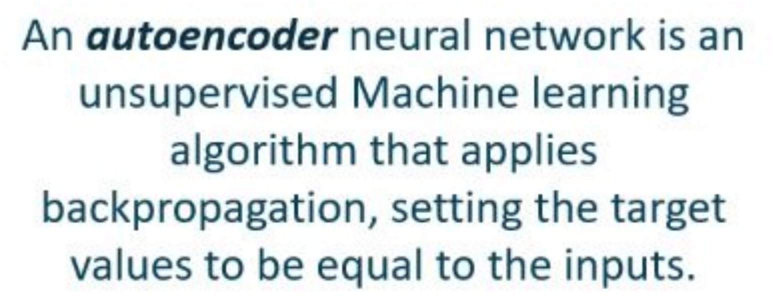
DEEP LEARNING METHODOLOGIES Part - II

Lecture 7

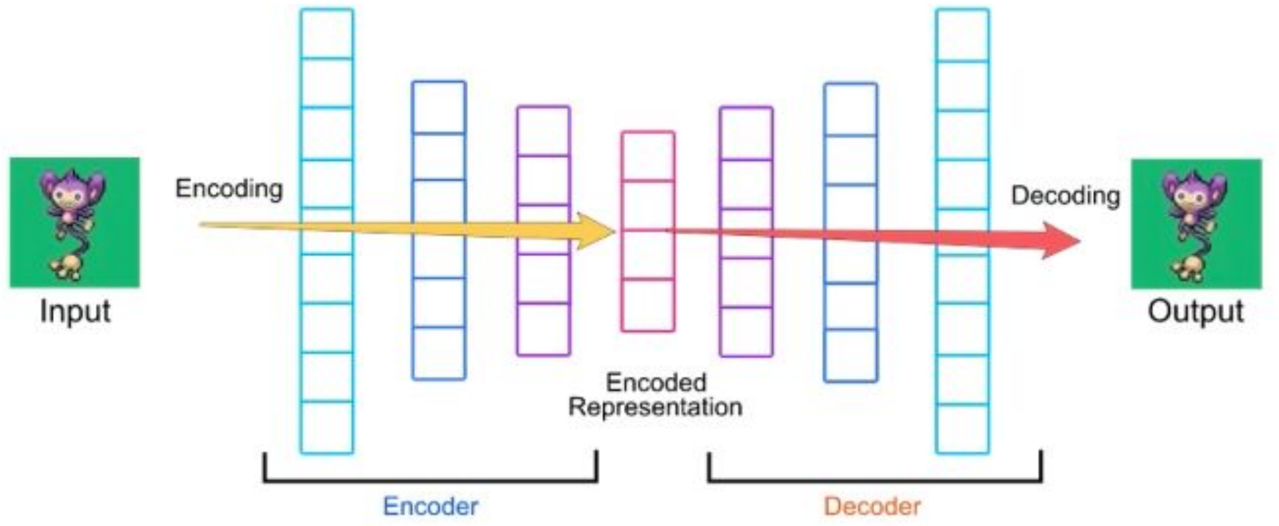
AutoEncoders

AutoEncoders



An ***autoencoder*** neural network is an unsupervised Machine learning algorithm that applies backpropagation, setting the target values to be equal to the inputs.

AutoEncoders



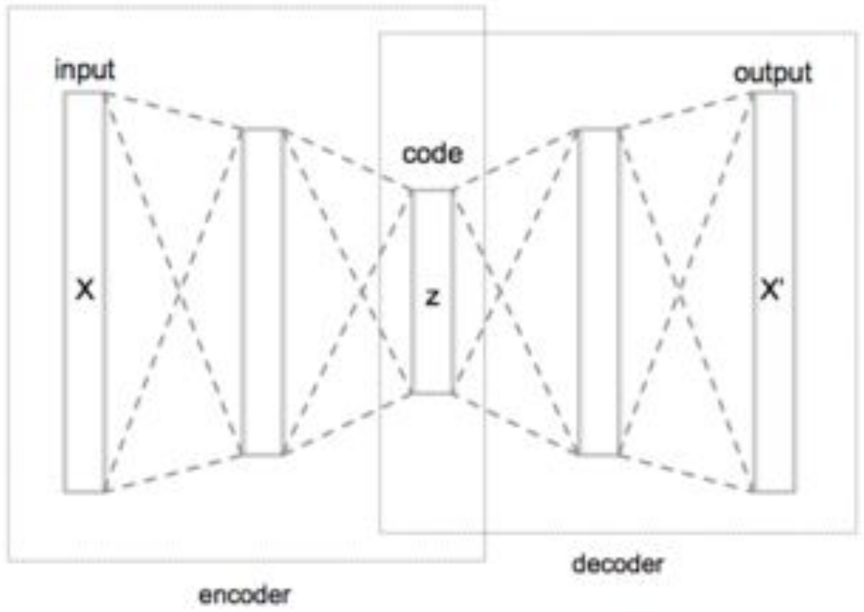
AutoEncoders

- An autoencoder is a simple neural network that is composed of two parts:
  - ***Encoder***
  - ***Decoder.***
- ***The encoder part will compress your input into a smaller dimension.***
- ***From this smaller dimension, it then tries to reconstruct the input using the decoder part of the model.***

AutoEncoders

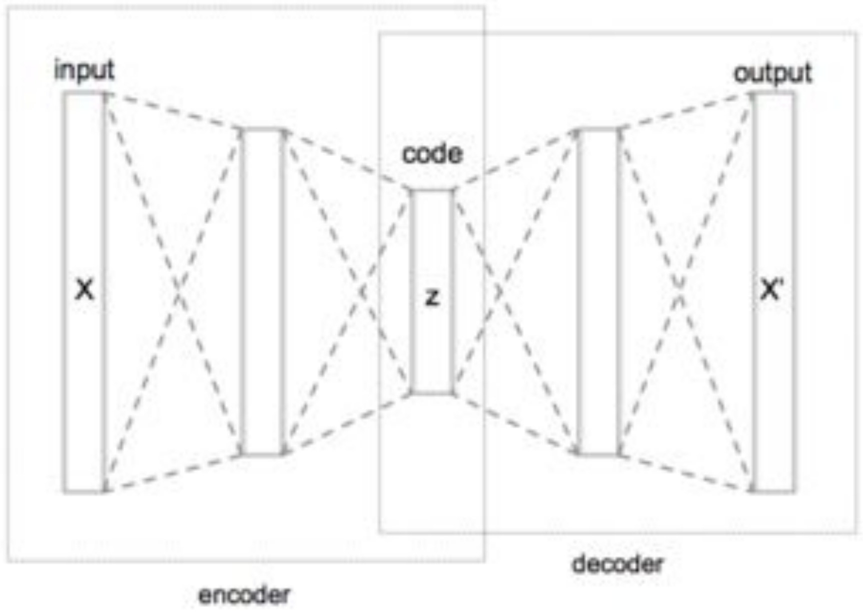
The smaller dimension is called by many names such as -

- ***Latent space***
- ***Hidden space***
- ***Embedding***
- ***Coding***



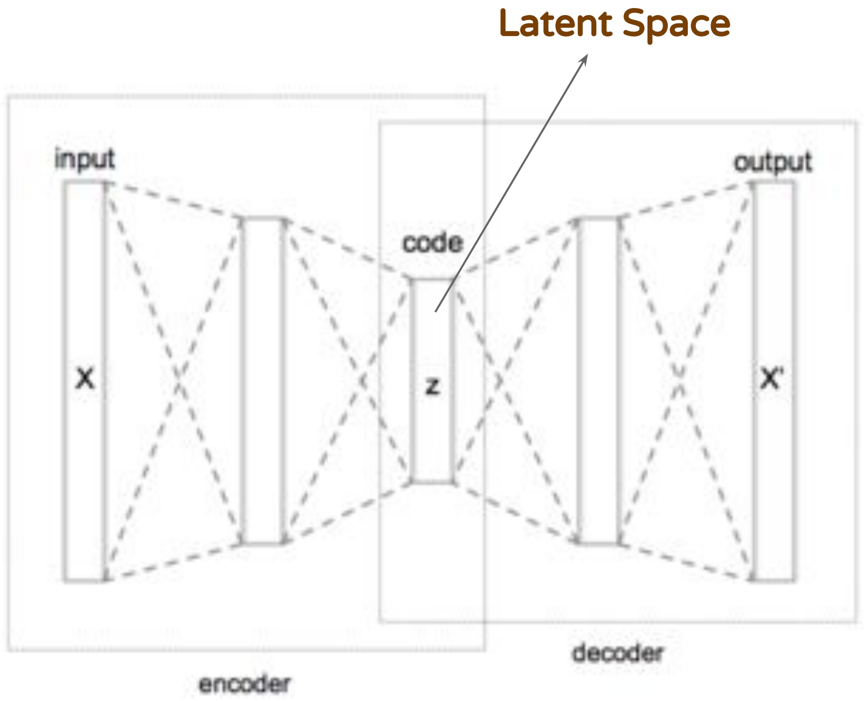
AutoEncoders

- This latent space encodes all the important information needed to represent the original data, but with the advantage of being a smaller dimension than the input.

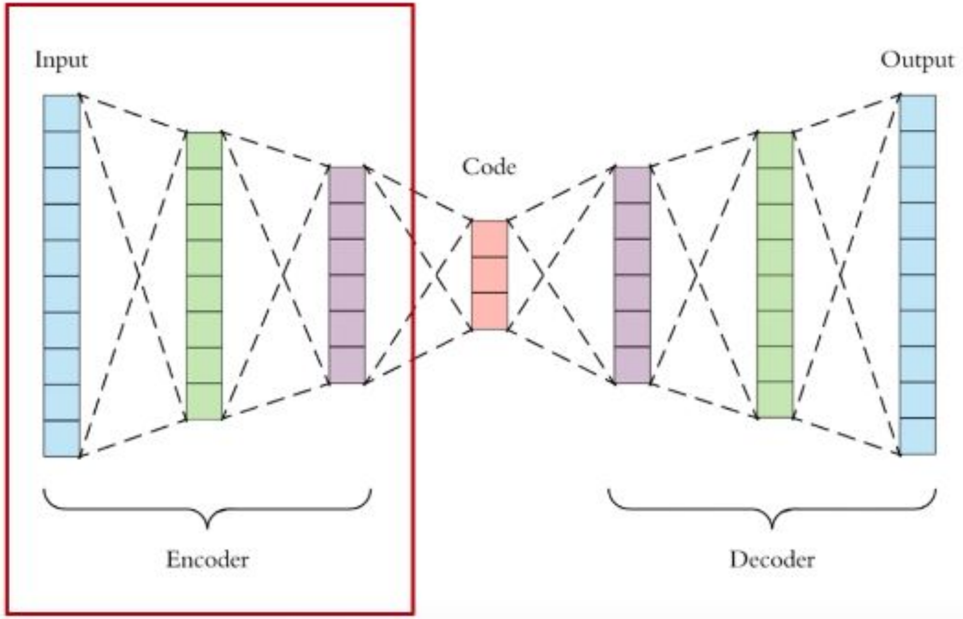


AutoEncoders

- The encoder can be thought of as a way of compressing the input data while the decoder is the way to uncompress it.



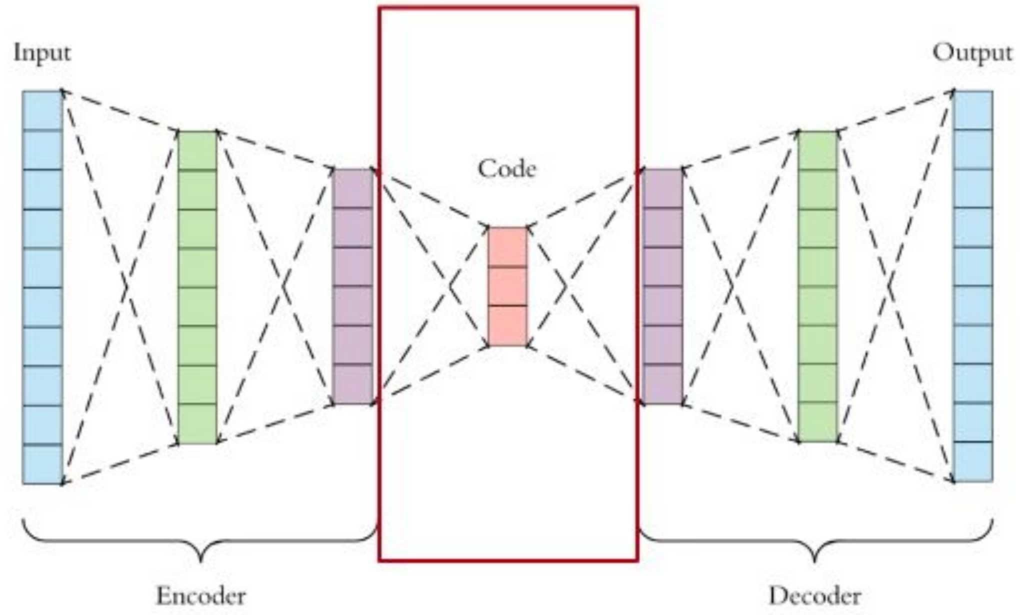
AutoEncoders



01 Encoder.

This is the part of the network that compresses the input into a latent space representation.

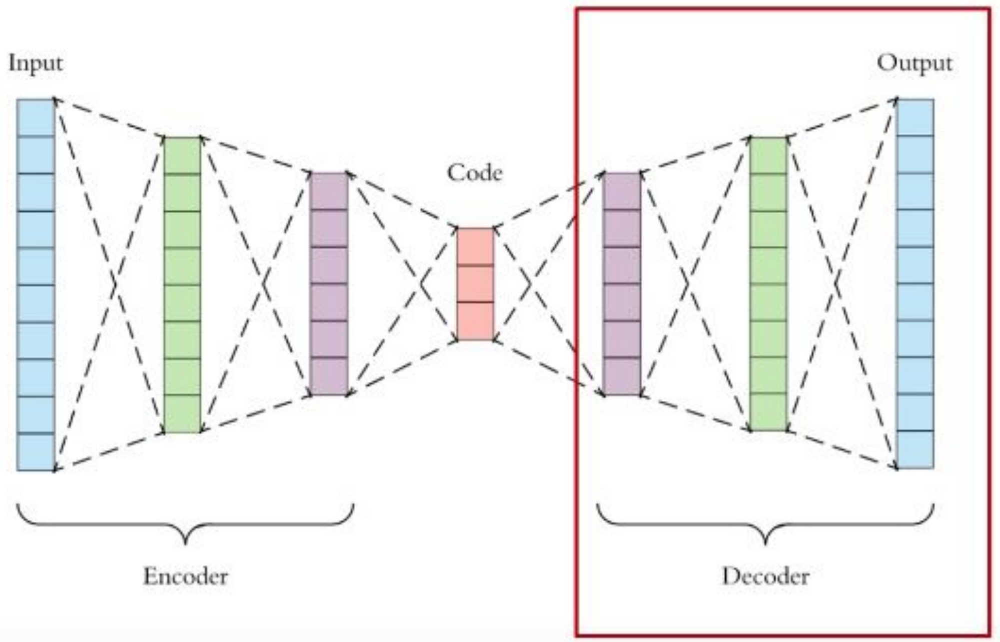
AutoEncoders



02 Code

This is the part of the network represents the compressed input that is fed to the decoder

AutoEncoders



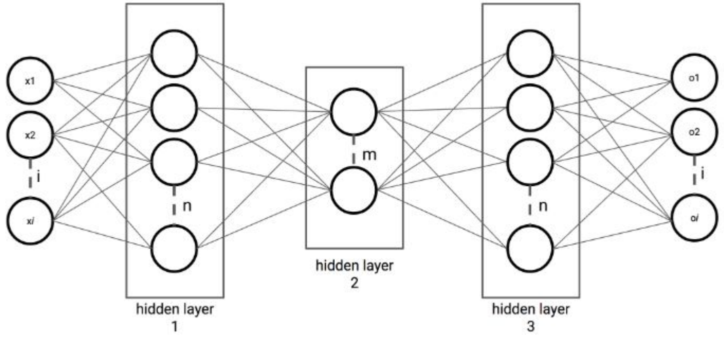
03 Decoder

This part aims to reconstruct the input from the latent space representation

AutoEncoders using CNN Layers

- Autoencoders can be extended to images as well by using convolutional layers.

