

# AI for Cybersecurity

Artificial Intelligence in the field of  
Cybersecurity

## AI in the context of cybersecurity

With the exponential increase in the spread of threats associated with the daily diffusion of new malware, it is practically impossible to think of dealing effectively with these threats using only analysis conducted by human operators. It is necessary to introduce algorithms that allow us to automate that introductory phase of analysis known as **triage**, that is to say, to conduct a **preliminary screening** of the threats to be submitted to the attention of the cybersecurity professionals, allowing us to respond in a timely and effective manner to ongoing attacks.

We need to be able to respond in a dynamic fashion, adapting to the changes in the context related to the presence of **unprecedented** threats. This implies not only that the analysts manage the tools and methods of cybersecurity, but that they can also correctly interpret and evaluate the results offered by AI and ML algorithms.

Cybersecurity professionals are therefore called to understand the **logic of the algorithms**, thus proceeding to the **fine tuning** of their learning phases, based on the results and objectives to be achieved.

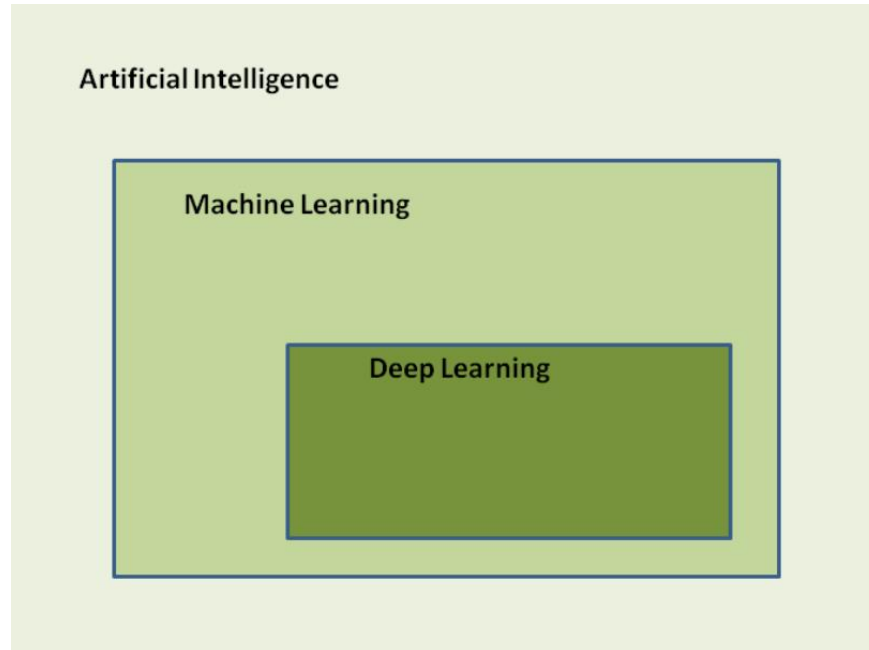
Some of the tasks related to the use of AI are as follows:

- **Classification:** This is one of the main tasks in the framework of cybersecurity. It's used to properly identify types of similar attacks, such as different pieces of **malware** belonging to the same family, that is, having common characteristics and behavior, even if their signatures are distinct (just think of **polymorphic malware**). In the same way, it is important to be able to adequately classify emails, distinguishing **spam** from legitimate emails.
- **Clustering:** Clustering is distinguished from classification by the ability to automatically identify the classes to which the samples belong when information about classes is not available in advance (this is a typical goal, as we have seen, of unsupervised learning). This task is of fundamental importance in **malware analysis** and **forensic analysis**.
- **Predictive analysis:** By exploiting NNs and DL, it is possible to identify threats as they occur. To this end, a **highly dynamic** approach must be adopted, which allows algorithms to optimize their learning capabilities automatically.

Possible uses of AI in cybersecurity are as follows:

- **Network protection:** The use of ML allows the implementation of highly sophisticated **intrusion detection systems (IDS)**, which are to be used in the network perimeter protection area.
- **Endpoint protection:** Threats such as **ransomware** can be adequately detected by adopting algorithms that learn the behaviors that are typical of these types of malware, thus overcoming the **limitations of traditional antivirus** software.
- **Application security:** Some of the most insidious types of attacks on web applications include **Server Side Request Forgery (SSRF)** attacks, **SQL injection**, **Cross-Site Scripting (XSS)**, and **Distributed Denial of Service (DDoS)** attacks. These are all types of threats that can be adequately countered by using AI and ML tools and algorithms.
- **Suspect user behavior:** Identifying attempts at **fraud** or compromising applications by malicious users at the very moment they occur is one of the emerging areas of application of DL.

## What exactly is AI?



## What exactly is AI?

In the case of ML (which, as we have seen, is a branch of research belonging to AI), it is common to distinguish between the following types of ML:

- Supervised learning
- Unsupervised learning
- Reinforcement learning

The differences between these learning modalities are attributable to the type of result (output) that we intend to achieve, based on the nature of the input required to produce it.

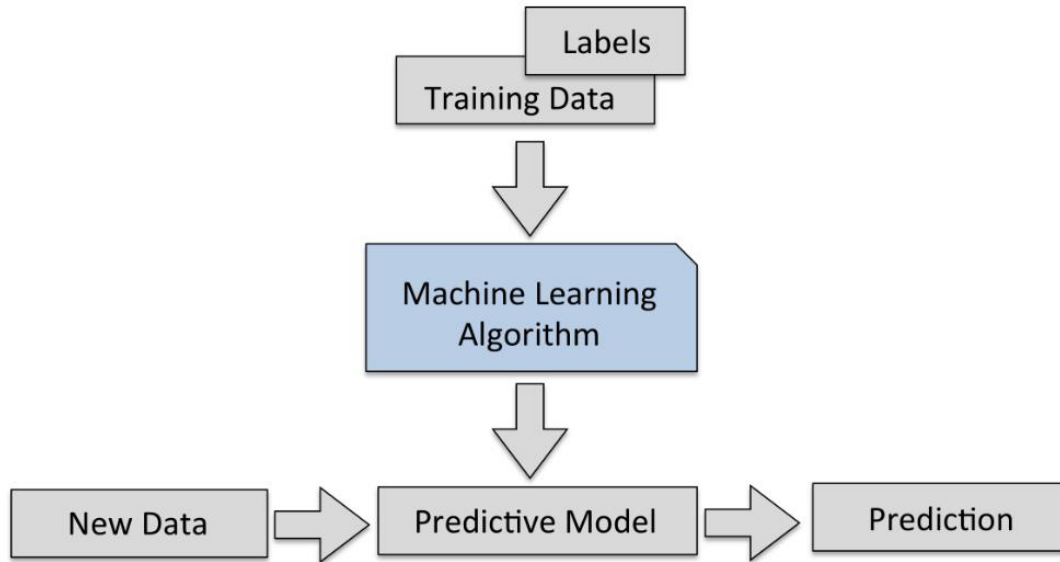
## Supervised Learning

An example of a supervised learning algorithm is classification algorithms, which are particularly used in the field of cybersecurity for **spam classification**.

A **spam filter** is in fact trained by submitting an input dataset to the algorithm containing many examples of emails that have already been previously classified as spam (the emails were malicious or unwanted) or ham (the emails were genuine and harmless).

The classification algorithm of the spam filter must therefore learn to classify the new emails it will receive in the future, referring to the spam or ham classes based on the training previously performed on the input dataset of the already classified emails.

## Supervised Learning





## Supervised Learning

Another example of supervised algorithms is regression algorithms. Ultimately, there are the following main supervised algorithms:

- Regression (linear and logistic)
- **k-Nearest Neighbors (k-NNs)**
- **Support vector machines (SVMs)**
- Decision trees and random forests
- **Neural networks (NNs)**

## Unsupervised learning

In the case of **unsupervised learning**, the algorithms must try to **classify the data independently**, without the aid of a previous classification provided by the analyst. In the context of cybersecurity, unsupervised learning algorithms are important for identifying new (not previously detected) forms of **malware** attacks, **frauds**, and **email spamming** campaigns.

