1. What is the difference between TRAINABLE and NON-TRAINABLE PARAMETERS?

As already mentioned, parameters usually refer to the weights in a neural network. **These parameters often come in two flavors, trainable and non-trainable parameters**. Trainable parameters are those that have their values calculated and updated during the training phase. **Whereas non-trainable parameters are parameters whose values are not updated during training**.

Suppose we have a CNN that classifies RGB images as either cat or dog. The architecture consists of an input layer and convolution and pooling layers connected to a simple feed-forward network like an MLP. The main task of the convolution layers, i.e. layers prior to the feed-forward network, is to learn the features of cats and dogs from the input data.

After the features are learned, the MLP has the task of classifying images as cat or dog. Now during training, after a couple of iterations, we freeze the convolution layers or the portion that learns the features to focus on fine-tuning the MLP for classification. In this case, the convolutional layers are non-trainable and hence do not learn further. This is a typical example of the use of non-trainable parameters:

**When Do We Use Them?**

**Non-trainable parameters are most commonly used in**[**pre-trained**](https://www.baeldung.com/cs/neural-network-pre-training)**models and transfer learning**. In pre-training, a neural network is first trained as a model on one task or dataset. Then, the parameters or models from this training are used to train another model on a different task or dataset.

Some well-known examples of pre-trained models are [BERT](https://arxiv.org/abs/1810.04805), [VGG-16](https://arxiv.org/abs/1409.1556), [ResNet50](https://arxiv.org/abs/1512.03385), [Inceptionv3](https://arxiv.org/abs/1512.00567), and [GPT-4](https://arxiv.org/abs/2303.08774).

In [transfer learning](https://www.baeldung.com/cs/neural-network-pre-training#1-transfer-learning), the model parameters are frozen and saved in the pre-trained model; hence, they are not trainable when applied to a new task. In this case, the model is only fine-tuned to the new task and new data. Hence, training does not begin from scratch.

**Should We Use Non-trainable Parameters?**

Whether or not to use non-trainable parameters depends on the model used and the task at hand. For example, a simple classification model will most likely not require the use of non-trainable parameters. Whereas a computer vision task for image segmentation might require the use of non-trainable parameters.

**Pros**

There are several benefits and drawbacks to using non-trainable parameters. We’ll start off with the benefits. Firstly, using non-trainable parameters shortens the time required to train models. Using non-trainable parameters means fewer parameter updates, hence faster training time.

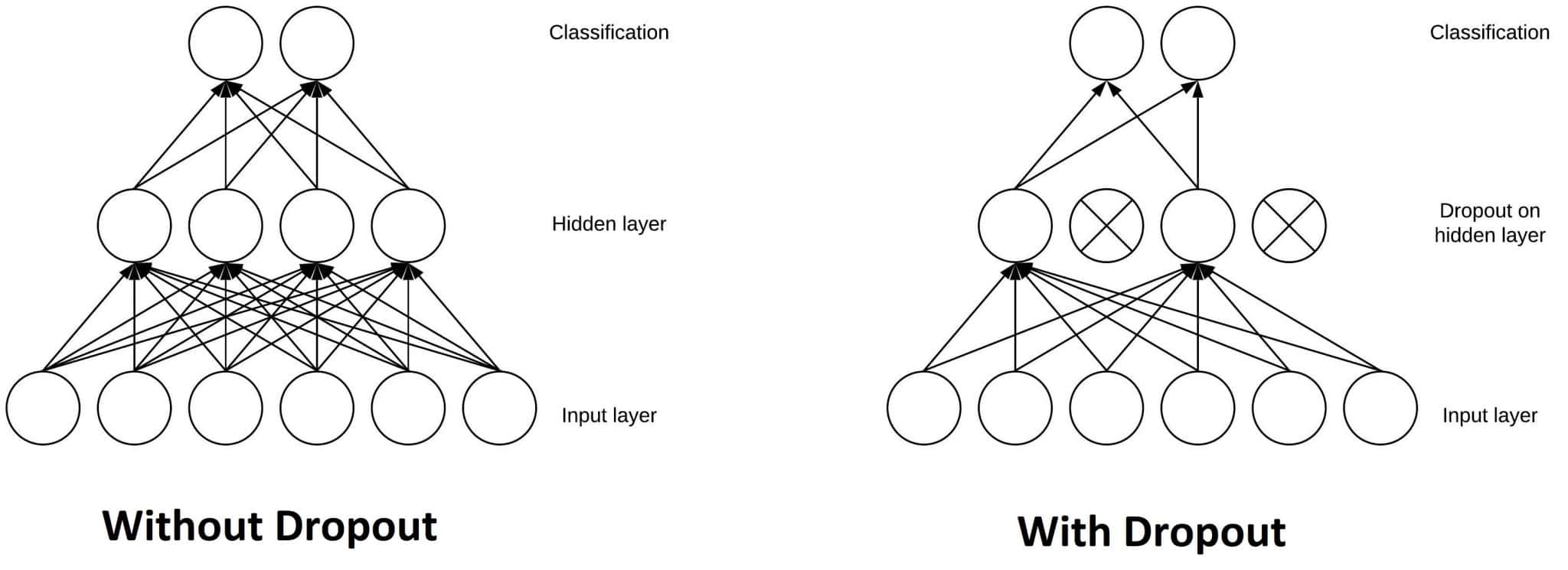
**Cons**

The most notable drawback to using non-trainable parameters is that model parameters cannot be adjusted when they are set to non-trainable. In this case, no further learning takes place in the model.

1. In the CNN architecture, where does the DROPOUT LAYER go?

Another typical characteristic of CNNs is a Dropout layer. **The Dropout layer is a mask that nullifies the contribution of some neurons towards the next layer and leaves unmodified all others**. We can apply a Dropout layer to the input vector, in which case it nullifies some of its features; but we can also apply it to a hidden layer, in which case it nullifies some hidden neurons.

