**REFERENCES:**

[1] G. Cybenko, “Approximation by superposition of a sigmoidal function,” Math. ControlSignals Systems, 1989.

[2] F. Chollet. *Deep Learning with Python*. Manning, Shelter Island, NY, USA, 1st edition, 2017

[3] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning (Adaptive Computation and Machine Learning Series*. MIT Press, Cambridge, MA, USA, 2016.

[4] <https://www.ibm.com/cloud/learn/gradient-descent>

[5] Sebastian Ruder. *An overview of gradient descent optimization algorithms*

[6] <https://builtin.com/data-science/gradient-descent>

*[7] P. Wang, P. Chen, Y. Yuan, D. Liu, Z. Huang, X. Hou, and G. Cottrell. Understanding convolution for semantic segmentation. In IEEE Winter Conference on Applications of Computer Vision (WACV), pages 1451 - 1460, March 2018.*

*[8] Kunihiko Fukushima, “Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position,”* Biological cybernetics*, vol. 36, no. 4, pp. 193–202, 1980.*

*[9]* [*https://cs231n.github.io/convolutional-networks/*](https://cs231n.github.io/convolutional-networks/)

*[10] Jonathan Long, Evan Shelhamer, and Trevor Darrell, “Fully convolutional networks for semantic segmentation,” in Proceedings of the IEEE conference on computer vision and pattern recognition, 2015, pp. 3431–3440.*

*[11] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi, “You only look once: Unified, real-time object detection,” in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 779–788.*

*[12] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” 2012.*

*[13] J. Han and C. Moraga, “The influence of the sigmoid function parameters on the speed of backpropagation learning,” in Natural to Artificial Neural Computation. IWANN 1995. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1995, pp. 195–201.*

*[14] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” Nature, vol. 521, no. 7553, pp. 436–444, 2015.*

*[15] B. Karlik and A. Vehbi, “Performance Analysis of Various Activation Functions in Generalized MLP Architectures of Neural Networks,” International Journal of Artificial Intelligence and Expert Systems (IJAE), vol. 1, no. 4, pp. 111–122, 2011.*

*[16] Y. N. Dauphin, A. Fan, M. Auli, and D. Grangier, “Language Modeling with Gated Convolutional Networks,” arXiv, 2017.*

*[17] V. Nair and G. E. Hinton, “Rectified linear units improve restricted boltzmann machines,” Haifa, 2010, pp. 807–814.*

*[18] M. D. Zeiler, M. Ranzato, R. Monga, M. Mao, K. Yang, Q. V. Le, and G. E. Hinton, “On rectified linear units for speech processing,” in International Conference on Acoustics, Speech and Signal Processing. IEEE, 2013, pp. 3517–3521*

*[19] X. Glorot, A. Bordes, and Y. Bengio, “Deep Sparse Rectifier Neural Networks,” in International Conference on Machine Learning, 2011*

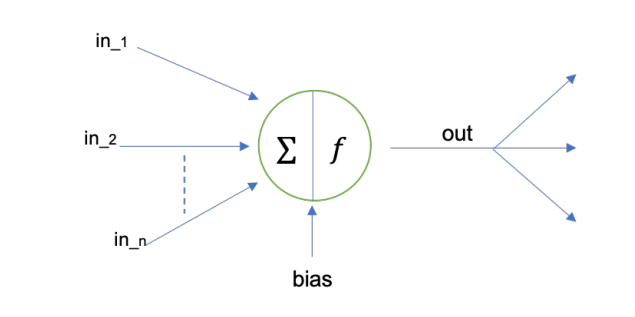
**Background Knowledge**

1. **Artificial Neural Networks (ANN)**

The concept of Artificial Neural Networks was built on the inspire of human brain biology characteristics, Cybenko [1] proved that in a neural network a single hidden layer containing a finite number of neurons is capable of approximating any continuous function to any desired precision. it included a group of machine learning models that are able to “learn” to execute or perform some specific tasks based on supplied examples. In general, these networks were created by a group of units or nodes connected each other like a simulating version of neurons in human brain, which called artificial neurons.

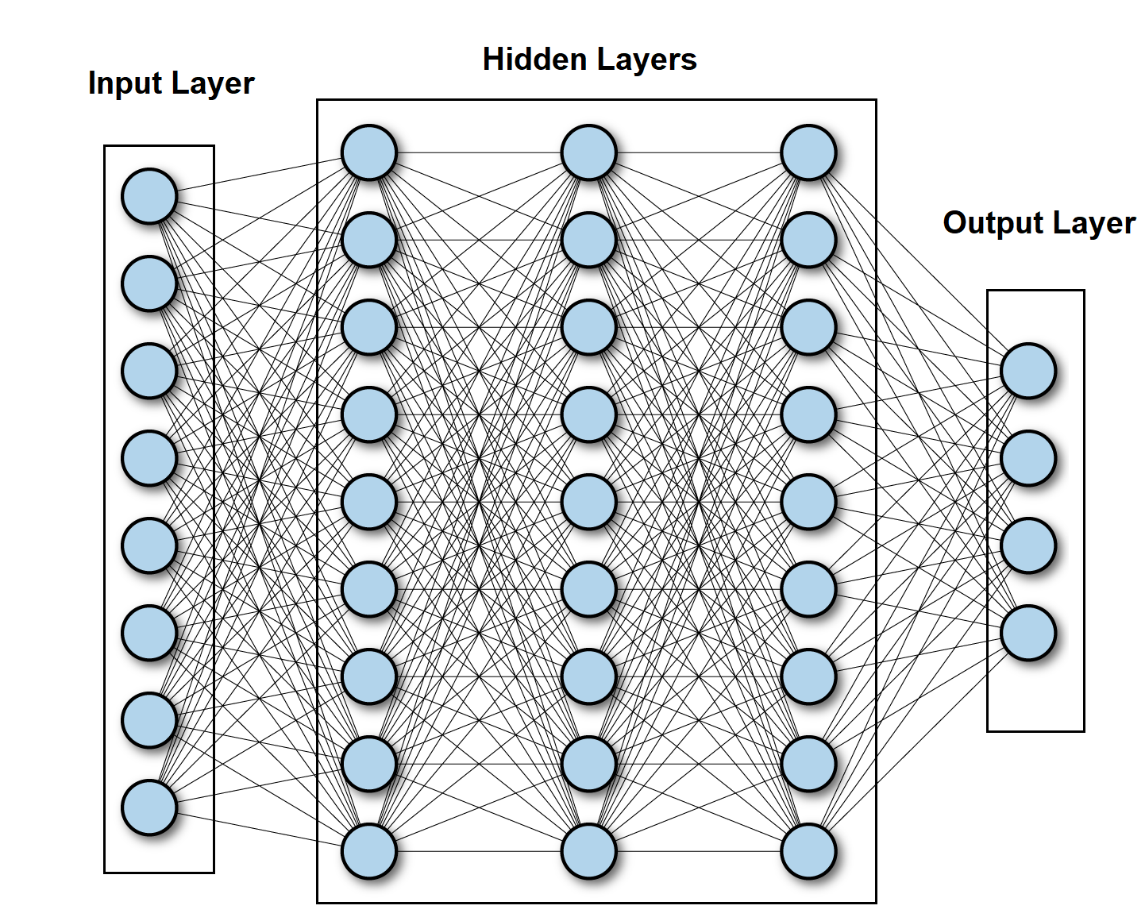
* 1. **Neuron**

Analog electrical signals flow inside biological neurons, while digital values present as the signals for artificial neurons. Each artificial neuron has an output being computed by using a function of the sum of the input signal values with generally added bias, the connection between the neurons as known as edges, in each connection contains an adjustable model parameter named weight that can be “learned” through a process called “model training”. The strength of the synaptic will determines the interaction’s level between the neurons, this function combines the neurons, bias and weights in the connections called “activation function”, and there are several activation functions that mean there is not only one fixed activation function for ANN, and we can choose the activation function for the network’s output. These functions will be mentioned later in this thesis.