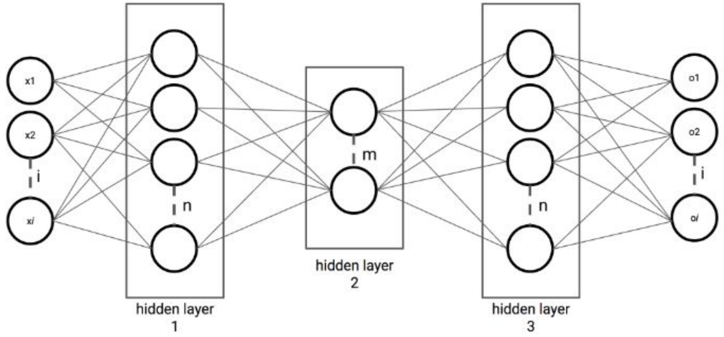
- The encoder will compress the input image to a smaller representation, and the decoder will try its best to recover the information.



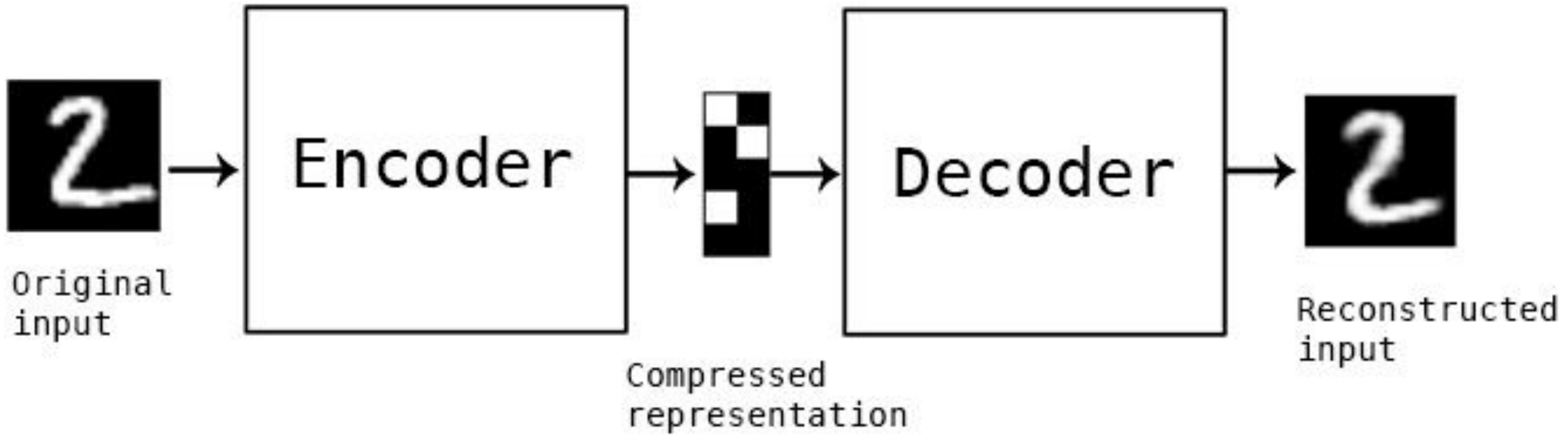
AutoEncoders using CNN Layers

- The encoder is now a CNN that compresses the data into a feature vector

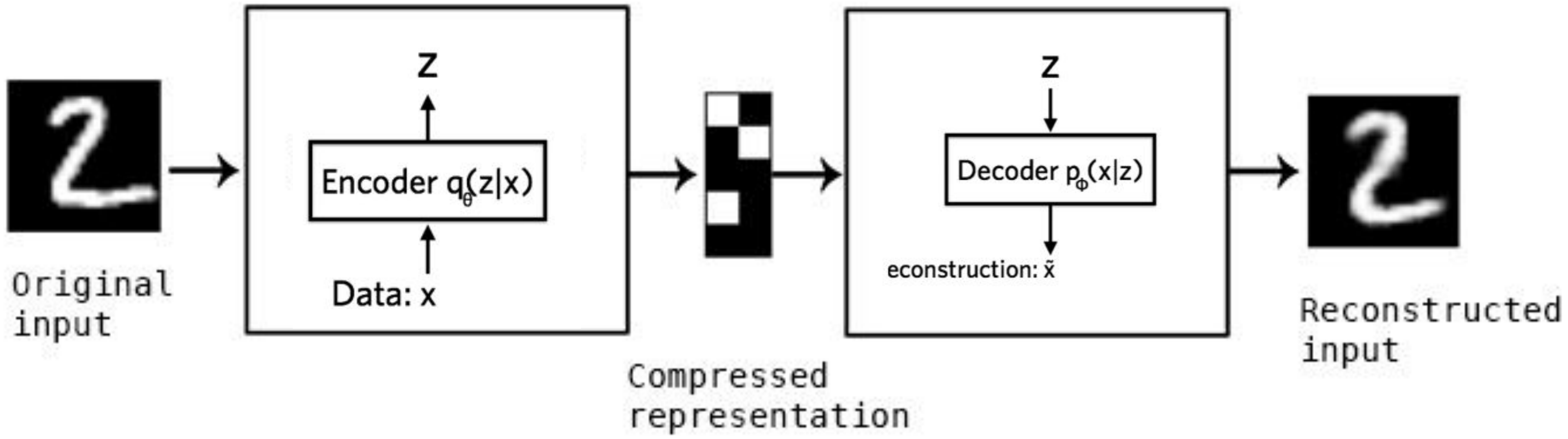
- The decoder will use transposed convolution layers to recreate the image from the encoding.



Working of AutoEncoders on Images

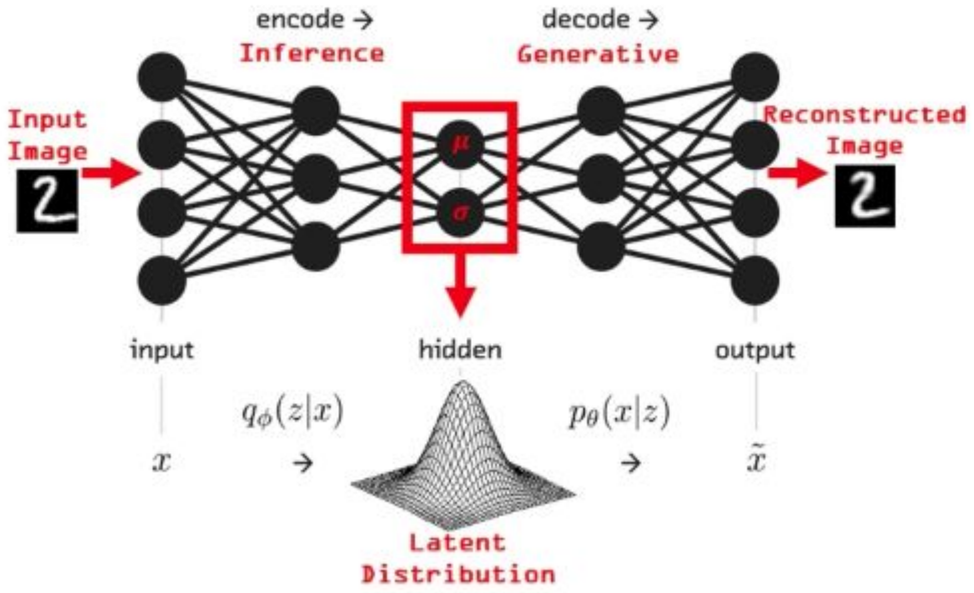


Working of AutoEncoders on Images

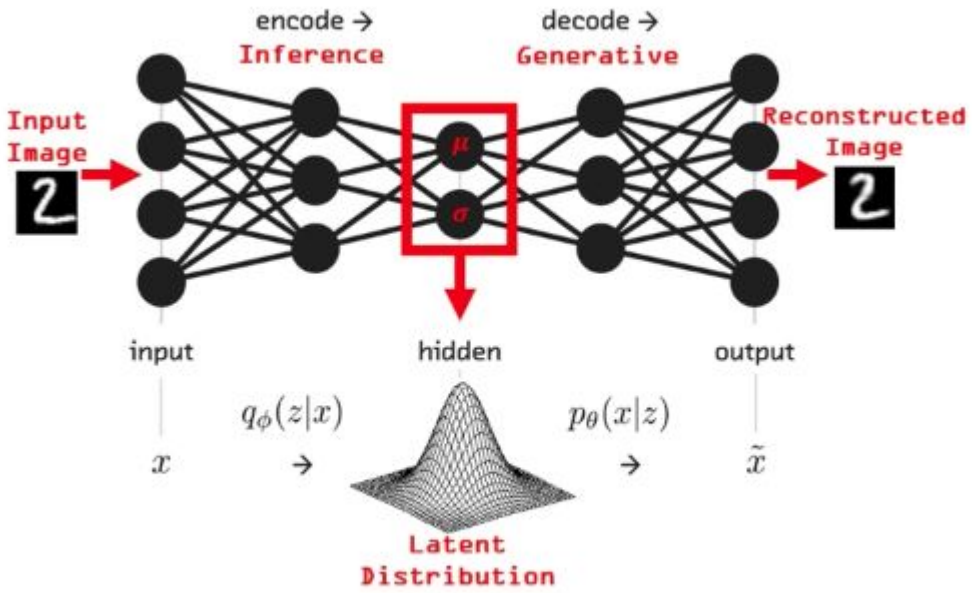


Working of AutoEncoders on Images

In the neural net world, the encoder is a neural network that outputs a representation *z* of data *x*.

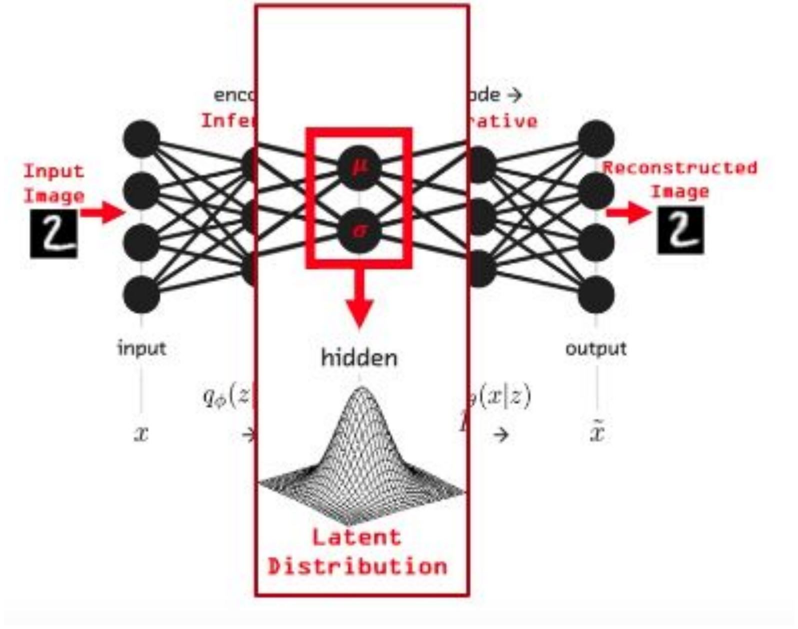


Working of AutoEncoders on Images

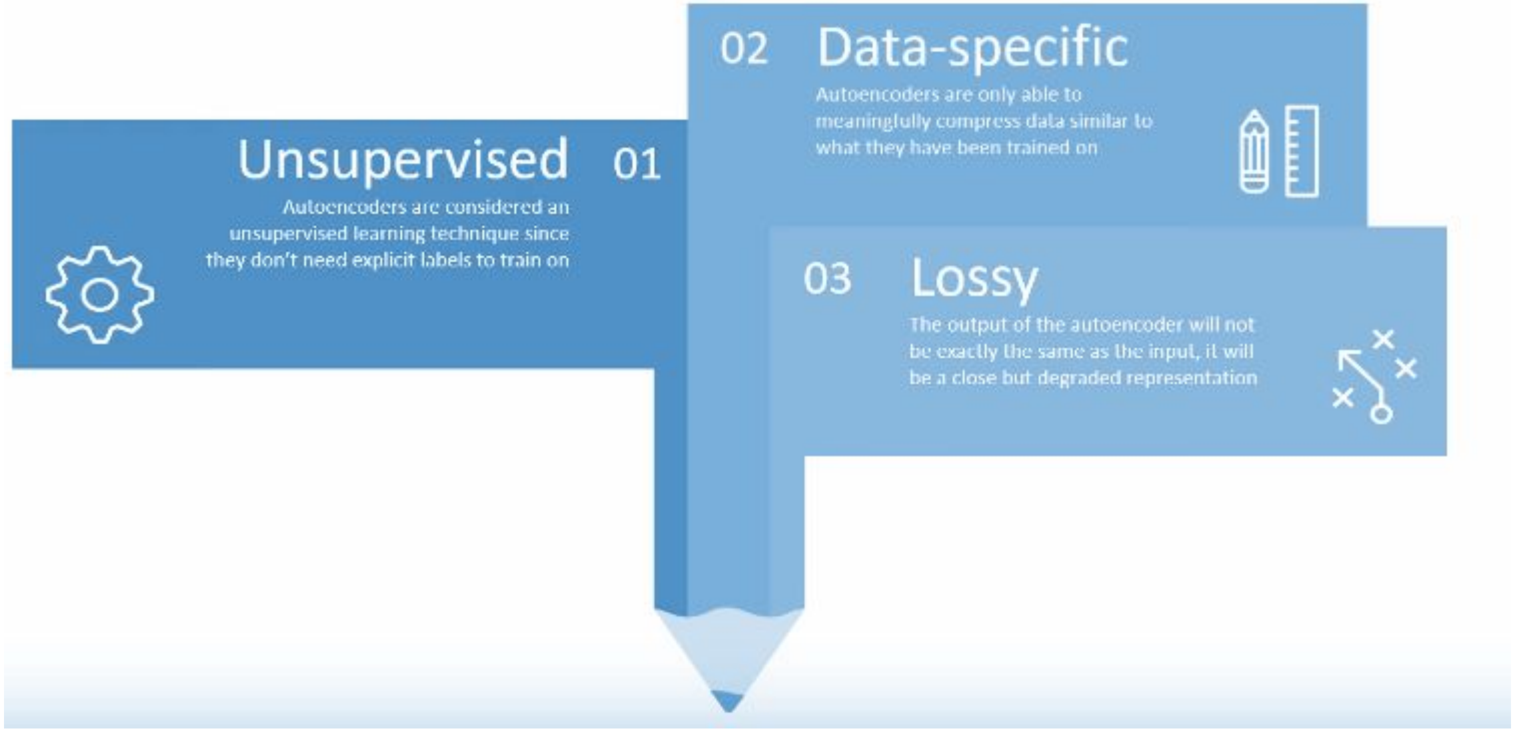


In deep learning, the decoder is a neural net that learns to reconstruct the data *x* given a representation *z*.