Here are some examples of unsupervised algorithms:

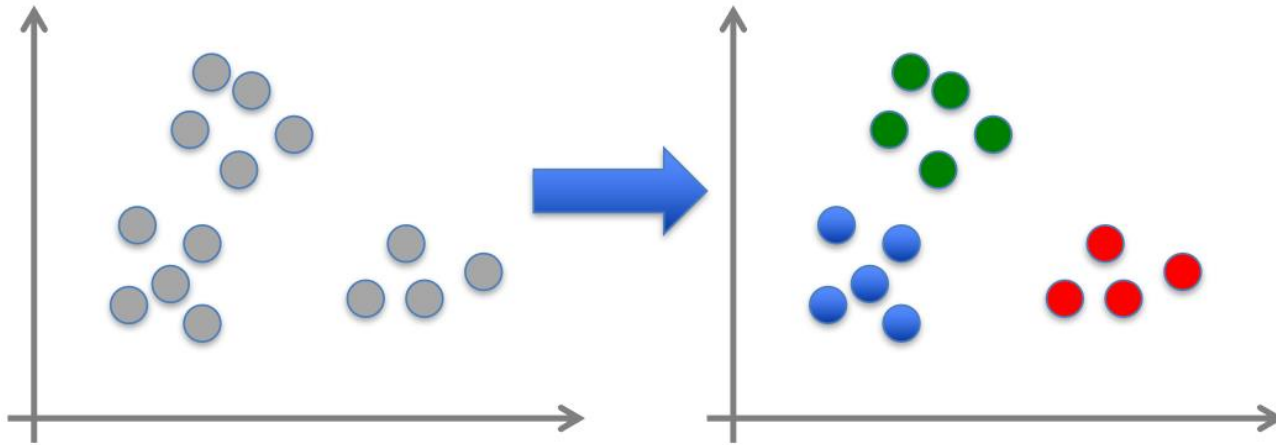
- Dimensionality reduction:
  - **Principal component analysis (PCA)**
  - PCA Kernel
- Clustering:
  - k-means
  - **Hierarchical cluster analysis (HCA)**

## Unsupervised Learning

Dealing with *unlabeled* data

Cluster analysis

Objects within a cluster share a degree of similarity



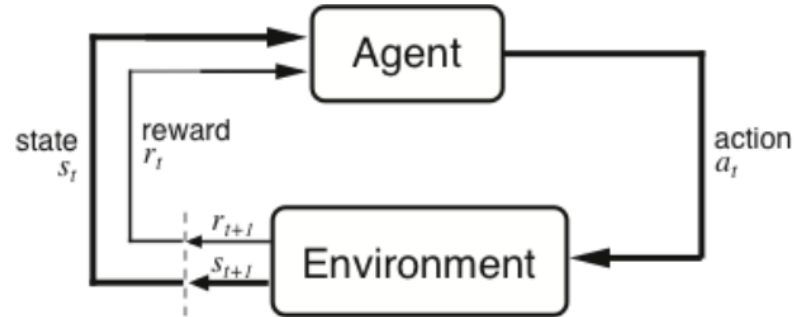
## Reinforcement learning

In the case of **reinforcement learning (RL)**, a different learning strategy is followed, which emulates the trial and error approach. Thus, drawing information from the feedback obtained during the learning path, with the aim of maximizing the reward finally obtained based on the number of correct decisions that the algorithm has selected.

In practice, the learning process takes place in an unsupervised manner, with the particularity that a **positive reward** is assigned to each correct decision (and a **negative reward** for incorrect decisions) taken at each step of the learning path. At the end of the learning process, the decisions of the algorithm are reassessed based on the final reward achieved.

## Reinforcement Learning

- Credit assignment problem
- Game playing
- Robot in a maze
- Balance a pole on your hand



## Reinforcement Learning

Given its dynamic nature, it is no coincidence that RL is more similar to the general approach adopted by AI than to the common algorithms developed in ML.

The following are some examples of RL algorithms:

- Markov process
- Q-learning
- **Temporal difference (TD)** methods
- Monte Carlo methods

## Algorithm training and optimization

When preparing automated learning procedures, we will often face a series of challenges. We need to overcome these challenges in order to recognize and avoid compromising the reliability of the procedures themselves, thus preventing the possibility of drawing erroneous or hasty conclusions that, in the context of cybersecurity, can have devastating consequences.

The management of false positives is particularly burdensome in the case of **detection systems** aimed at contrasting **networking threats**, given that the number of events detected are often so high that they absorb and saturate all the human resources dedicated to threat detection activities.

## How to find useful sources of data

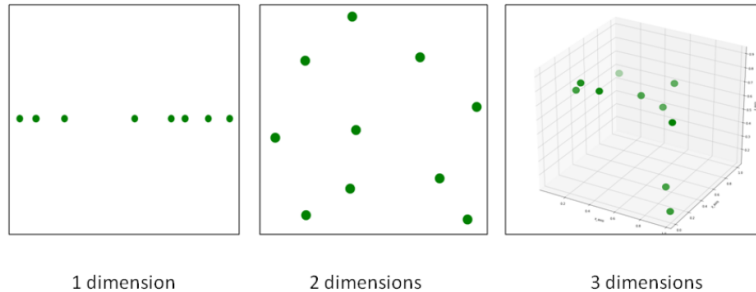
In the case of **anomaly detection**, for example, particular attention must be paid to the data being analyzed. An effective anomaly detection activity presupposes that the training data does not contain the anomalies sought, but that on the contrary, they reflect the normal situation of reference.

Given the increasing availability of raw data in real time, often the preliminary cleaning of data is considered a challenge in itself. In fact, it's often necessary to conduct a preliminary skim of the data, **eliminating irrelevant or redundant** information. We can then **present the data** to the algorithms in a correct form, which can improve their ability to learn, adapting to the form of data on the basis of the type of algorithm used.



For example, a **classification algorithm** will be able to identify a more representative and more effective model in cases in which the input data will be presented in a **grouped form**, or is capable of being **linearly separable**. In the same way, the presence of **variables** (also known as **dimensions**) containing **empty fields** weighs down the computational effort of the algorithm and produces less reliable predictive models due to the phenomenon known as the **curse of dimensionality**.

## The “Curse of Dimensionality”



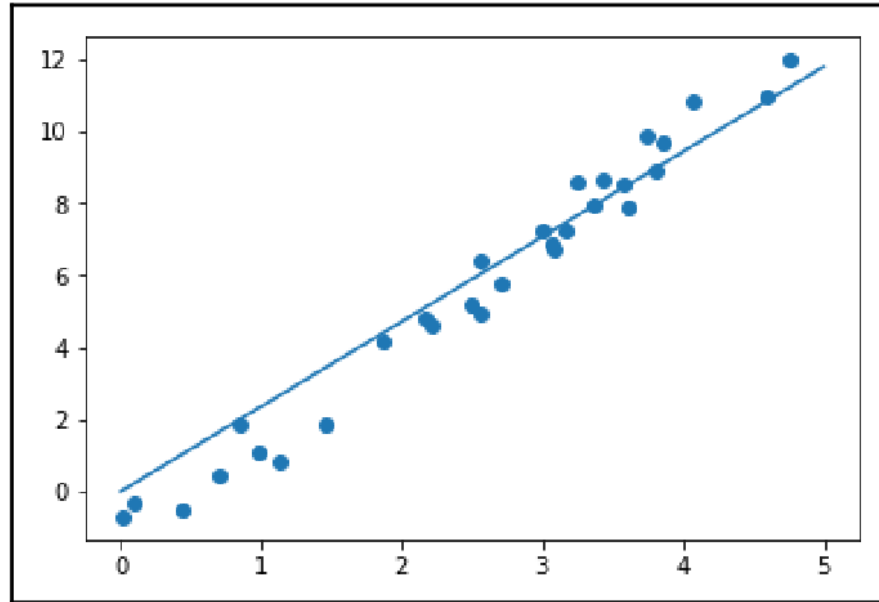
## Supervised learning example – linear regression

As our first example, we'll look at one of the most commonly used algorithms in the field of supervised learning, namely linear regression. Taking advantage of the `scikit-learn` Python library, we instantiate a linear regression object, by importing the `LinearRegression` class included in the `linear_model` package of the `scikit-learn` library.

The model will be trained with a training dataset obtained by invoking the `rand()` method of the `RandomState` class, which belongs to the `random` package of the Python `numpy` library. The training data is distributed following the linear model of,  $y = 3x + 2$ . The training of the model is carried out by invoking the `fit()` method on the `lreg` object of the `LinearRegression` class.