* 1. **Layer**

In typical, all neurons are linked directly in network, on the other hand, they are grouped into subsets of neurons known as layers, which are generally connected in a sequential way based on the activation function in every node to compute nodes’ outputs from node’s inputs. In detail, the neurons in a layer are connected to a set of neurons in previous layer and to another one in next layer. Because of the modeling of ANNs as a connection of a neurons’ structure in layer, neurons’ outputs at a certain layer become neurons’ inputs at the following layer, which networks are understood as feedforward neural networks. There are no loops in present layer, therefore the inputs are always sent forward. As a result, the neurons into a layer cannot be connected with each other. When all the neurons in a layer are connected to all other ones in previous layer and following layer, it is called “Fully Connected Layers”, this state of the connection is kept in whole network as a series of fully connected layers that will be called “Fully Connected Neural Network”.



There is an existing of 3 kinds of layer inside a network, they are input layer, hidden layers and output layer.

Input layer represent for the input of the network, the number of nodes as known as neurons are size of the input that call “input dimension”. The hidden layers lay between input lay and output layer, there are so many hidden layers in a network, however there is only one input layer and output layer, and the output layer represent for the output of the network. In addition, the bias contains in input and output layer, not in output layer. Although simple decisions are made at the first layer based on the input, sophisticated judgments are already made in the second layer depending on decisions made in the first layer. As one moves deeper into the network, more difficult and abstract judgments emerge. The phrase deep neural networks refer to a network with numerous hidden layers.

* 1. **Forward and Backward (back propagation) steps**

As the mentioning before, the signals travel from input layer to output layer traversing hidden layers, which operations performed over the values received from the nodes in previous layers (in all layers but in the input layer) and whole activation function results are sent forwards to the next layer till the output layer. However, it is not the end of process. After getting the result of output layer, there is a backwards named back-propagation applying in here as known as training step for ANN. The algorithm is used to effectively train a neural network through a technique called chain rule. In simple terms, after each forward pass through a network, backpropagation performs a backward pass while adjusting the model’s parameters (weights and biases), which helps minimize the error for network’s result, it is called “loss function” or “cost function” which is the evaluation between predicted output and expected output at the final point of forward step. Basically, this function calculates the gradient of the loss function to each parameter in network respectively that means it work from the top layer (output layer) to the bottom layer (input layer) to calculate the contribution of each parameter in that value [2], which update the weights until reaching the minimum values. Furthermore, the key point of calculating the minimum value for loss function is gradient descent, which is a process of determining the modifications needed to minimize the loss and optimize the model to a specific evaluation measure by calculating the derivative of the loss function [3].

* 1. **Gradient Descent:**

Gradient descent is an optimization algorithm which is commonly-used to train machine learning models and neural networks [4]. It is built on a convex function and iteratively changes its parameters to minimize a given function to its local minimum. On the other words, Gradient descent begins by defining the initialized parameter values, and then iteratively adjusts the values to minimize the given cost-function. Deeply, a gradient is a function’s derivative with several input variables. In terms of mathematics, a gradient is known as the function’s slope, and it simply measures the change in all weights in relation to the change in error. The steeper the slope and the faster a model can learn, the higher the gradient. However, if the slope is zero, the model will stop learning. A gradient is a partial derivative with regard to its inputs.

Imaging that a blindfolded man wants to hike down to the bottom of a valley from the top of a hill with as few steps as possible. He might begin going down the hill by taking large steps in the sharpest direction, which he can do as long as he is not close to the bottom. However, as he gets closer to the bottom, his steps will become smaller and smaller in order to prevent overshooting it. The gradient can be used to mathematically characterize this process. The learning rate η determines the size of the steps we take to reach a (local) minimum. In other words, we follow the direction of the slope of the surface created by the objective function downhill until we reach a valley [5].

To reach the local minimum for gradient descent, we should choose a relevant value for the learning rate, which is not both too low and too high. This is very significant because if the steps are too large, it could fail to reach the local minimum since it bounces between the convex function of gradient descent. Besides, gradient descent will eventually reach the local minimum if we set the learning rate to a very low value, but it may take a lot of time [6]. So, the key point is choosing an optimal learning rate that helps model converge to the minimum value without spending much time.



Gradient descent has three forms that differ in the amount of data used to compute the gradient of the objective function. We choose between the accuracy of the parameter update and the time it takes to complete an update based on the amount of data.

* + 1. **Batch Gradient Descent**

Batch gradient descent calculates the sum of the error for each point in a training set and updates the model only after all training instances have been evaluated. This procedure is known as a training epoch. Though this batching supplies a computation efficiency, it can still take a long time to execute big training datasets because it must still keep all of the data in memory [5]. Batch gradient descent also frequently yields a stable error gradient and convergence, although that convergence point isn't always the best, locating the local minimum rather than the global minimum.

* + 1. **Stochastic Gradient Descent**

Stochastic gradient descent (SGD) performs a training epoch for each example in the dataset and updates the parameters of each training example at a time. They are easier to remember because you only need to hold one training example [5]. In spite of the fact that these frequent updates provide more detail and speed, they can lead in computational efficiency losses when it be compared to batch gradient descent. Despite of ability of noisy gradients production, these frequent updates can also aid in escape the local minimum and locating the global one.

* + 1. **Mini-batch Gradient Descent**

Mini-batch gradient descent combines batch gradient descent with stochastic gradient descent ideas. It divides the training dataset into small batches and updates each of those batches. This method strikes a balance between batch gradient descent's computing efficiency and stochastic gradient descent's speed [5].

Besides, gradient descent has its own set of challenges such as: Local minima and saddle points, vanishing and exploding gradients.

* Local minima resemble global minima in shape with the slope of the cost function increasing on each side of the current location.
* Saddle points happen when the negative gradient exists just on one side of the point, reaching a local maximum on one side and a local minimum on the other. Its name was inspired by horse's saddle;
* Vanishing gradients happens when the gradient is too small. The gradient continues to shrink as we move backwards during backpropagation, leading earlier levels of the network to learn more slowly than later layers. When this occurs, the weight parameters are updated until they become inconsequential, i.e., 0, resulting in an algorithm that is no longer learning.
* Exploding gradients occurs when the gradient becomes too big, resulting in an unstable model. In this instance, the model weights will get excessively huge and will be represented as “NaN” finally. One approach to this problem is to use a dimensionality reduction technique, which can help to reduce model complexity.

