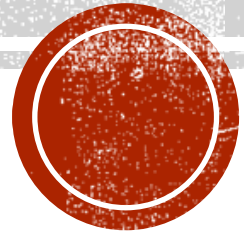
- L2 –
  - Computationally efficient due to analytical solutions and nonsparse outputs, but it does not do feature selection automatically.
  - Common and simple hyperparameter, adds a term to the objective function that decreases the squared weights.
  - Multiply half the sum of the squared weights by a coefficient called the weight-cost.
  - Improves generalization, smooths the output of the model as input changes, and helps the network ignore weights it does not use.

COMMON ARCHITECTURAL PRINCIPLES OF DEEP NETWORKS

Mini-batching-

- We send more than one input vector (a group or batch of vectors) to be trained in the learning system.
- Also allows us to compute certain linear algebra operations (specifically matrix-to-matrix multiplications) in a vectorized fashion.
- In this scenario we also have the option of sending the vectorized computations to GPUs if they are present.

BUILDING BLOCKS OF DEEP NETWORKS



BUILDING BLOCKS OF DEEP NETWORKS

Specific building blocks

- Feed-forward multilayer neural networks
- Restricted Boltzmann Machines (RBMs)
- Autoencoders

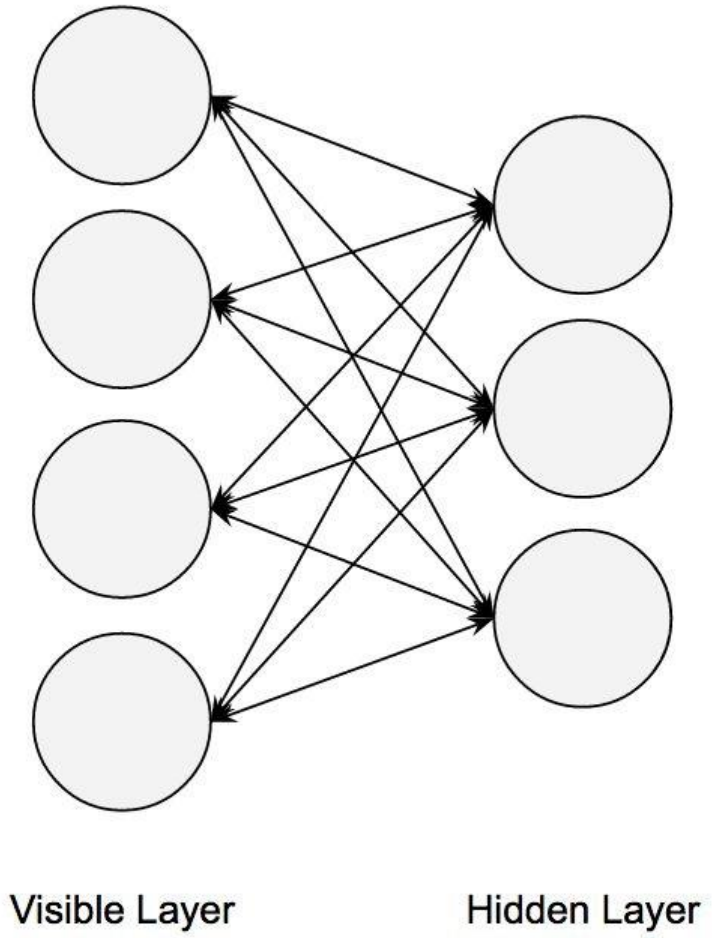
- Both RBMs and autoencoders are characterized by an extra layer-wise step for training.
- They are often used for the pretraining phase in other larger deep networks.

BUILDING BLOCKS OF DEEP NETWORKS

- **RBMs-**
  - Model probability and are great at feature extraction.
  - Feedforward networks in which data is fed through them in one direction with two biases rather than one bias as in traditional backpropagation feed-forward networks.
  - Used in deep learning for the following:
    - Feature extraction
    - Dimensionality reduction
- The “restricted” part of the name “Restricted Boltzmann Machines” - connections between nodes of the same layer are prohibited (e.g., there are no visible-visible or hidden-hidden connections along which signal passes).
- Geoff Hinton—” A network of symmetrically connected, neuron-like units that make stochastic decisions about whether to be on or off.”

BUILDING BLOCKS OF DEEP NETWORKS

- **RBMs-**
  - Also, a type of autoencoder
  - Used for pretraining layers in larger networks such as Deep Belief Networks.
- **Network layout**
- 5 main parts of a basic RBM:
  - Visible units
  - Hidden units
  - Weights
  - Visible bias units
  - Hidden bias units
- A standard RBM has a visible layer and a hidden layer
- Every visible unit is connected to every hidden unit,
- Each layer of an RBM can be imagined as a row of nodes.
- The nodes of the visible and hidden layers are connected by connections with associated weights



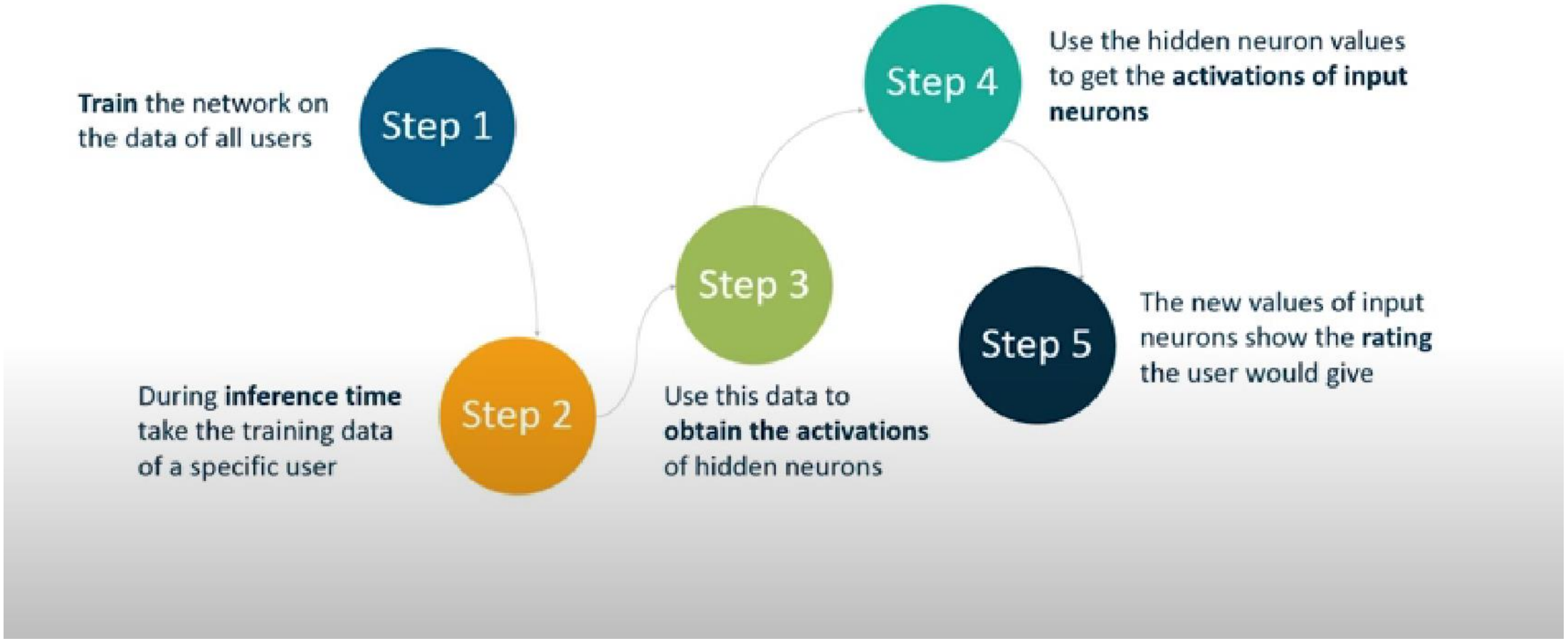
BUILDING BLOCKS OF DEEP NETWORKS

RBMs-

- **Network layout**
  - *Visible and hidden layers-*
  - *Connections and weights-*
  - *Biases-*
- **Training-**
- **Reconstruction**

