Unit 2
Introduction to Deep Learning & Architectures

Deep Learning V/S Machine Learning

- Both machine learning and deep learning have the potential to transform a wide range of industries, including healthcare, finance, retail, and transportation, by providing insights and automating decision-making processes.
- **Machine Learning:**
- Machine learning is a subset, an application of Artificial Intelligence (AI) that offers the ability of the system to learn and improve from experience without being programmed to that level.
- Machine Learning uses data to train and find accurate results. Machine learning focuses on the development of a computer program that accesses the data and uses it to learn from itself.

- **Deep Learning:**
- Deep Learning is a subset of Machine Learning where the artificial neural network and the recurrent neural network come in relation.
- The algorithms are created exactly just like machine learning but it consists of many more levels of algorithms.
- All these networks of the algorithm are together called the artificial neural network. In much simpler terms, it replicates just like the human brain as all the neural networks are connected in the brain, which exactly is the concept of deep learning. It solves all the complex problems with the help of algorithms and its process.

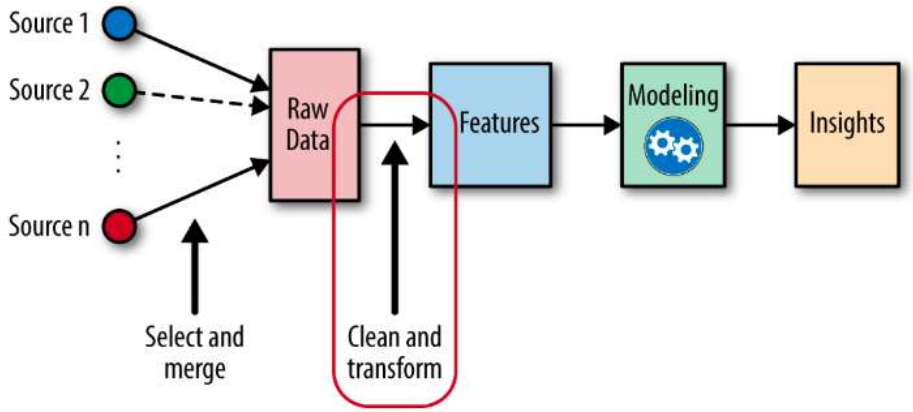
S. No.	Machine Learning	Deep Learning
1.	Machine Learning is a superset of Deep Learning	Deep Learning is a subset of Machine Learning
2.	The data represented in Machine Learning is quite different compared to Deep Learning as it uses structured data	The data representation used in Deep Learning is quite different as it uses neural networks(ANN).
34.	Machine learning consists of thousands of data points.	Big Data: Millions of data points.
5.	Outputs: Numerical Value, like classification of the score.	Anything from numerical values to free-form elements, such as free text and sound.

5.	Uses various types of automated algorithms that turn to model functions and predict future action from data.	Uses a neural network that passes data through processing layers to, interpret data features and relations.
6.	Algorithms are detected by data analysts to examine specific variables in data sets.	Algorithms are largely self-depicted on data analysis once they're put into production.
7.	Training can be performed using the CPU (Central Processing Unit).	A dedicated GPU (Graphics Processing Unit) is required for training.
8.	More human intervention is involved in getting results.	Although more difficult to set up, deep learning requires less intervention once it is running.

9.	Machine learning involves training algorithms to identify patterns and relationships in data.	Deep learning, on the other hand, uses complex neural networks with multiple layers to analyze more intricate patterns and relationships.
10.	Machine learning algorithms can range from simple linear models to more complex models such as decision trees and random forests.	Deep learning algorithms, on the other hand, are based on artificial neural networks that consist of multiple layers and nodes.
11.	Machine learning algorithms typically require less data than deep learning algorithms, but the quality of the data is more important.	Deep learning algorithms, on the other hand, require large amounts of data to train the neural networks but can learn and improve on their own as they process more data.
12.	Machine learning is used for a wide range of applications, such as regression, classification, and clustering.	Deep learning, on the other hand, is mostly used for complex tasks such as image and speech recognition, natural language processing, and autonomous systems.

Feature Engineering

- Feature engineering is the process of transforming raw data into features that are suitable for machine learning models. In other words, it is the process of selecting, extracting, and transforming the most relevant features from the available data to build more accurate and efficient machine learning models.



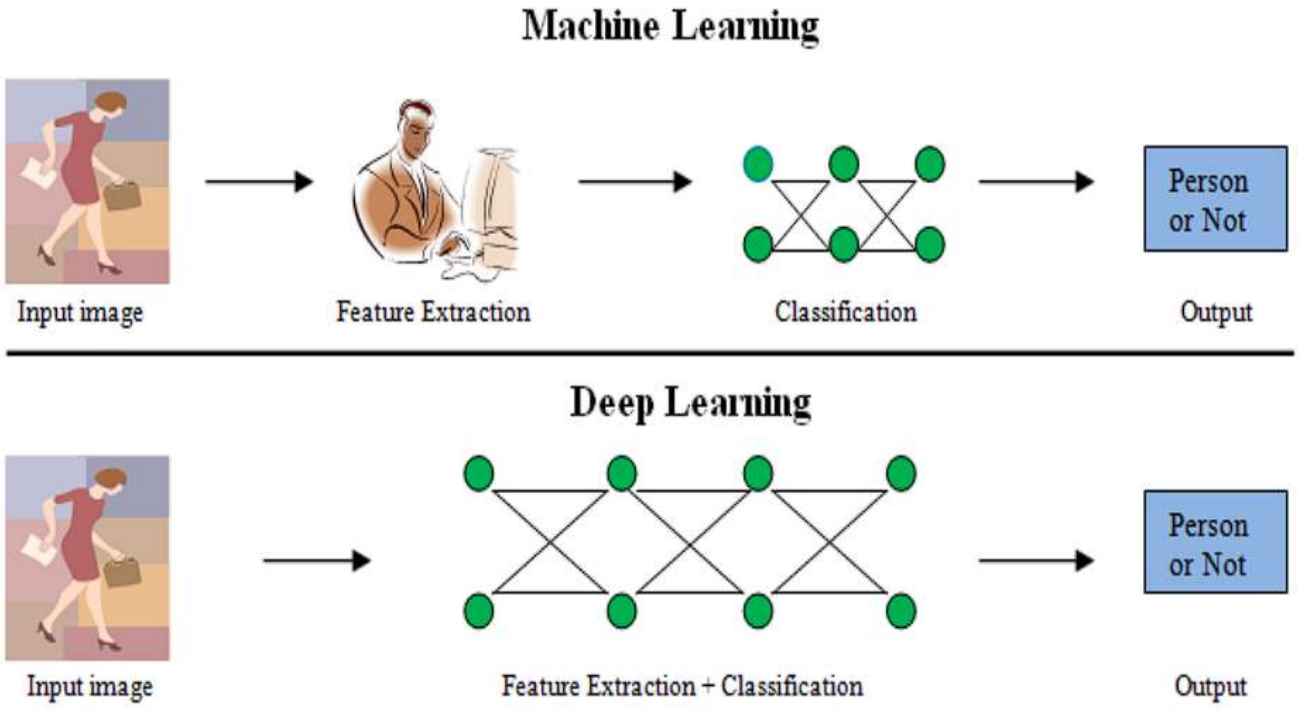
- The success of machine learning models heavily depends on the quality of the features used to train them.
- Feature engineering involves a set of techniques that enable us to create new features by combining or transforming the existing ones.
- These techniques help to highlight the most important patterns and relationships in the data, which in turn helps the machine learning model to learn from the data more effectively.