1. **Convolutional Neural Network**

Convolutional Neural Networks (CNNs), like ANNs, being inspired by the structure of the human visual cortex [8]. CNNs, which are a type of neural network typically utilized for computer vision tasks in deep learning, include convolutional layers, pooling layers and fully connected layers. These networks are identical to traditional neural networks, except that instead of generic matrix multiplication, a convolution kernel is used in one or more of their layers [3]. These networks have the role of reduction the images into a form which is easier to process, without losing features which are critical for getting a good prediction. Moreover, CNNs have ability to take image tensors as an input, identify which image characteristics or features are essential for classification or differentiation, and output these classifications [1, 7].

Therefore, the basic structure of a Convolutional Network in order starting from a Convolutional Layer or so-called convolution process (do not count the initial input layer), has the effect of picking out the 2-dimensional characteristics of the image input. Specifically, the features of the input original image are passed through the “convolutional layer”, at which there is a mechanism built up by matrix multiplication between the input image known as a matrix and one or more matrices with a certain size (usually smaller than the input matrix) being called “filters”, this matrix multiplication will be shifted by rows and columns with a certain rule that will give generate one or more new matrices based on the number of filter layers, the product from this process is called feature maps which contain the attributes of the original input image.

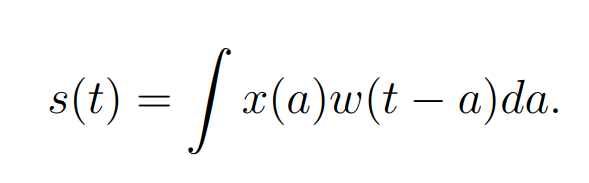
Followed by a pooling layer to help the network synthesize the features of the image in a deeply specific way (max pooling) or more objectively general (average pooling or mean pooling) from the output of the previous layer. Next, the size of the image has been significantly reduced that is very meaningful for memory in process of training model. Although the dimensions are decreased, the number of channels or feature maps are still kept unchanged. So, this process is as a compositing to reduce the size of the input image feature in length and width. At last, when the input size has been reduced to a reasonable value, the 2-dimensional image will be converted to a one-dimensional (one-dimensional vector) or known as "flatten", at which this process is implemented as a Fully Connected Layer, the number of units in this layer will correspond to the number of classes that defined for the network to distinguish as known as the final output of the network with a specific activation function for specific final result.

In general, using CNNs because of their capacity to learn translation-invariant patterns as well as spatial hierarchies utilizing appropriate filters. Learning translation invariant patterns means that once a pattern is learned in one area of a given image, the network will be able to recognize it anywhere else in the image. This improves image processing efficiency by requiring fewer training samples to learn generalizable representations [1].

Spatial hierarchies show how earlier layers in the network can learn tiny local patterns and grow into bigger patterns composed of these tiny patterns in the next layers, which also improves the network's ability to learn complicated and abstract concepts of visualization. In addition, this also allows these networks to deliver class predictions with far higher accuracy than vectorizing a complex image with pixel dependencies throughout. Therefore, CNNs have to be used for any image classification task so as to track these crucial properties and correlations between features [1].

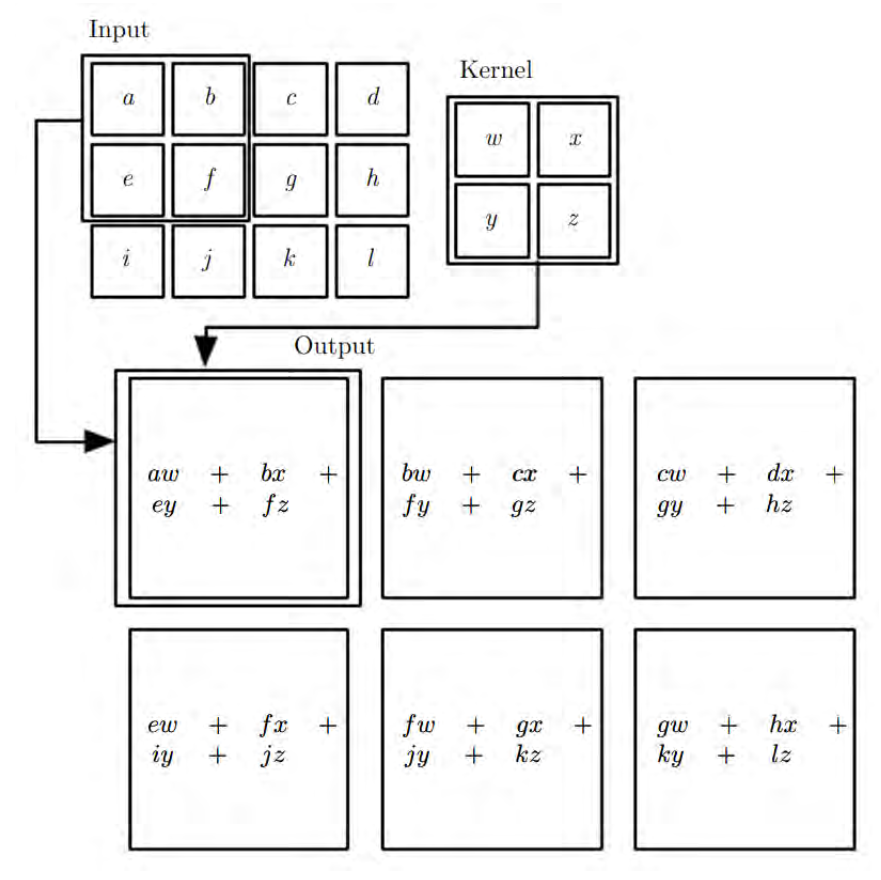
* 1. **Convolution Layer**

Convolution is an operation that produces a new function by combining two functions of a real-valued argument. Let s(t) is the output estimate function based on time t, x(a) is the age-based input position function, and w(a) is the weighting function that prioritizes recent measurements. This is the general formula for convolution utilizing this defined function:

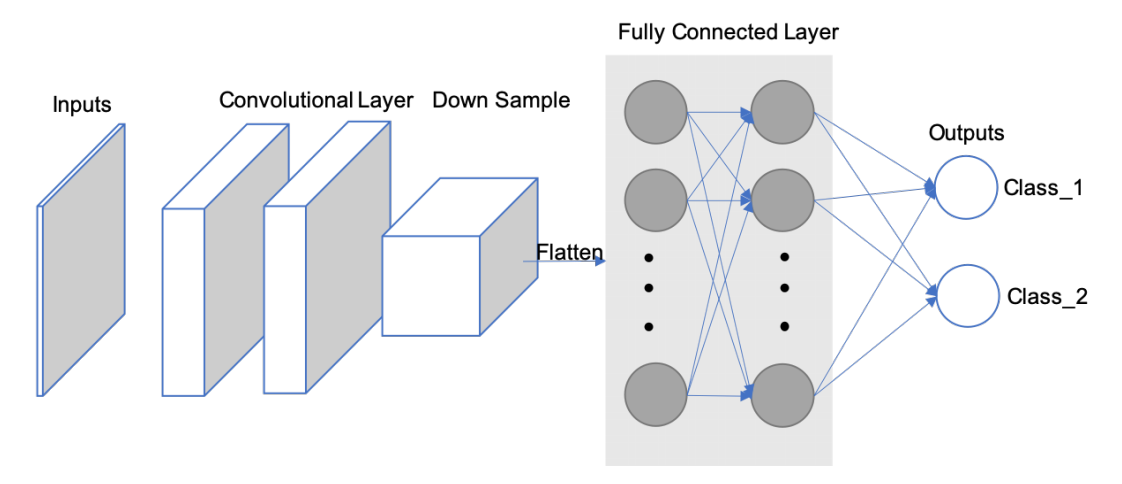


In detail, the input is the function x(a), the kernel is the weighting function w(a), and the output s(t) is the feature map in the CNN. the CNNs use convolutions that are performed over two-dimensional tensors that are made up of the input images' height, width, and channels of colour. These techniques take out patches and alter the input to build a feature map with varied depth. One of the two critical criteria for defining these convolutions is the size of these extracted patches. The depth of the output feature map is the second parameter to consider. This is the number of filters built by the layer that can encode diverse features of the input data.

2D Convolutions are computed by moving a square window of defined height and width across all pixels of the input feature map, which connect to each local region or receptive field of the image corresponding to the size of the applied filters. For images, the input feature map has three dimensions: height, width and colour bands, which commonly correspond to red, green, and blue. When these feature maps perform 2D convolution, two dimensional patches of surrounding features are created. These patches are then turned into a one-dimensional vector reflecting the output depth via a convolution kernel. After that, the vectors are reassembled spatially into a two-dimensional output map that corresponds to all points in the input map. The convolution process is depicted in the figure below.



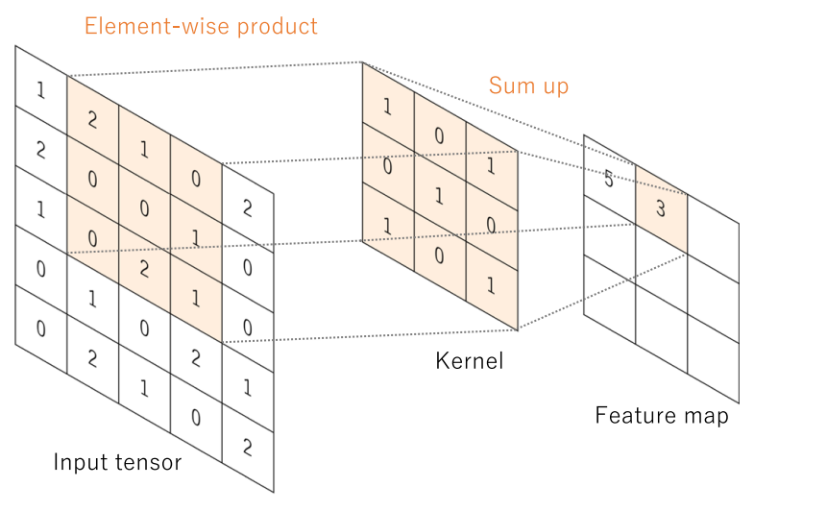
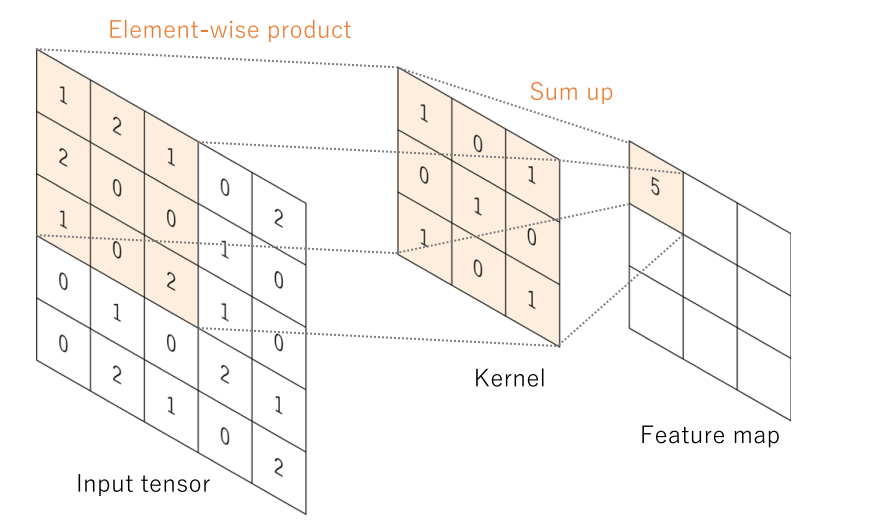
In other words, a feature map is the output of a convolutional layer, which is typically a three-dimensional tensor with dimensions width, height, and depth. To process inputs or feature maps, the convolutional layer employs a filter, similar to the filtering procedure in picture pre-processing. The primary distinction is that in standard image pre-processing, the filter weights (values) are hand-crafted, whereas in CNNs, during training, data helps they learn. Alternatively, the receptive field in the input feature map corresponds to the output of a convolutional layer, and the size of the receptive field is governed by the kernel size and dilation. It is not essential to flatten images while using CNNs. However, flattening is required to feed the features maps, being the outputs of convolutional layers, to the fully connected layers. Fully connected layers are the same as the ANNs mentioned above, however ANN is a catch-all term for all artificial neural networks, including CNNs. Fully connected layers understand the linear and non-linear correlations between extracting features and classifying each sample into many abstract classes. This figure shows a simple CNN architecture.

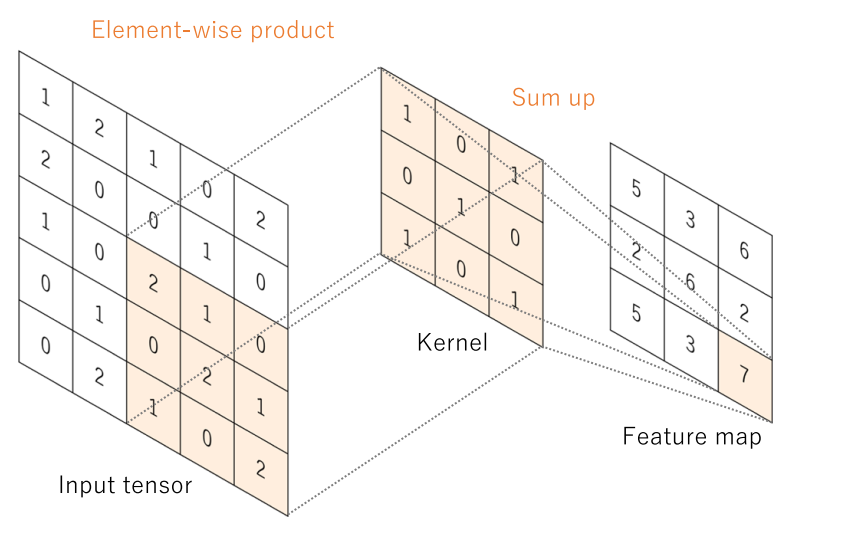


The size of the output volume is controlled by hyperparameters which are depth, stride, and zero-padding.