Working of AutoEncoders on Images



Properties of AutoEncoders



01 **Unsupervised**

Autoencoders are considered an unsupervised learning technique since they don't need explicit labels to train on

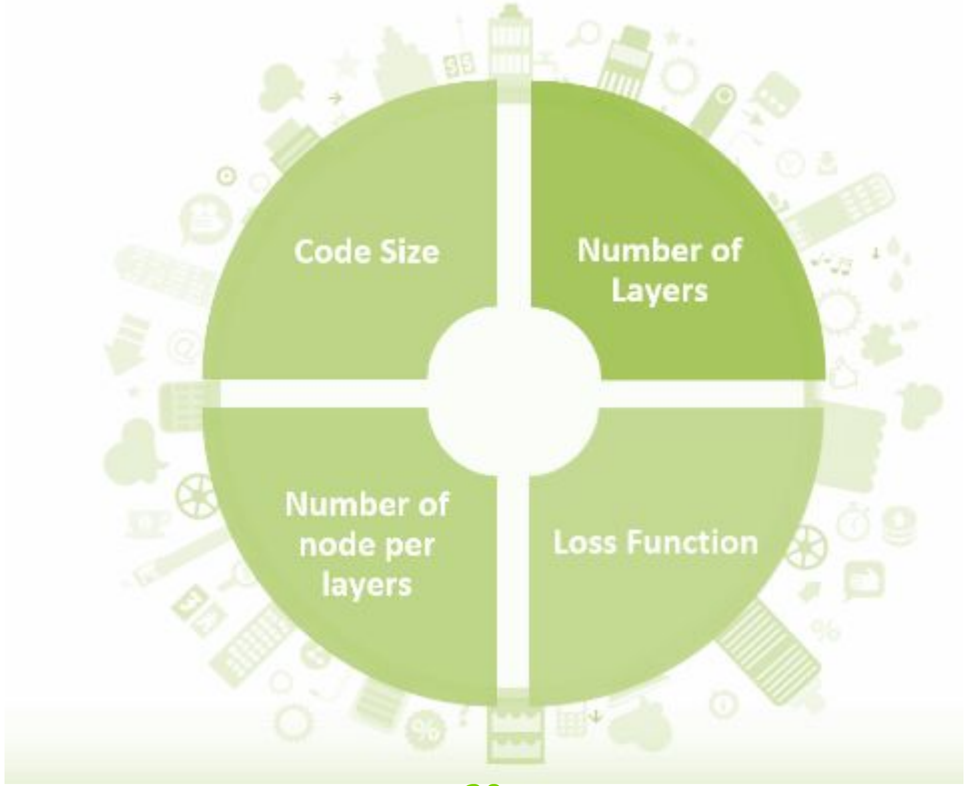
02 **Data-specific**

Autoencoders are only able to meaningfully compress data similar to what they have been trained on

03 **Lossy**

The output of the autoencoder will not be exactly the same as the input, it will be a close but degraded representation

Training AutoEncoders



Training AutoEncoders

Code Size

Smaller size results in more compression

Number of node per layers

Stacked autoencoders look like a sandwich

Number of Layers

The autoencoder can have many layers

Loss Function

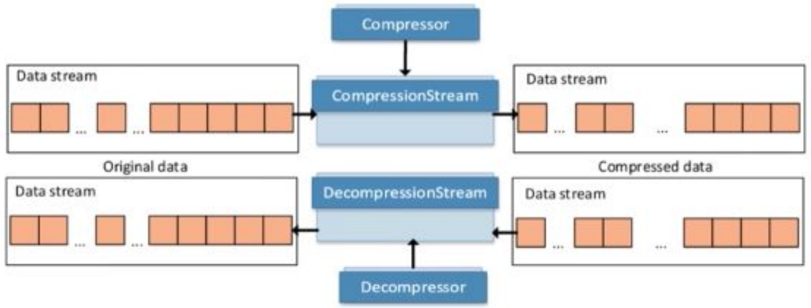
Mean squared error or binary cross entropy

Uses of AutoEncoders

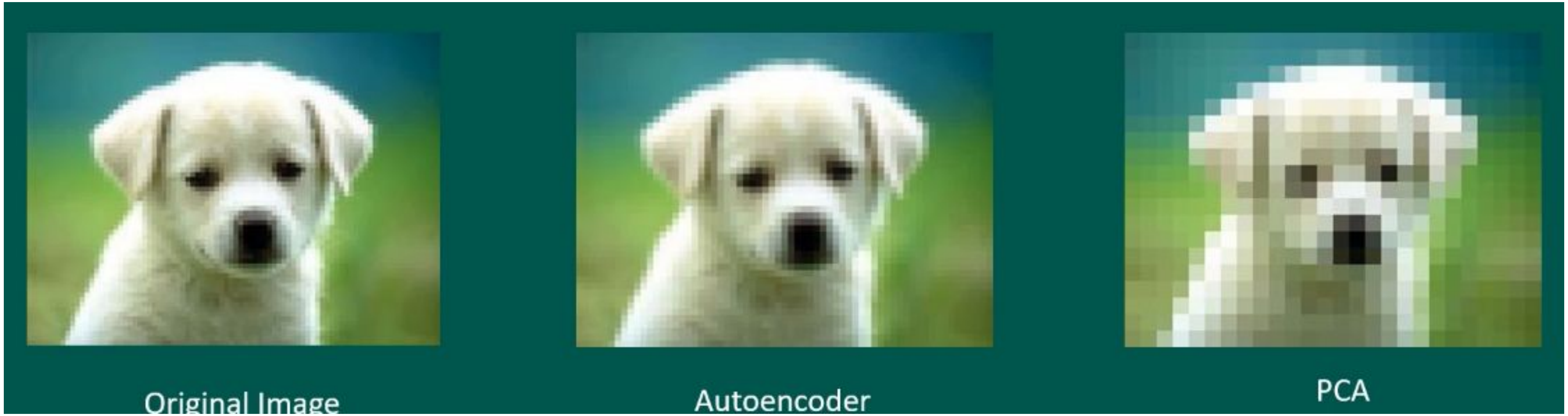
- **Compression Of Data**

- The autoencoder can be used to reduce the dimensionality down to two or three dimensions and then try to visualize your inputs in the latent space to see whether it shows you anything useful.

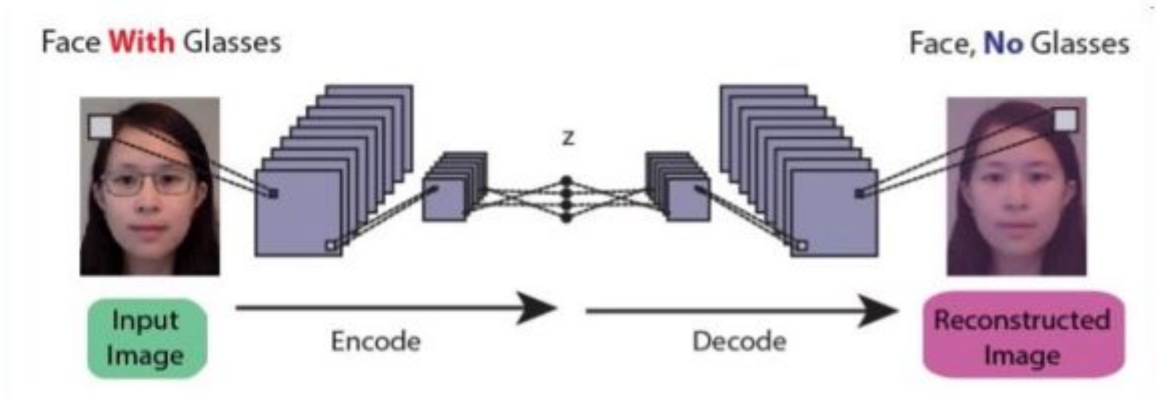
Uses of AutoEncoders : Data Compression



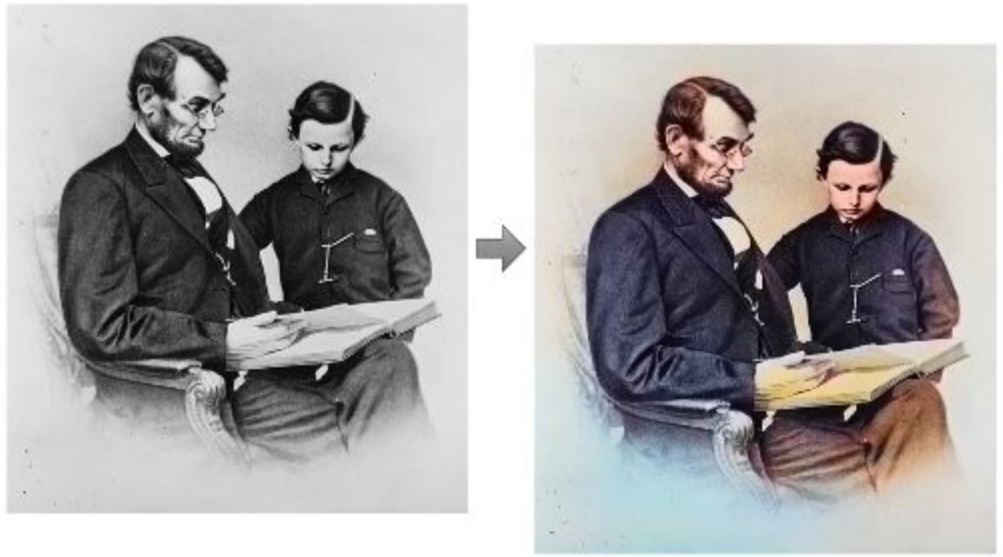
Uses of AutoEncoders : Better Compression



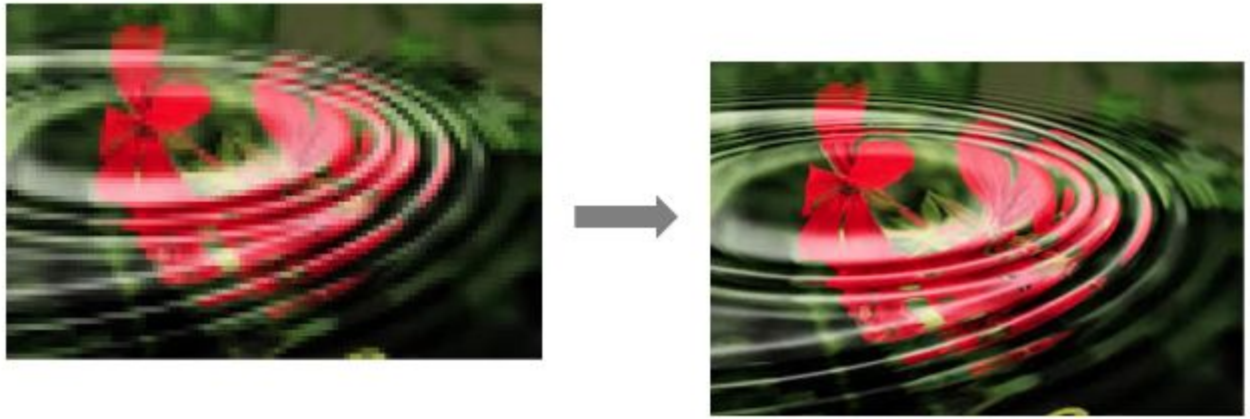
Uses of AutoEncoders : Image Reconstruction



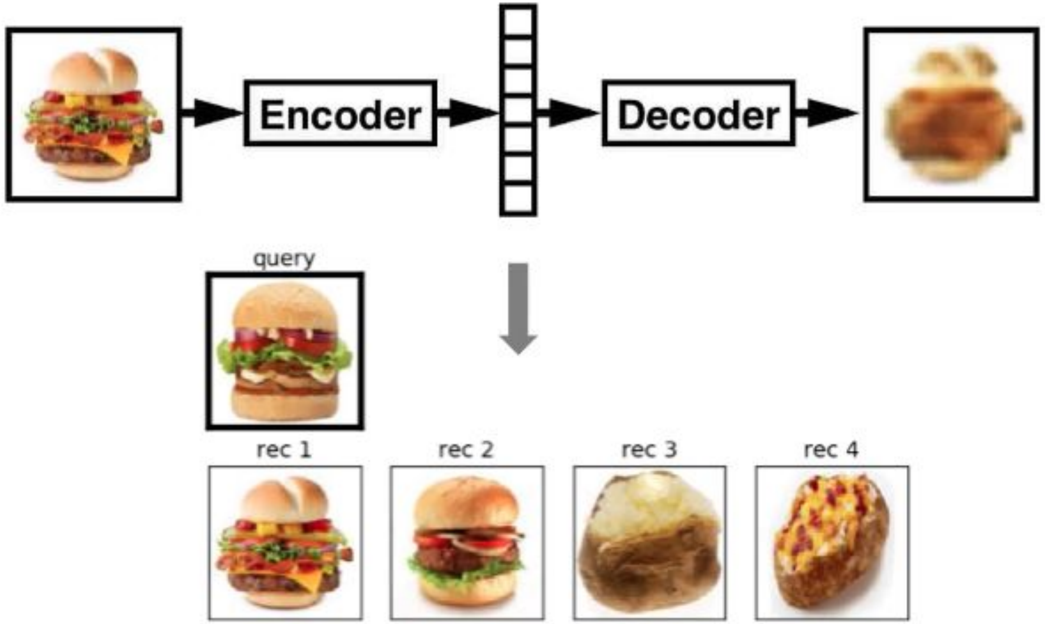
Uses of AutoEncoders : Image Coloring



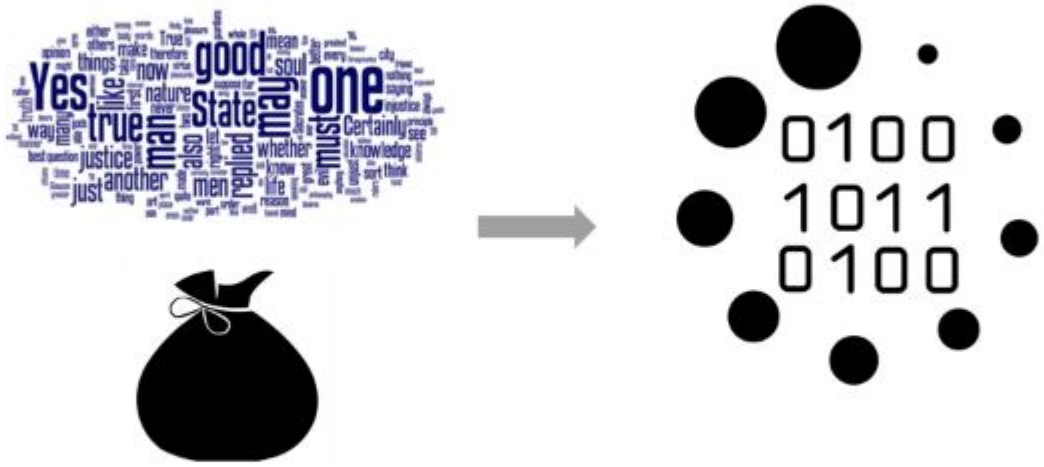
Uses of AutoEncoders : Generating High Resolution Images



Uses of AutoEncoders : Image Search



Uses of AutoEncoders : Topic Modeling



Generative Adversarial Networks

GANs

- **Generative adversarial networks (GAN)**

- GAN is composed of two networks together: a generator network and a discriminator network.

- During training, they both play a zero-sum game, where the discriminator network tries to discover whether the images input to it are real or fake.

- At the same time, the generator network tries to create fake images that are good enough to fool the discriminator.