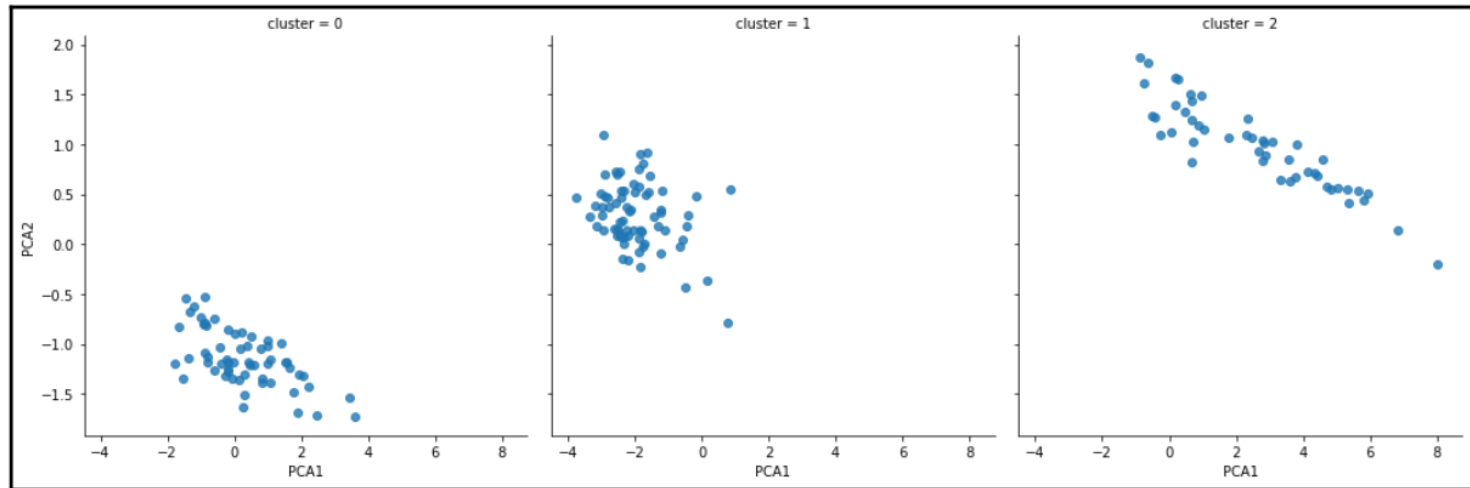
# AI for Cybersecurity

The preceding code generates the following output, which shows how well the data samples are approximated by the straight line returned by the LinearRegression model:



## Unsupervised learning example – clustering

As an example of unsupervised learning, we use the `GaussianMixture` clustering model. Through this type of model, we will try to bring the data back to a collection of **Gaussian blobs**.



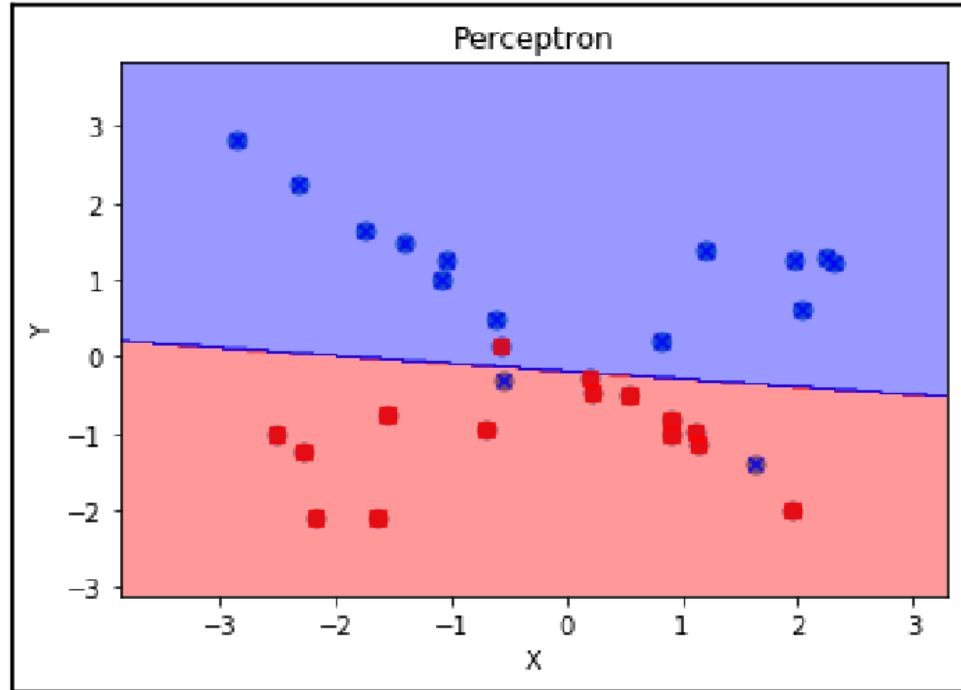
## Simple NN example – perceptron

In this section, we will show a simple NN model, known as a **perceptron**.

However rudimentary it is, a perceptron is nonetheless able to adequately classify samples that tend to group together (in technical terms, those that are **linearly separable**).

One of the most common uses of a perceptron in the field of cybersecurity, as we will see, is in the area of **spam filtering**.

# AI for Cybersecurity



## Detecting spam with Perceptrons

One of the first concrete and successful applications of AI in the field of cybersecurity was spam detection, and one of the most famous open source tools is **SpamAssassin**.

The strategies that can be implemented for effective spam detection are different, as we will see in the course of the chapter, but the most common and simpler one uses **Neural Networks (NNs)** in the most basic form; that is, the Perceptron.

Spam detection also provides us with the opportunity to introduce theoretical concepts related to NNs in a gradual and accessible way, starting with the Perceptron.

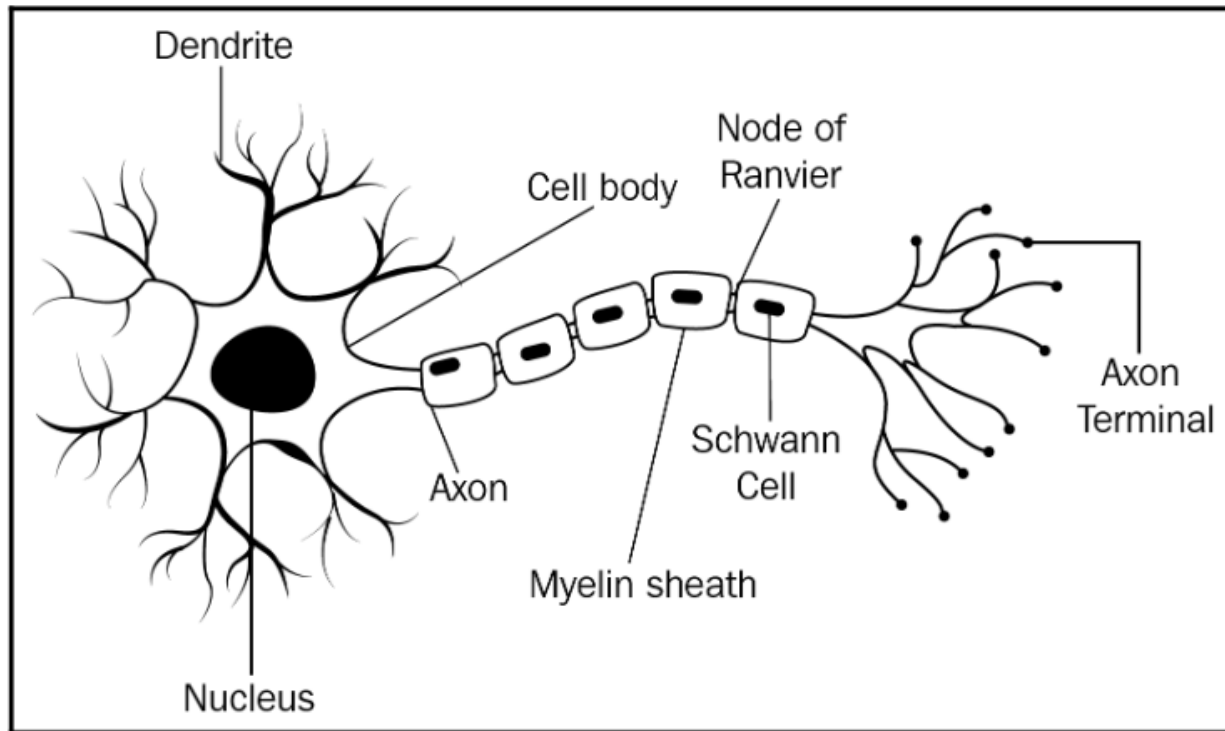
The peculiar characteristic that unites all NNs (regardless of their implementation complexity) is that they conceptually mimic the behavior of the human brain. The most basic structure we encounter when we analyze the behavior of the brain, is undoubtedly the neuron.