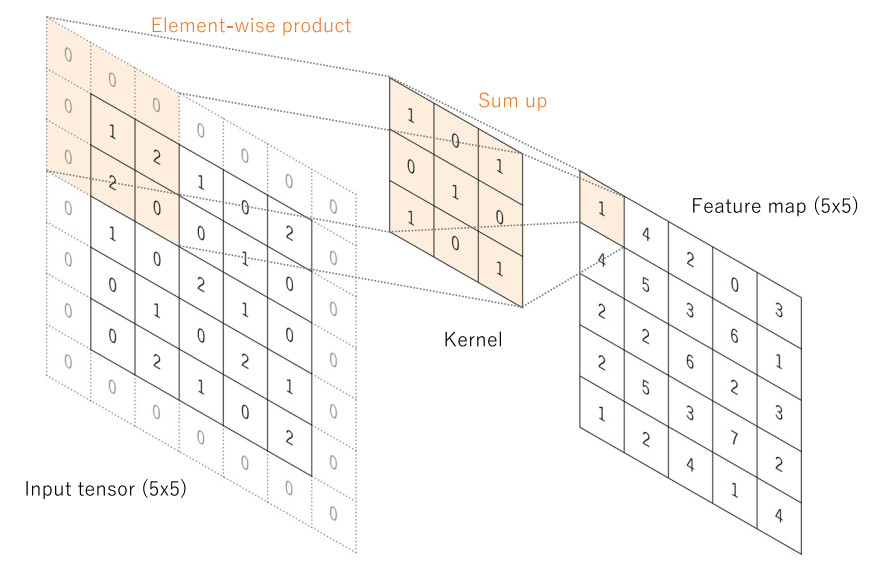
The first component of the output volume decision is depth, which is the number of filters desiring to utilize, each learning to search for something different in the input. For example, if the first Convolutional Layer receives the raw image as an input, distinct neurons along the depth dimension may fire in the presence of different oriented edges or colour blobs. A column of depth is a group of neurons being all staring at the same region of the input [9].

Stride aids in picture and video data compression tuning. The stride of the convolution is a characteristic which can contribute to a different output size in CNNs, which is a convolution operation parameter that defines the distance between patches derived from the input feature map. With a stride of 2, the output feature map's width and height are down-sampled by a factor of two with no padding. For example, if the stride of a neural network is set to 1, step by step, the filter will make a reflection onto the matrix of input, which move one pixel or one unit at a time from left to right in the row till the end of horizontal point and top to bottom in the column at the end of the matrix. Because the size of the filter influences the encoded output volume, stride is frequently a positive integer rather than a fraction or decimal.

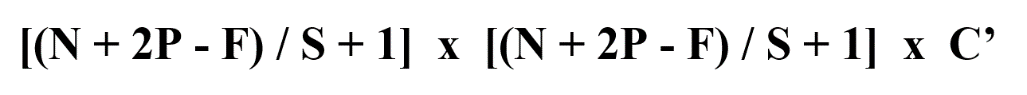




It is sometimes useful to pad the input volume with zeros around the border. This zero-size padding's is a hyperparameter. The great thing about zero padding is that it allows to tweak the spatial size of the output volumes, which most typically using it to precisely preserve the spatial size of the input volume so as to be same of the input and output width and height [9].



Combination of these components will give out the formula for calculating the size of Convolution layer:



The spatial dimension of the output volume may be calculated as a function of the input volume size ***(N)*** which represent for the dimension by width and height of input image, the receptive field size of the Conv Layer neurons ***(F)*** as the dimension of filter by width and height, the amount of zero padding placed the border for input image ***(P)***, the positive integer number of strides with which they are applied ***(S)***, and the number of filter ***(C’)***.

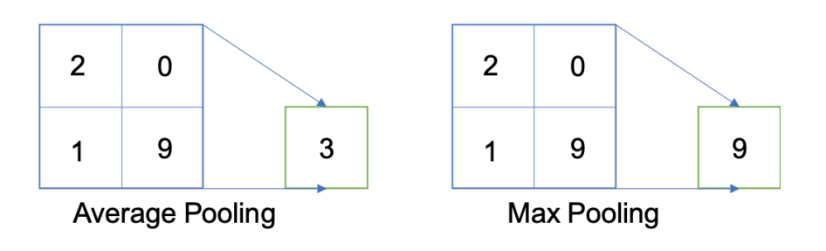
The results of a linear operation, such as convolution, are subsequently processed by a nonlinear activation function. Although smooth nonlinear functions such as the sigmoid or hyperbolic tangent (tanh) function were previously used because they are mathematical representations of biological neuron behaviour, the rectified linear unit (ReLU) function is now the most commonly used nonlinear activation function, which simply computes the function: f(x) = max (0, x)

In convolutional neural networks, rectified linear units (ReLUs) provide three key advantages:

* + The ReLUs efficiently transmit the gradient, reducing the risk of a vanishing gradient problem, which is prevalent in deep neural architectures.
  + The ReLUs set negative values to zero, so solving the problem of cancellation and resulting in a much sparser activation volume at its output. Sparsity is helpful for a variety of reasons; However, it is most commonly used to supply robustness to tiny adjustments in input such as noise.
  + The ReLUs that including just simple computations (primarily comparisons) and are thus substantially more efficient to implement in convolutional neural networks.
  1. **Pooling Layer**

There are also pooling layers between convolutional layers. They are used to down sample the feature maps and reduce the number of parameters. The later layers after feature extraction will need a large number of parameters because the depth is determined by the number of channels in the later layers often increasing exponentially. That increases the number of parameters and the amount of computation in the neural network. Therefore, to reduce the computational load we will need to reduce the dimensions of the input matrix block or reduce the number of layer units. Since each unit would be a representative result of applying a filter to find a specific feature, reducing the number of units would not be feasible. Reducing the input matrix block size by finding a representative value for each spatial region that the filter passes through will not change the main contours of the image but reduce the size of the image. Hence the matrix reduction process is applied.

Average pooling and max pooling are the two most commonly used types of pooling layers helping make this reduction purpose. As the suggestion of their names, average pooling takes out the average (mean) of the values on the feature map corresponding to a kernel size. Besides, max pooling calculates the maximum value in the neighbourhood corresponding to the kernel size position on the feature map. Both of them travel through the whole feature map by a stride size without overlapping local region position. In addition, the pooling layers do not have any parameters which need to be learned during training and the stride in the convolutional layer is able to also be used to replace the pooling layers. Figure below shows the difference between these two pooling types.