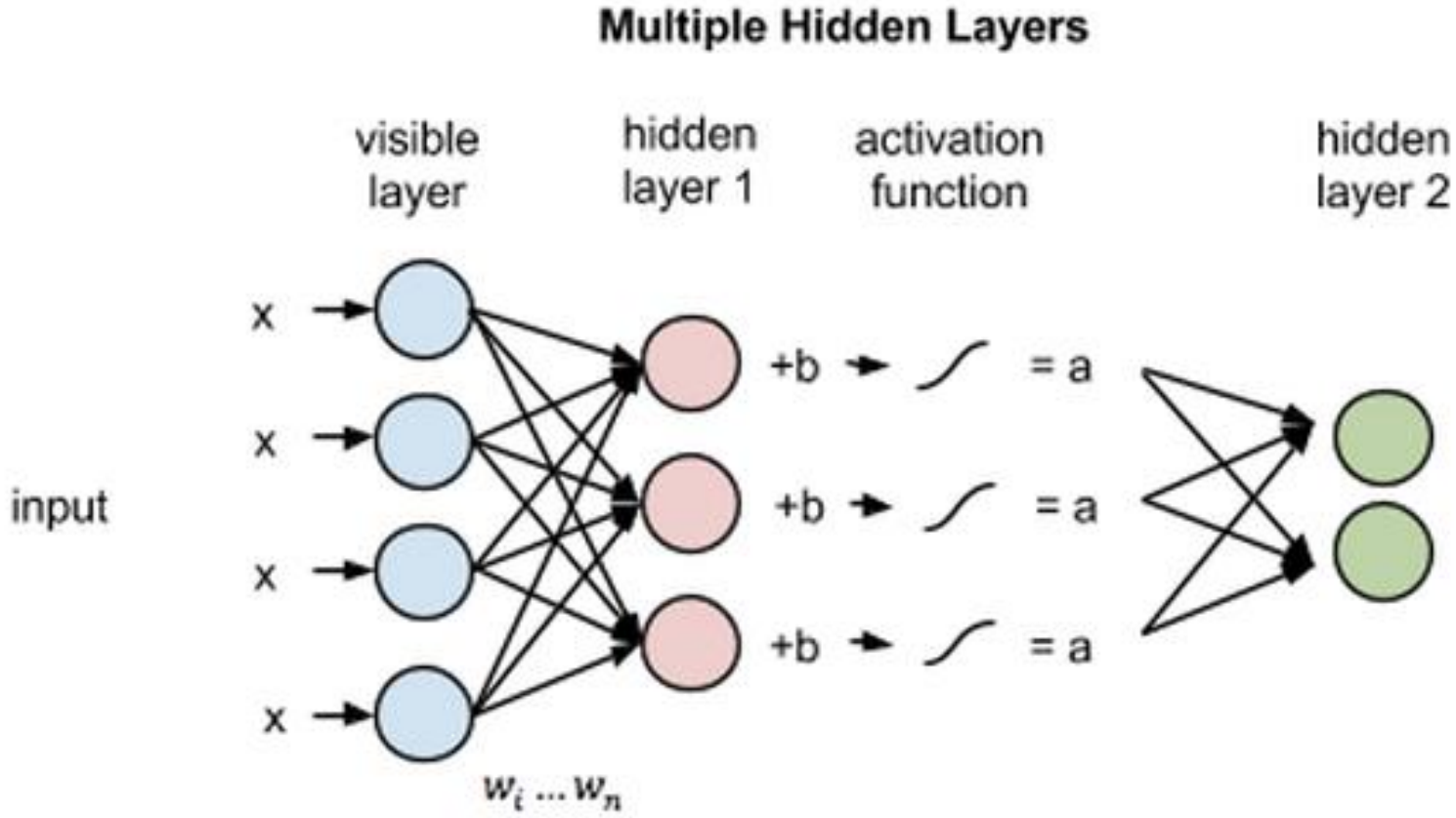
BUILDING BLOCKS OF DEEP NETWORKS

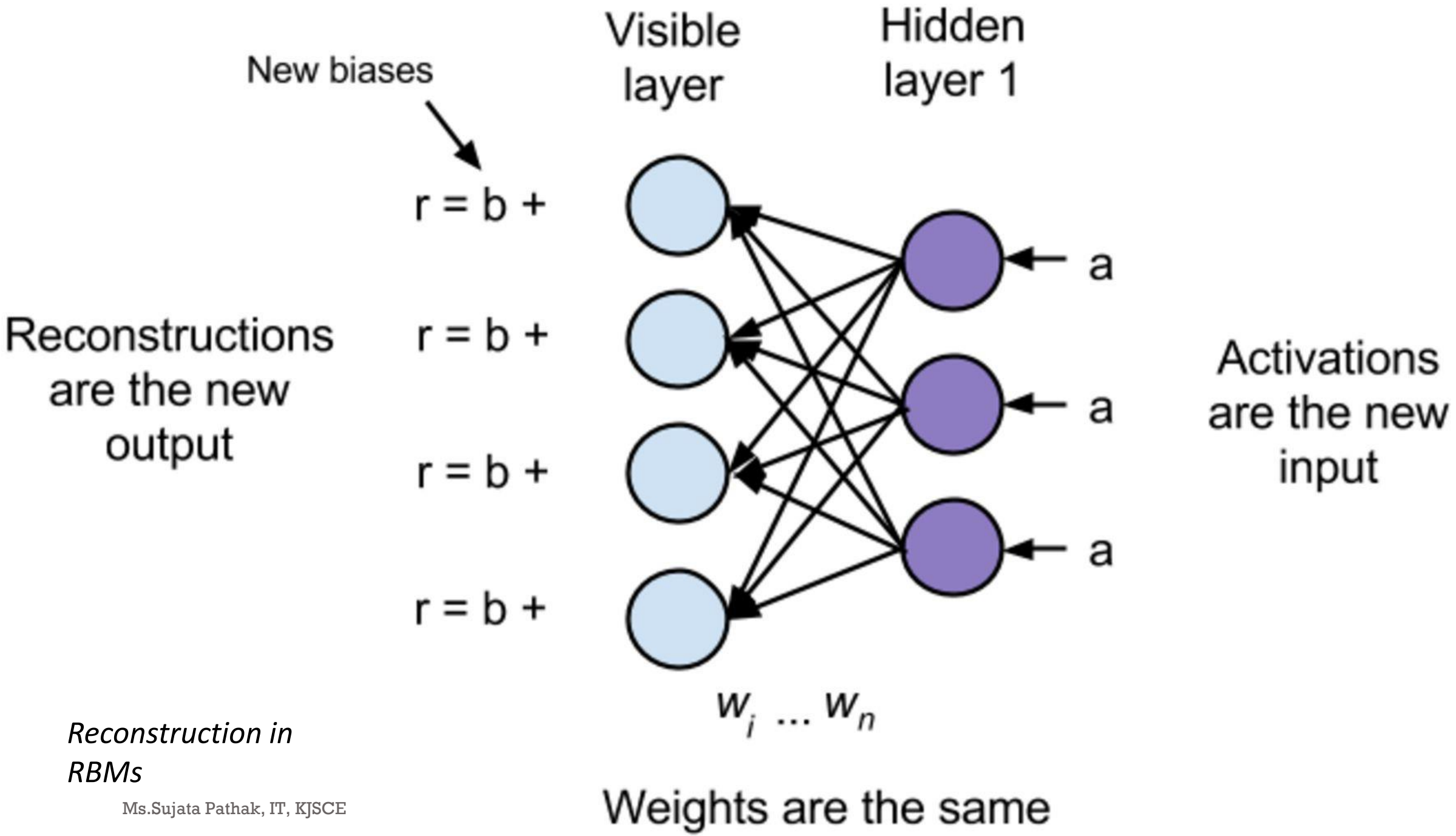
RBMS-



BUILDING BLOCKS OF DEEP NETWORKS

- **RBMs-**

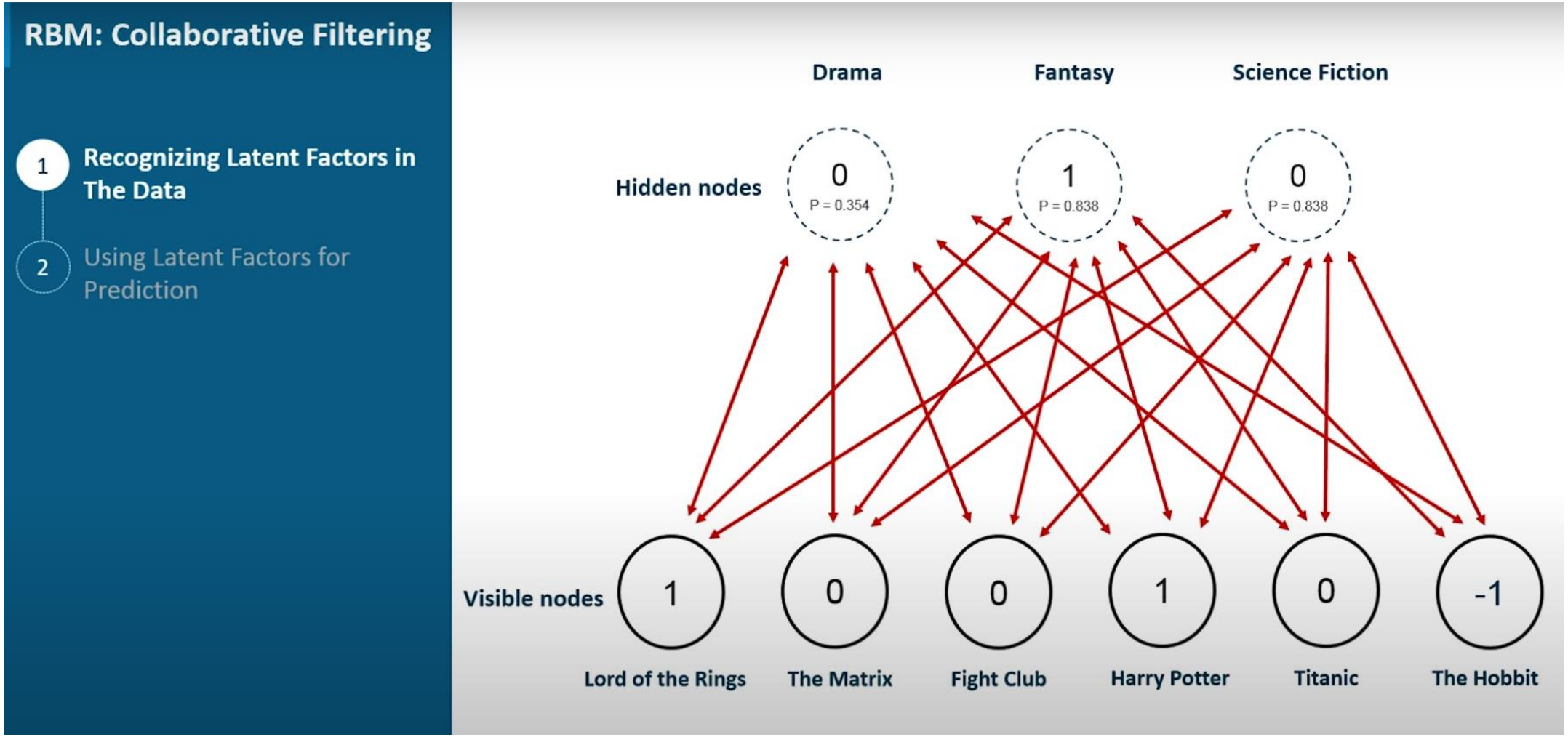




Reconstruction in RBMs

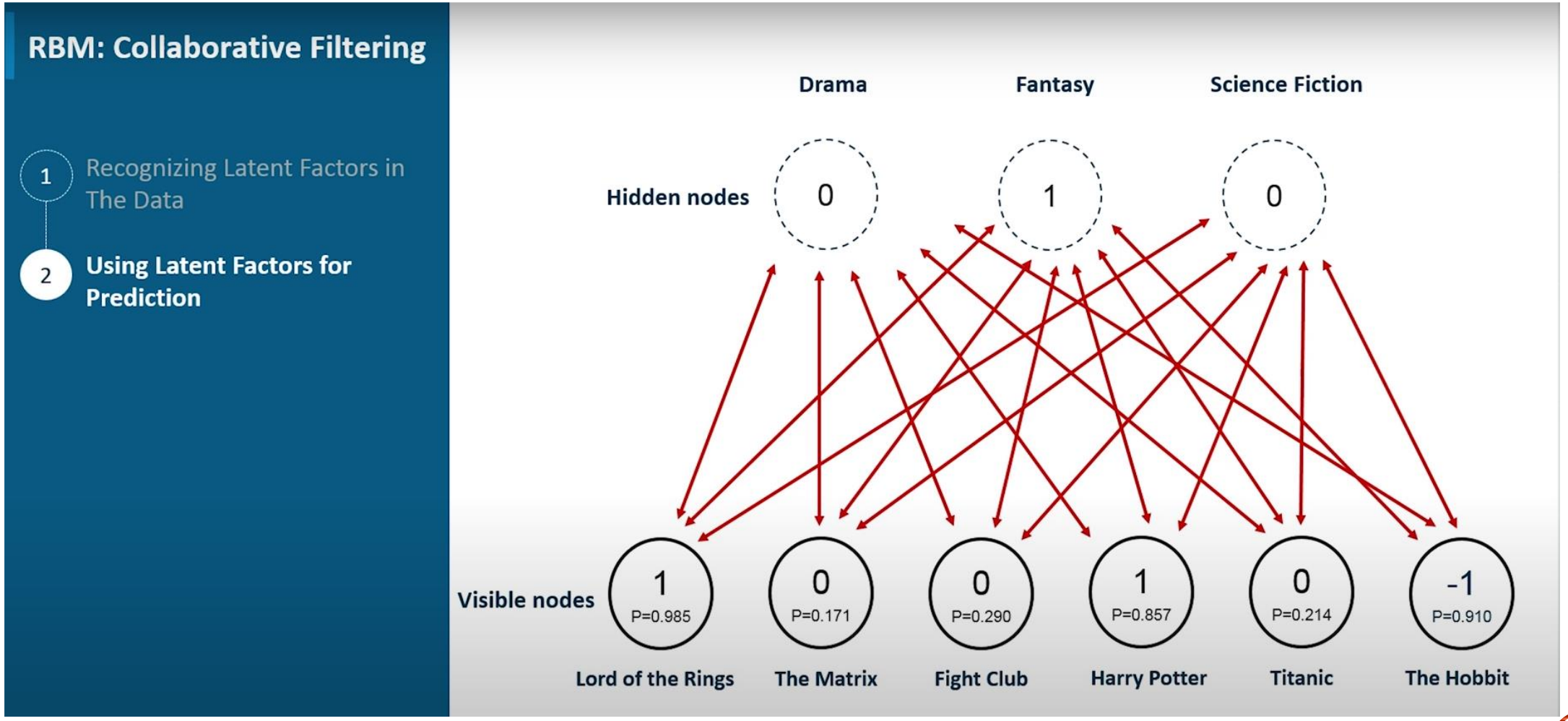
BUILDING BLOCKS OF DEEP NETWORKS

RBMS-



BUILDING BLOCKS OF DEEP NETWORKS

RBMS-

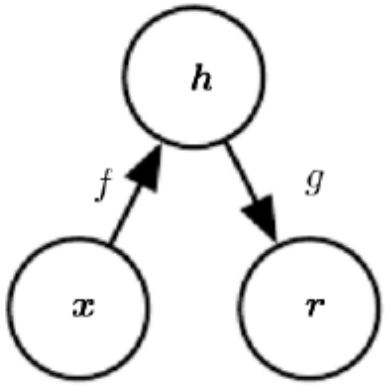


BUILDING BLOCKS OF DEEP NETWORKS

- **Other uses of RBMs-**
  - Dimensionality reduction
  - Classification
  - Regression
  - Collaborative filtering
  - Topic modelling
  - Imbalance data problem
  - Unstructured data
  - Noisy label problems

BUILDING BLOCKS OF DEEP NETWORKS