Representation Learning

- Representation Learning is a process in machine learning where algorithms extract meaningful patterns from raw data to create representations that are easier to understand and process.
- These representations can be designed for interpretability, reveal hidden features, or be used for transfer learning.
- They are valuable across many fundamental machine learning tasks like image classification and retrieval.
- Deep neural networks can be considered representation learning models that typically encode information which is projected into a different subspace. These representations are then usually passed on to a linear classifier to, for instance, train a classifier.

- Representation learning works by reducing high-dimensional data to low-dimensional data, making it easier to discover patterns and anomalies while also providing a better understanding of the data's overall behavior.
- Basically, Machine learning tasks such as classification frequently demand input that is mathematically and computationally convenient to process, which motivates representation learning.
- Real-world data, such as photos, video, and sensor data, has resisted attempts to define certain qualities algorithmically.
- An approach is to examine the data for such traits or representations rather than depending on explicit techniques.

- Representation learning is a method of training a machine learning model to discover and learn the most useful representations of input data automatically.
- These representations, often known as **“features”,** are internal states of the model that effectively summarize the input data, which help the algorithm to understand the underlying patters of this data better.
- Deep Neural Network models could learn complex, hierarchical representations of data through multiple layers. Eg, CNN, RNN, Autoencoder, and Transformers.


Machine Learning
Person or Not
Input image
Feature Extraction
Classification
Output
Deep Learning
Person or Not
Input image
Feature Extraction + Classification
Output

- A good representation has three characteristics: **Information, compactness, and generalization.**
- **Information:** The representation encodes important features of the data into a compressed form.
- **Compactness:**
- **Low Dimensionality**: Learned embedding representations from raw data should be much smaller than the original input. This allows for efficient storage and retrieval, and also discards noise from the data, allowing the model to focus on relevant features and converge faster.
- **Preserves Essential Information:** Despite being lower-dimensional, the representation retains important features. This balance between dimensionality reduction and information preservation is essential.

- **Generalization (Transfer Learning):** The aim is to learn versatile representations for transfer learning, starting with a pre-trained model and then fine-tuning it for specific tasks requiring less data.

- Representation learning can be divided into:
- **Supervised representation learning:**
- Leverages Labeled Data: Uses labeled data. The labels guide the learning algorithm about the desired outcome.
- Focuses on Specific Tasks: The learning process is tailored towards a specific task, such as image classification or sentiment analysis. The learned representations are optimized to perform well on that particular task.
- Examples: Training a Convolutional Neural Network (CNN) to classify objects in images (e.g., dog, cat) using labeled image datasets, or a Recurrent Neural Network (RNN) for sentiment analysis of text data (positive, negative, neutral) with labeled reviews or sentences.

- **Unsupervised Representation Learning:**
- Without Labels: Works with unlabeled data. The algorithm identifies patterns and relationships within the data itself.
- Focuses on Feature Extraction: The goal is to learn informative representations that capture the underlying structure and essential features of the data. These representations can then be used for various downstream tasks (transfer learning).
- Examples: Training an autoencoder to compress and reconstruct images, learning a compressed representation that captures the key features of the image. Using Word2Vec or GloVe on a massive text corpus to learn word embeddings, where words with similar meanings have similar representations in a high-dimensional space. BERT to learn contextual representation of words.

Width Vs. Depth of Neural Networks

- In deep learning, the terms "depth" and "width" refer to different aspects of the architecture of a neural network:

1. **Depth**:
    1. **Definition**: The depth of a neural network is defined by the number of layers it has. This includes all layers that have learnable parameters, such as convolutional layers, fully connected layers, and recurrent layers.
    2. **Implication**: A deeper network, meaning one with more layers, can capture more complex patterns in the data. This is because each additional layer can build on the representations learned by previous layers, allowing the network to understand more intricate structures and hierarchies in the input data.
    3. **Example**: If a neural network has an input layer, 5 hidden layers, and an output layer, it is considered to be 7 layers deep.