The Perceptron is one of the first successful implementations of a neuron in the field of **Artificial Intelligence (AI)**. Just like a neuron in the human brain, it is characterized by a layered structure, aimed at associating a result in output to certain input levels, as shown in the following diagram:



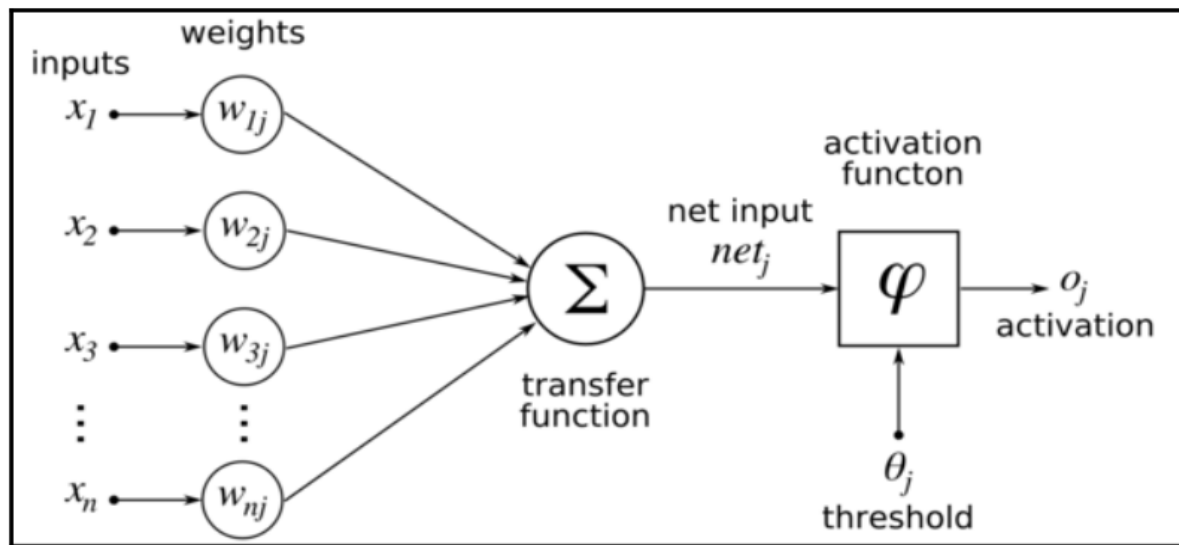
# AI for Cybersecurity

Innovation  
eXploited



# AI for Cybersecurity

In the same way, the artificial representation of the neuron implemented through the Perceptron model is structured in such a way as to associate a given output value to one or more levels of input data:



## It's all about finding the right weight!

One of the differences in the approach between the statistical models and the AI algorithms is that the algorithms implement an optimization strategy based on the iteration. At each iteration, in fact, the algorithm tries to adjust its own estimate of the values, attributing to them a greater or lesser weight depending on the cost function that we must minimize. One of the aims of the algorithm is to identify precisely an optimal weight vector to be applied to the estimated values in order to obtain reliable future predictions on unknown future data.

To fully understand the power of AI algorithms applied to spam detection, we must first clarify the ideas on which tasks we should perform a spam filter.

## Spam filters in a nutshell

To understand the tasks performed by a spam filter, let's look at an example. Imagine separating the emails we receive, categorizing them based on the presence or the absence of particular keywords occurring within the text of the emails with a certain frequency. To this end, we could list all the messages we receive in our inbox within a table. But how will we proceed with classifying our messages as ham or spam?

As we said, we will look for the number of occurrences of the suspicious keywords within the text of the email messages. We will then assign a score to the individual messages identified as spam, based on the number of occurrences of identified keywords. This score will also provide us with a reference to classify subsequent email messages.

We will identify a threshold value that allows us to separate spam messages. If the calculated score exceeds the threshold value, the email will automatically be classified as spam; otherwise, it will be accepted as a legitimate message, and thus classified as ham. This threshold value (as well as the assigned scores) will be constantly redetermined to take into account the new series of spam messages that we will meet in the future.

## Spam filters in action

How does an anti-spam algorithm actually behave in the classification of emails? First of all, let's classify the emails based on suspicious keywords. Let's imagine, for the sake of simplicity, that the list of the most representative suspicious keywords is thus reduced to only two words: buy and sex.

At this point, we will classify the email messages within a table, showing the number of occurrences of the individual keywords identified within the text of the emails, indicating the messages as spam or ham:

# AI for Cybersecurity

Email	Buy	Sex	Spam or Ham?
1	1	0	H
2	0	1	H
3	0	0	H
4	1	1	S

At this point, we will assign a score to every single email message.

This score will be calculated using a scoring function that takes into account the number of occurrences of suspicious keywords contained within the text.

A possible scoring function could be the sum of the occurrences of our two keywords, represented in this case by the  $B$  variable instead of the word buy, and the  $S$  variable instead of the word sex.

The scoring function therefore becomes the following:

$$y = B + S;$$

We can also attribute different weights to the representative variables of the respective keywords, based on the fact that, for example, the keyword sex contained within the message is indicative of a greater probability of spam than the word buy.

It is clear that if both words are present in the text of the email, the probability of it being spam increases. Therefore, we will attribute a lower weight of 2 to the  $B$  variable and a greater weight of 3 to the  $S$  variable.

Our scoring function, corrected with the relative weights assigned to the variables/keywords, therefore becomes the following:

$$y = 2B + 3S;$$



# AI for Cybersecurity

Now let's try to reclassify our emails, calculating the relative scores with our scoring function:

Email	B	S	$2B + 3S$	Spam or Ham?
1	1	0	2	H
2	0	1	3	H
3	0	0	0	H
4	1	1	5	S

At this point, we must try to identify a threshold value that effectively separates spam from ham. Indeed, a threshold value between 4 and 5 allows us to properly separate the spam from the ham. In other words, in the event that a new email message scores a value equal to or greater than 4, we would most likely be faced with spam rather than ham.

How can we effectively translate the concepts we have just seen into mathematical formulas that can be used in our algorithms?

We will discuss further the implementation of Perceptrons, but first, we will introduce the concept of a linear classifier, useful for mathematically representing the task performed by a common spam detection algorithm.

## Detecting spam with linear classifiers

As known from linear algebra, the equation that represents the function used to determine the score to be associated with every single email message is as follows:

$$y = 2B + 3S;$$

This identifies a straight line in the Cartesian plane; therefore, the classifier used by our spam filter to classify emails is called a **linear classifier**. Using the known mathematical formalization commonly adopted in statistics, it is possible to redefine the previous equation in a more compact form by introducing the sum operator  $\sum$ , substituting in place of the  $B$  and  $S$  variables a matrix of indexed values  $x_i$ , and a vector of weights  $w_i$  associated with it:

$$y = \sum w_i x_i$$

With the index  $i$ , which takes the values from 1 to  $n$ , this formalization is nothing more than the compact form of the previous summation between the variables and our relative weights:

$$y = w_1x_1 + w_2x_2 + \dots + w_nx_n;$$

This way, we have generalized our linear classifier to an unspecified number of variables,  $n$ , rather than limiting ourselves to 2 as in the previous case. This compact representation is also useful for exploiting linear algebra formulas in the implementation of our algorithms.

As we have seen, to adequately classify email messages, we need to identify an appropriate threshold value that correctly splits spam messages from ham messages: if the score associated with a single email message is equal to or higher than the threshold value, the message email will be classified as spam (and we will assign it the value  $+1$ ); otherwise, it will be classified as ham (to which we will assign the value  $-1$ ).

The preceding conditions are nothing but the following:

$$w_1 x_1 + w_2 x_2 + \dots + w_n x_n \geq \theta \rightarrow y = +1;$$

$$w_1 x_1 + w_2 x_2 + \dots + w_n x_n < \theta \rightarrow y = -1;$$

Taking up the previous mathematical expression that determines the conditions of activation of the Perceptron:

$$\text{if } wx \geq \theta \rightarrow f(y) = +1;$$

$$\text{if } wx < \theta \rightarrow f(y) = -1;$$

We see that it is the product of the  $wx$  values (that is, the input data for the corresponding weights) that has to overcome the  $\theta$  threshold to determine the activation of the Perceptron. Since the  $x_i$  input data is by definition prefixed, it is the value of the corresponding weights that helps to determine if the Perceptron has to activate itself or not.

But how are weights updated in practice, thus determining the Perceptron learning process?