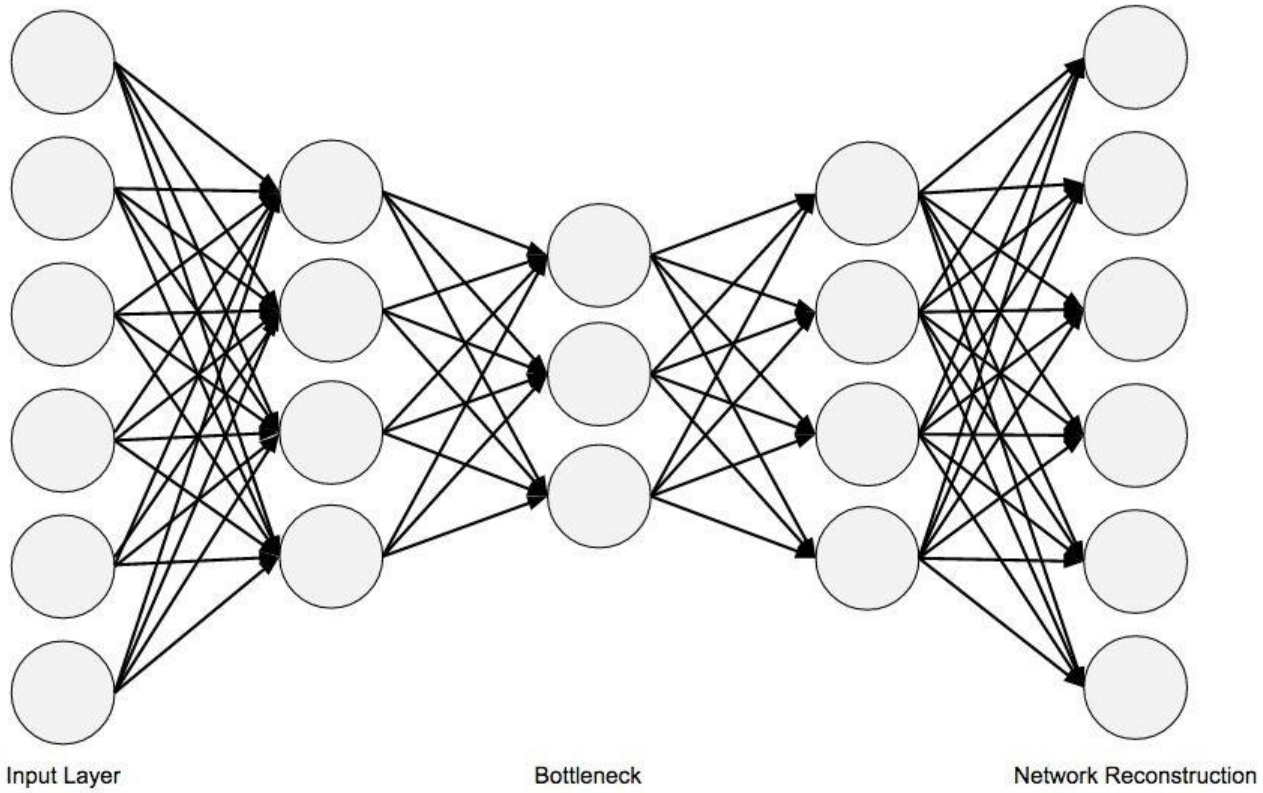
- Autoencoder is a neural network, which is trained to reproduce its input.
  - Encoder: $h = f(x)$
  - Decoder: $r = g(h)$



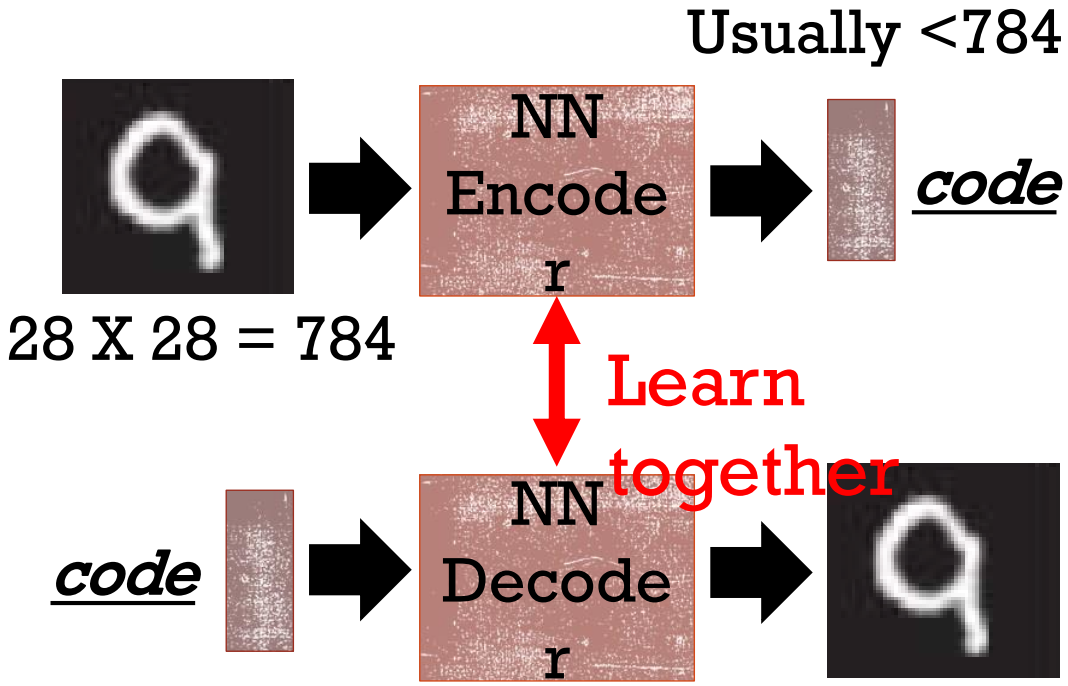
BUILDING BLOCKS OF DEEP NETWORKS

Autoencoders

- Used to learn compressed representations of datasets.
- To reduce a dataset's dimensionality.
- The output of the autoencoder network is a reconstruction of the input data in the most efficient form.
- **Similarities to multilayer perceptrons**
- Input layer, hidden layers of neurons, and then an output layer.
- Key diff- output layer in an autoencoder has the same number of units as the input layer does