GANs

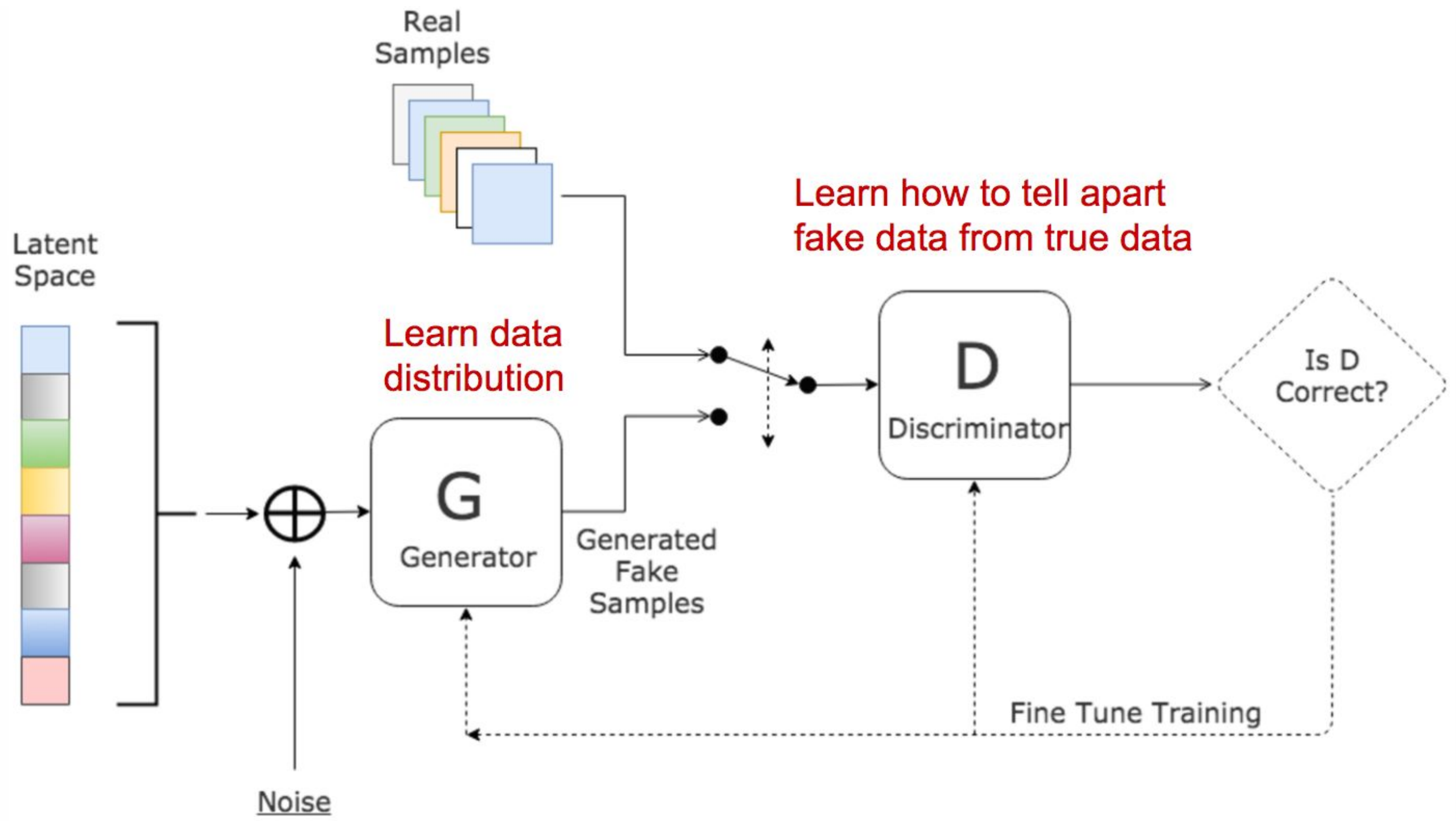
- **Generative adversarial networks (GAN)**

- The idea is that after some time of training, both the discriminator and the generator become very good at their tasks.

- As a result, the generator is forced to try and create images that look closer and closer to the original dataset.

- To be able to do this, it must capture the probability distribution of the dataset.

Generative adversarial networks (GAN)



The GAN Model

- **Generator:**
  - Creates images similar to the real images dataset
- **Discriminator:**
  - Verify that the image given to it is real or a fake generated one

Uses of GAN

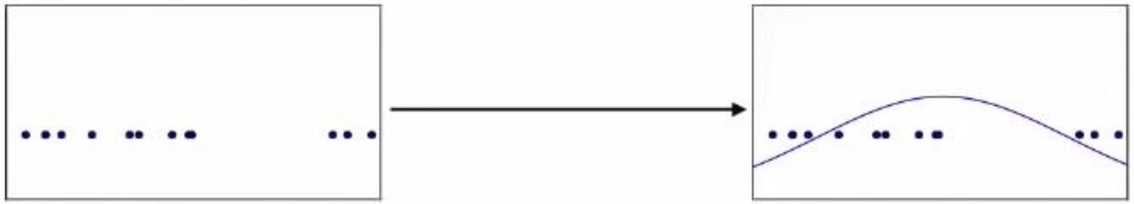
- Worst case input for one network is produced by another network

- Use the generator network to create new images, possibly to augment your dataset if you do not have sufficient data

- Use the discriminator as a loss function (potentially, better than L1/L2 for images)

Uses of GAN : Generative Modeling

- Density estimation



- Sample generation

Training examples Model samples

Tuning the Loss Function

- Both the discriminator and generator have their own loss functions that depend on the output of each others' networks.

- Think of the GAN as playing a minimax game between the discriminator and the generator

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))]$$

Here, D is our discriminator, G is our generator, z is a random vector input to the generator, and x is a real image.

Tuning the Loss Function

In order to train the GAN, we will alternate gradient step updates between the discriminator and the generator.

When updating the discriminator, we want to try and maximize the probability that the discriminator makes the right choice.

When updating the generator, we want to try and minimize the probability that the discriminator makes the right choice.

Tuning the Loss Function

- **Generator Loss :**
  - We want to maximize this loss function when training our GAN. When the loss is maximized, it means the generator is capable of generating images that can fool the discriminator.
- **Discriminator Loss :**
  - The discriminator wants to be able to distinguish between real and generated images. It wants to output 1 for real image and 0 for generated images.

Tuning the Loss Function

Since, it is impossible to maximize both discriminator loss and the generator loss at the same time, the model will hopefully reach some equilibrium, where the generator has to produce really good quality images in order to fool the discriminator.

Backpropagation

Backpropagation

Introduction

1. Backpropagation is short for backwards propagation of errors.
2. This is a supervised learning algorithm for artificial neural networks using gradient descent.
3. Backpropagation is used for training neural nets.

Backpropagation

Motive

1. *The connection between nodes of adjacent layers have weights associated with them.*
2. *The goal of learning is to assign correct weights to these edges.*
3. *Given the input vector, these weights determine the output vector.*

Backpropagation

Idea behind Backpropagation

1. *Output of Neural Network is evaluated against desired output.*
2. *If results are not satisfactory, connection or weights between layers are modified.*
3. *This Process is repeated until error becomes small enough.*

Backpropagation

Algorithm

1. *Initially all the weights are randomly assigned.*
2. *For every input in the training set, the ANN is activated and its output is observed.*
3. *This output is compared with the output we correctly know to gain deeper insight into the contributing weight of each input neuron.*

Backpropagation

Algorithm

4. *You compare the predictions of the neural network with the desired output and then compute the gradient of the errors with respect to the weights of the neural network. This gives you a direction in the parameter weight space in which the error would become smaller.*

5. Due to the layered structure of the neural network and the chain rule for derivatives, the formulas you get can be interpreted as propagating the error back through the network.

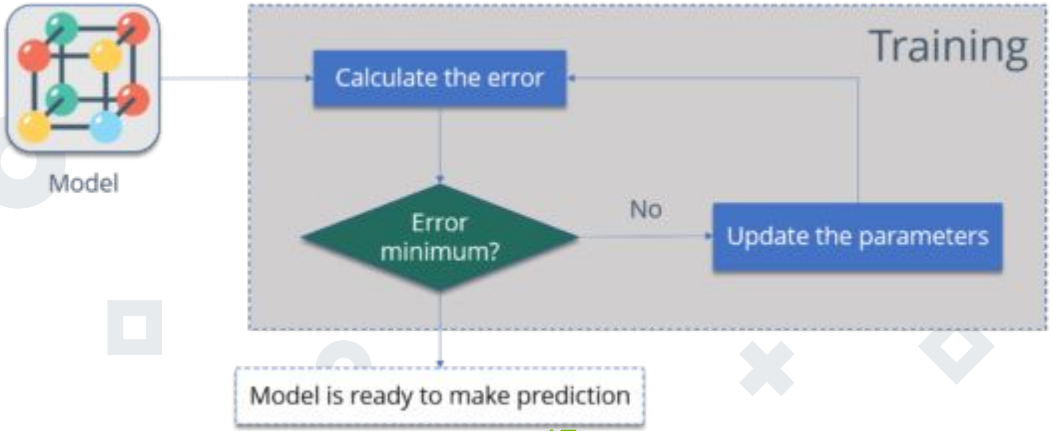
6. ***The error is propagated back to the previous layer and the weights are adjusted accordingly.***

7. ***The process is repeated until the output error is below a predetermined***

While designing a Neural Network, in the beginning, we initialize weights with some random values or any variable for that fact.

it’s not necessary that whatever weight values we have selected will be correct, or it fits our model the best.

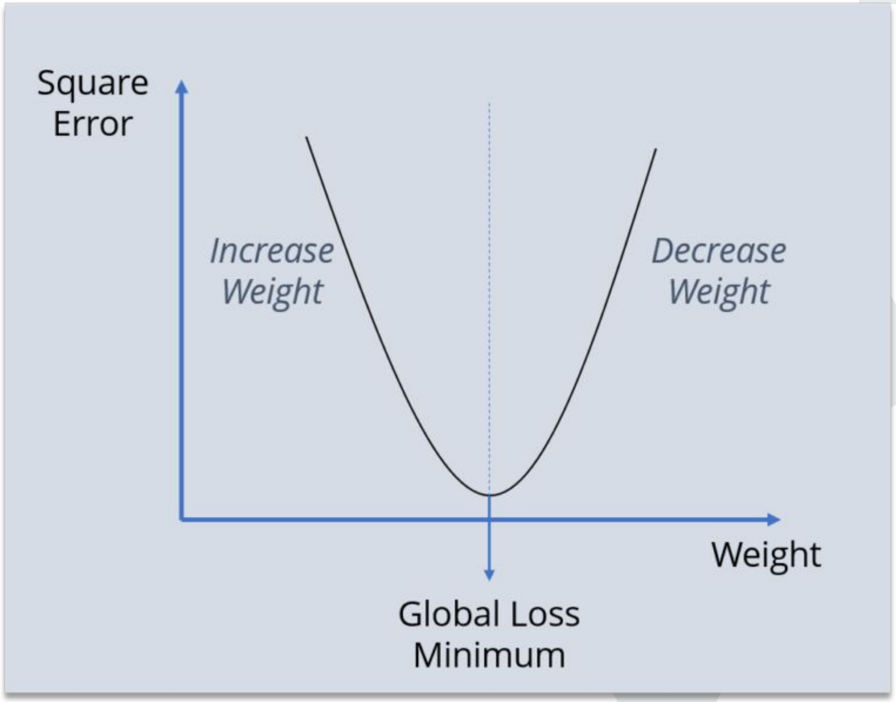
If we have selected some weight values in the beginning, but our model output is way different than our actual output i.e. the error value is huge, then we need to somehow explain the model to change the parameters (weights), such that error becomes minimum.

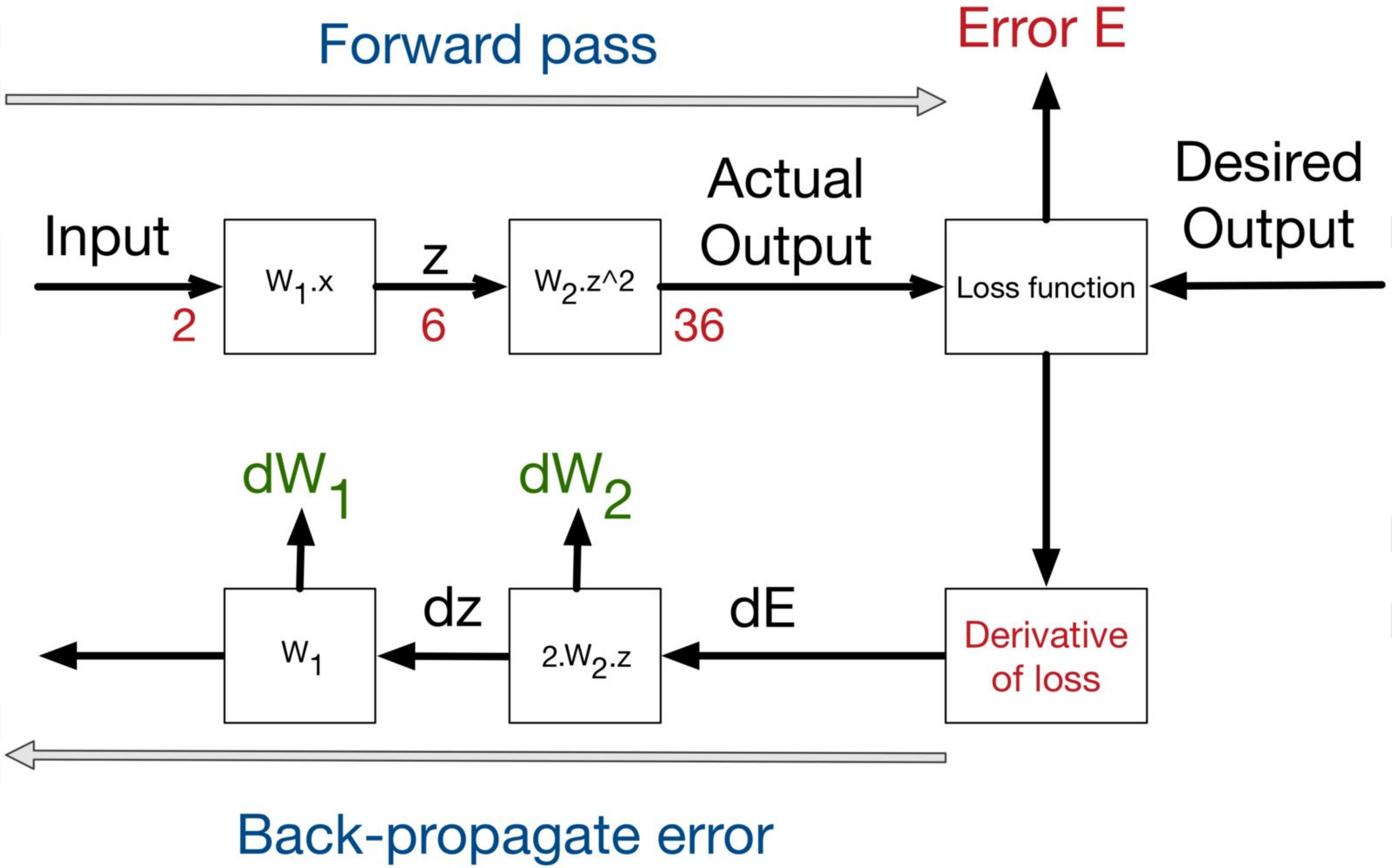


- **Calculate the error –** How far is your model output from the actual output.
- **Error minimum?** – Check whether the error is minimized or not.
- **Update the parameters** – If the error is huge then, update the parameters (weights and biases). After that again check the error. Repeat the process until the error becomes minimum.
- **Model is ready to make a prediction** – Once the error becomes minimum, you can feed some inputs to your model and it will produce the output.