The Perceptron learning process can be synthesized in the following three phases:

- Initializing the weights to a predefined value (usually equal to 0)
- Calculating the output value,  $y_i$ , for each corresponding training sample,  $x_i$
- Updating the weights on the basis of the distance between the expected output value (that is, the  $y$  value associated with the original class label of the corresponding input data,  $x_i$ ) and the predicted value (the  $y_i$  value estimated by the Perceptron)

In practice, the individual weights are updated according to the following formula:

$$w_i = w_i + \Delta w_i;$$

Here, the  $\Delta w_i$  value represents the deviation between the expected ( $y$ ) value and the predicted value ( $y_i$ ):

$$\Delta w_i = \lambda(y - y_i)x_i;$$



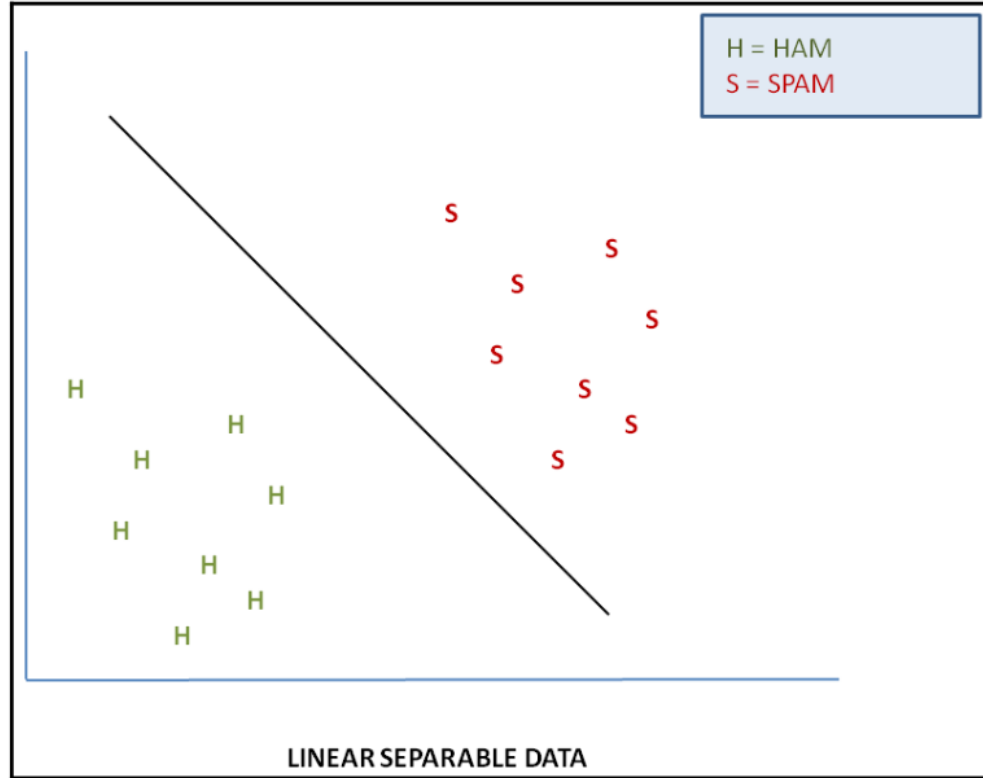
As is evident from the preceding formula, the deviation between the expected  $y$  value and predicted  $y_i$  value is multiplied by the value of input  $x_i$ , and by the  $\lambda$  constant, which represents the learning rate assigned to the Perceptron. The  $\lambda$  constant usually assumes a value between 0.0 and 1.0, a value that is assigned at the Perceptron initialization phase.

As we will see, the value of the learning rate is crucial for the learning of the Perceptron, and it is therefore necessary to carefully evaluate (even by trial and error) the value to be attributed to the  $\lambda$  constant to optimize the results returned from the Perceptron.

## Pros and cons of Perceptrons

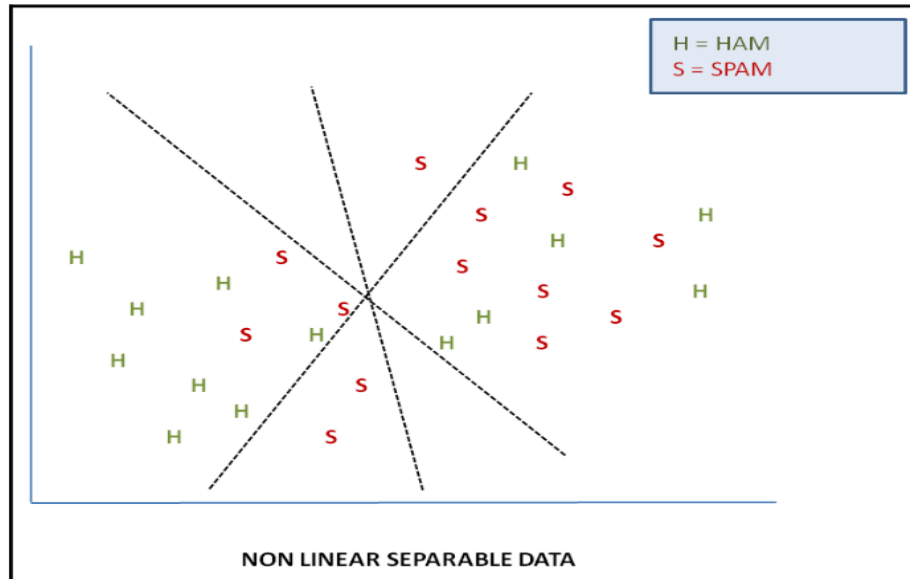
Despite the relative simplicity of the implementation of the Perceptron (simplicity here constitutes the strength of the algorithm, if compared to the accuracy of the predictions provided), it suffers from some important limitations. Being essentially a binary linear classifier, the Perceptron is able to offer accurate results only if the analyzed data can be linearly separable; that is, it is possible to identify a straight line (or a hyperplane, in case of multidimensional data) that completely bisects the data in the Cartesian plane:

# AI for Cybersecurity

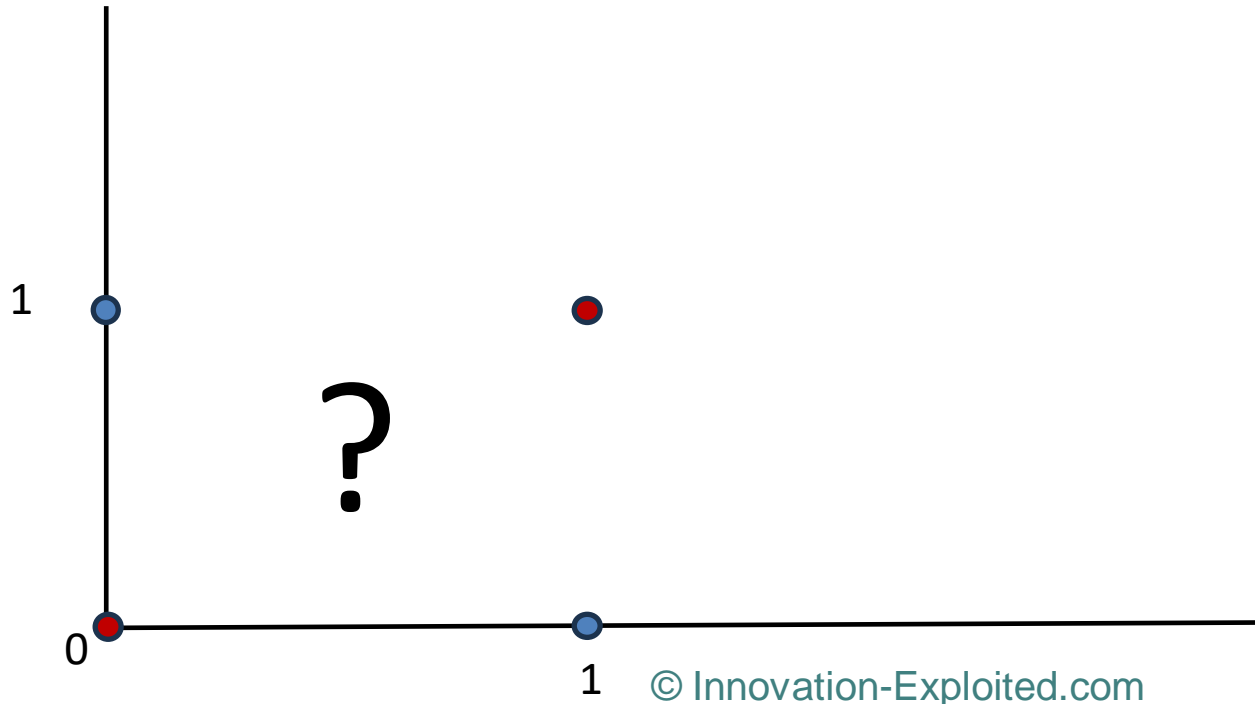


# AI for Cybersecurity

If instead (and this is so in the majority of real cases) the analyzed data was not linearly separable, the Perceptron learning algorithm would oscillate indefinitely around the data, looking for a possible vector of weights that can linearly separate the data (without, however, being able to find it):