Dropout layers are important in training CNNs because they prevent overfitting on the training data. If they aren’t present, the first batch of training samples influences the learning in a disproportionately high manner. This, in turn, would prevent the learning of features that appear only in later samples or batches:



Say we show ten pictures of a circle, in succession, to a CNN during training. The CNN won’t learn that straight lines exist; as a consequence, it’ll be pretty confused if we later show it a picture of a square. We can prevent these cases by adding Dropout layers to the network’s architecture, in order to prevent overfitting.

**A CNN With *ReLU* and a Dropout Layer**

This flowchart shows a typical architecture for a CNN with a ReLU and a Dropout layer. This type of architecture is very common for image classification tasks:

3. What is the optimal number of hidden layers to stack?

As far as I can imagine, it might be difficult to judge the maximum number of layers for a task without a proper ablation study. If the data is linearly separable then you don't need any hidden layers at all. If data is less complex and is having fewer dimensions or features then neural networks with 1 to 2 hidden layers would work. If data is having large dimensions or features then to get an optimum solution, 3 to 5 hidden layers can be used. It should be kept in mind that increasing hidden layers would also increase the complexity of the model and choosing hidden layers such as 8, 9, or in two digits may sometimes lead to overfitting.

4. In each layer, how many secret units or filters should there be?

most of deep learning choices on the tuning of hyper parameters have to do with empirical evidence and testing. Since the literature of deep learning have yet to devise a theory of what are the optimal hyper parameters, often times these hyperparameter (hidden layer size, filter size, batch size,etc…) are determined using trial and error. The 2 most common trial and error methods are random search or grid search. Although grid search (trialing using value with fixed intervals) might seem to logically be superior to random search (trailing using completely random values), however in practice, most of the time the performance of an algorithm is often heavily influenced by one of these parameters and employing random search will would provide the most bang for the buck (compute resource)

5. What should your initial learning rate be?

The learning rate will interact with many other aspects of the optimization process, and the interactions may be nonlinear. Nevertheless, in general, smaller learning rates will require more training epochs. Conversely, larger learning rates will require fewer training epochs. Further, smaller [batch sizes](https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/) are better suited to smaller learning rates given the noisy estimate of the error gradient.

A traditional default value for the learning rate is 0.1 or 0.01, and this may represent a good starting point on your problem.

6. What do you do with the activation function?

## What is an Activation Function?

An activation function in the context of neural networks is a mathematical function applied to the output of a neuron. The purpose of an activation function is to introduce non-linearity into the model, allowing the network to learn and represent complex patterns in the data. Without non-linearity, a neural network would essentially behave like a linear regression model, regardless of the number of layers it has.

The activation function decides whether a neuron should be activated or not by calculating the weighted sum and further adding bias to it. The purpose of the activation function is to introduce non-linearity into the output of a neuron.

**Explanation:** We know, the neural network has neurons that work in correspondence with weight, bias, and their respective activation function. In a neural network, we would update the weights and biases of the neurons on the basis of the error at the output. This process is known as [***back-propagation***](https://www.geeksforgeeks.org/backpropagation-in-data-mining/). Activation functions make the back-propagation possible since the gradients are supplied along with the error to update the weights and biases.

## ****Elements of a Neural Network****

**Input Layer:**This layer accepts input features. It provides information from the outside world to the network, no computation is performed at this layer, nodes here just pass on the information(features) to the hidden layer.

**Hidden Layer*:***Nodes of this layer are not exposed to the outer world, they are part of the abstraction provided by any neural network. The hidden layer performs all sorts of computation on the features entered through the input layer and transfers the result to the output layer.

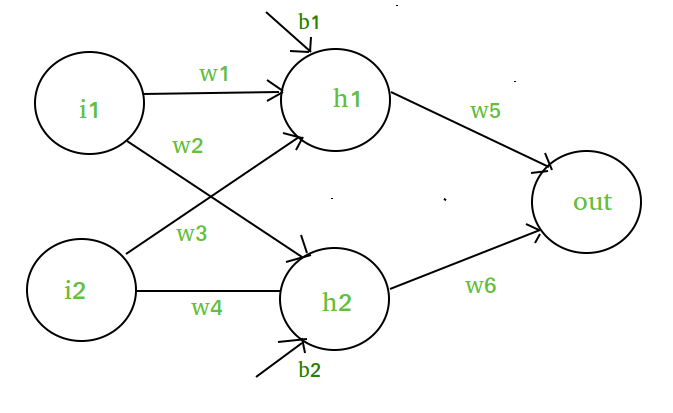
**Output Layer:**This layer bring up the information learned by the network to the outer world.

## ****Why do we need Non-linear activation function?****

A neural network without an activation function is essentially just a linear regression model. The activation function does the non-linear transformation to the input making it capable to learn and perform more complex tasks.

### ****Mathematical proof****

Suppose we have a Neural net like this :-



Elements of the diagram are as follows:

**Hidden layer i.e. layer 1:**

*z(1) = W(1)X + b(1) a(1)*

*Here,*

* *z(1) is the vectorized output of layer 1*
* *W(1) be the vectorized weights assigned to neurons of hidden layer i.e.*w1, w2, w3 and w4
* *X be the vectorized input features i.e.*i1 and i2
* *b is the vectorized bias assigned to neurons in hidden layer i.e.*b1 and b2
* *a(1) is the vectorized form of any linear function.*