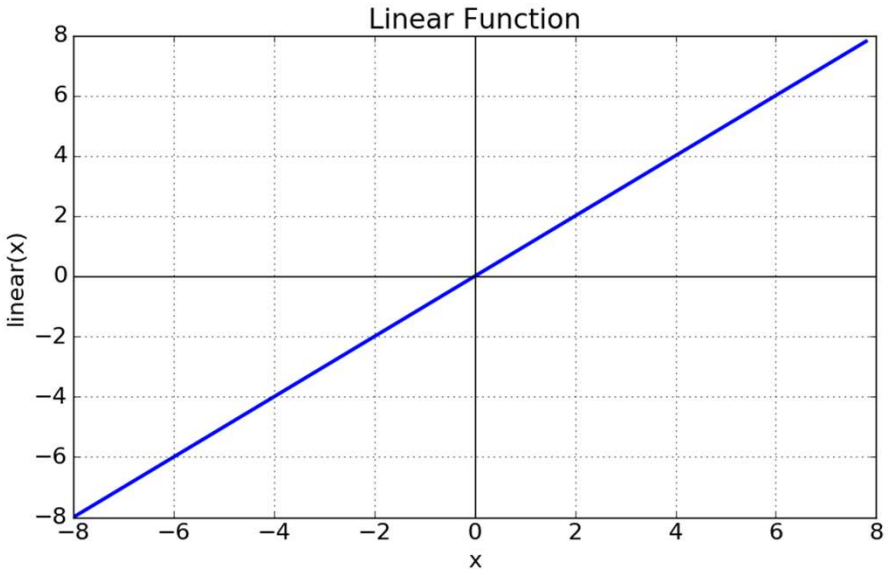
- **Width**:
- **Definition**: The width of a neural network refers to the number of units (neurons) in each layer. It typically refers to the layers with the most neurons.
- **Implication**: A wider network, meaning one with more neurons per layer, can learn more features in parallel. This can be beneficial for capturing more detailed information about the data, especially if each neuron can learn a distinct feature or pattern.
- **Example**: If a neural network layer has 512 neurons, it is said to be 512 units wide.

- **Balance**: Both depth and width need to be balanced based on the specific problem and the amount of available data. Very deep or very wide networks can lead to overfitting, especially with insufficient training data.
- **Computational Cost**: Increasing either depth or width increases the computational cost and memory requirements of training and using the neural network.
- **Architecture Choice**: Different tasks may benefit more from deeper networks (e.g., image classification with convolutional neural networks) or wider networks (e.g., natural language processing with transformer models).
- **Examples of Architectures**:
- **Deep Networks**: ResNet-50, VGG-16 (these networks have many layers).
- **Wide Networks**: Wide ResNet (these networks have fewer layers but more neurons per layer compared to their deep counterparts).

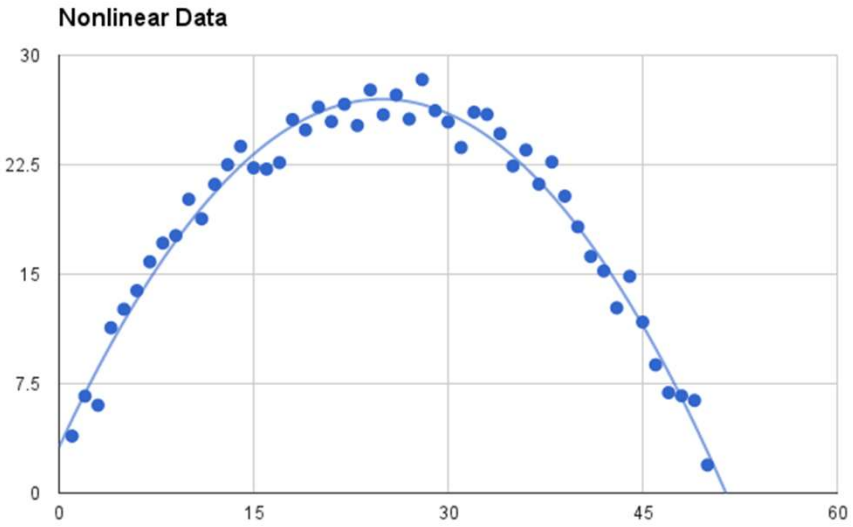
Activation Functions

- We use activation functions to propagate the output of one layer's nodes forward to the next layer (up to and including the output layer).
- Activation functions are a scalar-to-scalar function, yielding the neuron's activation.
- We use activation functions for hidden neurons in a neural network to introduce nonlinearity into the network's modeling capabilities.
- It's just a thing function that you use to get the output of node. It is also known as **Transfer Function.**

- It is used to determine the output of neural network like yes or no. It maps the resulting values in between 0 to 1 or -1 to 1 etc. (depending upon the function).
- The Activation Functions can be basically divided into 2 types-
- **Linear Activation Function**
- **Non-linear Activation Functions**



- As you can see the function is a line or linear. Therefore, the output of the functions will not be confined between any range.
- **Equation :** f(x) = x
- **Range :** (-infinity to infinity)
- It doesn't help with the complexity or various parameters of usual data that is fed to the neural networks.

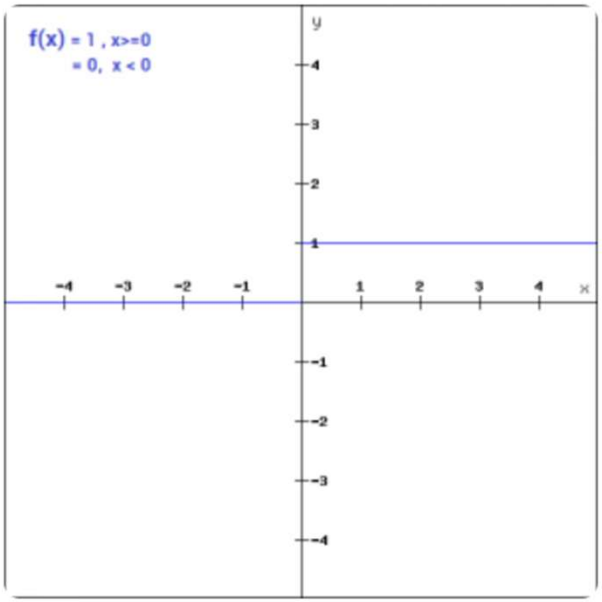


- The Nonlinear Activation Functions are the most used activation functions.
- It makes it easy for the model to generalize or adapt with variety of data and to differentiate between the output.
- The main terminologies needed to understand for nonlinear functions are:
- **Derivative or Differential:** Change in y-axis w.r.t. change in x-axis. It is also known as slope.
- **Monotonic function:** A function which is either entirely non-increasing or non-decreasing.

- Imagine a neural network without the activation functions. In that case, every neuron will only be performing a linear transformation on the inputs using the weights and biases.
- Although linear transformations make the neural network simpler, but this network would be less powerful and will not be able to learn the complex patterns from the data.
- A neural network without an activation function in deep learning is essentially just a linear regression model.
- Thus we use a non linear transformation to the inputs of the neuron and this non-linearity in the network is introduced by an activation function.