## XOR linear separability problem



XOR

X1	X2	Y
0	0	0
0	1	1
1	0	1
1	1	0

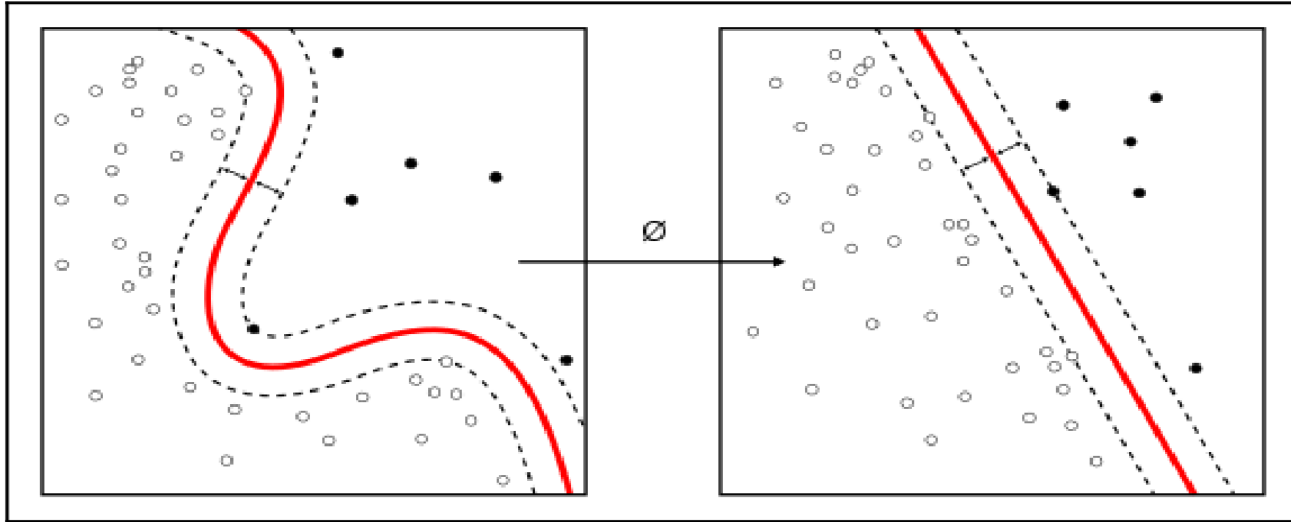
Therefore, the convergence of the Perceptron is only possible in the presence of linearly separable data, and in the case of a small learning rate. If the classes of data are not linearly separable, it is of great importance to set a maximum number of iterations (corresponding to the `max_iter` parameter) in order to prevent the algorithm from oscillating indefinitely in search of an (nonexistent) optimal solution.

One way to overcome the Perceptron's practical limitations is to accept a **wider margin** of data separation between them. This is the strategy followed by SVMs, a topic we'll encounter in the next section.

## Spam detection with SVMs

SVMs are an example of *supervised* algorithms (as well as the Perceptron), whose task is to identify the hyperplane that best separates classes of data that can be represented in a **multidimensional space**. It is possible, however, to identify different hyperplanes that correctly separate the data from each other; in this case, the choice falls on the hyperplane that **optimizes the prefixed margin**, that is, the distance between the hyperplane and the data.

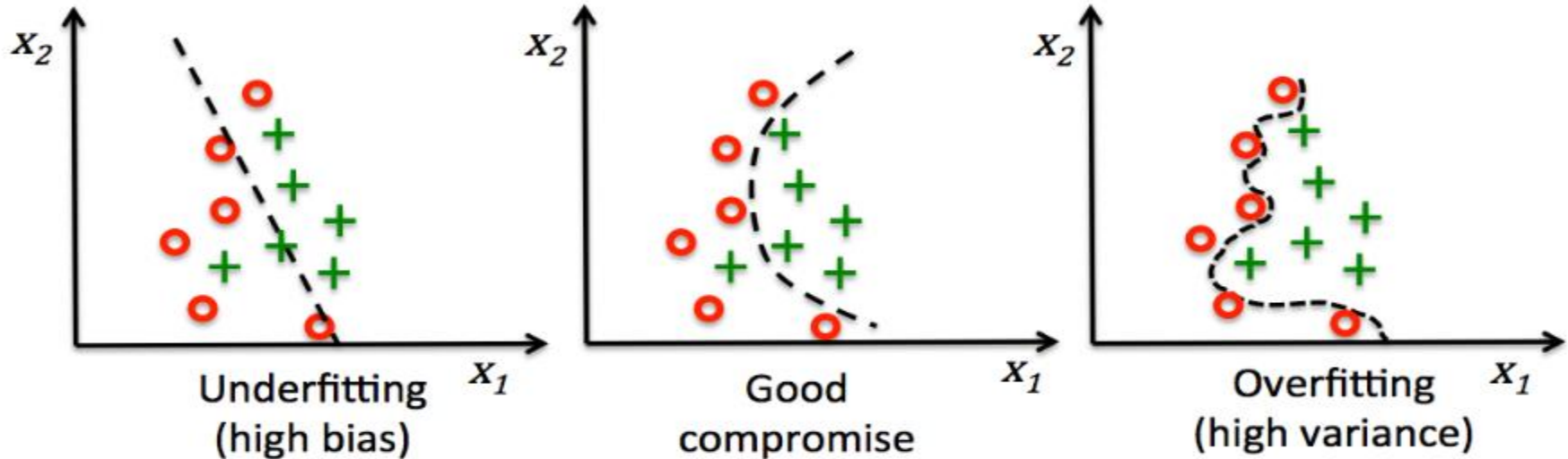
One of the advantages of the SVM is that the identified hyperplane is **not limited** to the linear model (unlike the Perceptron), as shown in the following screenshot:



The SVM can be considered as an extension of the Perceptron, however. While in the case of the Perceptron, our goal was to **minimize** classification errors, in the case of SVM, our goal instead is to **maximize** the margin, that is, the **distance** between the hyperplane and the training data *closest* to the hyperplane (the nearest training data is thus known as a **support vector**).



## Bias – Variance tradeoff



## Advantages of Naive Bayes for spam detection

The aforementioned features are well suited to the task of spam detection. Without the need to resort to large datasets, in fact, the spam detection algorithms based on Naive Bayes can exploit the emails already present in the inbox, constantly updating the probability estimations on the base of new email messages that are progressively added to those already existing.

The constant process of updating the probability estimates is based on the well-known Bayes rule:

$$P(A|E) = \frac{P(A) P(E|A)}{P(E)}$$

The preceding equation describes the relationship between the probability of the occurrence of an event,  $A$ , conditioned to the evidence,  $E$ .

## MLP & the Universal Approximation Theorem

Why does the MLP classifier show considerably better results in terms of prediction accuracy?

The answer lies in the fact that it represents an **artificial neural network (ANN)**.

ANNs constitute the fundamental element of deep learning and are at the base of high potential that characterizes deep learning algorithms, allowing, for example, the classification of enormous amounts of data, the performance of face and speech recognition, or beating a world chess champion such as Kasparov.

## MLP & the Universal Approximation Theorem

In fact, an MLP is made up of multiple layers of artificial neurons, each implemented by perceptrons.

An MLP can have three or more layers of fully connected artificial neurons, which, as a whole, constitute a feedforward network. Importantly, an MLP can approximate any continuous mathematical function; we can, thus, add an arbitrary number of hidden layers that amplify its overall predictive power.

## A Glimpse into Deep Learning

The concepts of DL and NNs are not new, but only in recent years have they found concrete practical, as well as theoretical, application, thanks to the progress achieved in the field of digital architectures, which have benefited from increased computational capacity, as well as the possibility of fully exploiting distributed computing through cloud computing, together with the almost unlimited availability of training data made possible by big data analytics.

The potential of DL has been recognized not only in the research and business sector, but also in the field of cybersecurity, where it is increasingly essential to use solutions capable of dynamically adapting to changes in context, adopting not only static detection tools, but algorithms that are able to dynamically learn how to recognize new types of attacks autonomously, finding possible threats by analyzing the most representative features within the often noisy datasets.