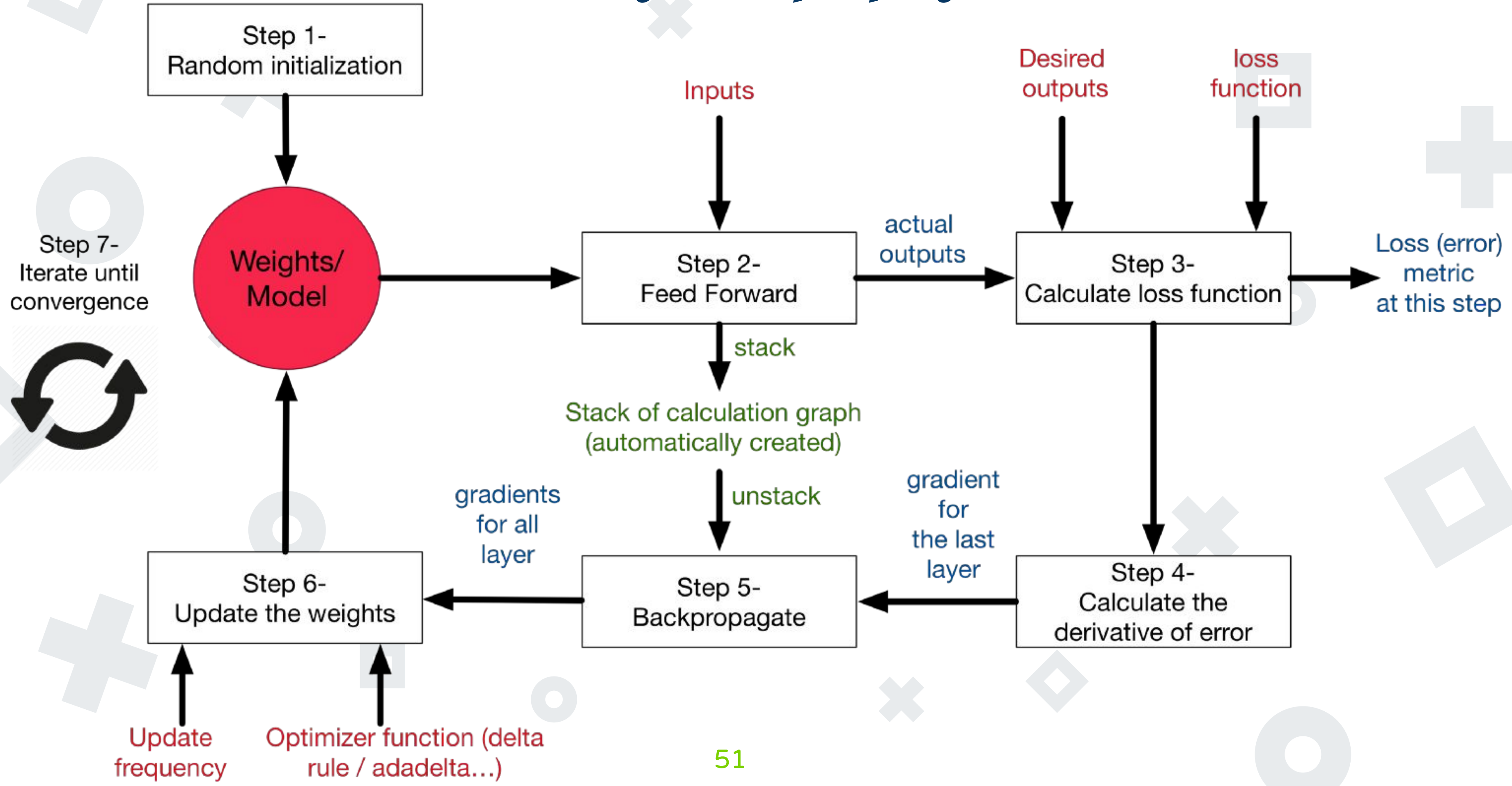
The Backpropagation algorithm looks for the minimum value of the error function in weight space using a technique called the delta rule or **gradient descent.**

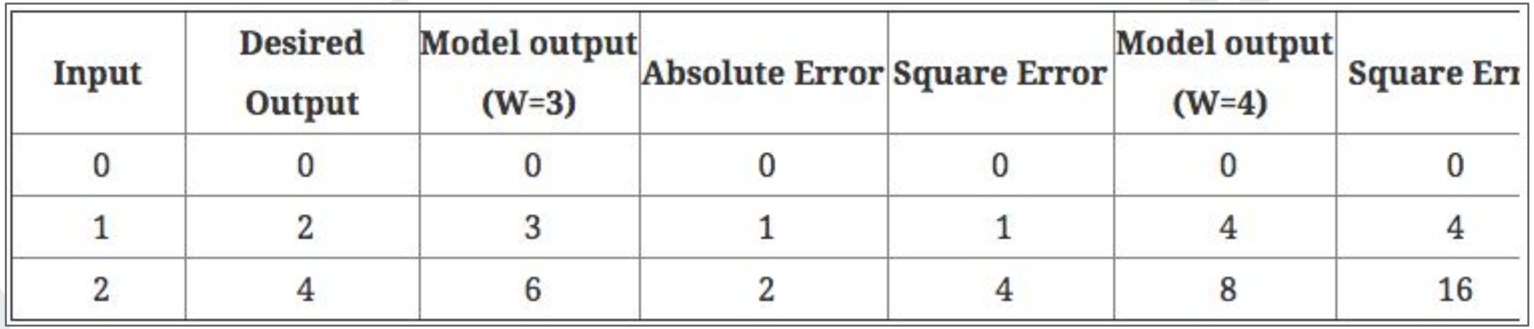
The weights that minimize the error function is then considered to be a solution to the learning problem.




Forward pass
Error E
Desired Output
Actual Output
Input
2
W1.x
z
6
W2.z^2
36
Loss function
dW1
dW2
dz
dE
W1
2.W2.z
Derivative of loss
Back-propagate error

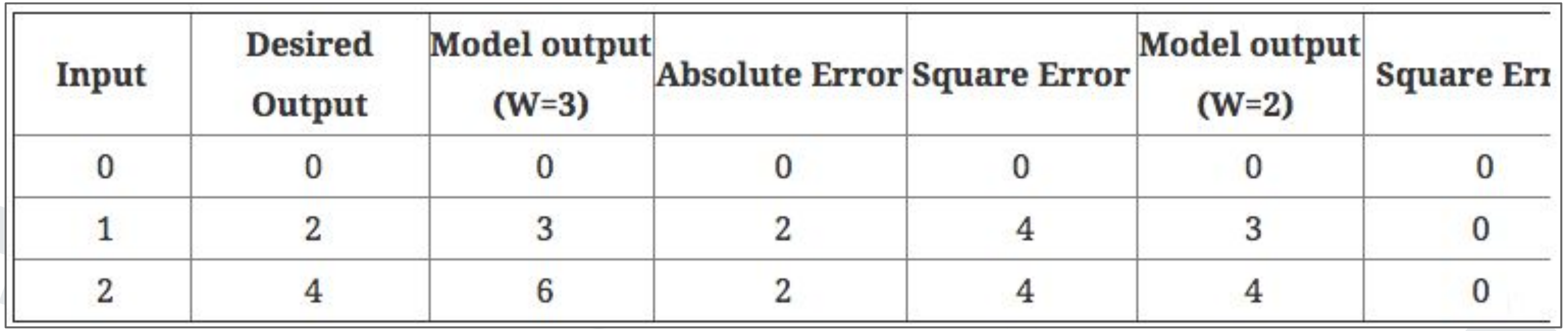
Flowchart of Backpropagation





Input	Desired Output	Model output (W=3)	Absolute Error	Square Error	Model output (W=4)	Square Err
0	0	0	0	0	0	0
1	2	3	1	1	4	4
2	4	6	2	4	8	16

When we increase the value of ‘W’ the error has increased. So, obviously there is no point in increasing the value of ‘W’ further. But, what happens if we decrease the value of ‘W’

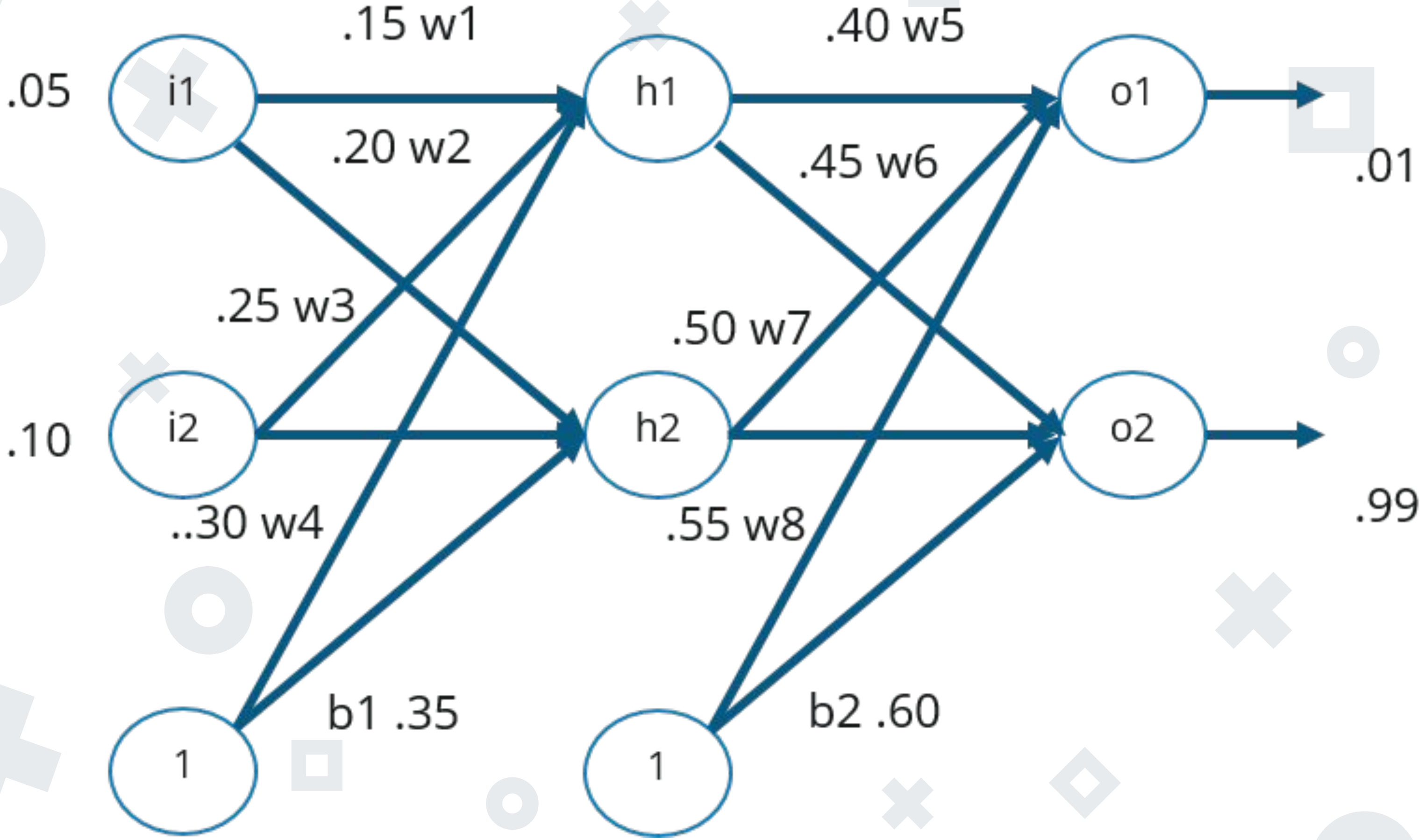


Input	Desired Output	Model output (W=3)	Absolute Error	Square Error	Model output (W=2)	Square Err
0	0	0	0	0	0	0
1	2	3	2	4	3	0
2	4	6	2	4	4	0

- We first initialized some random value to ‘W’ and propagated forward.
- Then, we noticed that there is some error. To reduce that error, we propagated backwards and increased the value of ‘W’.
- After that, also we noticed that the error has increased. We came to know that, we can’t increase the ‘W’ value.
- So, we again propagated backwards and we decreased ‘W’ value.
- Now, we noticed that the error has reduced.

- Basically, we need to figure out whether we need to increase or decrease the weight value.
- Once we know that, we keep on updating the weight value in that direction until error becomes minimum.
- One might reach a point, where if you further update the weight, the error will increase.
- At that time you need to stop, and that is your final weight value.

Mathematical Explanation


.15 w1
.40 w5
.05
i1
h1
o1
.20 w2
.45 w6
.01
.25 w3
.50 w7
.10
i2
h2
o2
..30 w4
.55 w8
.99
b1 .35
b2 .60
1
1

The above network contains the following:

Two inputs

Two hidden neurons

Two output neurons

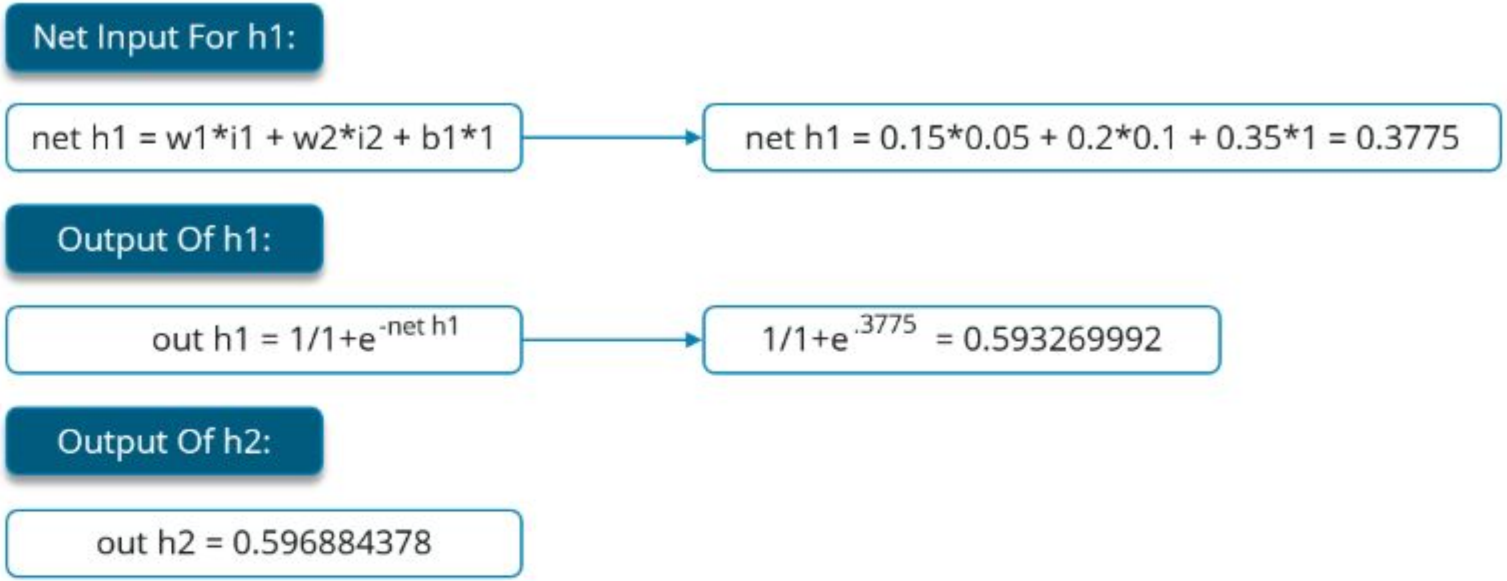
Two biases

Below are the steps involved in Backpropagation:

- Step – 1: Forward Propagation
- Step – 2: Backward Propagation
- Step – 3: Putting all the values together and calculating the updated weight value

Step – 1: Forward Propagation

We will start by propagating forward.