- **Binary Step Function**
- The first thing that comes to our mind when we have an activation function would be a threshold based classifier i.e. whether or not the neuron should be activated based on the value from the linear transformation.
- In other words, if the input to the activation function is greater than a threshold, then the neuron is activated, else it is deactivated, i.e. its output is not considered for the next hidden layer.

```
f(x) = 1, x>=0
     = 0, x<0
```



```
def binary_step(x):
    if x<0:
        return 0
    else:
        return 1
```

```
binary_step(5), binary_step(-1)
```

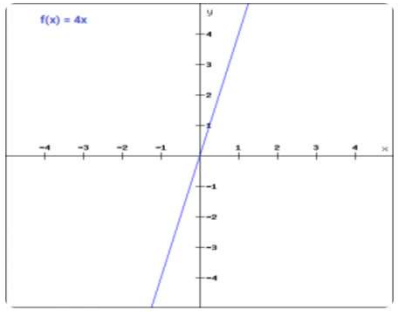
Output:

```
(1,0)
```

- The binary step function can be used as an activation function while creating a binary classifier.
- As you can imagine, this function will not be useful when there are multiple classes in the target variable. That is one of the limitations of binary step function.
- Moreover, the gradient of the step function is zero which causes a hindrance in the back propagation process. That is, if you calculate the derivative of f(x) with respect to x, it comes out to be 0.
- Gradients are calculated to update the weights and biases during the backprop process. Since the gradient of the function is zero, the weights and biases don’t update.

- **Linear Function**
- We saw the problem with the step function, the gradient of the function became zero. This is because there is no component of x in the binary step function.
- Instead of a binary function, we can use a linear function. We can define the function as-

```
f(x)=ax
```



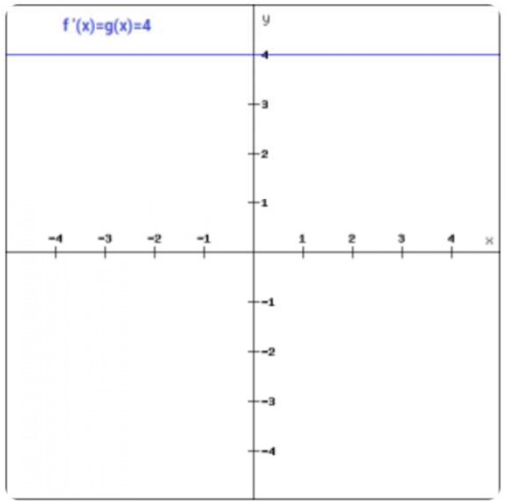
- Here the activation is proportional to the input. The variable ‘a’ in this case can be any constant value.

```
def linear_function(x):
    return 4*x
```

```
linear_function(4), linear_function(-2)
```

- When we differentiate the function with respect to x, the result is the coefficient of x, which is a constant.

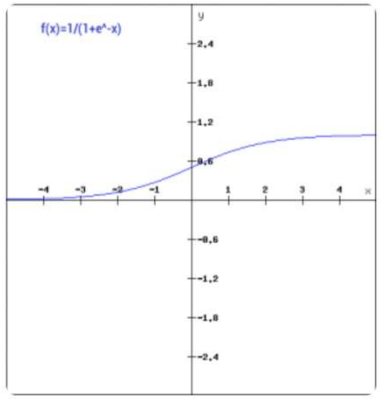
```
f'(x) = a
```



- Although the gradient here does not become zero, but it is a constant which does not depend upon the input value x at all.
- This implies that the weights and biases will be updated during the backpropagation process but the updating factor would be the same.
- In this scenario, the neural network will not really improve the error since the gradient is the same for every iteration.
- The network will not be able to train well and capture the complex patterns from the data. Hence, linear function might be ideal for simple tasks where interpretability is highly desired.

- **Sigmoid Activation Function**
- The next activation function in deep learning that we are going to look at is the Sigmoid activation function. It is one of the most widely used non-linear activation function. Sigmoid transforms the values between the range 0 and 1.

```
f(x) = 1/(1+e^-x)
```



- unlike the binary step and linear functions, sigmoid is a non-linear function. This essentially means -when I have multiple neurons having sigmoid function as their activation function, the output is non linear as well.

```
import numpy as np
def sigmoid_function(x):
    z = (1/(1 + np.exp(-x)))
    return z
```

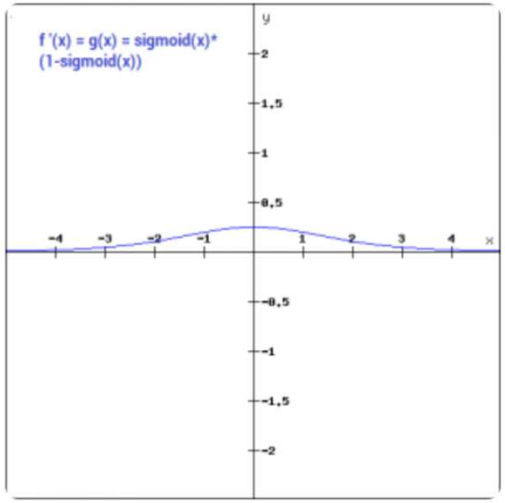
```
sigmoid_function(7),sigmoid_function(-22)
```

Output:

```
(0.9990889488055994, 2.7894680920908113e-10)
```

- this is a smooth S-shaped function and is continuously differentiable. The derivative of this function comes out to be (sigmoid(x)*(1-sigmoid(x))
- .

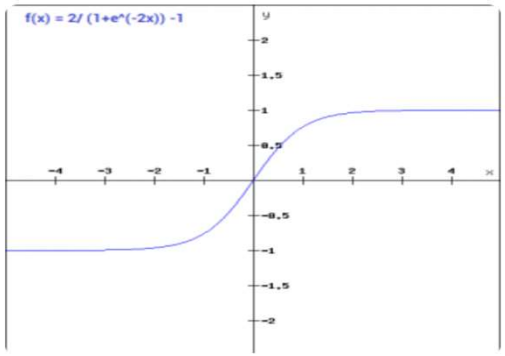
```
f'(x) = sigmoid(x)*(1-sigmoid(x))
```



- The gradient values are significant for range -3 and 3 but the graph gets much flatter in other regions.
- This implies that for values greater than 3 or less than -3, will have very small gradients. As the gradient value approaches zero, the network is not really learning.
- Additionally, the sigmoid function is not symmetric around zero. So output of all the neurons will be of the same sign. T
- his can be addressed by scaling the sigmoid function which is exactly what happens in the tanh function. Let’s read on.