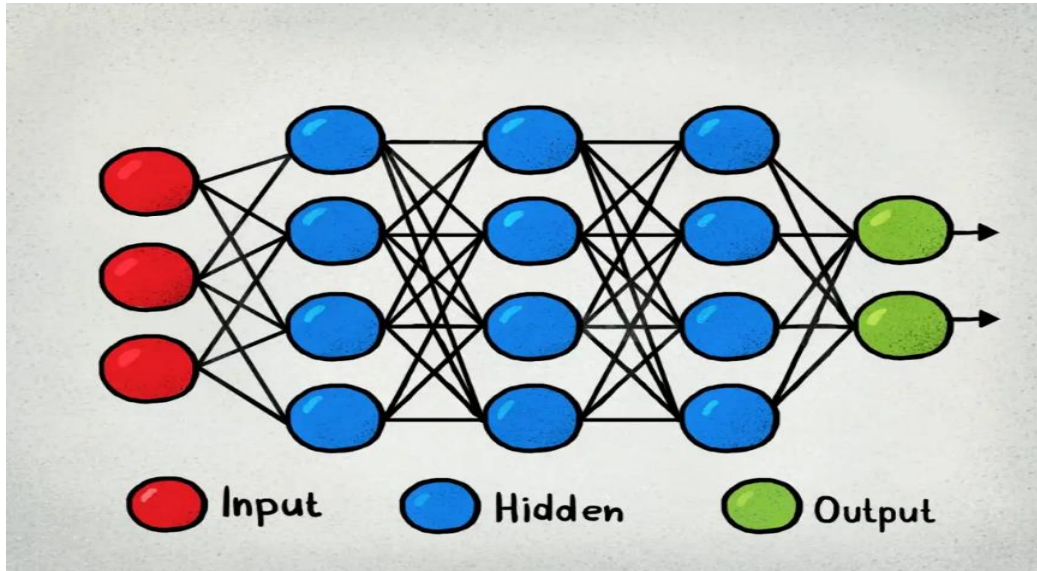
## A Glimpse into Deep Learning

Compared to traditional ML, DL is also characterized by a greater complexity from a mathematical point of view, especially regarding its widespread use of calculus and linear algebra. However, compared to ML, DL is able to achieve much better results in terms of accuracy and the potential reuse of algorithms in different application sectors.

The layers of artificial neurons that constitute DL analyze the data and features received as input and share them with the various inner layers, and these, in turn, process the output data of the outer layers. In this way, the original features extracted from the datasets are recombined, giving rise to new features that are optimized for analysis.

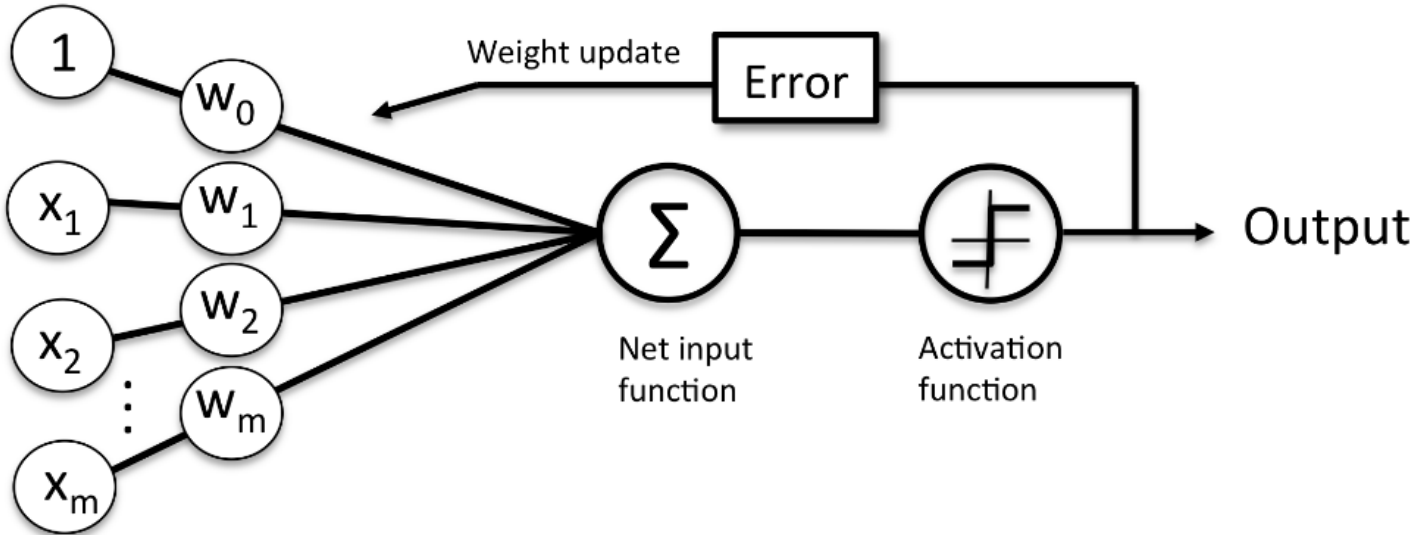
The greater the number of internal layers that are interconnected, the greater the depth and ability to recombine the features and adapt to the complexity of the problem, thereby reducing it to more specific and more manageable subtasks.

## A Glimpse into Deep Learning





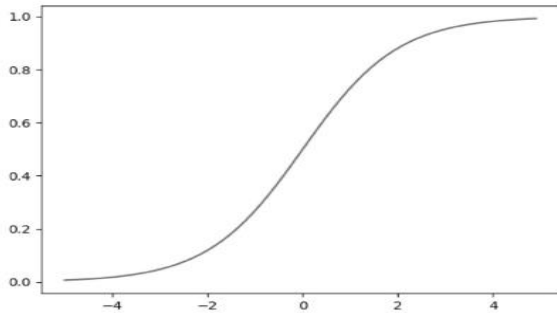
## ANN activation function & error propagation





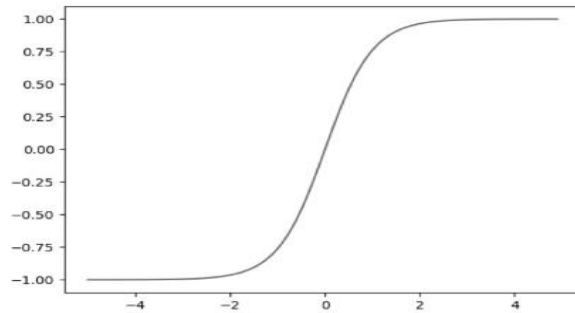
## Common activation function

Sigmoid



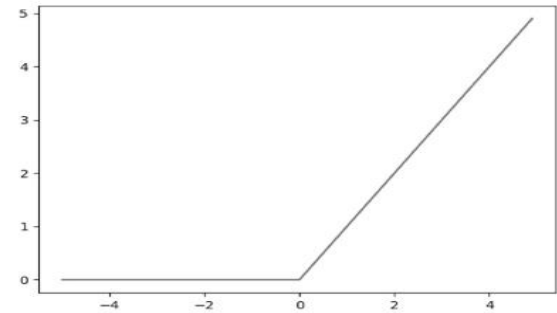
$$f(a) = \frac{1}{1 + e^{-ha}}$$

Tanh



$$f(a) = \frac{e^{2ha} - 1}{e^{2ha} + 1}$$

Rectified Linear Unit (ReLU)