*(****Note:****We are not considering activation function here)*

**Layer 2 i.e. output layer :-**

***Note :****Input for layer 2 is output from layer 1*

*z(2) = W(2)a(1) + b(2)*

*a(2) = z(2)*

#### ****Calculation at Output layer****

*z(2) = (W(2) \* [W(1)X + b(1)]) + b(2)*

*z(2) = [W(2) \* W(1)] \* X + [W(2)\*b(1) + b(2)]*

*Let,*

*[W(2) \* W(1)] = W*

*[W(2)\*b(1) + b(2)] = b*

*Final output : z(2) = W\*X + b*

*which is again a linear function*

This observation results again in a linear function even after applying a hidden layer, hence we can conclude that, doesn’t matter how many hidden layer we attach in neural net, all layers will behave same way because ***the composition of two linear function is a linear function itself***. Neuron can not learn with just a linear function attached to it. A non-linear activation function will let it learn as per the difference w.r.t error. **Hence we need an activation function.**

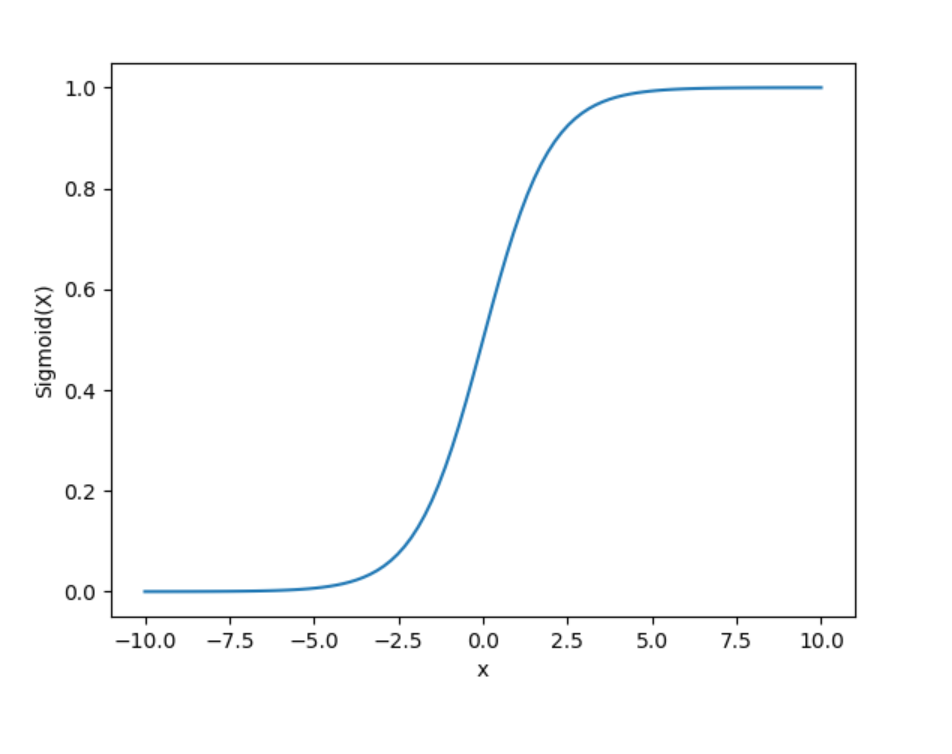
## ****Variants of Activation Function****

### ****Linear Function****

* **Equation :**Linear function has the equation similar to as of a straight line i.e. **y = x**
* No matter how many layers we have, if all are linear in nature, the final activation function of last layer is nothing but just a linear function of the input of first layer.
* **Range :** -inf to +inf
* **Uses : Linear activation function** is used at just one place i.e. output layer.
* **Issues :**If we will differentiate linear function to bring non-linearity, result will no more depend on input “x” and function will become constant, it won’t introduce any ground-breaking behavior to our algorithm.

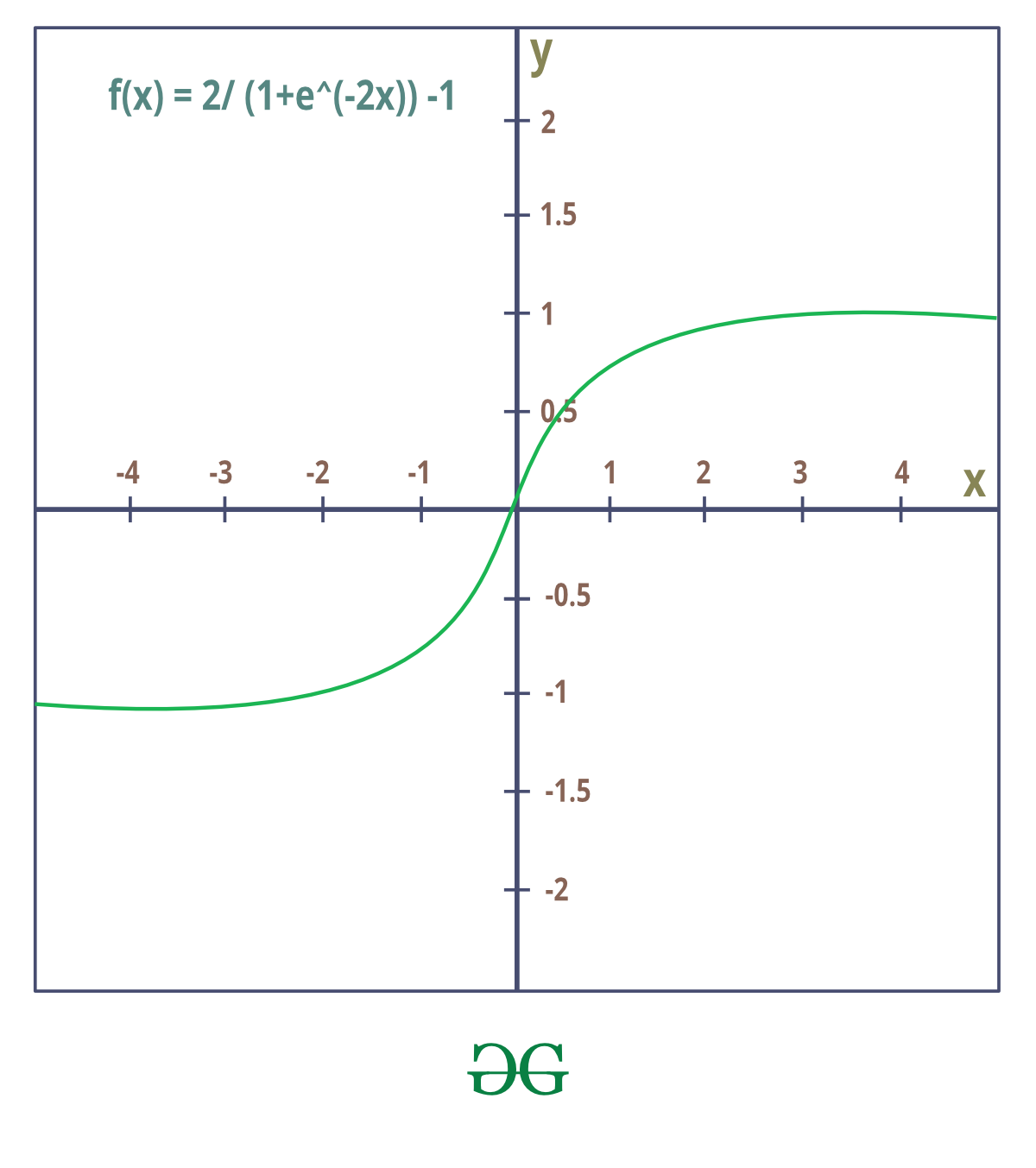
**For example :** Calculation of price of a house is a regression problem. House price may have any big/small value, so we can apply linear activation at output layer. Even in this case neural net must have any non-linear function at hidden layers.