AUTO-ENCODER



Compact representation of the input object

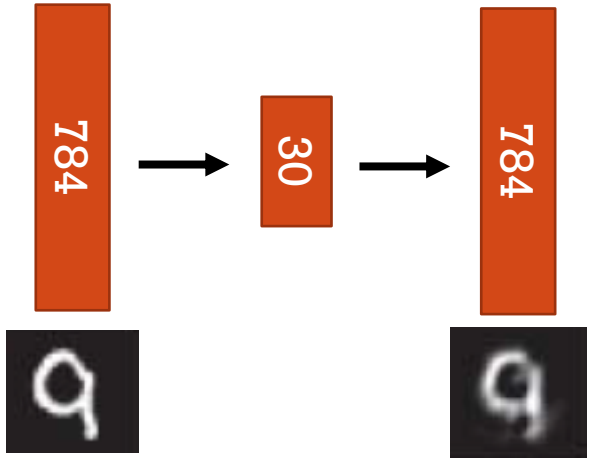
Can reconstruct the original object

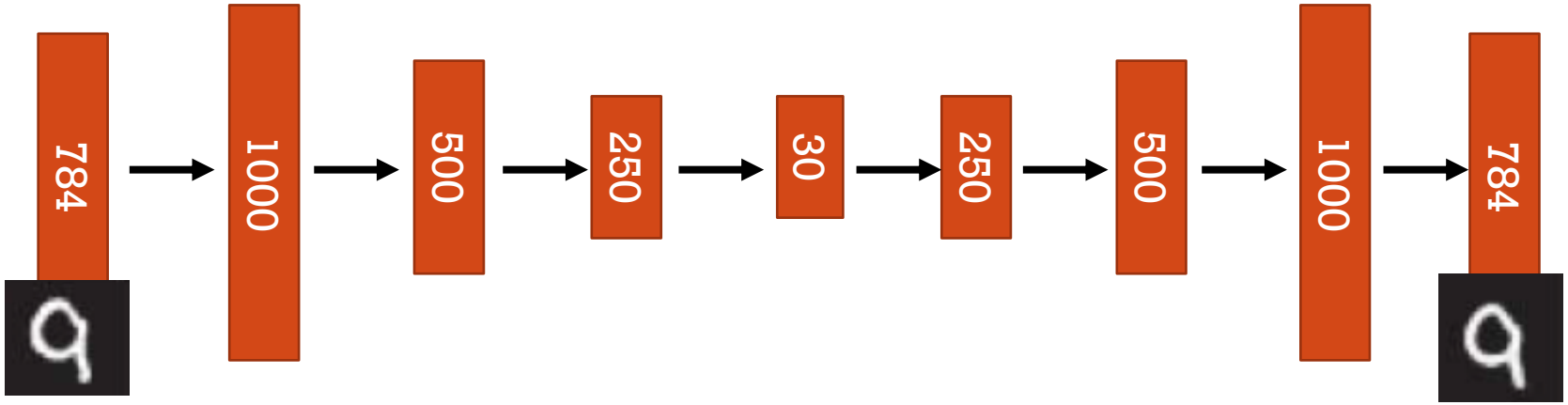
DEEP AUTO-ENCODER

Original Image

PCA

Deep Auto-encoder





BUILDING BLOCKS OF DEEP NETWORKS

Autoencoders-

Defining features of autoencoders

- Autoencoders differ from multilayer perceptrons in a couple of ways:
  - They use unlabeled data in unsupervised learning.
  - They build a compressed representation of the input data.
- ***Unsupervised learning of unlabeled data***
  - The autoencoder learns directly from unlabeled data.
  - Connected to the second major difference between multilayer perceptrons and autoencoders.
- ***Learning to reproduce the input data***
  - The goal of a multilayer perceptron network is to generate predictions over a class (e.g., fraud versus not fraud).
  - An autoencoder is trained to reproduce its own input data.

BUILDING BLOCKS OF DEEP NETWORKS

Autoencoders-

- **Training autoencoders**
- Autoencoders rely on backpropagation to update their weights.
- The main difference between RBMs and the more general class of autoencoders is in how they calculate the gradients

BUILDING BLOCKS OF DEEP NETWORKS

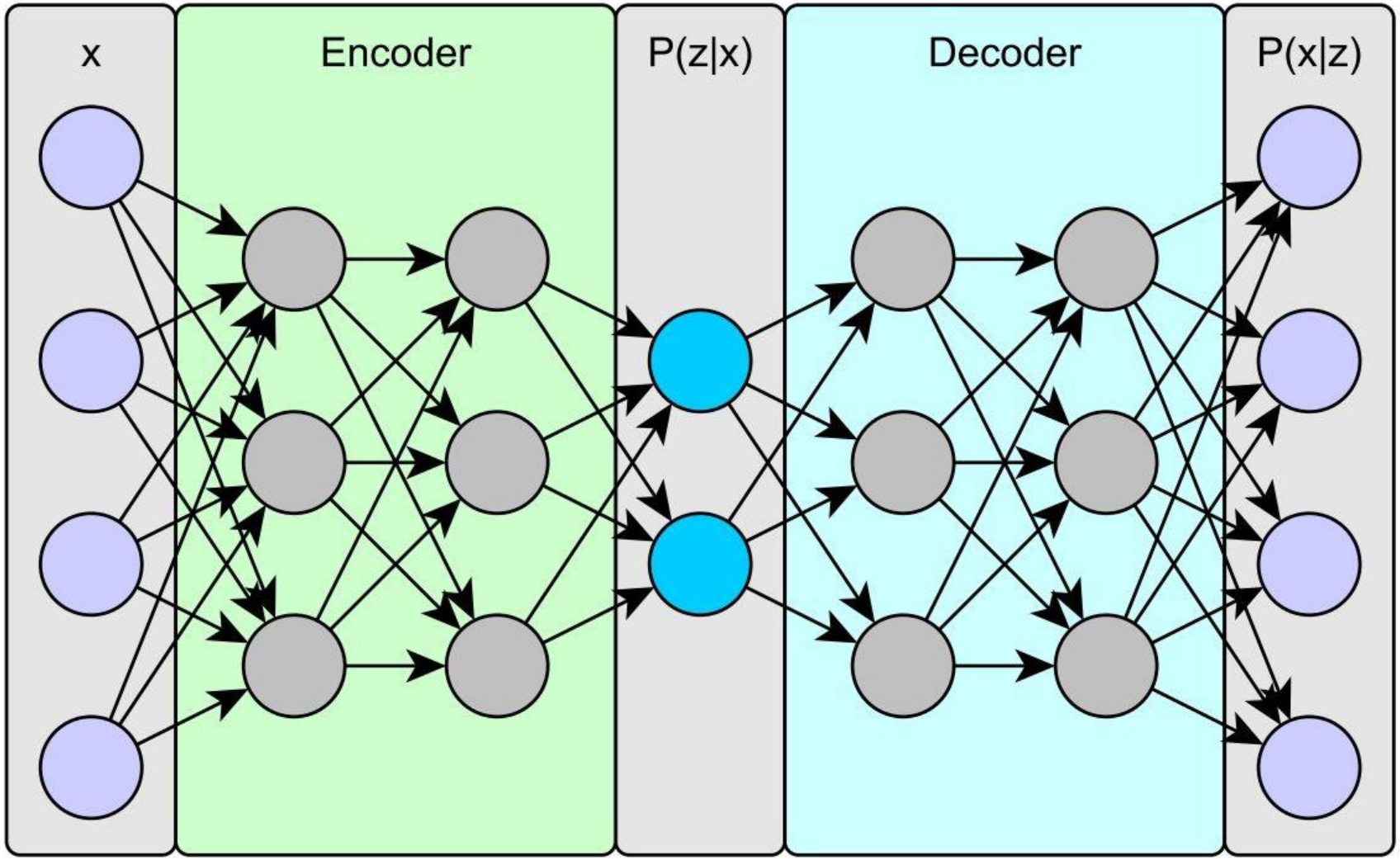
Common variants of autoencoders

- *Compression autoencoders*
- *Denoising autoencoders*.

- *Compression autoencoders*
  - The network input must pass through a bottleneck region of the network before being expanded back into the output representation.

- *Denoising autoencoders*
  - Autoencoder is given a corrupted version (e.g., some features are removed randomly) of the input and the network is forced to learn the uncorrupted output.

BUILDING BLOCKS OF DEEP NETWORKS

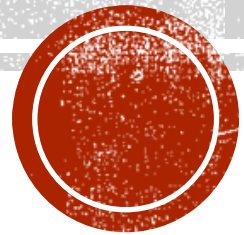


VAE network architecture

BUILDING BLOCKS OF DEEP NETWORKS

- Applications:
  - dimensionality reduction
    - visualization
    - feature extraction
    - ↑ prediction accuracy
    - ↑ speed of prediction
    - ↓ memory requirements
  - semantic hashing
  - unsupervised pretraining

MAJOR ARCHITECTURES OF DEEP NETWORKS



MAJOR ARCHITECTURES OF DEEP NETWORKS

- Unsupervised Pretrained Networks (UPNs)
- Convolutional Neural Networks (CNNs)
- Recurrent Neural Networks
- Recursive Neural Networks

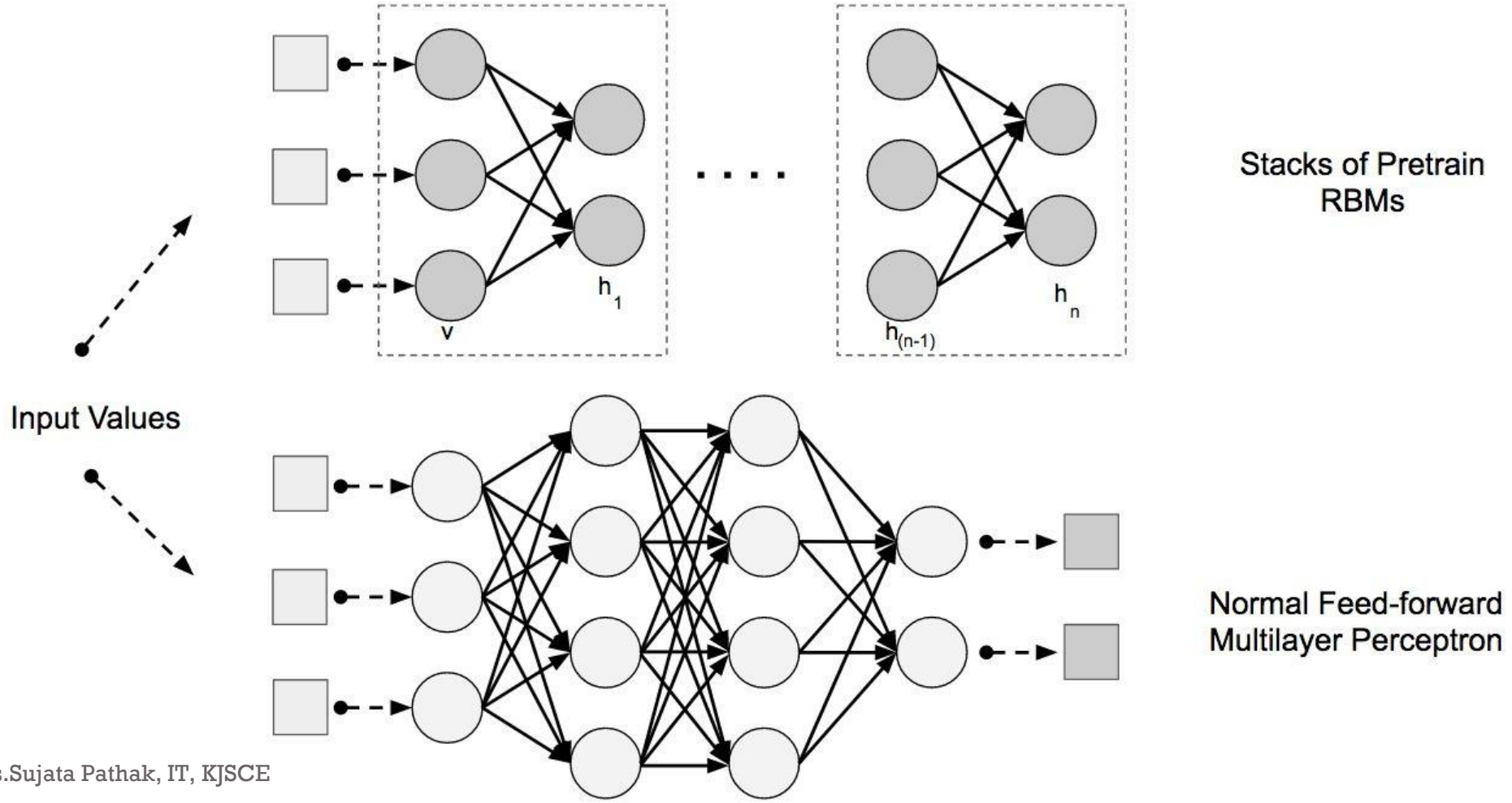
UNSUPERVISED PRETRAINED NETWORKS

Specific architectures:

- Autoencoders
- Deep Belief Networks (DBNs)
- Generative Adversarial Networks (GANs)

DEEP BELIEF NETWORKS

- Composed of layers of Restricted Boltzmann Machines (RBMs) for the pretrain phase and then a feed-forward network for the fine-tune phase

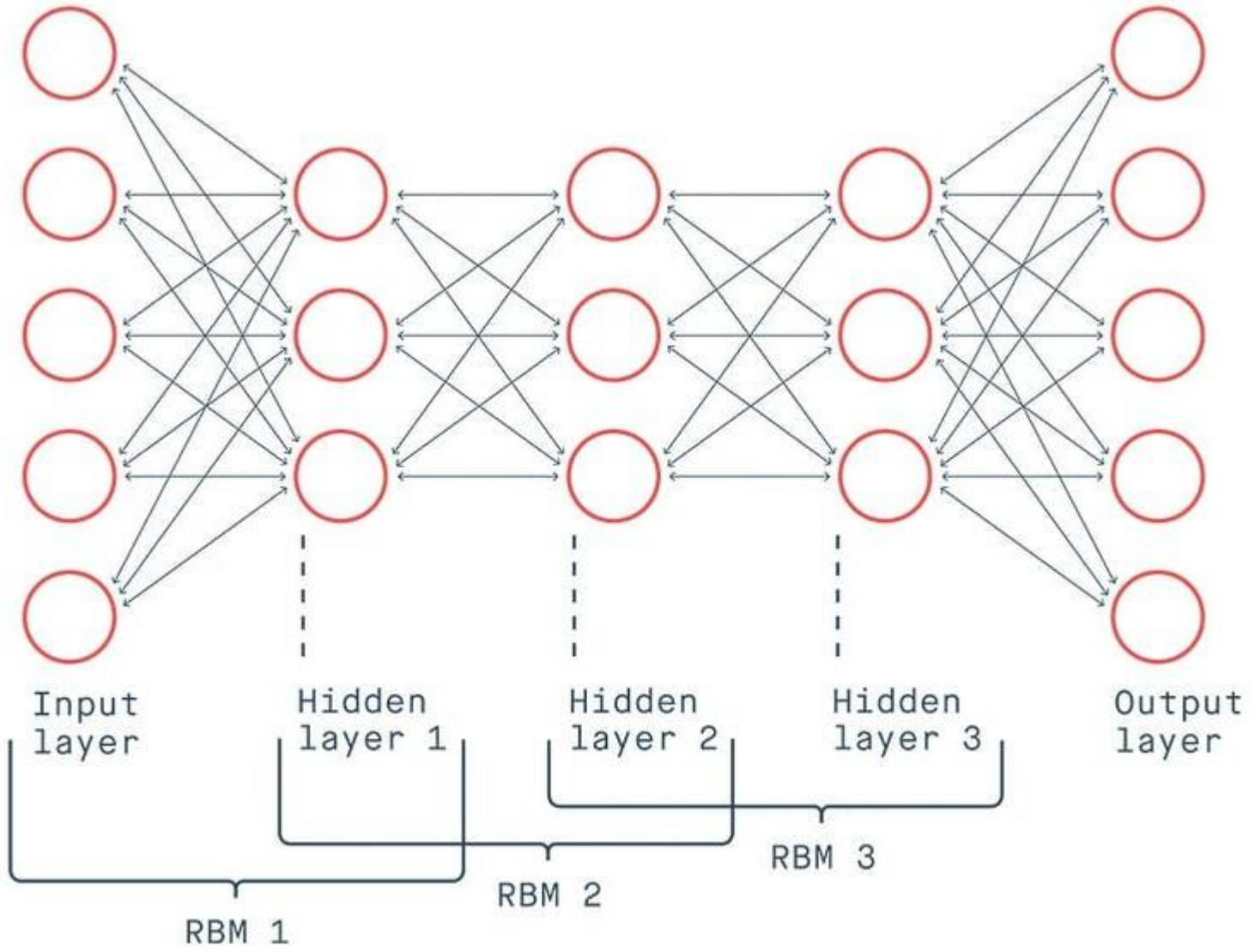


DEEP BELIEF NETWORKS

- Backpropagation-local minima
- Solution-Pre-training
- Pre-training-Back propagation-reduced error rate

DEEP BELIEF NETWORKS

- Network structure- DBN identical to MLP
- Training- DBN differs with MLP
- Role of Hidden nodes



DEEP BELIEF NETWORKS

Training-

- Greedy learning algorithm- from bottom layer moving upside, fine tuning generative weights
- Greedy learning algorithms- Quick and efficient, Optimize weights at each layer
- Each layer learns entire input
- DBN work globally and regulate each layer in order, as the model slowly improves

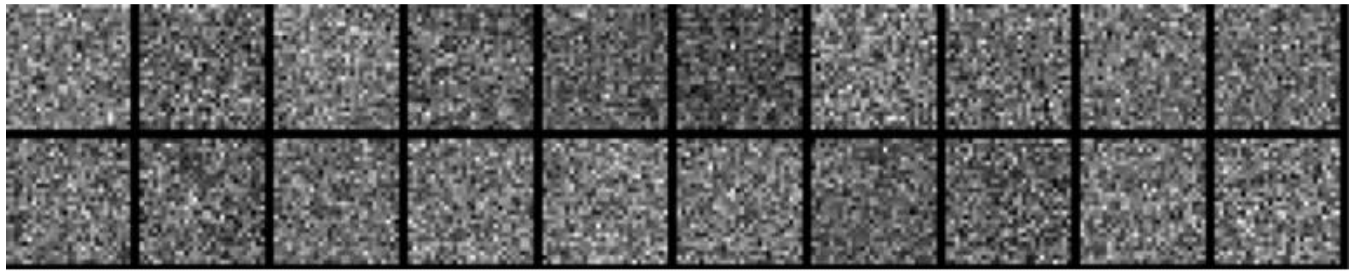
DEEP BELIEF NETWORKS

- **Feature Extraction with RBM Layers-**
  - RBMs -to extract higher-level features from the raw input vectors.
  - The fundamental purpose of RBMs in the context of deep learning and DBNs -to learn these higher-level features of a dataset in an unsupervised training fashion.

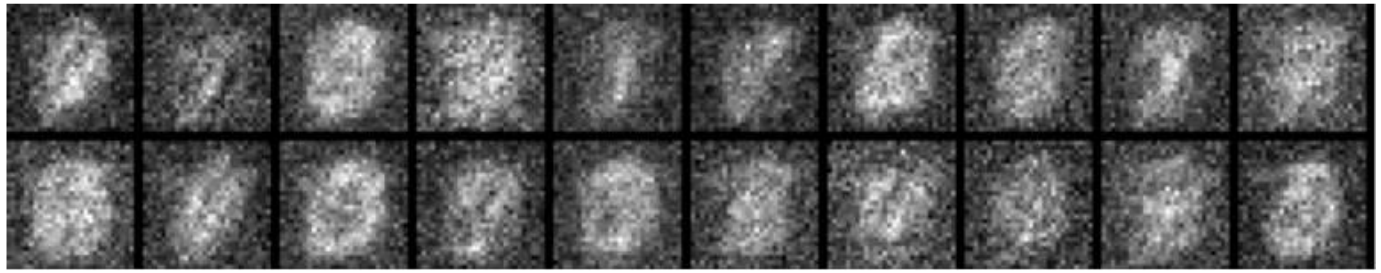
Learning higher-order features automatically-

Initializing the feed-forward network-*fine-tune phase* of DBNs

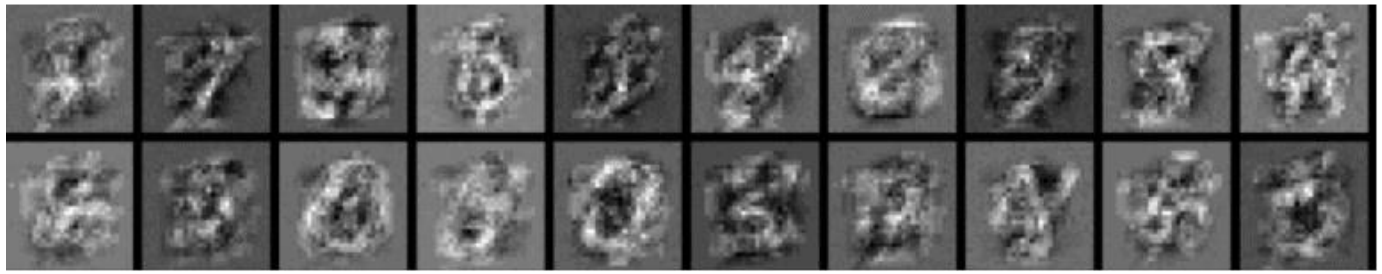
DEEP BELIEF NETWORKS



Activation render at the beginning of training



Features emerge in later activation render Figure



Portions of MNIST digits emerge towards end of training

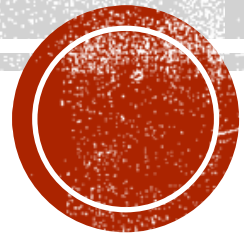
DEEP BELIEF NETWORKS

- **Fine-tuning a DBN with a feed-forward multilayer neural network-**
  - *Gentle backpropagation*
  - *The output layer*

Applications-

- *Image recognition*
- *Video recognition*
- *Motion capture data*

GENERATIVE ADVERSARIAL NETWORKS (GAN)