* 1. **Batch Normalization:**

Another critical component of a CNN architecture is batch normalization (BN). Although this is not a required component in CNN, it is a relatively important component with many benefits for training the model to reach the best results. BN allows faster training and stabilization of deep neural networks by stabilizing the distribution of layer inputs during training. This approach mainly involves Internal Covariate Shift (ICS). To improve model training, it is important to reduce ICS by controlling the means and variances of the input data layers. In detail, BN normalizes the feature maps at the output of a convolutional layer in the same way which it normalizes the input pictures. BN makes subtractions of the batch mean and does divisions of the feature map values by the batch standard deviation to ensure that the distribution of feature maps is the same. As a result, BN lowers value shift in hidden layers and BN allows you additional flexibility in the initialization of kernel weights. Furthermore, because of guarantee of BN which activation does not diverge to very large values, greater learning rates can be employed. In addition, there is a similar to a dropout layer which also has some regularization effects, because of adding noise to the feature maps. During training and evaluation, both the BN and dropout layers will behave differently.

When we would like the function of probability density (FPD) of the model to be closest the true real distribution of data as possible in deep learning. Unfortunately, in fact that we only have a narrow amount of training data at our disposal. As a result, the goal is to get the FPD of the model as near to the FPD of the train as possible by training on a dataset that replicates the domain of application in the real-world. A sufficiently big training dataset is required to get improved representation. When the training dataset is insufficiently large, there will be a significant difference in the error between the FPD of real and the FPD of train.

In other expressions, even if we produce a model FPD that is very close to the FPD of the train, it will not perform with the FPD of test skilfully, being a sample of the FPD of the real. This is referred to as the overfitting problem. Overfitting occurs when a model fails to fit subsequent observations. Dropout layer is applied to stay away from overfitting by randomly selecting some neurons to brush aside throughout the training process. A dropped-out node's incoming and outgoing edges are also eliminated. Thus, dropout not only reduces the problem of overfitting, but it also simultaneously speeds up training by eliminating training with all nodes.

* 1. **Full Connected Layer:**

Typically, the activation function applied to the last connected layer differs from the others. Each job necessitates the selection of an appropriate activation function. A soft-max function is used as an activation function in the multiclass classification job to normalize output real values from the final fully connected layer to target class probabilities, where each value ranges between 0 and 1 and all values will be made a sum to 1. Typical final layer activation function selections for various sorts of jobs.

In addition, the majority of the trainable parameters are located in fully connected layers in CNN. Fully connected layers can give out outputs that are categories for image classification or localization via bounding boxes [11]. Convolutional layer work can also be used as fully connected layers to execute the same duty more efficiently. Instead of a class, the result of an image segmentation job is a mask with the same resolution as the input pictures. We may use a 1x1 kernel as the output layer to reduce the depth to the same level as the target classes [10], giving the model pixel-wise classification capability.

Training a CNN consists of two steps: forward and backward. We feed the CNN with input information flow in the forward. To acquire the input for the following layer, feature maps are generated in each layer with predefined weights and biases. A loss value will be computed at the output layer by applying a loss function to the difference between the calculated output and the ground truth. The chain rule is utilized in the backward step to find out the gradients of loss value for each trainable parameter. After that, the gradients are utilized to adjust each parameter for the following iteration. This type of update is what allows CNNs to “learn”.

After processing enough iterations, at which the loss value converges to an acceptable small number that means it is approximately close to zero, the training process can be terminated. The loss function computes the difference or error between the outputs and the ground truth. Minimizing loss implies minimizing the difference between pdf(model) and pdf(train) when the loss function is applied as a guide. Moreover, the loss function will change depending on the application and objective. Because of its ability of quantify the similarity/difference between two distributions, cross-entropy is a widely used loss function.

Otherwise, there are several optimizer options to help you decide how to update the network parameters. Adam and Stochastic Gradient Descent (SGD) are two options. In general, SGD will produce better results while learning at a slow pace. It will take several iterations to achieve a satisfactory outcome. SGD has a fixed learning rate that does not vary during training. Adam, on the other hand, converges quicker since it employs a vector of learning rates that are adjusted as learning continues during training progresses. They will be mentioned more detail in the next sections of Optimizer.

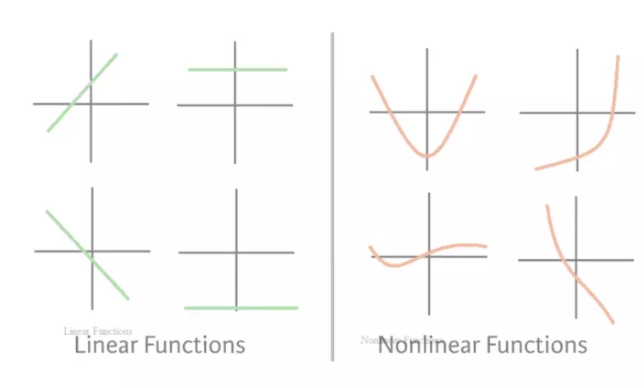
1. **Activation Functions:**

Activation functions (AFs) in Neural Network are a component which is very essential in Deep Learning. AFs help determine the output of a Deep Learning model, the accuracy and computational efficiency of the training model – which will make a decision on the success or failure of a Neural Network system. AFs also have a great influence on the convergence and convergence speed of neural networks, or in some cases, AFs can prevent neural networks from converging in the first place.

AFs are presented in mathematical equations and this function is attached to each neuron in the network and determines whether it should be activated, based on whether the input data fits the model prediction, which also help normalize the output of each neuron to a range of 1 to 0 or between -1 and 1, depending on the type of activation function being used in the layer or network. Additionally, another aspect of AFs is that they have to have efficiency in computation because of being computed over thousands or even millions of neurons for each data sample. And modern neural networks use backpropagation technique to train the model.

The main reason that neural network models stand out from machine learning models is their ability to solve non-linear data problems through AFs of hidden layers, at which execute the task of solving problems for complex nonlinear relationships between data features and model outputs.

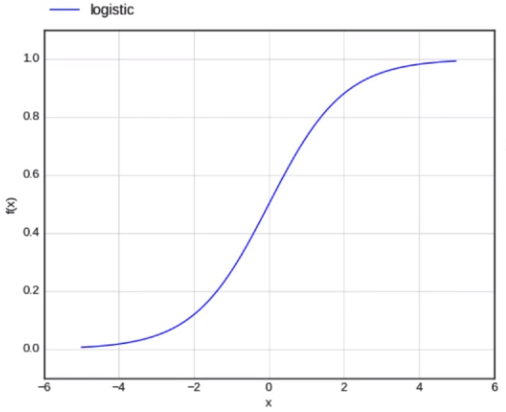
As mentioned earlier, in a neural network, the input data, is fed to the neurons in the input layer. Each neuron has a weight and multiplying the input by the weight gives the output of the neuron, which is then passed on to the next layer. So, AFs are applied in neural networks to be on duty of computation for the weight input's sum and biases, being utilized to make a decision on whether or not a neuron can be made activated. It manipulates the supplied data via several gradient processing, or gradient descent in typical and afterwards give out a production for a neural network's output, including the parameters. These AFs are usually known as a reference for a transfer function in some documentation or research papers. Furthermore, depending on the function they deputize, AFs can be either linear or non-linear, and they are used to modulate the outputs of neural networks in a variety of areas ranging such as: object recognition, classification, segmentation, speech recognition, cancer detection systems, finger print detection, and so on… [12]



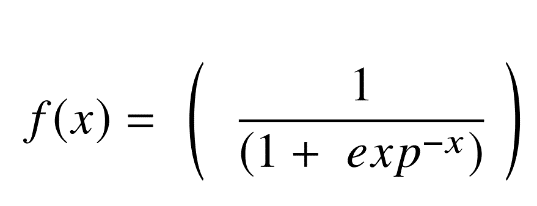
Although in fact that there are many types of AFs, in this article I will only mention some of the AFs that are commonly used in current CNNs architectures such as: sigmoid, softmax, hyperbolic tangent (Tanh), rectified linear unit (ReLU) and Leaky rectified linear unit (LReLU).