Net Input For h1:

net h1 = w1*i1 + w2*i2 + b1*1 $\rightarrow$ net h1 = 0.15*0.05 + 0.2*0.1 + 0.35*1 = 0.3775

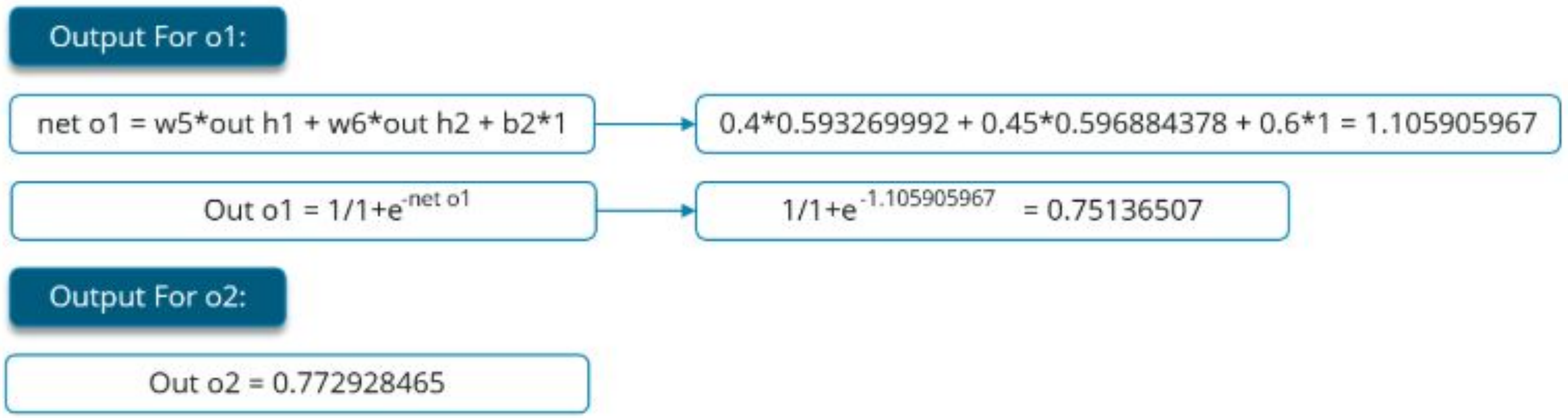
Output Of h1:

out h1 = $1/1+e^{-\text{net h1}}$ $\rightarrow$ $1/1+e^{.3775}$ = 0.593269992

Output Of h2:

out h2 = 0.596884378

We will repeat this process for the output layer neurons, using the output from the hidden layer neurons as inputs.



Output For o1:

net o1 = w5*out h1 + w6*out h2 + b2*1 $\rightarrow$ 0.4*0.593269992 + 0.45*0.596884378 + 0.6*1 = 1.105905967

Out o1 = $1/1+e^{-\text{net o1}}$ $\rightarrow$ $1/1+e^{-1.105905967}$ = 0.75136507

Output For o2:

Out o2 = 0.772928465

Let's see what is the value of the error:

Error For o1:

$E\ o1 = \Sigma 1/2(target\ - output)^2 \longrightarrow$ ½ $(0.01 - 0.75136507)^2 = 0.274811083$

Error For o2:

$E\ o2 = 0.023560026$

Total Error:

$E_{total} = E\ o1 + E\ o2 \longrightarrow 0.274811083 + 0.023560026 = 0.298371109$

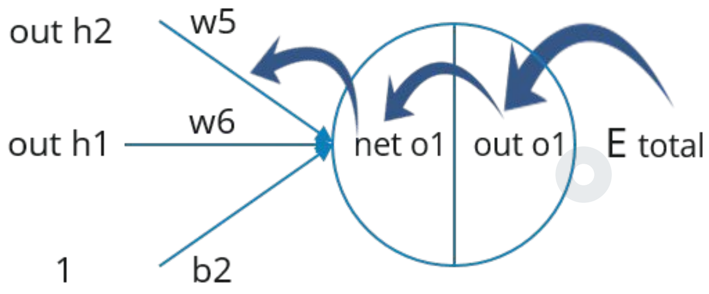
Step – 2: Backward Propagation

“

Now, we will propagate backwards. This way we will try to reduce the error by changing the values of weights and biases.

Consider W5, we will calculate the rate of change of error w.r.t change in weight W5.

$$\frac{\delta Etotal}{\delta w5} = \frac{\delta Etotal}{\delta out\ o1} * \frac{\delta out\ o1}{\delta net\ o1} * \frac{\delta net\ o1}{\delta w5}$$



Since we are propagating backwards, first thing we need to do is, calculate the change in total errors w.r.t the output O1 and O2.

“

$$E_{total} = 1/2(\text{target } o1 - \text{out } o1)^2 + 1/2(\text{target } o2 - \text{out } o2)^2$$

$$\frac{\delta Etotal}{\delta out\ o1} = -(\text{target } o1 - \text{out } o1) = -(0.01 - 0.75136507) = 0.74136507$$

Now, we will propagate further backwards and calculate the change in output O1 w.r.t to its total net input.

“

$$\text{out o1} = 1/1+\text{e}^{-neto1}$$

$$\frac{\delta out\ o1}{\delta net\ o1} = \text{out o1 (1 - out o1)} = 0.75136507\ (1 - 0.75136507) = 0.186815602$$

Let’s see now how much does the total net input of O1 changes w.r.t W5?

“

$$\text{net o1} = \text{w5} * \text{out h1} + \text{w6} * \text{out h2} + \text{b2} * 1$$

$$\frac{\delta net\ o1}{\delta w5} = 1 * \text{out h1}\ w5^{(1-1)} + 0 + 0 = 0.593269992$$

“

Step – 3: Putting all the values together and calculating the updated weight value

Now, let’s put all the values together

$$\boxed{\frac{\delta Etotal}{\delta w5} = \frac{\delta Etotal}{\delta out\ o1} * \frac{\delta out\ o1}{\delta net\ o1} * \frac{\delta net\ o1}{\delta w5}} \rightarrow 0.082167041$$

Let's calculate the updated value of W5:

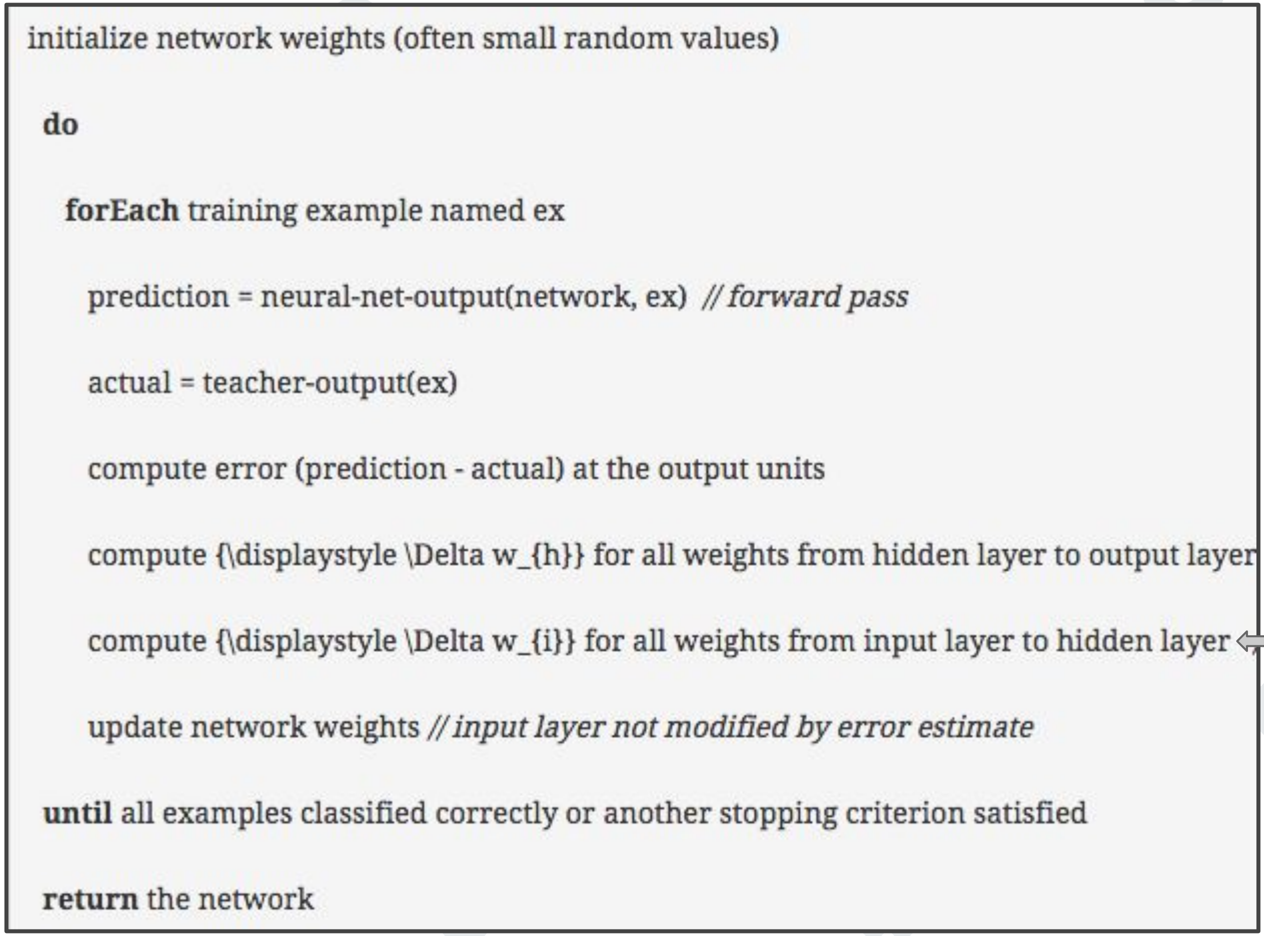
$$\boxed{w5^{+} = w5 - n\frac{\delta Etotal}{\delta w5}} \rightarrow \boxed{w5^{+} = 0.4 - 0.5 * 0.082167041}$$

$$\text{Updated w5} \rightarrow 0.35891648$$

- Similarly, we can calculate the other weight values as well.
- After that we will again propagate forward and calculate the output. Again, we will calculate the error.
- If the error is minimum we will stop right there, else we will again propagate backwards and update the weight values.
- This process will keep on repeating until error becomes minimum.

“

Summary of Backpropagation Algorithm : Pseudocode



```
initialize network weights (often small random values)
 do
   forEach training example named ex
     prediction = neural-net-output(network, ex)  // forward pass
     actual = teacher-output(ex)
     compute error (prediction - actual) at the output units
     compute {\displaystyle \Delta w_{h}} for all weights from hidden layer to output layer
     compute {\displaystyle \Delta w_{i}} for all weights from input layer to hidden layer
     update network weights // input layer not modified by error estimate
 until all examples classified correctly or another stopping criterion satisfied
 return the network
```

Backward Pass

Recurrent Neural Networks

RNN

- RNN is a variant of ANN where the connections between neurons can form a cycle.
- Unlike feed-forward networks, RNNs can use internal memory for their processing.

RNN

Recurrent networks have two input sources --

Present

and

The ***Recent Past***

RNN

- So, Recurrent networks take as their input, current input data and also what they have experienced over time.

- These two input sources are combined to determine how they respond to new data.

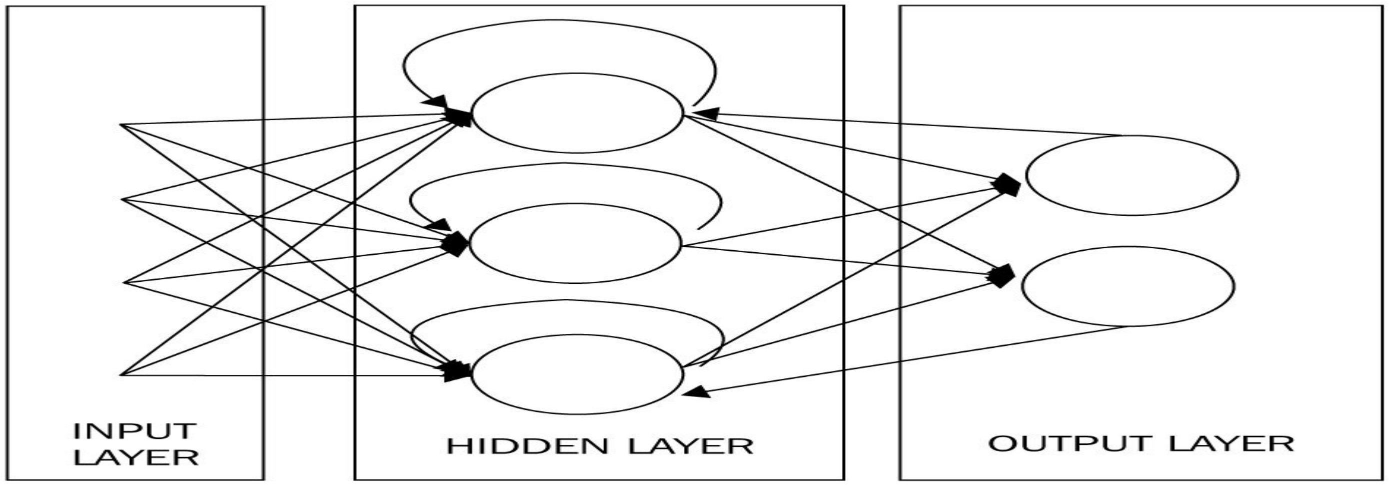
RNN

- Recurrent Neural Nets are distinguished by the feedback loop linked to their past decisions.

- Adding memory to neural networks has a purpose: there is information in the sequence itself and recurrent networks use it to perform the tasks that feed-forward networks cannot.

RNN

- In RNN connections between neurons form a directed cycle.

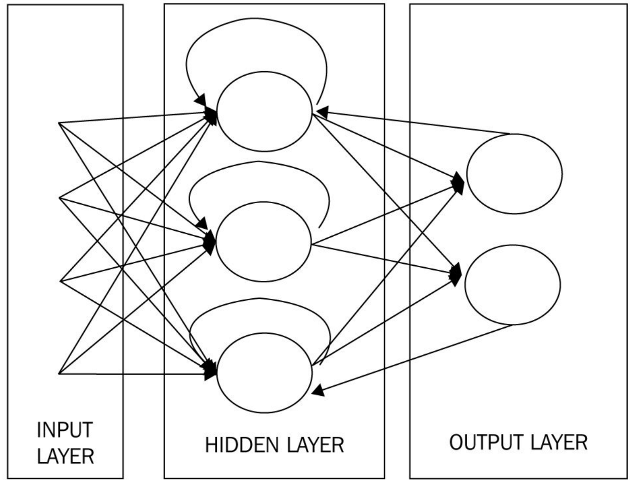


RECURRENT NEURAL NETWORKS

RNN

The output of one instance is taken as input for the next instance for the same neuron.

The way the data is kept in memory and flows at different time periods makes RNNs powerful and successful.

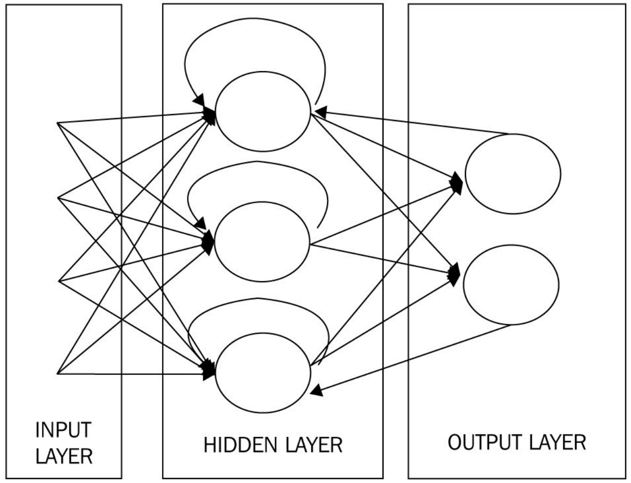


RECURRENT NEURAL NETWORKS

RNN

The output of one instance is taken as input for the next instance for the same neuron.