### ****Sigmoid Function****



* It is a function which is plotted as **‘S’** shaped graph.
* **Equation :**A = 1/(1 + e-x)
* **Nature :** Non-linear. Notice that X values lies between -2 to 2, Y values are very steep. This means, small changes in x would also bring about large changes in the value of Y.
* **Value Range :**0 to 1
* **Uses :**Usually used in output layer of a binary classification, where result is either 0 or 1, as value for sigmoid function lies between 0 and 1 only so, result can be predicted easily to be ***1*** if value is greater than **0.5** and ***0*** otherwise.

### ****Tanh Function****



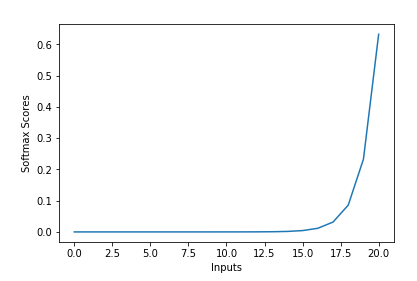
* The activation that works almost always better than sigmoid function is Tanh function also known as **Tangent Hyperbolic function**. It’s actually mathematically shifted version of the sigmoid function. Both are similar and can be derived from each other.
* **Equation :-**  
  f(x) = tanh(x) = 2/(1 + e-2x) – 1  
  OR  
  tanh(x) = 2 \* sigmoid(2x) – 1
* **Value Range :-**-1 to +1
* **Nature :-**non-linear
* **Uses :-**Usually used in hidden layers of a neural network as it’s values lies between **-1 to 1**hence the mean for the hidden layer comes out be 0 or very close to it, hence helps in centering the data by bringing mean close to 0. This makes learning for the next layer much easier.

### ****RELU Function****

* It Stands for Rectified linear unit. It is the most widely used activation function. Chiefly implemented in hidden layers of Neural network.
* **Equation :- *A(x) = max(0,x)***. It gives an output x if x is positive and 0 otherwise.
* **Value Range :-**[0, inf)
* **Nature :-**non-linear, which means we can easily backpropagate the errors and have multiple layers of neurons being activated by the ReLU function.
* **Uses :-**ReLu is less computationally expensive than tanh and sigmoid because it involves simpler mathematical operations. At a time only a few neurons are activated making the network sparse making it efficient and easy for computation.

In simple words, RELU learns much faster than sigmoid and Tanh function.

### ****Softmax Function****



The softmax function is also a type of sigmoid function but is handy when we are trying to handle multi- class classification problems.

* **Nature :-**non-linear
* **Uses :-**Usually used when trying to handle multiple classes. the softmax function was commonly found in the output layer of image classification problems.The softmax function would squeeze the outputs for each class between 0 and 1 and would also divide by the sum of the outputs.
* **Output:-**The softmax function is ideally used in the output layer of the classifier where we are actually trying to attain the probabilities to define the class of each input.
* The basic rule of thumb is if you really don’t know what activation function to use, then simply use RELU as it is a general activation function in hidden layers and is used in most cases these days.
* If your output is for binary classification then, sigmoid function is very natural choice for output layer.
* If your output is for multi-class classification then, Softmax is very useful to predict the probabilities of each classes.

7. What is NORMALIZATION OF DATA?

# Normalization in Machine Learning

**Normalization is one of the most frequently used data preparation techniques, which helps us to change the values of numeric columns in the dataset to use a common scale.**

Although **Normalization** is no mandate for all datasets available in machine learning, it is used whenever the attributes of the dataset have different ranges. It helps to enhance the performance and reliability of a machine learning model. In this article, we will discuss in brief various Normalization techniques in machine learning, why it is used, examples of normalization in an ML model, and much more. So, let's start with the definition of Normalization in Machine Learning.

## What is Normalization in Machine Learning?

Normalization is a scaling technique in Machine Learning applied during data preparation to change the values of numeric columns in the dataset to use a common scale. It is not necessary for all datasets in a model. It is required only when features of machine learning models have different ranges.