* 1. **Sigmoid function:**

The Sigmoid Function is also known as the Sigmoid curve. This is a mathematical function characterized by an S-shaped curve, which is one of the non-linear activation functions most widely used. If being familiar with some machine learning models, probably still remembering Logistic Regression - a one of the simple algorithms for binary classification problems, which is quite effective. The "soul" of Regression is this Sigmoid function. The sigmoid is a continuously nonlinear function, allows to pass real numbers as its input and gives a production of results in the range 0 to 1, is considered probabilistic in some problems. Suppose you trained a *Neural Network* to images classification of birds and dogs, what a classic problem, where bird is 1 and dog is 0. Basically, when your Model returns a value greater than 0.5 meaning the image is of a bird, or the value less than 0.5 means the image is of a dog.



In the Sigmoid function, a small adjustment from the input results in a no relative change for output. It gives a smoother and more continuous output than input, so, it has derivative in everywhere [13]. The Figure below shows the formular of sigmoid function:

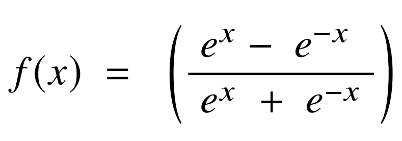


In the formula, x represents for the network’s output, and f(x) is its function, and exp stands for the Number of Euler. As a result, if the value of x approaches to the infinity of positive, the expected value of f(x) becomes 1. In contrast, if x approaches to the infinity of negative, the predicted value of f(x) becomes 0. And if the result of the sigmoid function is greater than 0.5, that label is classified as class 1 or positive side, and if it is less than 0.5, classified as class 0 or negative side.

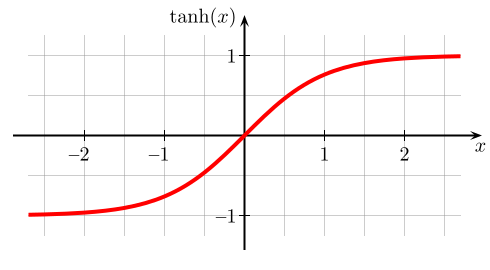
On the other hand, the sigmoid has significant shortcomings such as sharp damp gradients while doing backpropagation starting from deeper hidden layers back to input layers, gradient saturation, sluggish convergence, and output of having non-zero centred, which causes gradient updates to propagate in various passageways. Other types of AF, such as the hyperbolic tangent function (Tanh), have been recommended to address some of the shortcomings of the Sigmoid AF.

* 1. **Hyperbolic tangent function (Tanh):**

Another form of AF utilized in DL is the hyperbolic tangent function, which has several versions employed in DL applications [14]. The hyperbolic tangent function, often known as the tanh function, is a smoother zero-centred function with a range of -1 to 1, and its output is given by:



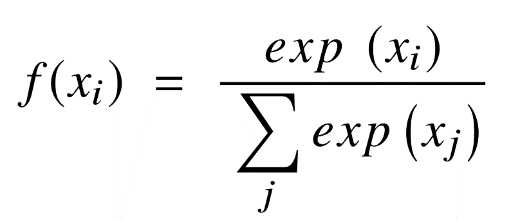
Because of supplying higher training performance for multilayer neural networks than the sigmoid function, the tanh function has become the favoured function over the sigmoid function [15]. Otherwise, the tanh function, like the sigmoid functions, was unable to overcome the vanishing gradient problem. The function's key advantage is that it generates zero-centred output, which aids in the operation of back-propagation.



The tanh function has the peculiarity of only being able to achieve a gradient of 1 only when the input's value is equal to 0, that equivalent to x equal zero. As a result, the tanh function generates some inactivated neurons during computation. The unworked neuron is a condition in which the activation weight rarely employed as a result of a zero gradient. In addition, this obstruction of the tanh function prompted additional study in activation functions to remedy the problem, giving rise to the ReLU activation function. Furthermore, the tanh function has mostly been employed in recurrent neural networks for natural language processing and speech recognition [16].

* 1. **Softmax function:**

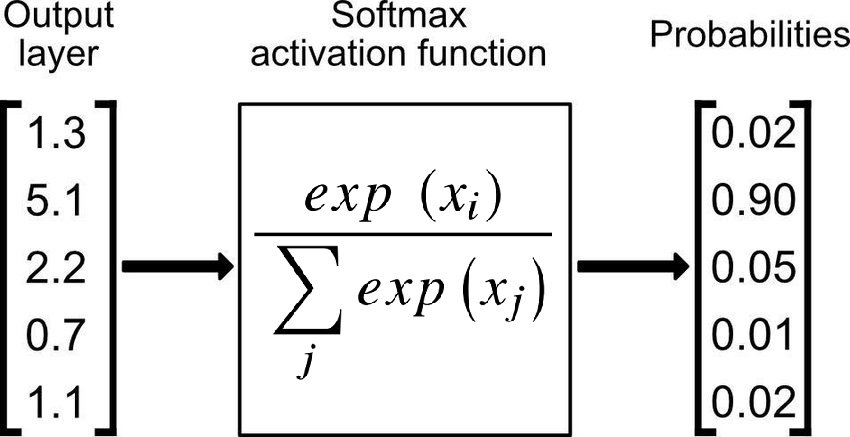
Softmax Function (SF) is an exponential averaging function that calculates the probability of an event occurring. In general, the SF calculates the probability of a class occurring out of the total of all possible classes. This probability will then be used to determine the target class for the inputs. Specifically, the SF turns a k-dimensional vector of any real values ​​into a real-valued k-dimensional vector that sums to 1. The input value can be positive, negative, zero, or greater than 1, but the SF will always turn them into a value in the range (0:1]. The SF is calculated following this formula:



Where (xi are vector of input values for the function from x0 to xn, in which all the values can be any real number, positive number, negative number or zero. Next, exp(xi) is the standard exponentiation function is applied to each input value returning a positive value greater than 0. This value will be very small if the input is negative, and very large if the input is positive and it is an uncertain number in the range (0:1] as the requirement of a probability.