The way the data is kept in memory and flows at different time periods makes RNNs powerful and successful.

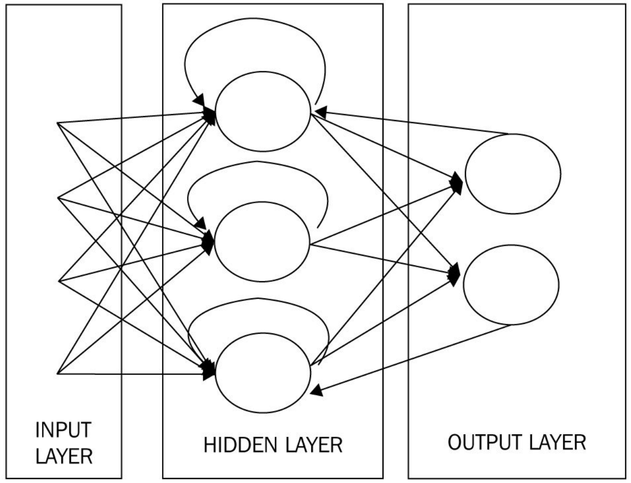


RECURRENT NEURAL NETWORKS

RNN

Recurrent networks are designed to recognize patterns as a sequence of data and are helpful in prediction and forecasting.

They can work on text, images, speech, and time series data.

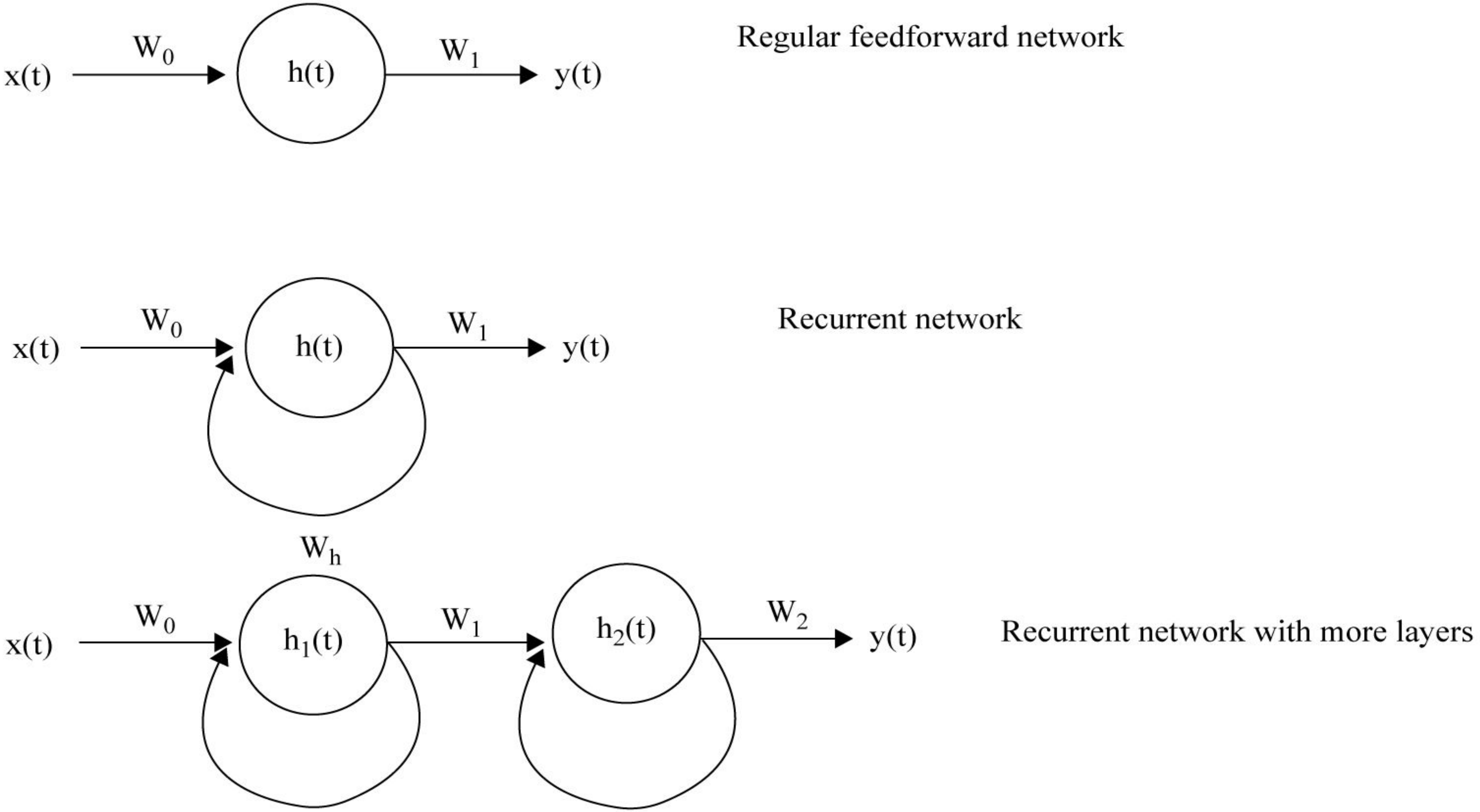


RECURRENT NEURAL NETWORKS

RNN

- Recurrent networks take inputs from the current input (like a feed-forward network) and the output that was calculated previously.

RNNs are among the powerful ANNs that represent the biological brain, including memory with processing power.


x(t)
W0
h(t)
W1
y(t)
Regular feedforward network
x(t)
W0
h(t)
W1
y(t)
Recurrent network
Wh
x(t)
W0
h1(t)
W1
h2(t)
W2
y(t)
Recurrent network with more layers

RNN

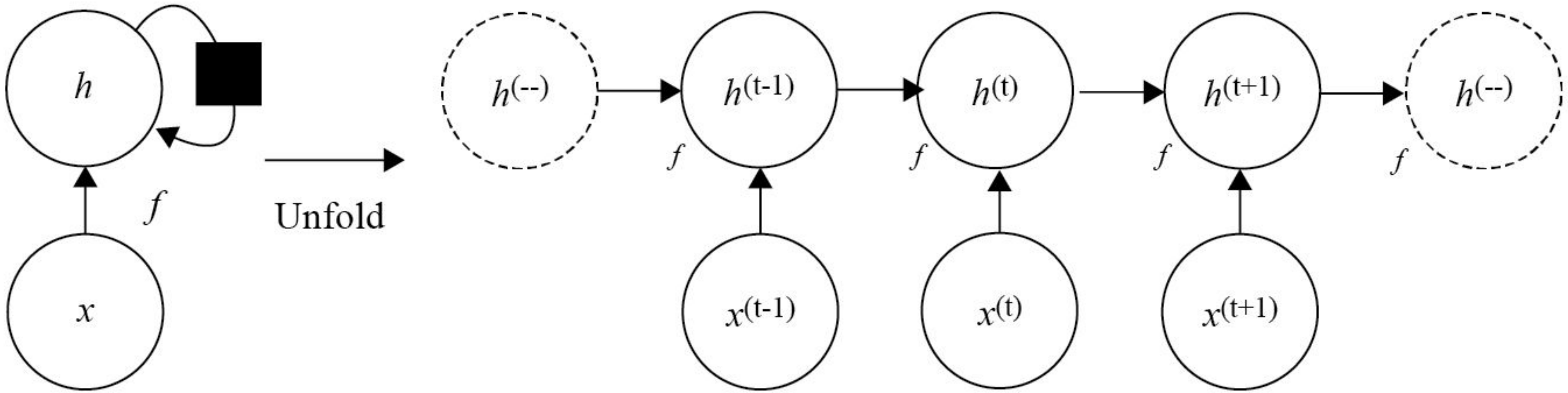
Consider the RNN as a network of neural networks, and the cyclic nature is unfolded in the following manner.

The state of a neuron h is considered at different time periods

(-t-1, t, t+1 and so on)

until convergence or the total number of epochs is reached.

RNN



RNN

- *RNNs are mainly used for sequence modeling.*

- *The inputs and outputs are treated as vectors.*

- *RNNs are basically ANNs with loops that allow information to persist in the network.*

- *The looping allows information to be passed from state* ***t*** *to state* ***t+1.***

RNN

- *When we persist the information, as the patterns change, RNN is able to predict the t+1 value.*

- *This is particularly useful for analyzing time-series-based problems.*

- *RNN can learn the pattern and do the prediction.*

- *The internal state of the RNN is updated for every time step of the learning process*

Uses of RNN

- *Image Captioning*
- *Document Classification*
- *Video Cassifcation Frame by Frame*
- *Forecasting Time Series*

Backpropagation Through Time

Pseudocode

- ✓ *Unfold the RNN to contain n feed-forward networks.*
- ✓ *Initialize the weights w to random values.*
- ✓ *Perform the following until the stopping criteria is met or you are done with the required number of epochs.*
- ✓ *Set inputs to each network with values as x_i.*

Backpropogation Through Time

Pseudocode …

- ✓ *Forward-propagate the inputs over the whole unfolded network.*
- ✓ *Back-propagate the error over the unfolded network.*
- ✓ *Update all the weights in the network.*
- ✓ *Average out the weights to find the final weight in the folded network.*

Problems with RNN

i. *Loss of information through Time*

ii. *Vanishing Gradients*

Solutions

i. *Gating Units - LSTM, RNN*
ii. *Gradient Clipping*
iii. *Better Optimizers*
iv. *Steeper Gates*

Long Short Term Memory Units

We have seen that RNNs have a memory that uses persistent previous information to be used in the current neural network processing.

The previous information is used in the present task.

However, the memory is short-term and we do not have a list of all of the previous information available for the neural node.

Long Short Term Memory Units

When we introduce a long-term memory into the RNN, we are able to remember a lot of previous information and use it for the current processing.

This concept is called LSTM model of RNN

LSTM has numerous use cases in video, audio, text prediction, and various other applications.

Long Short Term Memory Units

The LSTM network is trained using BPTT and diminishes the vanishing gradient problem.

LSTMs have powerful applications in time series predictions and can create large, recurrent networks to address difficult sequence problems in machine learning.

LSTM have gates that make the long/short term memory possible.

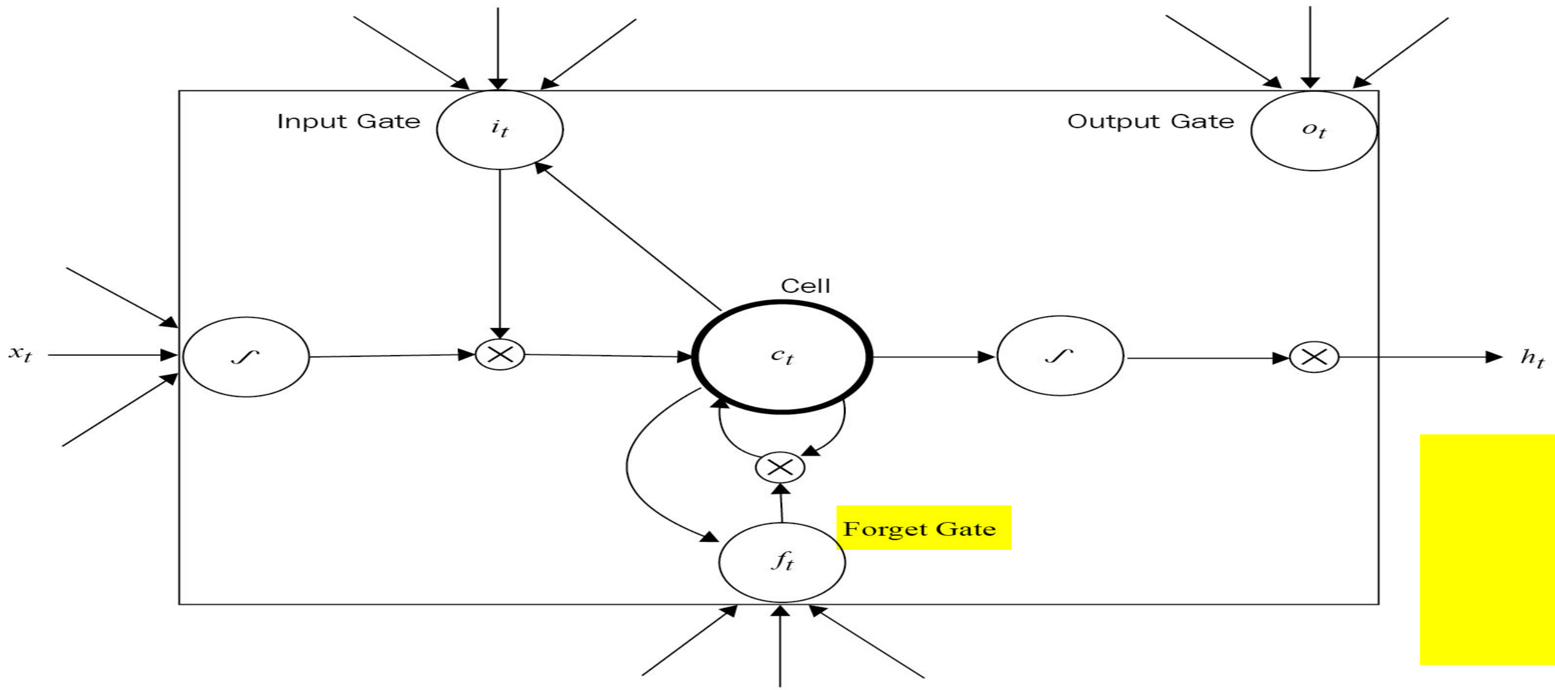
Long Short Term Memory Units

These gates are contained in memory blocks connected through layers:

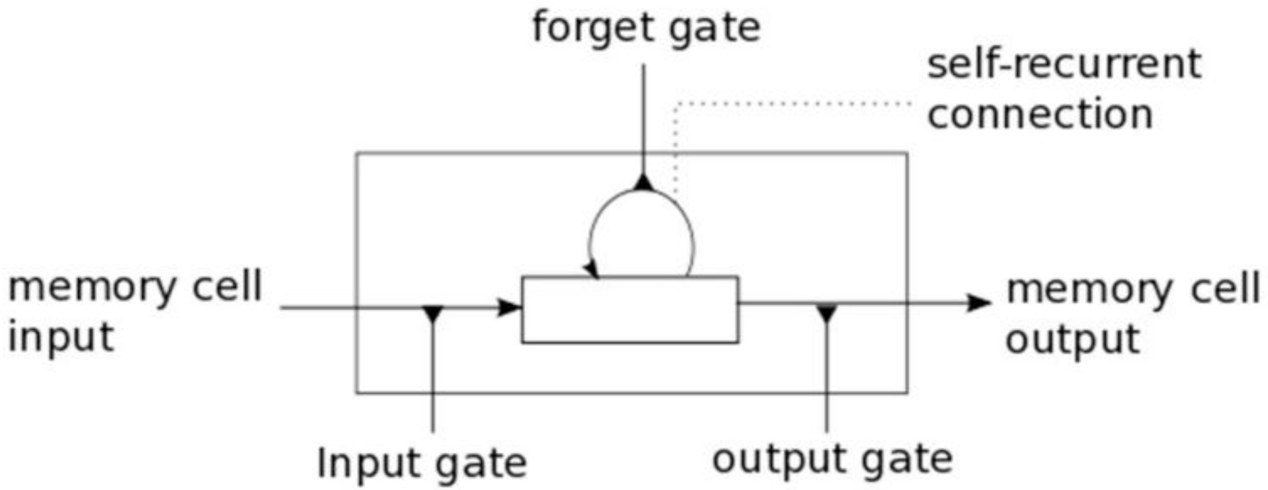
There are three types of gates within a unit:

- Input Gate: Scales input to cell (write)
- Output Gate: Scales output to cell (read)
- Forget Gate: Scales old cell value (reset)

Long Short Term Memory Units



Long Short Term Memory Units



- Each gate is like a switch that controls the read/write, thus incorporating the long-term memory function into the LSTM model.