- **Tanh**
- The tanh function is very similar to the sigmoid function. The only difference is that it is symmetric around the origin.
- The range of values in this case is from -1 to 1. Thus the inputs to the next layers will not always be of the same sign. The tanh function is defined as-

$$\tanh(x)=2\,\text{sigmoid}(2x)-1$$



```
tanh(x) = 2sigmoid(2x)-1
```

```
tanh(x) = 2/(1+e^(-2x)) -1
```

And here is the python code for the same:

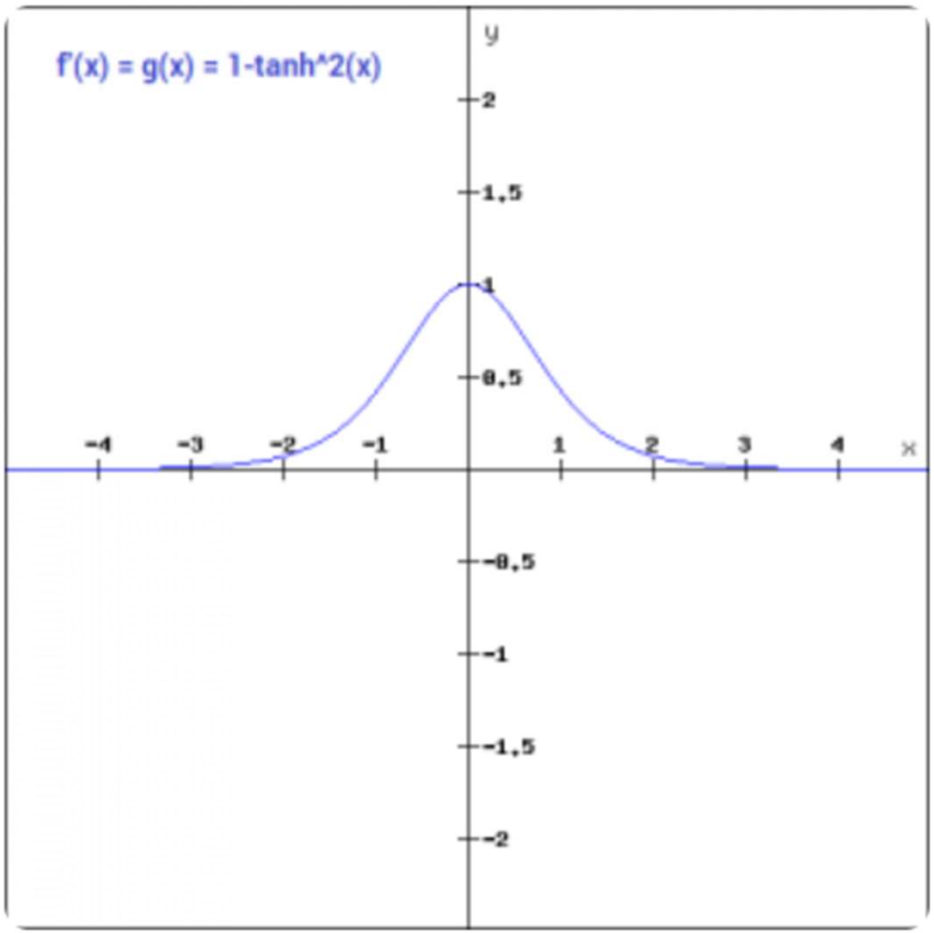
```
def tanh_function(x):
    z = (2/(1 + np.exp(-2*x))) -1
    return z
```

```
tanh_function(0.5), tanh_function(-1)
```

Output:

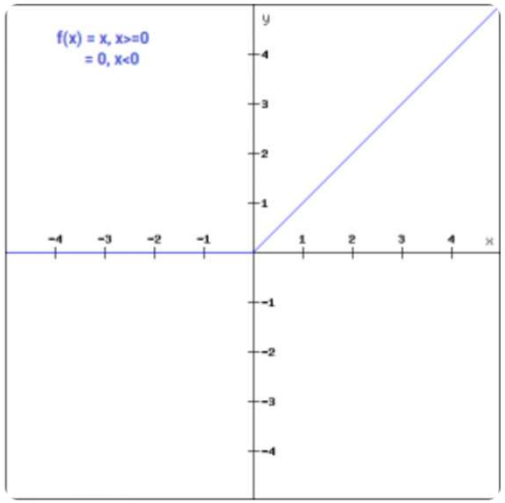
```
(0.4621171572600098, -0.7615941559557646)
```

- the range of values is between -1 to 1. Apart from that, all other properties of tanh function are the same as that of the sigmoid function.
- Similar to sigmoid, the tanh function is continuous and differentiable at all points.
- The gradient of the tanh function is steeper as compared to the sigmoid function. You might be wondering, how will we decide which activation function to choose? Usually tanh is preferred over the sigmoid function since it is zero centered and the gradients are not restricted to move in a certain direction.


f'(x) = g(x) = 1-tanh^2(x)
y
x
2
1,5
1
0,5
-0,5
-1
-1,5
-2
-4
-3
-2
-1
1
2
3
4

- **ReLU Activation Function**
- The ReLU Activation function is another non-linear activation function that has gained popularity in the deep learning domain.
- ReLU stands for Rectified Linear Unit. The main advantage of using the ReLU function over other activation functions is that it does not activate all the neurons at the same time.
- This means that the neurons will only be deactivated if the output of the linear transformation is less than 0.

```
f(x)=max(0,x)
```



- For the negative input values, the result is zero, that means the neuron does not get activated.
- Since only a certain number of neurons are activated, the ReLU function is far more computationally efficient when compared to the sigmoid and tanh function.