Mathematically, we can calculate normalization with the below formula:

1. Xn = (X - Xminimum) / ( Xmaximum - Xminimum)

* Xn = Value of Normalization
* Xmaximum = Maximum value of a feature
* Xminimum = Minimum value of a feature

**Example:** Let's assume we have a model dataset having maximum and minimum values of feature as mentioned above. To normalize the machine learning model, values are shifted and rescaled so their range can vary between 0 and 1. This technique is also known as **Min-Max scaling**. In this scaling technique, we will change the feature values as follows:

**Case1-** If the value of X is minimum, the value of Numerator will be 0; hence Normalization will also be 0.

1. Xn = (X - Xminimum) / ( Xmaximum - Xminimum)

Put X =Xminimum in above formula, we get;

Xn = Xminimum- Xminimum/ ( Xmaximum - Xminimum)

Xn = 0

**Case2-** If the value of X is maximum, then the value of the numerator is equal to the denominator; hence Normalization will be 1.

1. Xn = (X - Xminimum) / ( Xmaximum - Xminimum)

Put X =Xmaximum in above formula, we get;

Xn = Xmaximum - Xminimum/ ( Xmaximum - Xminimum)

Xn = 1

**Case3-** On the other hand, if the value of X is neither maximum nor minimum, then values of normalization will also be between 0 and 1.

Hence, Normalization can be defined as a scaling method where values are shifted and rescaled to maintain their ranges between 0 and 1, or in other words; it can be referred to as **Min-Max scaling technique**.

## Normalization techniques in Machine Learning

Although there are so many feature normalization techniques in Machine Learning, few of them are most frequently used. These are as follows:

* **Min-Max Scaling:** This technique is also referred to as scaling. As we have already discussed above, the Min-Max scaling method helps the dataset to shift and rescale the values of their attributes, so they end up ranging between 0 and 1.
* **Standardization scaling:**

Standardization scaling is also known as **Z-score** normalization, in which values are centered around the mean with a unit standard deviation, which means the attribute becomes zero and the resultant distribution has a unit standard deviation. Mathematically, we can calculate the standardization by subtracting the feature value from the mean and dividing it by standard deviation.

Hence, standardization can be expressed as follows:

Normalization in Machine Learning

Here, **µ** represents the mean of feature value, and **σ** represents the standard deviation of feature values.

However, unlike Min-Max scaling technique, feature values are not restricted to a specific range in the standardization technique.

This technique is helpful for various machine learning algorithms that use distance measures such as **KNN, K-means clustering, and Principal component analysis**, etc. Further, it is also important that the model is built on assumptions and data is normally distributed.

## Difference between Normalization and Standardization

|  |  |
| --- | --- |
| **Normalization** | **Standardization** |
| This technique uses minimum and max values for scaling of model. | This technique uses mean and standard deviation for scaling of model. |
| It is helpful when features are of different scales. | It is helpful when the mean of a variable is set to 0 and the standard deviation is set to 1. |
| Scales values ranges between [0, 1] or [-1, 1]. | Scale values are not restricted to a specific range. |
| It got affected by outliers. | It is comparatively less affected by outliers. |
| Scikit-Learn provides a transformer called MinMaxScaler for Normalization. | Scikit-Learn provides a transformer called StandardScaler for Normalization. |
| It is also called Scaling normalization. | It is known as Z-score normalization. |
| It is useful when feature distribution is unknown. | It is useful when feature distribution is normal. |

## When to use Normalization or Standardization?

Which is suitable for our machine learning model, Normalization or Standardization? This is probably a big confusion among all data scientists as well as machine learning engineers. Although both terms have the almost same meaning choice of using normalization or standardization will depend on your problem and the algorithm you are using in models.

1. Normalization is a transformation technique that helps to improve the performance as well as the accuracy of your model better. Normalization of a machine learning model is useful when you don't know feature distribution exactly. In other words, the feature distribution of data does not follow a **Gaussian** (bell curve) distribution. Normalization must have an abounding range, so if you have outliers in data, they will be affected by Normalization.

Further, it is also useful for data having variable scaling techniques such as **KNN, artificial neural network**s. Hence, you can't use assumptions for the distribution of data.

2. Standardization in the machine learning model is useful when you are exactly aware of the feature distribution of data or, in other words, your data follows a Gaussian distribution. However, this does not have to be necessarily true. Unlike Normalization, Standardization does not necessarily have a bounding range, so if you have outliers in your data, they will not be affected by Standardization.

Further, it is also useful when data has variable dimensions and techniques such as **linear regression, logistic regression, and linear discriminant analysis**.

**Example:** Let's understand an experiment where we have a dataset having two attributes, i.e., age and salary. Where the age ranges from 0 to 80 years old, and the income varies from 0 to 75,000 dollars or more. Income is assumed to be 1,000 times that of age. As a result, the ranges of these two attributes are much different from one another.

Because of its bigger value, the attributed income will organically influence the conclusion more when we undertake further analysis, such as multivariate linear regression. However, this does not necessarily imply that it is a better predictor. As a result, we normalize the data so that all of the variables are in the same range.

Further, it is also helpful for the prediction of credit risk scores where normalization is applied to all numeric data except the class column. It uses the **tanh transformation** technique, which converts all numeric features into values of range between 0 to 1.

8. What is IMAGE AUGMENTATION and how does it work?

**What is Data Augmentation?**

Data augmentation is a technique of artificially increasing the training set by creating modified copies of a dataset using existing data. It includes making minor changes to the dataset or using deep learning to generate new data points.

**Augmented vs. Synthetic data**

**Augmented data** is driven from original data with some minor changes. In the case of image augmentation, we make geometric and color space transformations (flipping, resizing, cropping, brightness, contrast) to increase the size and diversity of the training set.

**Synthetic data**is generated artificially without using the original dataset. It often uses DNNs (Deep Neural Networks) and GANs (Generative Adversarial Networks) to generate synthetic data.