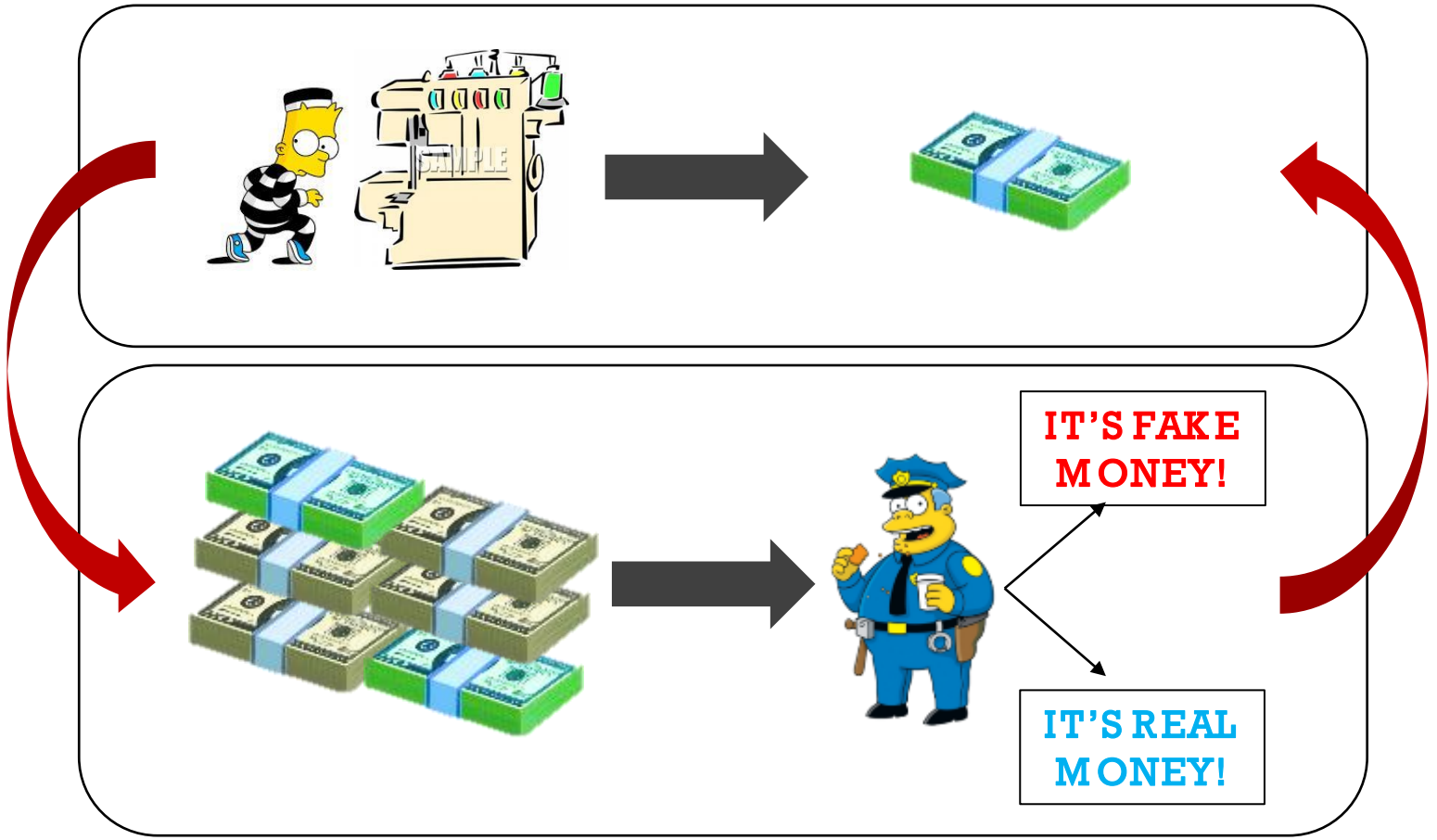


WHAT ARE GANS?

- System of two neural networks competing against each other in a zero-sum game framework.
- They were first introduced by Ian Goodfellow *et al.* in 2014.
- Can learn to draw samples from a model that is similar to data that we give them.

The intuition behind GAN: Counterfeiters vs Police Game

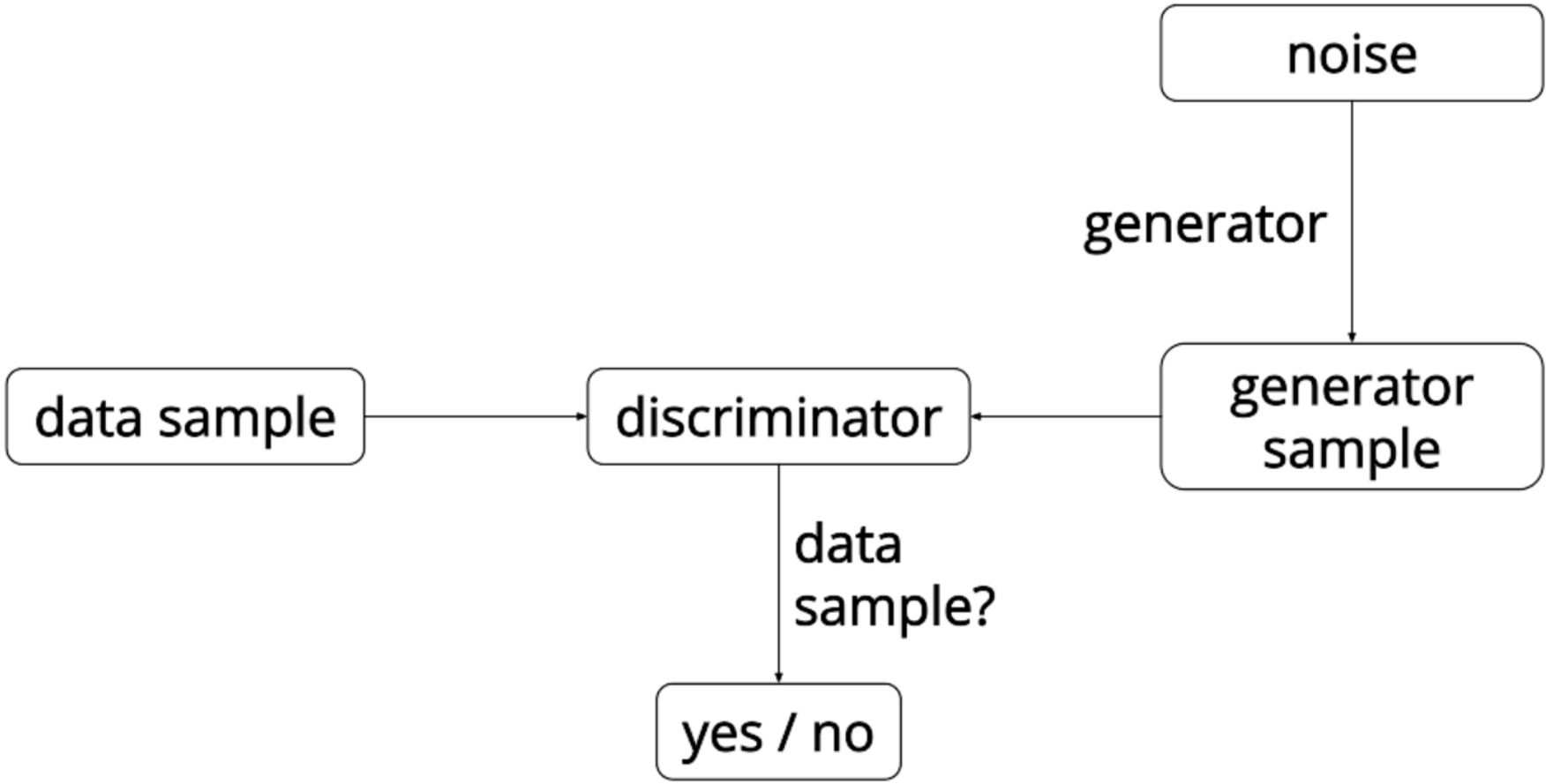
- **The generative model can be thought of as analogous to a team of counterfeiters, trying to produce fake currency, while the discriminative model is analogous to the police, trying to detect the counterfeit currency. Competition in this game drives both teams to improve their methods until the counterfeits are indistinguishable from the genuine articles.**



-Goodfellow et. al., "Generative Adversarial Networks" (2014)

What are GANs?

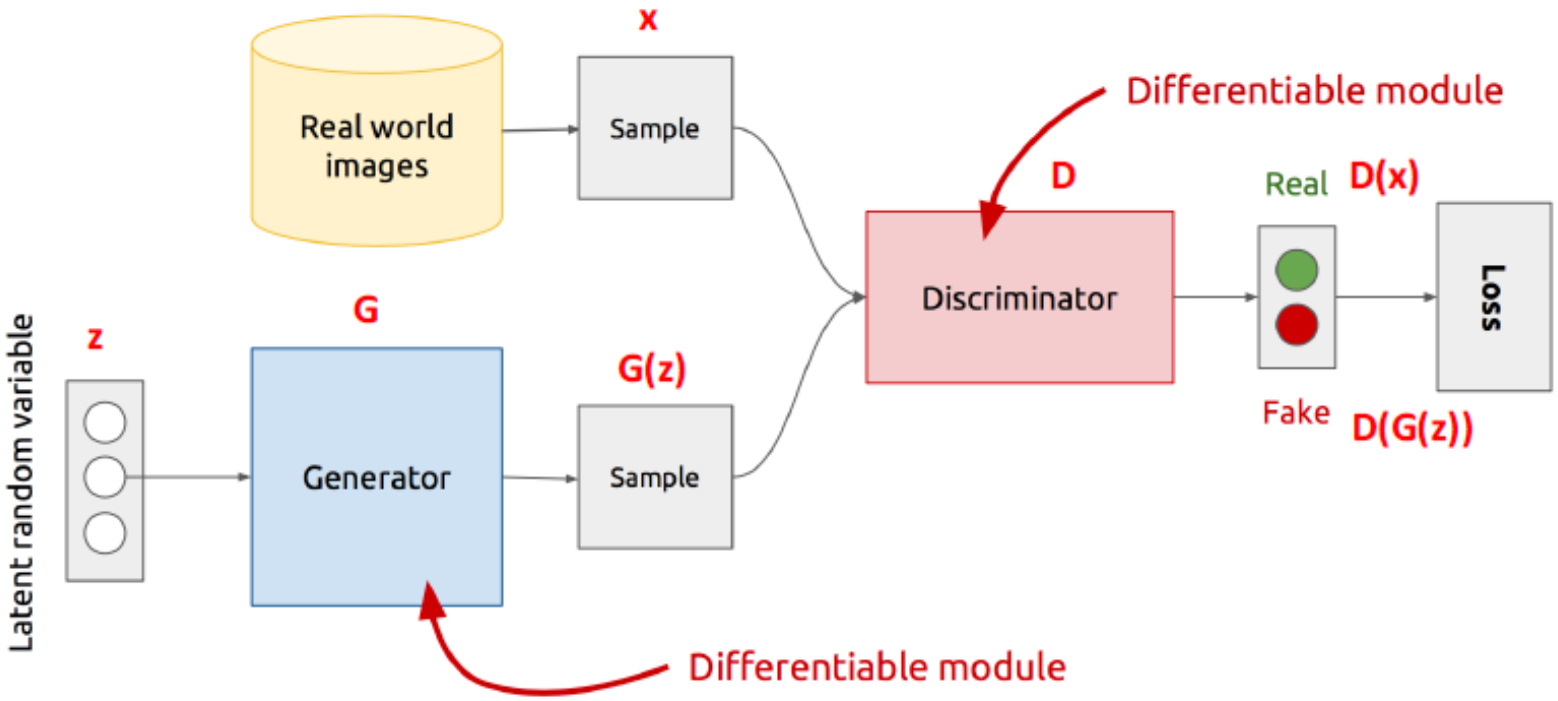
- **GAN is a new framework for estimating generative models via an adversarial process, in which we simultaneously train two models:**
  - **a generative model G that captures the data distribution.**
  - **a discriminative model D that estimates the probability that a sample came from the training data rather than G.**
- **The training procedure for G is to maximize the probability of D making a mistake. This framework corresponds to a minimax two-player game.**
- **In the case where G and D are defined by multilayer perceptrons, the entire system can be trained with backpropagation.**



Overview of GANs

Source: https://ishmaelbelghazi.github.io/ALI

GAN'S ARCHITECTURE



- **Z** is some random noise (Gaussian/Uniform).
- **Z** can be thought as the latent representation of the image.