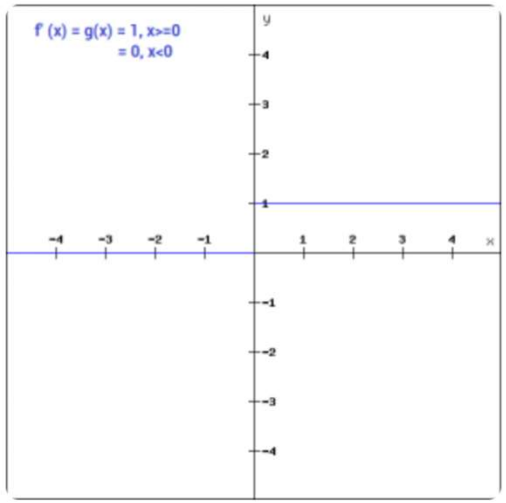
```
def relu_function(x):
    if x<0:
        return 0
    else:
        return x
```

```
relu_function(7), relu_function(-7)
```

Output:

```
(7, 0)
```

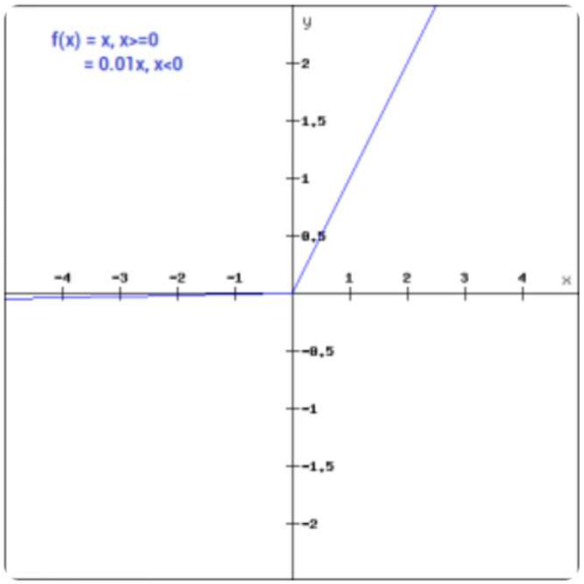
```
f'(x) = 1, x>=0
      = 0, x<0
```



- If you look at the negative side of the graph, you will notice that the gradient value is zero.
- Due to this reason, during the backpropagation process, the weights and biases for some neurons are not updated.
- This can create dead neurons which never get activated. This is taken care of by the ‘Leaky’ ReLU function.

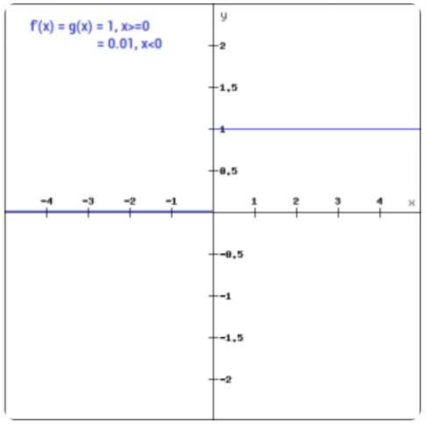
- **Leaky ReLU**
- Leaky ReLU function is nothing but an improved version of the ReLU function.
- As we saw that for the ReLU function, the gradient is 0 for $x<0$, which would deactivate the neurons in that region.
- Leaky ReLU is defined to address this problem. Instead of defining the Relu function as 0 for negative values of x, we define it as an extremely small linear component of x.

```
f(x)= 0.01x, x<0
    =   x, x>=0
```



- By making this small modification, the gradient of the left side of the graph comes out to be a non zero value. Hence we would no longer encounter dead neurons in that region. Here is the derivative of the Leaky ReLU function

```
f'(x) = 1, x>=0
      =0.01, x<0
```



- Since Leaky ReLU is a variant of ReLU, the python code can be implemented with a small modification-

```
def leaky_relu_function(x):
    if x<0:
        return 0.01*x
    else:
        return x
```

```
leaky_relu_function(7), leaky_relu_function(-7)
```

Output:

```
(7, -0.07)
```

Unsupervised Training of Neural Networks

- Unsupervised learning is an intriguing area of machine learning that reveals hidden structures and patterns in data without requiring labelled samples. Because it investigates the underlying relationships in data, it's an effective tool for tasks like anomaly identification, dimensionality reduction, and clustering.
- There are several uses for unsupervised learning in domains like computer vision, natural language processing, and data analysis.
- Through self-sufficient data interpretation, it provides insightful information that enhances decision-making and facilitates comprehension of intricate data patterns.