**Note**: the augmentation techniques are not limited to images. You can augment audio, video, text, and other types of data too.

**When Should You Use Data Augmentation?**

1. To prevent models from overfitting.
2. The initial training set is too small.
3. To improve the model accuracy.
4. To Reduce the operational cost of labeling and cleaning the raw dataset.

**Limitations of Data Augmentation**

* The biases in the original dataset persist in the augmented data.
* Quality assurance for data augmentation is expensive.
* Research and development are required to build a system with advanced applications. For example, generating high-resolution images using GANs can be challenging.
* Finding an effective data augmentation approach can be challenging.

**Become a ML Scientist**

**Master Python skills to become a machine learning scientist**

[**Start Learning for Free**](https://www.datacamp.com/tracks/machine-learning-scientist-with-python)

**Data Augmentation Techniques**

In this section, we will learn about audio, text, image, and advanced data augmentation techniques.

**Audio Data Augmentation**

1. **Noise injection**: add gaussian or random noise to the audio dataset to improve the model performance.
2. **Shifting**: shift audio left (fast forward) or right with random seconds.
3. **Changing the speed**: stretches times series by a fixed rate.
4. **Changing the pitch**: randomly change the pitch of the audio.

**Text Data Augmentation**

1. **Word or sentence shuffling**: randomly changing the position of a word or sentence.
2. **Word replacement**: replace words with synonyms.
3. **Syntax-tree manipulation**: paraphrase the sentence using the same word.
4. **Random word insertion**: inserts words at random.
5. **Random word deletion**: deletes words at random.

**Image Augmentation**

***Learn more about image transformation and manipulation with hands-on exercises in our***[***Image Processing with Python skill track***](https://www.datacamp.com/tracks/image-processing)***.***

1. **Geometric transformations**: randomly flip, crop, rotate, stretch, and zoom images. You need to be careful about applying multiple transformations on the same images, as this can reduce model performance.
2. **Color space transformations**: randomly change RGB color channels, contrast, and brightness.
3. **Kernel filters**: randomly change the sharpness or blurring of the image.
4. **Random erasing**: delete some part of the initial image.
5. **Mixing images**: blending and mixing multiple images.

**Advanced Techniques**

1. **Generative adversarial networks** (GANs): used to generate new data points or images. It does not require existing data to generate synthetic data.
2. **Neural Style Transfer**: a series of convolutional layers trained to deconstruct images and separate context and style.

**Data Augmentation Applications**

Data augmentation can apply to all machine learning applications where acquiring quality data is challenging. Furthermore, it can help improve model robustness and performance across all fields of study.

**Healthcare**

Acquiring and labeling medical imaging datasets is time-consuming and expensive. You also need a subject matter expert to validate the dataset before performing data analysis. Using geometric and other transformations can help you train robust and accurate machine-learning models.

For example, in the case of Pneumonia Classification, you can use random cropping, zooming, stretching, and color space transformation to improve the model performance. However, you need to be careful about certain augmentations as they can result in opposite results. For example, random rotation and reflection along the x-axis are not recommended for the X-ray imaging dataset.

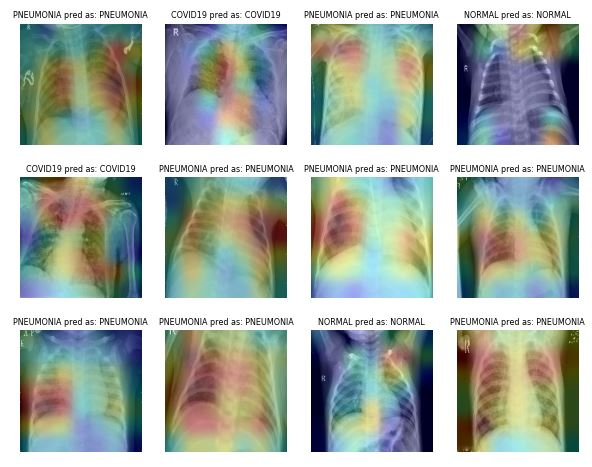


Image from [**ibrahimsobh.github.io**](https://ibrahimsobh.github.io/kaggle-COVID19-Classification/) | kaggle-COVID19-Classification

**Self-Driving Cars**

There is limited data available on self-driving cars, and companies are using simulated environments to generate synthetic data using reinforcement learning. It can help you train and test machine learning applications where data security is an issue.

Image by [**David Silver**](https://medium.com/self-driving-cars/autonomous-visualization-system-from-uber-atg-50957e7101f0) | Autonomous Visualization System from Uber ATG

The possibilities of augmented data as a simulation are endless, as it can be used to generate real-world scenarios.

**Natural Language Processing**

Text data augmentation is generally used in situations with limited quality data, and improving the performance metric takes priority. You can apply synonym augmentation, word embedding, character swap, and random insertion and deletion. These techniques are also valuable for low-resource languages.

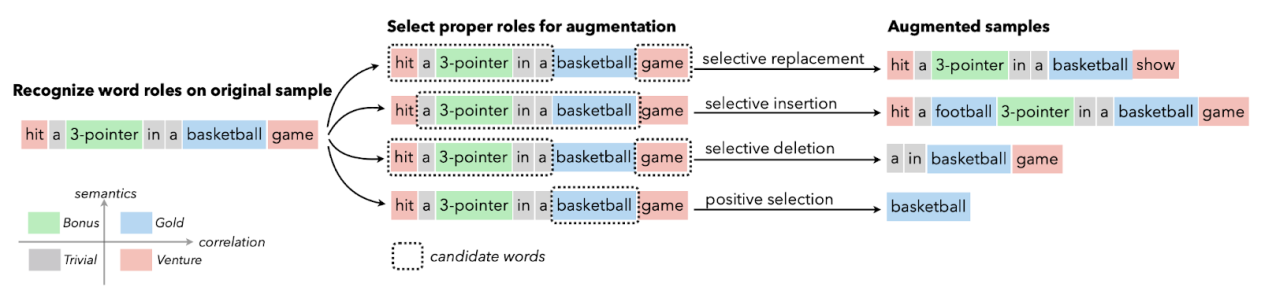


Image from [**Papers With Code**](https://paperswithcode.com/paper/selective-text-augmentation-with-word-roles) | Selective Text Augmentation with Word Roles for Low-Resource Text Classification.