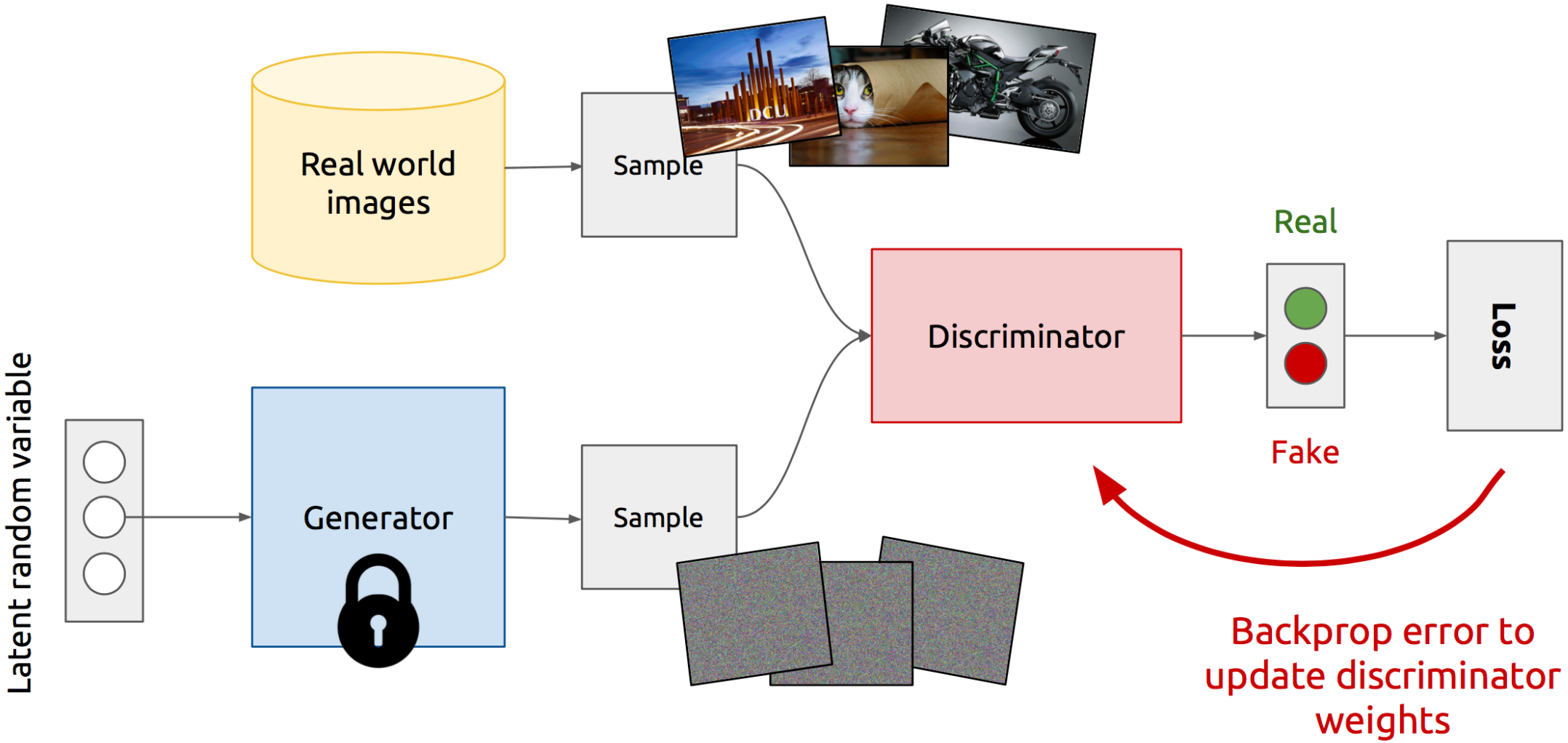
https://www.slideshare.net/xavigiro/deep-learning-for-computer-vision-generative-models-and-adversarial-training-upc-2016

DISCRIMINATIVE MODELS

- A **discriminative** model learns a function that maps the input data (x) to some desired output class label (y).

- In probabilistic terms, they directly learn the conditional distribution $P(y|x)$.

TRAINING DISCRIMINATOR

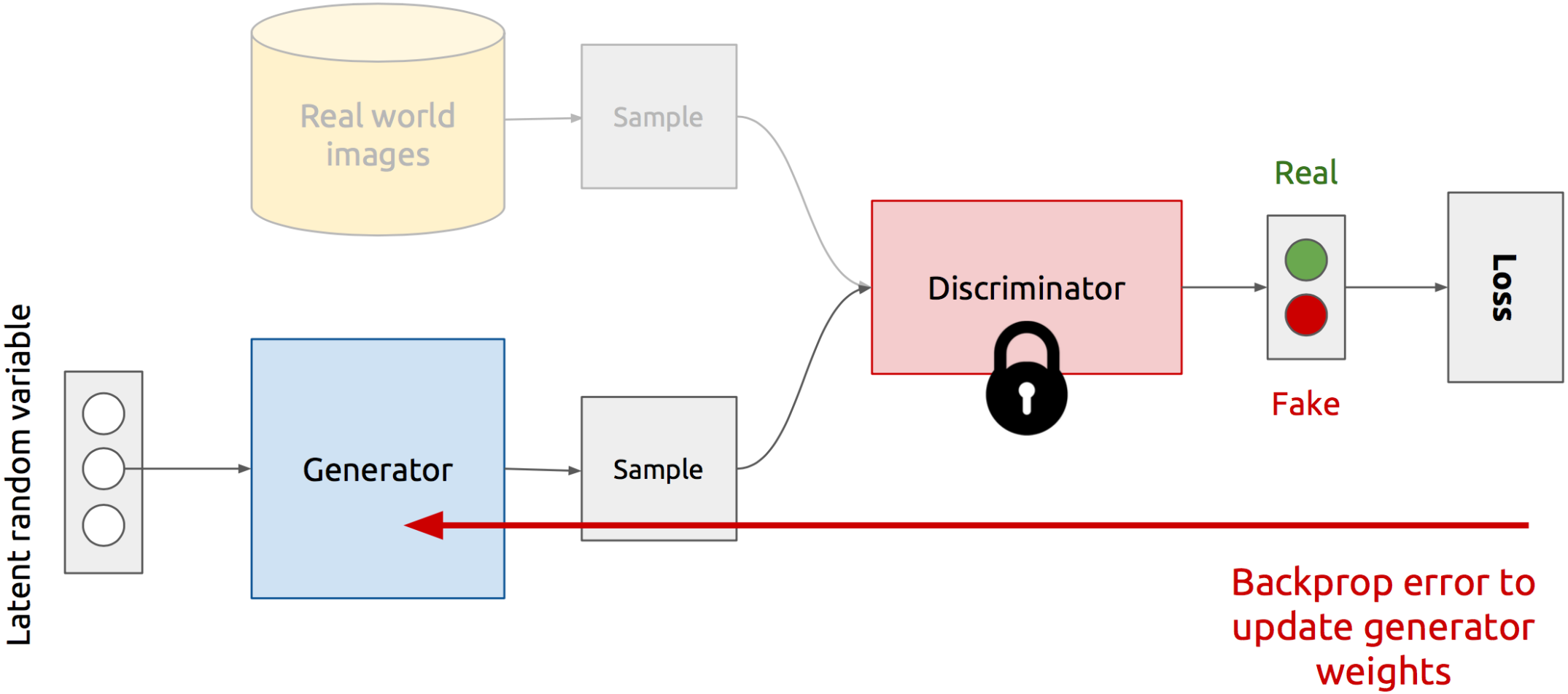


GENERATIVE MODELS

- A **generative** model tries to learn the joint probability of the input data and labels simultaneously i.e. *P(x,y)*.

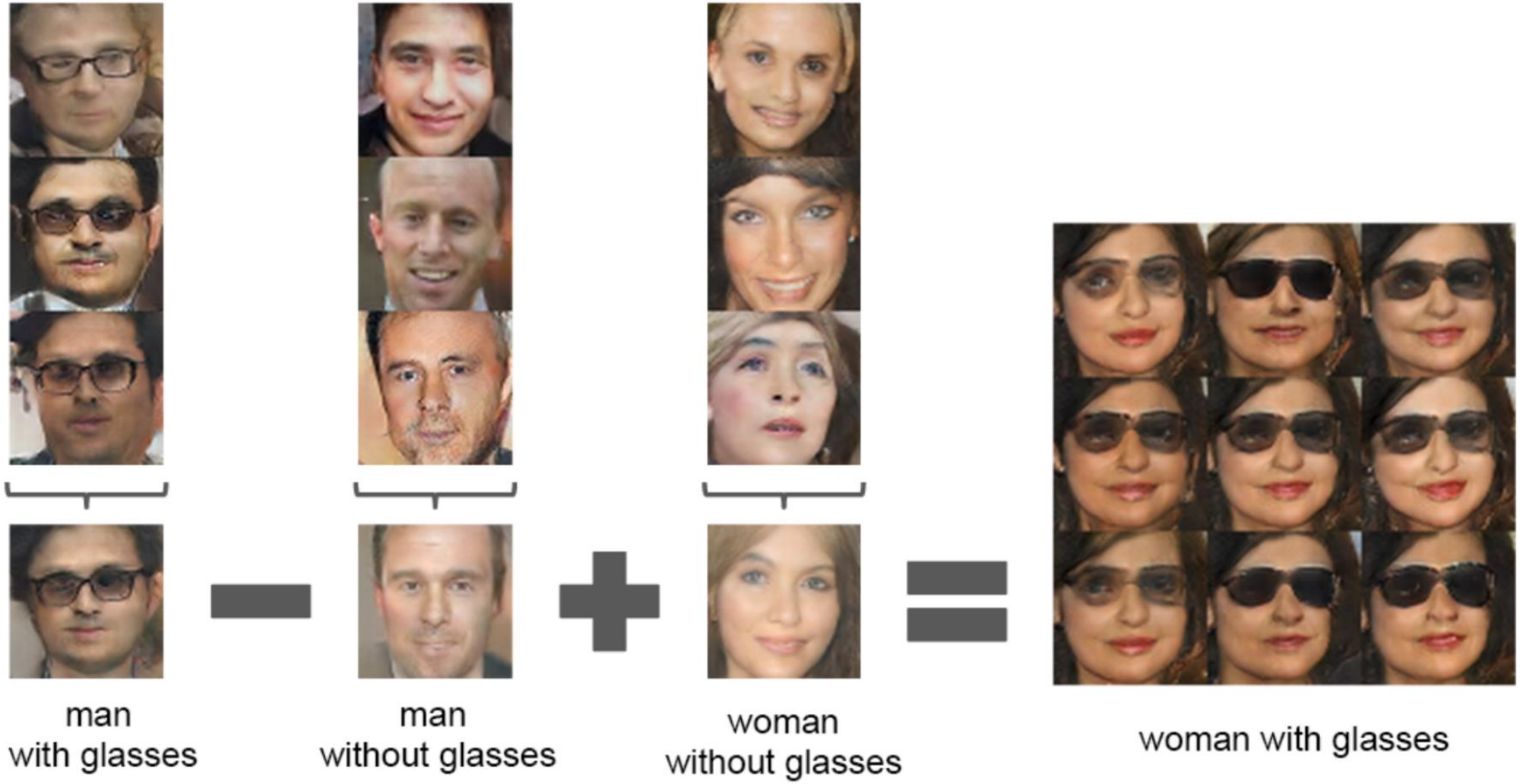
- Potential to understand and explain the underlying structure of the input data even when there are no labels.