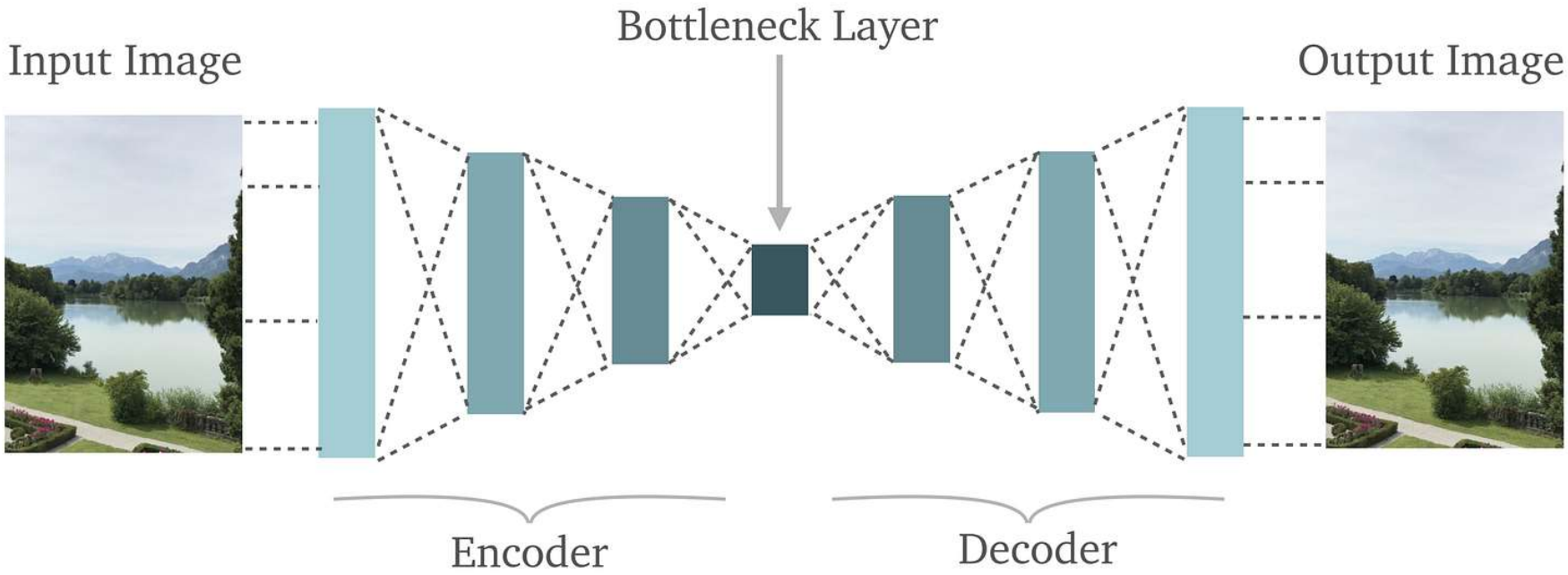
- An unsupervised neural network is a type of artificial neural network (ANN) used in unsupervised learning tasks.
- Unlike supervised neural networks, trained on labeled data with explicit input-output pairs, unsupervised neural networks are trained on unlabeled data.
- In unsupervised learning, the network is not under the guidance of features. Instead, it is provided with unlabeled data sets (containing only the input data) and left to discover the patterns in the data and build a new model from it.
- Here, it has to figure out how to arrange the data by exploiting the separation between clusters within it. These neural networks aim to discover patterns, structures, or representations within the data without specific guidance.

- There are several components of unsupervised learning. They are:
- **Encoder-Decoder:** As the name itself suggests that it is used to encode and decode the data.
- Encoder basically responsible for transforming the input data into lower dimensional representation on which the neural network works.
- Whereas decoder takes the encoded representation and reconstruct the input data from it. There architecture and parameters are learned during the training of the network.
- **Latent Space:** It is the immediate representation created by the encoder. It contains the abstract representation or features that captures important information about the data's structures. It is also known as the latent space.

- **Training algorithm:** Unsupervised neural network model use specific training algorithms to get the parameters. Some of the common optimization algorithms are Stochastic gradient descent, Adam etc. They are used depending on the type of model and loss function.
- **Loss Function:** It is a common component among all the machine learning models. It basically calculates the model's output and the actual/measured output. It quantifies how well the model understands the data.

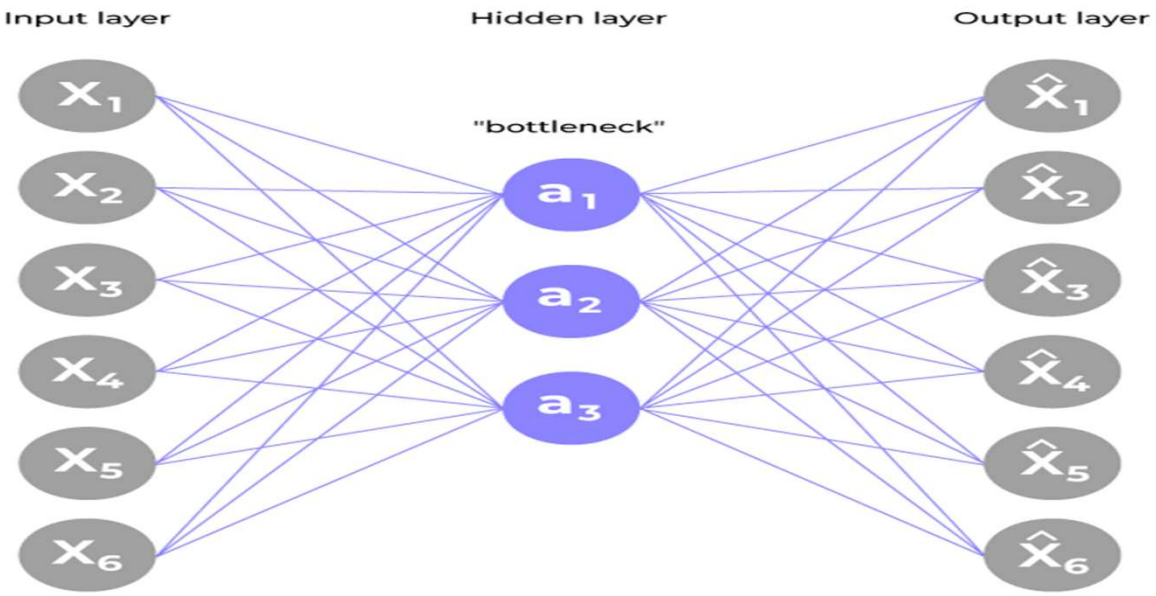
Autoencoder

- Autoencoders are a specialized class of algorithms that can learn efficient representations of input data with no need for labels.
- It is a class of artificial neural networks designed for unsupervised learning. Learning to compress and effectively represent input data without specific labels is the essential principle of an automatic decoder.
- This is accomplished using a two-fold structure that consists of an encoder and a decoder.
- The encoder transforms the input data into a reduced-dimensional representation, which is often referred to as “latent space” or “encoding”. From that representation, a decoder rebuilds the initial input. For the network to gain meaningful patterns in data, a process of encoding and decoding facilitates the definition of essential features.


Input Image
Bottleneck Layer
Output Image
Encoder
Decoder

Architecture of Autoencoder in Deep Learning

- The general architecture of an autoencoder includes an encoder, decoder, and bottleneck layer.