As such, they can be called “probabilities”. If one of the input values ​​is very small or negative, the SF turns them into a small probability. If an input is large, it will be converted to a large probability. But the probability is always greater than 0 and less than 1, or equal to 1. So, the sum of all probabilities is always 1 and the SF brings a lot of benefits such as:

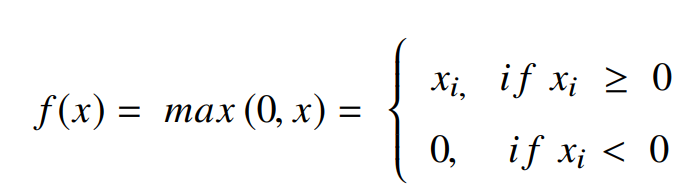
* Optimized when calculating the maximum probability in the model parameter.
* The property of the SF makes it suitable for probabilistic interpretation, which is very useful in Machine Learning.
* Softmax normalization is a way to minimize the effect of extreme values ​​or outliers in data without having to modify the original data.



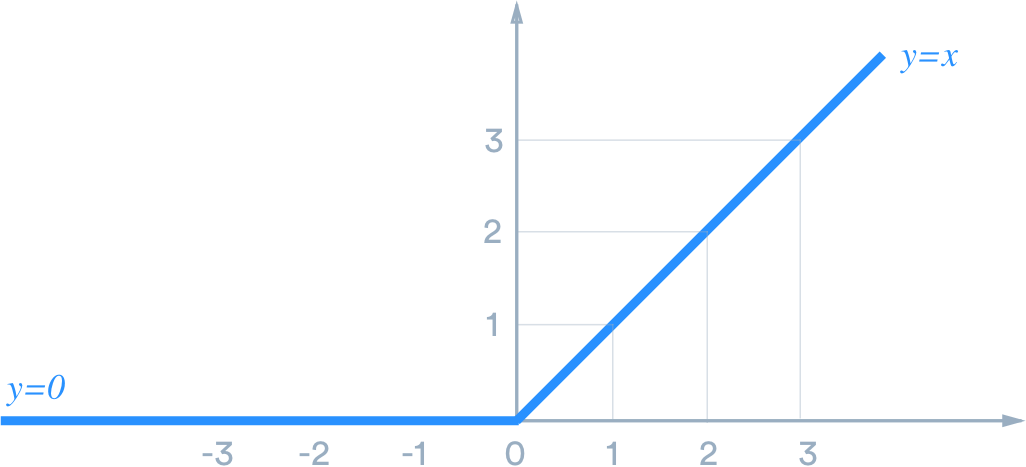
For the reasons mentioned above, the SF returns the value of probabilities for each class in multi-class models, with the target class having the highest likelihood. The SF may be found in practically all of the deep learning's output layers architectures where it is used. Adding, the Sigmoid and Softmax vary in that the Sigmoid is used for binary classification and the Softmax is utilized for the tasks of multivariate classification.

* 1. **Rectified Linear Unit (ReLU) Function:**

In 2010, Nair and Hinton introduced the rectified linear unit (ReLU) activation function, which has been the most extensively utilized activation function for applications in the DL field, bringing a state-of-the-art result till currently [17]. In general, The ReLU has a expressive speed of learning AF proven to be the most extensively and successfully applied function [14]. In DL, it outperforms and generalizes the Sigmoid and Tanh activation functions [18]. In particular, there is a representation of a roughly linear function in ReLU function and hence retains the features of linear models that make them easier to optimize using the approaches of gradient decent. The mechanism of ReLU AF, which applies a functioning of threshold to each element of input having values smaller than zero to be replaced to zero, and the formular of the ReLU as follows:



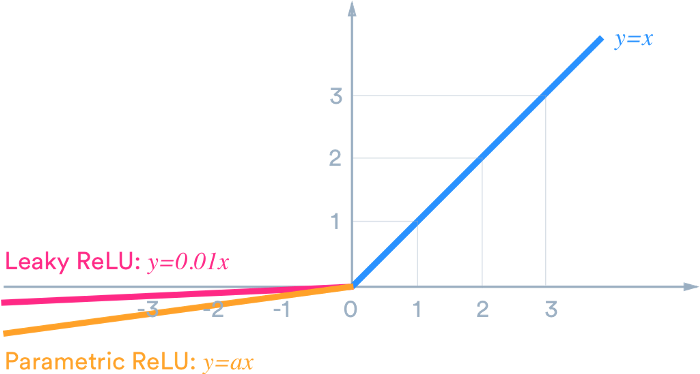
The ReLU will correct the inputs' values which are smaller than zero, driving them to zero and removing the issue about vanishing gradient that was noticed in the previous types of AFs. Mainly, the ReLU function is put to utilize into the hidden units of deep neural networks. Besides, another AF as sigmoid or softmax is offen applied in the network's output layers with popular purpose of classification in image [12] and speech recognition.



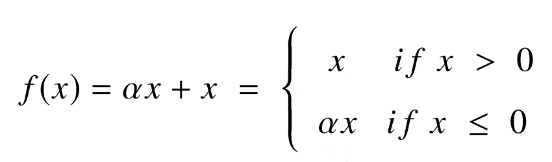
There are many advantages when using ReLU. In detail, the fundamental advantage of ReLU in computation is that they bring a more expeditious computation because they do not need to quantify exponentials and divisions, resulting in increased overall computation performance [18]. creation of sparsity in the hidden units is another feature of the ReLU, which squishes the values from zero to maximum. The ReLU has a problem in that it readily overfits when compared to the sigmoid function, despite the fact that the approach of dropout has been used to lessen the overfitting's influence of ReLUs and the rectified networks enhanced deep neural network performances [19].

There is a notable weakness in the ReLU, which it is occasionally brittle during training, leading to the death of the several gradients. This causes will make neurons to be dead and direct weight updates to inactivate data points in the future, hampered learning because of dead neurons result in zero activation [3]. So that, the leaky ReLU was offered as a solution to the dead neuron problem.

* 1. **Leaky ReLU:**



The leaky ReLU was released in 2013 that give out several small negative slope to the ReLU to maintain and stay survived the weight updates throughout the phase of propagation. The building of the alpha parameter serves for the solution to issue of dead neuron of the ReLUs, which ensure that the gradients are not fixed zero anymore during training. The computation of the LReLU bases on the gradient with a tiny constant value for the negative gradient α in the range of 0.01; hence, the LReLU AF is computed following this formula:



The comparasion of the ReLU to an exception of no having zero gradients over the entire of time, which report a similar result. Therefore, excepting of distribution and sparsity in the comparasion of ReLU to Tanh, no having any noteworthy enhancement of result. As a result, Leaky ReLU offers two advantages:

* Because it lacks zero-slope portions, it solves the "dying ReLU" issue, so training process is expedited. And, the "mean activation" near to zero is an evidence for speeding up training. (It aids in keeping the diagonal entries of the Fisher information matrix short, but it can disregard judiciously.) Unlike ReLU, leaky ReLU is more "balanced," and as a result, it may learn faster.
* Keep in mind that the outcome is not always consistent. Leaky ReLU isn't necessarily better than regular ReLU, and it should only be used as a last resort.