TRAINING GENERATOR



HOW GANS ARE BEING USED?

- Applied for modelling natural images.
- Performance is fairly good in comparison to other generative models.
- Useful for unsupervised learning tasks.


man
with glasses
man
without glasses
woman
without glasses
woman with glasses

HOW TO TRAIN GANS?

- Objective of generative network - increase the error rate of the discriminative network.
- Objective of discriminative network – decrease binary classification loss.
- Discriminator training - backprop from a binary classification loss.
- Generator training - backprop the negation of the binary classification loss of the discriminator.

LOSS FUNCTIONS

$$\mathcal{L}(\hat{x}) = \min_{x \in data} (x - \hat{x})^2$$

Generator

$$D_G^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_g(x)}$$

Discriminator

"IMPROVED TECHNIQUES FOR TRAINING GANS" BY SALIMANS ET. AL

- One-sided Label smoothing - replaces the 0 and 1 targets for a classifier with smoothed values, like .9 or .1 to reduce the vulnerability of neural networks to adversarial examples.

- Virtual batch Normalization - each example x is normalized based on the statistics collected on a reference batch of examples that are chosen once and fixed at the start of training, and on x itself.

VARIATIONS TO GANS

- Several new concepts built on top of GANs have been introduced –
  - InfoGAN – Approximate the data distribution and learn interpretable, useful vector representations of data.
  - Conditional GANs - Able to generate samples taking into account external information (class label, text, another image). Force *G* to generate a particular type of output.

MAJOR DIFFICULTIES

- Networks are difficult to converge.
- Ideal goal – Generator and discriminator to reach some desired equilibrium but this is rare.
- GANs are yet to converge on large problems (E.g. Imagenet).

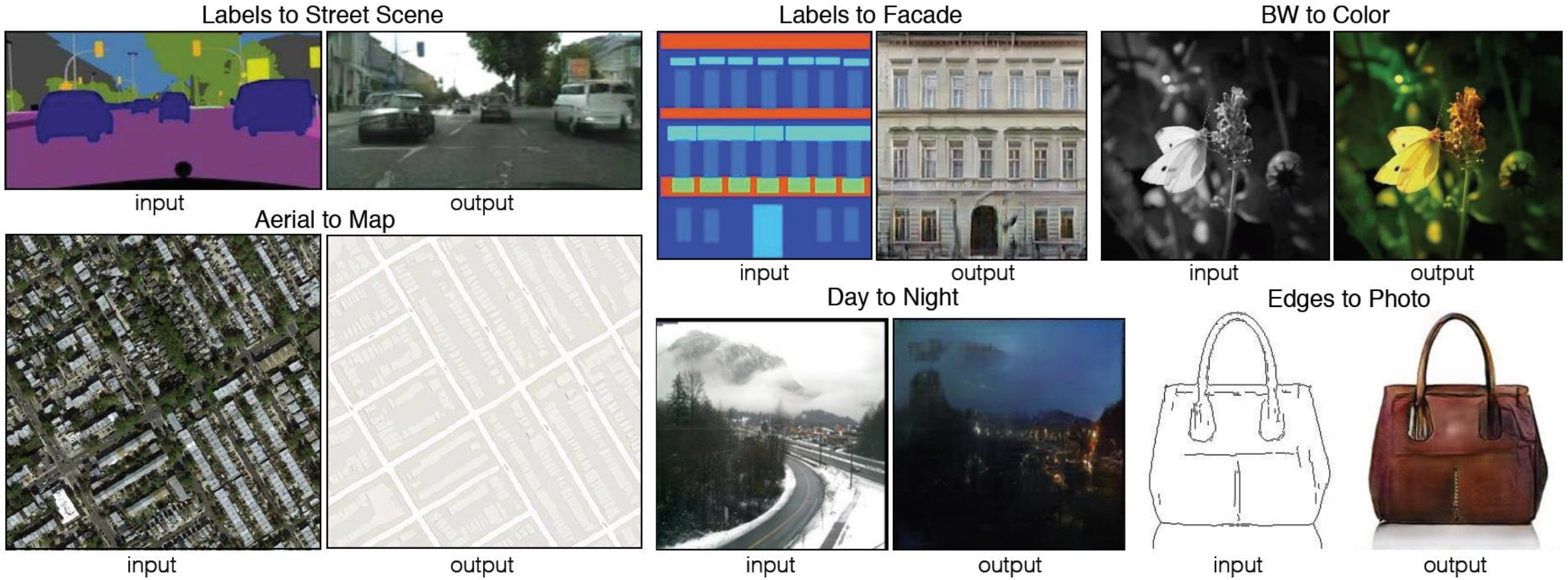
COMMON FAILURE CASES

- The discriminator becomes too strong too quickly and the generator ends up not learning anything.
- The generator only learns very specific weaknesses of the discriminator.
- The generator learns only a very small subset of the true data distribution.

SO WHAT CAN WE DO?

- Normalize the inputs
- A modified loss function
- BatchNorm
- Avoid Sparse Gradients: ReLU, MaxPool
- Use Soft and Noisy Labels
- DCGAN / Hybrid Models
- Track failures early (D loss goes to 0: failure mode)
- If you have labels, use them
- Add noise to inputs, decay over time

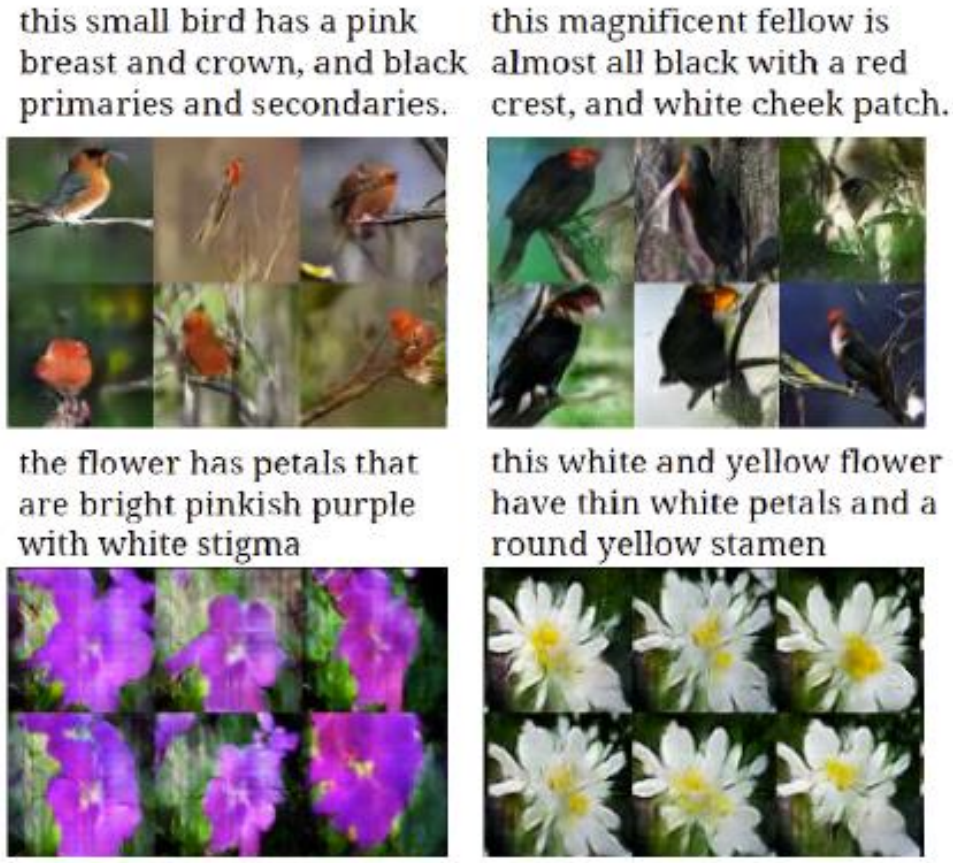
APPLICATIONS OF GANS



Isola, P., Zhu, J. Y., Zhou, T., & Efros, A. A. "**Image-to-image translation with conditional adversarial networks**". arXiv preprint arXiv:1611.07004. (2016).

APPLICATIONS OF GANS

- Motivation
- Given a text description, generate images closely associated.
- Uses a conditional GAN with the generator and discriminator being condition on “dense” text embedding.



WHY USE GANS FOR GENERATION?

- Can be trained using back-propagation for Neural Network based Generator/Discriminator functions.
- Sharper images can be generated.
- Faster to sample from the model distribution: *single* forward pass generates a *single* sample.

CONCLUSIONS

- GANs are generative models that are implemented using two stochastic neural network modules: **Generator** and **Discriminator**.
- **Generator** tries to generate samples from random noise as input
- **Discriminator** tries to distinguish the samples from Generator and samples from the real data distribution.
- Both networks are trained adversarially (in tandem) to fool the other component. In this process, both models become better at their respective tasks.

CONCLUSIONS

- Train GAN – Use discriminator as base model for transfer learning and the fine-tuning of a production model.

- A well-trained generator has learned the true data distribution well - Use generator as a source of data that is used to train a production model.

REFERENCES

- https://tryolabs.com/blog/2016/12/06/major-advancements-deep-learning-2016/
- https://blog.waya.ai/introduction-to-gans-a-boxing-match-b-w-neural-nets-b4e5319cc935#.6l7zh8u50
- https://en.wikipedia.org/wiki/Generative_adversarial_networks
- http://blog.aylien.com/introduction-generative-adversarial-networks-code-tensorflow/
- https://github.com/soumith/ganhacks
- https://www.youtube.com/watch?v=i64KpxyaLpo