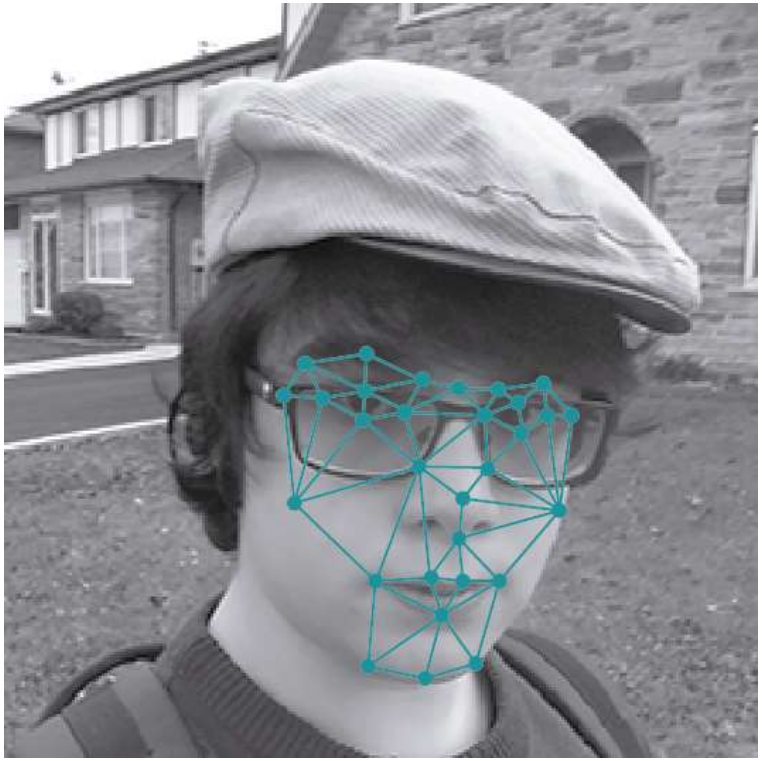
- **Encoder**
- Input layer take raw input data
- The hidden layers progressively reduce the dimensionality of the input, capturing important features and patterns. These layer compose the encoder.
- The bottleneck layer (latent space) is the final hidden layer, where the dimensionality is significantly reduced. This layer represents the compressed encoding of the input data.

- **Decoder**
- The bottleneck layer takes the encoded representation and expands it back to the dimensionality of the original input.
- The hidden layers progressively increase the dimensionality and aim to reconstruct the original input.
- The output layer produces the reconstructed output, which ideally should be as close as possible to the input data.

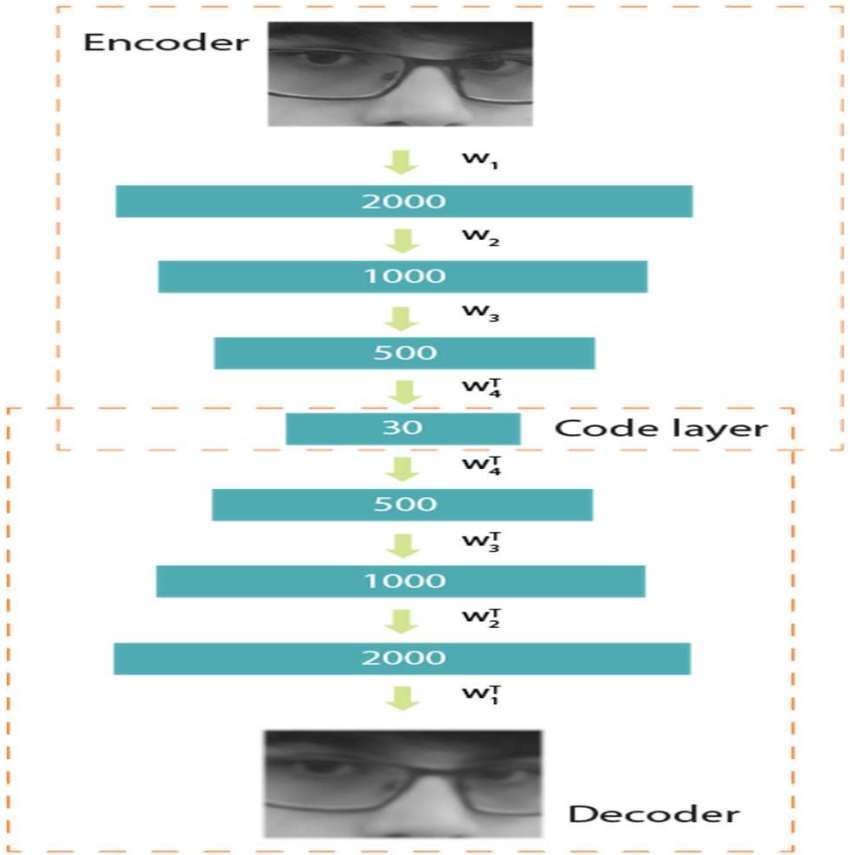
- The loss function used during training is typically a reconstruction loss, measuring the difference between the input and the reconstructed output. Common choices include mean squared error (MSE) for continuous data or binary cross-entropy for binary data.
- During training, the autoencoder learns to minimize the reconstruction loss, forcing the network to capture the most important features of the input data in the bottleneck layer.

- After the training process, only the encoder part of the autoencoder is retained to encode a similar type of data used in the training process. The different ways to constrain the network are: –
- **Keep small Hidden Layers:** If the size of each hidden layer is kept as small as possible, then the network will be forced to pick up only the representative features of the data thus encoding the data.
- **Regularization:** In this method, a loss term is added to the cost function which encourages the network to train in ways other than copying the input.
- **Denoising:** Another way of constraining the network is to add noise to the input and teach the network how to remove the noise from the data.
- **Tuning the Activation Functions:** This method involves changing the activation functions of various nodes so that a majority of the nodes are dormant thus, effectively reducing the size of the hidden layers.

- Using the following 256x256 pixel grayscale picture as an example
- But when using this picture we start running into a bottleneck! Because this image being 256x256 pixels in size correspond with an input vector of 65536 dimensions!
- If we used an image produced with conventional cellphone cameras, that generates images of 4000 x 3000 pixels, we would have 12 million dimensions to analyze



- As you can see, it increases exponentially! Returning to our example, we don’t need to use all of the 65,536 dimensions to classify an emotion.
- A human identifies emotions according to some specific facial expression, some key features, like the shape of the mouth and eyebrows.


Encoder
w_1
2000
w_2
1000
w_3
500
w_4^T
30
Code layer
w_4^T
500
w_3^T
1000
w_2^T
2000
w_1^T
Decoder