Long Short Term Memory Units

LSTMs can be used to solve the following sequence prediction problems:

- Direct sequence prediction
- Sequence classification
- Sequence generation
- Sequence to sequence prediction

GRU and LSTM

- The key differences between GRU and LSTM are:

- A GRU has two gates, whereas an LSTM has three gates.
- GRUs don't possess any internal memory that is different from the exposed hidden state.
- They don't have the output gate, which is present in LSTMs.
- There is no second nonlinearity applied when computing the output in GRU.

Deep Reinforcement Learning

- Learning via interaction and feedback.
- For building agents that can perceive and interpret the world around it and take action and interact with it.

Deep-Q-Network

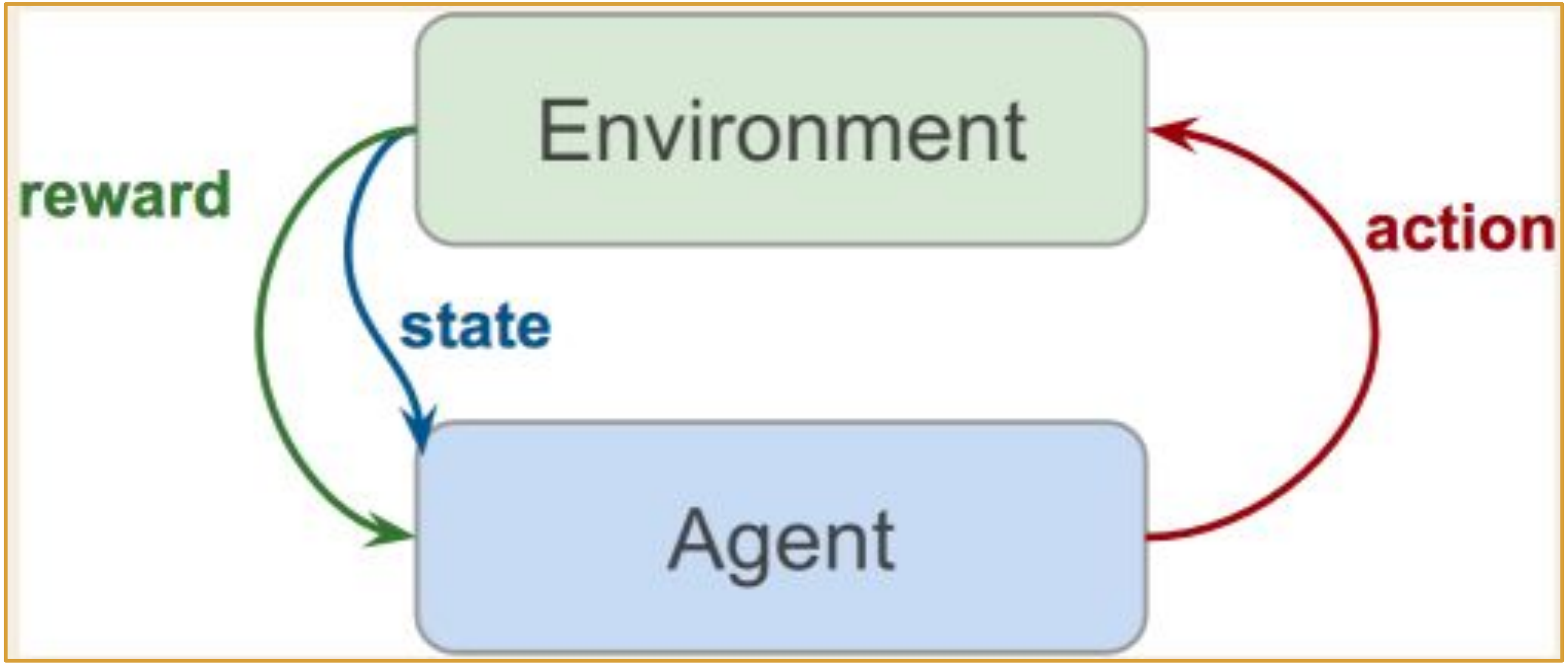
- DQN was the first large scale successful application of reinforcement learning with deep neural nets.

- DQN was capable of learning 49 different ATARI games with different goals, rules and gameplays.

Deep-Q-Network

A deep reinforcement learning agent playing breakout.

Reinforcement Learning Setup



What is Reinforcement Learning

- Learning by interacting with the environment.
- Involves an actor, an environment and a reward signal.
- The actor chooses to take an action in the environment for which the actor is rewarded accordingly.

What is Reinforcement Learning

- The way an actor chooses its actions

  Is called *policy*.
- The actor wants to increase its rewards.
- The goal of the actor is to learn an optimal policy for

  interacting with the environment.

Comparing Learning Techniques

Supervised

Input: DATA + LABELS

Output: Predict Labels

Unsupervised

Input: DATA

Output: Discover underlying structure

Reinforcement

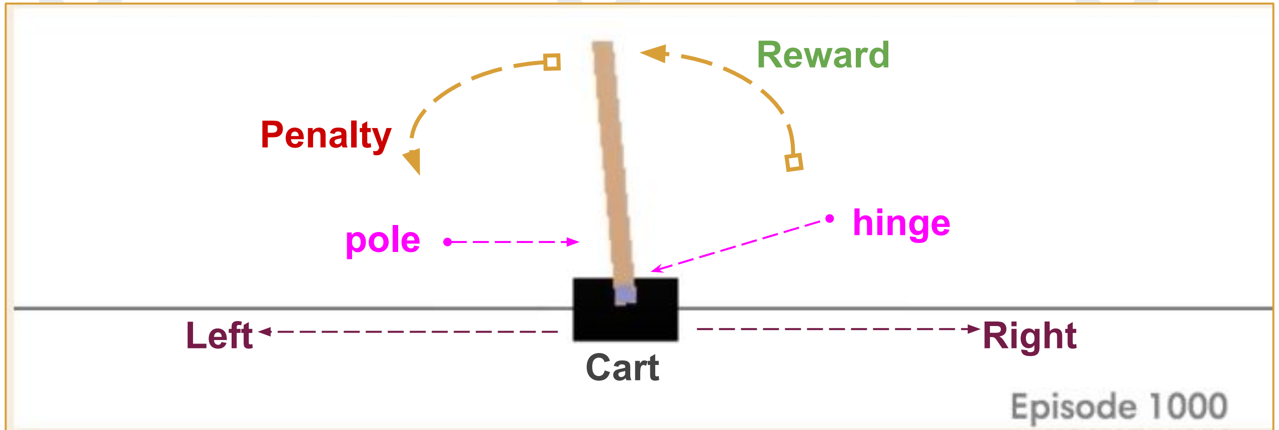
Input: Reward Signals by Environment

Output: Plan of Action to maximize rewards

The Pole Balancing Problem

- A cart with a pole connected by a hinge so the pole can swing around the cart.
- An agent can control the cart moving it left or right.
- The environment rewards the agent when the pole is pointed upward
- The environment penalizes the agent when the pole falls over.

$(x_{cart}, \theta_{pole})$.



State - The cart has a range of possible places on the x-plane where it can be. The pole has a range of possible angles.

Action - The agent can move the cart either left or right.

State Transition - The environment changes—the cart changes velocity and the pole changes angle.

Reward - When the agent acts, the environment changes the cart moves and the pole changes angle and velocity.

An MDP is defined as the following --

S finite set of possible states

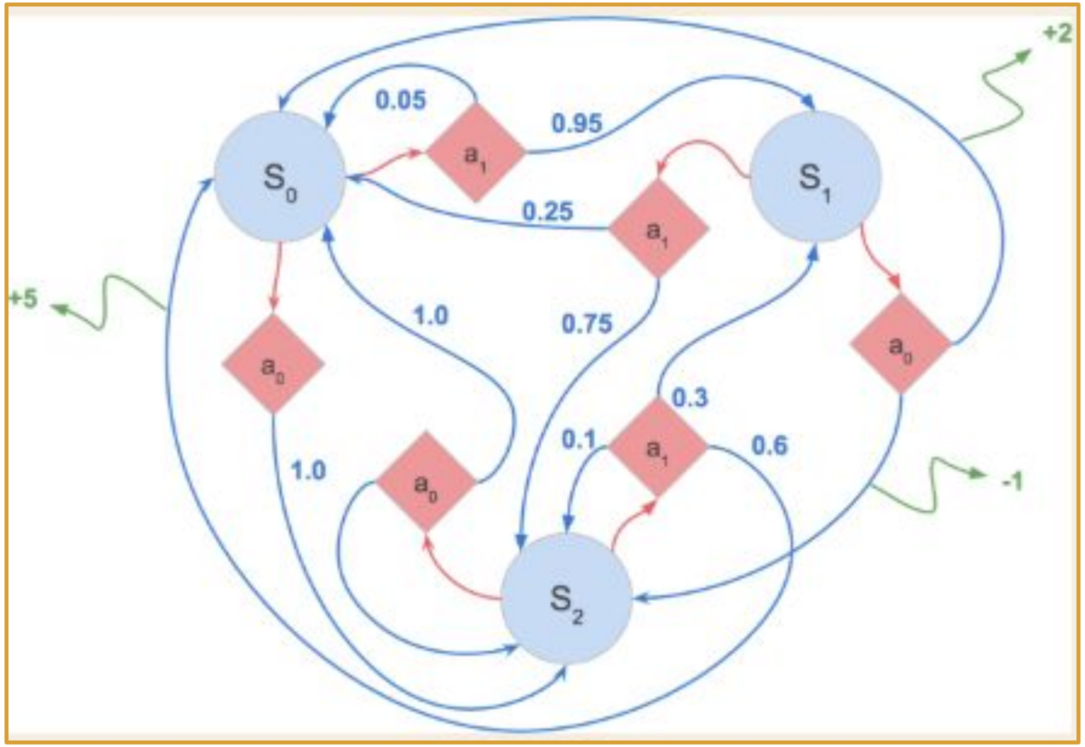
A a finite set of actions

$P(r, s' \mid s, a)$ a state transition function

R reward function

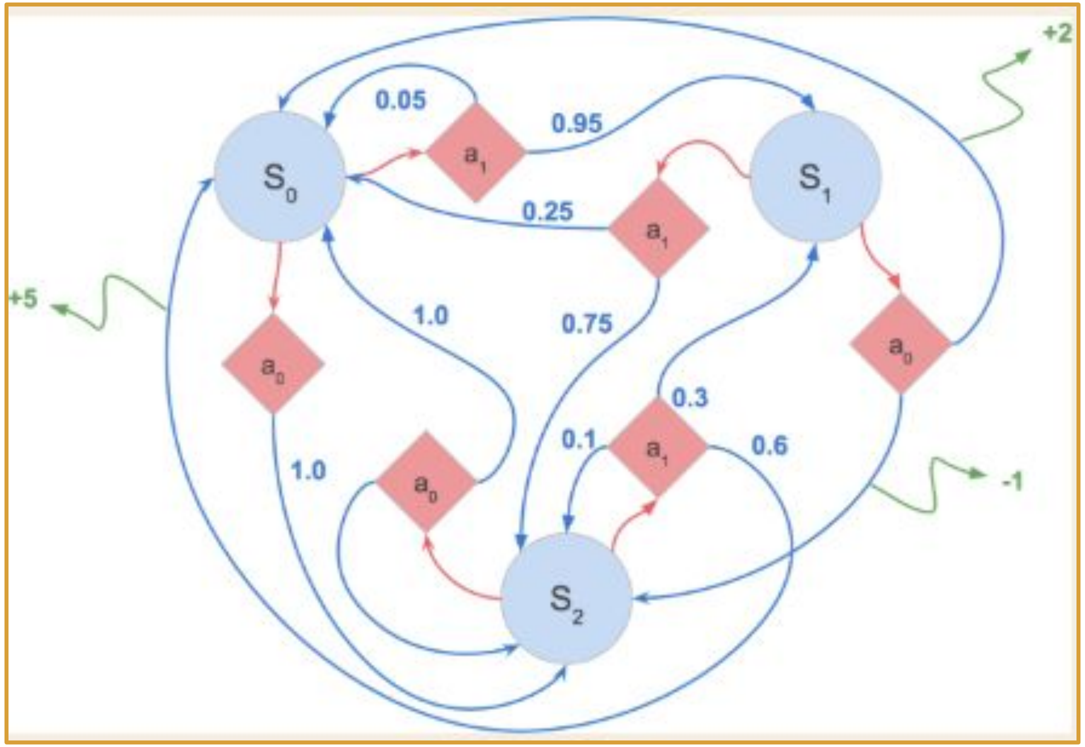
Formalizing Markov Decision Process

- Blue circles represent the states of the environment.
- Red diamonds represent actions that can be taken.
- The edges from diamonds to circles represent the transition from one state to the next.



MDP : Mathematical Form of Decision Making

- The numbers along these edges represent the probability of taking a certain action.
- The numbers at the end of the green arrows represent the reward given to the agent for making the given transition.



MDP : Mathematical Form of Decision Making

Episode

- An episode consists of series of tuples of states, actions, and rewards.

- As an agent takes action in an MDP framework, it forms an episode

- Episodes run until the environment reaches a terminal state, like the "Game Over" screen in Atari games.

  E.g., When the pole hits the ground in pole-cart example.

Equation of Episode

$$(s_0, a_0, r_0), (s_1, a_1, r_1), \ldots (s_n, a_n, r_n)$$

Policy

- **Aim of MDP** : Find an optimal policy for the agent

- Policies are the way in which our agent acts based on its current state.

- policies can be represented as a function f that chooses the action A that the agent will take in state S.

Policy

- Objective of our MDP is to find a policy to maximize the expected future return.

- Policies can be represented as a function f that chooses the action A that the agent will take in state S.

Future Return

- Future return is how we consider the rewards of the future

- Choosing the best action requires consideration of not only the immediate effects of that action, but also the long-term consequences

Future Return

- Sometimes the best action actually has a negative immediate effect, but a better long-term result.
- For example, a mountain-climbing agent that is rewarded by its altitude may actually have to climb downhill to reach a better path to the mountain’s peak.

Optimizing Future Return

- the agent must consider the future consequences of its actions.

  For example, in a game of Pong, the agent receives a reward when the ball passes into the opponent’s goal.
- However, the actions responsible for this reward (the inputs that position the racquet to strike scoring hit) happen many time steps before the reward is received. The reward for each of those actions is delayed.

Incorporating Delayed Awards

- We can incorporate delayed rewards into our overall reward signal by constructing a return for each time step that takes into account future rewards as well as immediate rewards.

- A naive approach for calculating future return for a time step may be a simple sum like so:

$$R_t = \sum_{k=0}^{T} r_{t+k}$$

Applications of Deep Reinforcement Learning

- Robotic Motor Controls.
- Self Driving Cars.
- Game Playing
- Air Conditioning Control
- Ad-Placement Optimization
- Stock Market Trading Strategies

List of Reading Materials

http://deeplearning.net/reading-list/

https://project.inria.fr/deeplearning/files/2016/05/deepLearning.pdf

Any questions?

You can find me at

ankita.sinha 8118@gmail.com

Thank You