Researchers use text augmentation for the language models in high error recognition scenarios, sequence-to-sequence data generation, and text classification.

**Automatic Speech Recognition**

In sound classification and speech recognition, data augmentation works wonders. It improves the model performance even on low-resource languages.

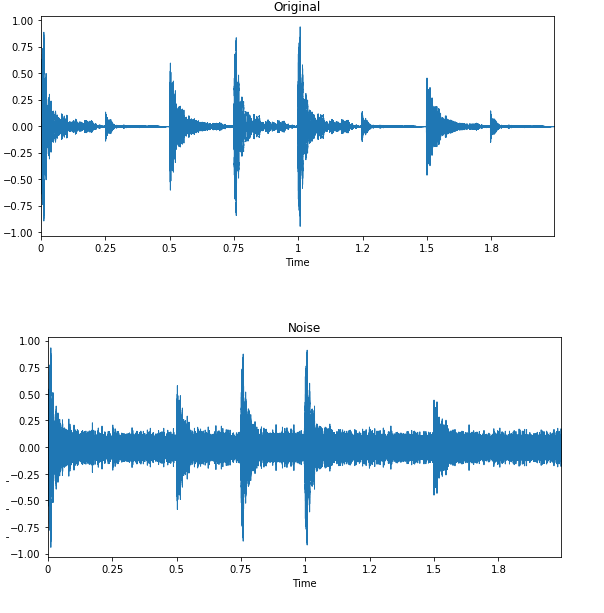


Image by [**Edward Ma**](https://medium.com/@makcedward/data-augmentation-for-audio-76912b01fdf6) | Noise Injection

The random noise injection, shifting, and changing the pitch can help you produce state-of-the-art speech-to-text models. You can also use GANs to generate realistic sounds for a particular application.

**Data Augmentation with Keras and TensorFlow**

In this tutorial, we are going to learn how to augment image data using Keras and Tensorflow. Furthermore, you will learn how to use your augmented data to train a simple binary classifier. The code mentioned below is the modified version of **[TensorFlow’s official example](https://www.tensorflow.org/tutorials/images/data_augmentation" \t "_blank)**.

9. What is DECLINE IN LEARNING RATE?

# Learning Rate Decay

Imagine you’re looking for a coin you dropped in a big room. At first, you take big steps, covering a lot of ground quickly. But as you get closer to the coin, you take tinier steps to look more precisely. This is similar to how learning rate decay works in machine learning.

In training a machine learning model, the “learning rate” decides how much we adjust the model in response to the error it made. Start with a high learning rate, and the model might learn quickly, but it can overshoot and miss the best solution. Start too low, and it might be too slow or get stuck. So, instead of keeping the learning rate constant, we gradually reduce it. This method is called “learning rate decay.” We start off taking big steps (high learning rate) when we’re far from the best solution. But as we get closer, we reduce the learning rate, taking smaller steps, and ensuring we don’t miss the optimal solution. This approach helps the model train faster and more accurately.

There are various ways to reduce the learning rate: some reduce it gradually over time, while others drop it sharply after a set number of training rounds. The key is to find a balance that lets the model learn efficiently without missing the best possible solution.

## Learning Rate Decay

[Learning rate decay](https://www.geeksforgeeks.org/impact-of-learning-rate-on-a-model/)is a technique used in [machine learning](https://www.geeksforgeeks.org/machine-learning/) models, especially [deep neural networks](https://www.geeksforgeeks.org/deep-learning-tutorial/). It is sometimes referred to as learning rate scheduling or learning rate annealing. Throughout the training phase, it entails gradually lowering the learning rate. Learning rate decay is used to gradually adjust the learning rate, usually by lowering it, to facilitate the optimization algorithm’s more rapid convergence to a better solution. This method tackles problems that are frequently linked to a fixed learning rate, such as oscillations and sluggish convergence.

Learning rate decay can be accomplished by a variety of techniques, such as step decay, exponential decay, and 1/t decay. Degradation strategy selection is based on the particular challenge and architecture. When training deep learning models, learning rate decay is a crucial hyperparameter that, when used properly, can result in faster training, better convergence, and increased model performance.

## How Learning Rate Decay works

Learning rate decay is like driving a car towards a parking spot. At first, you drive fast to reach the spot quickly. As you get closer, you slow down to park accurately. In machine learning, the learning rate determines how much the model changes based on the mistakes it makes. If it’s too high, the model might miss the best fit; too low, and it’s too slow. Learning rate decay starts with a higher learning rate, letting the model learn fast. As training progresses, the rate gradually decreases, making the model adjustments more precise. This ensures the model finds a good solution efficiently. Different methods reduce the rate in various ways, either stepwise or smoothly, to optimize the training process.

### Mathematical representation of Learning rate decay

A basic learning rate decay plan can be mathematically represented as follows:

Assume that the starting learning rate is and that the learning rate at epoch t is .

A typical decay schedule for learning rates is based on a constant decay rate , where , applied at regular intervals (e.g., every n epochs):

Where,

* is the learning rate at epoch t.
* is the initial learning rate at the start of training.
* is the fixed decay rate, typically a small positive value, such as 0.1 or 0.01.
* t is the current epoch during training.
* The learning ratedecreases as t increases, leading to smaller step size as training progresses.

The learning rate is decreased by a percentage of its previous value at each epoch in this formula, which depicts a basic learning rate decay schedule. A timetable like this facilitates the optimization process by enabling the model to converge more quickly at first, then fine-tuning in smaller increments as it gets closer to a local minimum.

### Basic decay schedules

In order to enhance the convergence of machine learning models, learning rate decay schedules are utilized to gradually lower the learning rate during training. Here are a few simple schedules for learning rate decay:

* **Step Decay**: In [step decay](https://www.geeksforgeeks.org/rate-of-decay-formula/), after a predetermined number of training epochs, the learning rate is decreased by a specified factor (decay rate). The mathematical formula for step decay is:
* **Exponential Decay**: The learning rate is progressively decreased over time by [exponential decay](https://www.geeksforgeeks.org/exponential-decay-formula/). At each epoch, a factor is used to adjust the learning rate. The mathematical formula for Exponential decay is:
* **Inverse Time Decay**: A factor inversely proportional to the number of epochs is used to reduce the learning rate through inverse decay. The mathematical formula for Inverse Time decay is:
* **Polynomial Decay**: When a polynomial function, usually a power of the epoch number, is followed, [polynomial decay](https://www.geeksforgeeks.org/exponential-functions/) lowers the learning rate.The mathematical formula for Polynomial decay is:

In simple words, these schedules adjust the learning rate during training. They help in starting with big steps and taking smaller steps as we get closer to the best solution, ensuring efficiency and precision.

## ****Steps Needed**** to implement Learning Rate Decay

* **Set Initial Learning Rate**: Start by establishing a base learning rate. It shouldn’t be too high to cause drastic updates, nor too low to stall the learning process.
* **Choose a Decay Method**: Common methods include exponential decay, step decay, or inverse time decay. The choice depends on your specific machine learning problem.
* **Implement the Decay**: Apply the chosen decay method after a set number of epochs, or based on the performance of the model.
* **Monitor and Adjust**: Keep an eye on the model’s performance. If it’s not improving, you might need to adjust the decay rate or the method.

## Implementing Learning Rate Decay

Certainly, let’s see a simple example of implementing learning rate decay using[TensorFlow](https://www.geeksforgeeks.org/introduction-to-tensorflow/). In this script, we’ll use a basic neural network model for the classification task on the MNIST dataset, which is a dataset of handwritten digits.

10. What does EARLY STOPPING CRITERIA mean?

# Regularization by Early Stopping

**Regularization**is a kind of regression where the learning algorithms are modified, to reduce overfitting. This may incur a higher bias but will lead to lower variance when compared to non-regularized models i.e. increases generalization of the training algorithm.

## Why Regularisation is needed?

In a general learning algorithm, the dataset is divided into a **training set** and a **test set.** After each epoch of the algorithm, the parameters are updated accordingly after understanding the dataset. Finally, this trained model is applied to the test set.

Generally, the training set error will be less compared to the test set error. This is because of overfitting whereby the algorithm memorizes the training data and produces the right results on the training set. So, the model becomes highly exclusive to the training set and fails to produce accurate results for other datasets including the test set.

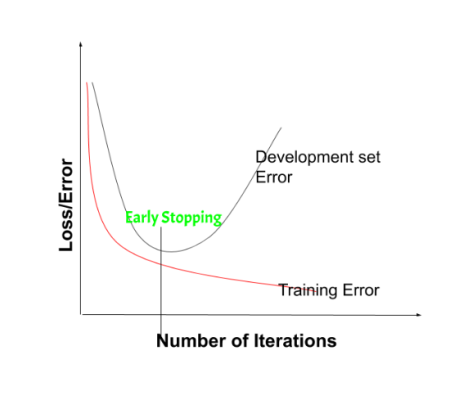
Regularization techniques are used in such situations to **reduce overfitting** and**increase the model’s performance** on any general dataset.

*To understand about underfitting and overfitting in machine learning in detail follow the link below-*

## What is Early Stopping?

In [Regularization](https://www.geeksforgeeks.org/regularization-in-machine-learning/)by Early Stopping, we stop training the model when the performance on the validation set is getting worse- increasing loss decreasing accuracy, or poorer scores of the scoring metric. By plotting the error on the training dataset and the validation dataset together, both the errors decrease with a number of iterations until the point where the model starts to overfit. After this point, the training error still decreases but the validation error increases.

So, even if training is continued after this point, early stopping essentially returns the set of parameters that were used at this point and so is equivalent to stopping training at that point. So, the final parameters returned will enable the model to have low variance and better generalization. The model at the time the training is stopped will have a better generalization performance than the model with the least training error.

on the validation set is getting worse- increasing loss or decreasing accuracy or poorer scores Early stopping can be thought of as **implicit regularization**, contrary to regularization via weight decay. This method is also efficient since it requires less amount of training data, which is not always available. Due to this fact, early stopping requires lesser time for training compared to other regularization methods. Repeating the early stopping process many times may result in the model overfitting the validation dataset, just as similar as overfitting occurs in the case of training data.

The number of iterations(i.e. epoch) taken to train the model can be considered a **hyperparameter**. Then the model has to find an optimum value for this hyperparameter (by hyperparameter tuning) for the best performance of the learning model.

***Tip:****The downside of early stopping are as follows:*

*By stopping early , we can’t able to optimize Cost function(J) much for the training set. So, we use a different concept Known as****Orthogonalisation****is used.*

### Benefits of Early Stopping:

* Helps in reducing overfitting
* It improves generalisation
* It requires less amount of training data
* Takes less time compared to other regularisation models
* It is simple to implement

### Limitations of Early Stopping:

* If the model stops too early, there might be risk of underfitting
* It may not be beneficial for all types of models
* If validation set is not chosen properly, it may not lead to the most optimal stopping