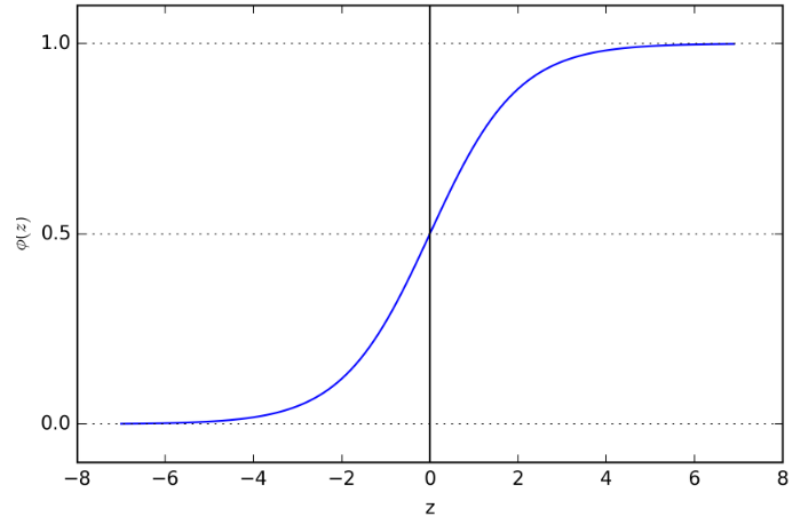


$$f(a) = \max\{0, ha\}$$

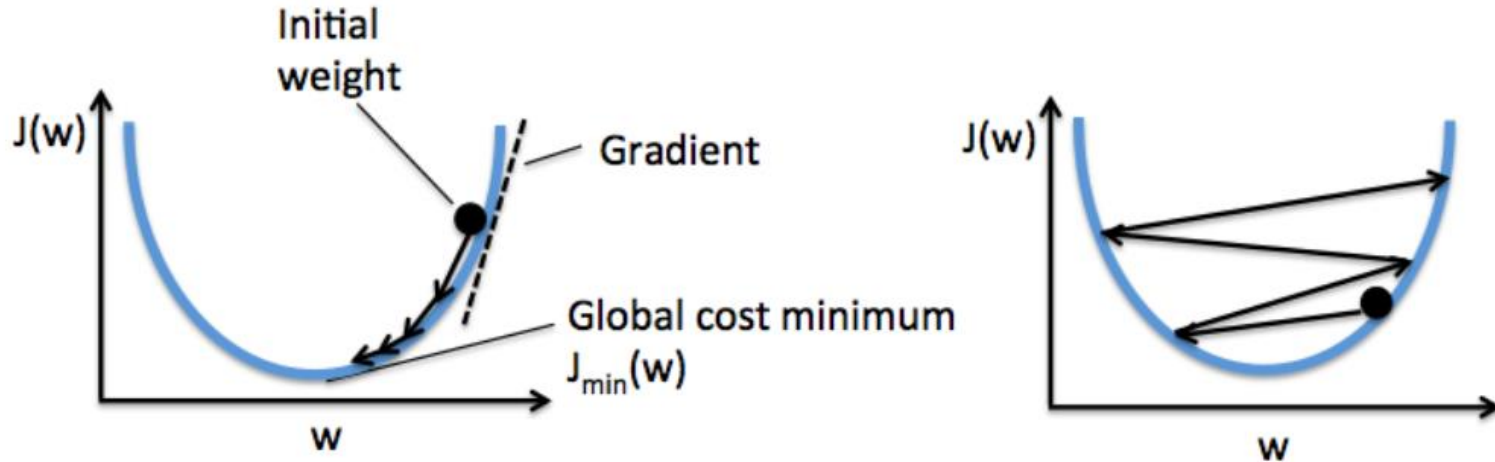
## Algorithm optimization: Logistic function

Logistic function (aka sigmoid function)

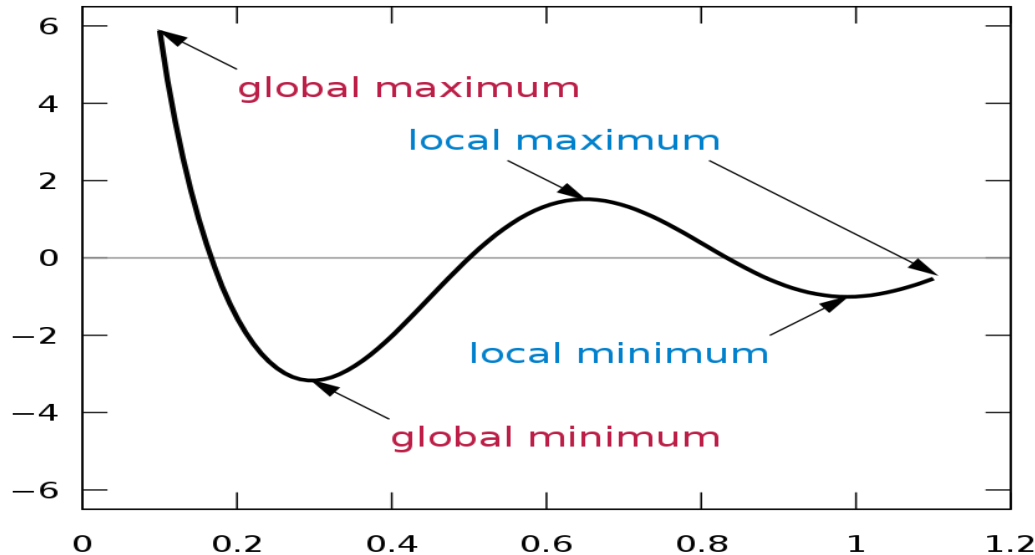
$$\phi(z) = \frac{1}{1 + e^{-z}}.$$



## Algorithm optimization: Gradient Descent



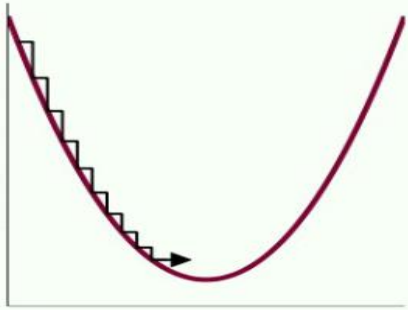
## Algorithm optimization: Gradient Descent



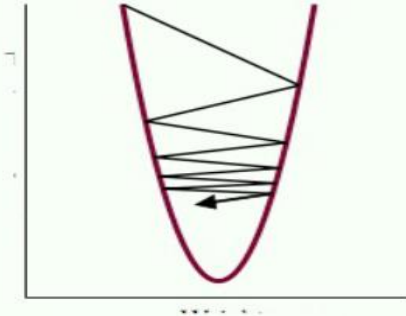
[https://commons.wikimedia.org/wiki/File:Extrema\\_example.svg](https://commons.wikimedia.org/wiki/File:Extrema_example.svg)

## Algorithm fine-tuning: Learning rate

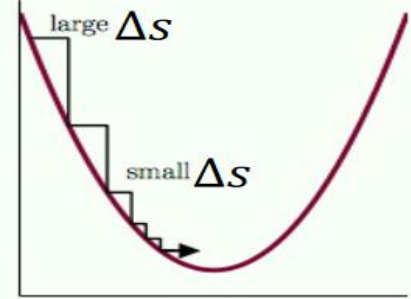
$\Delta s$  too small



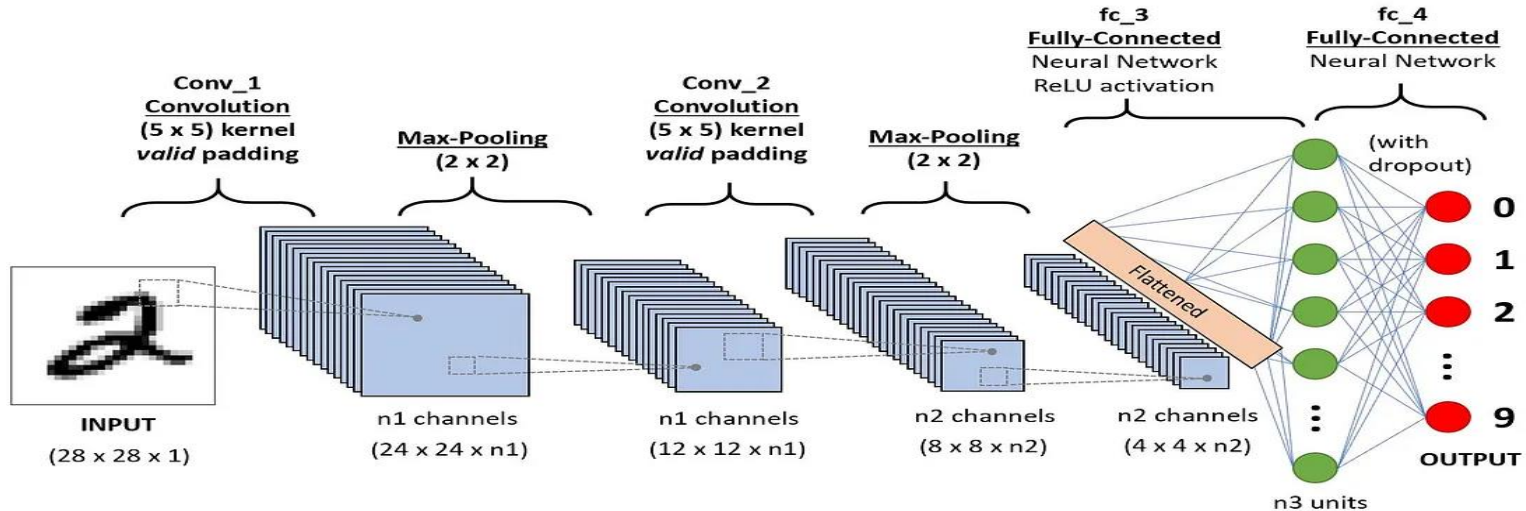
$\Delta s$  too large



Just right

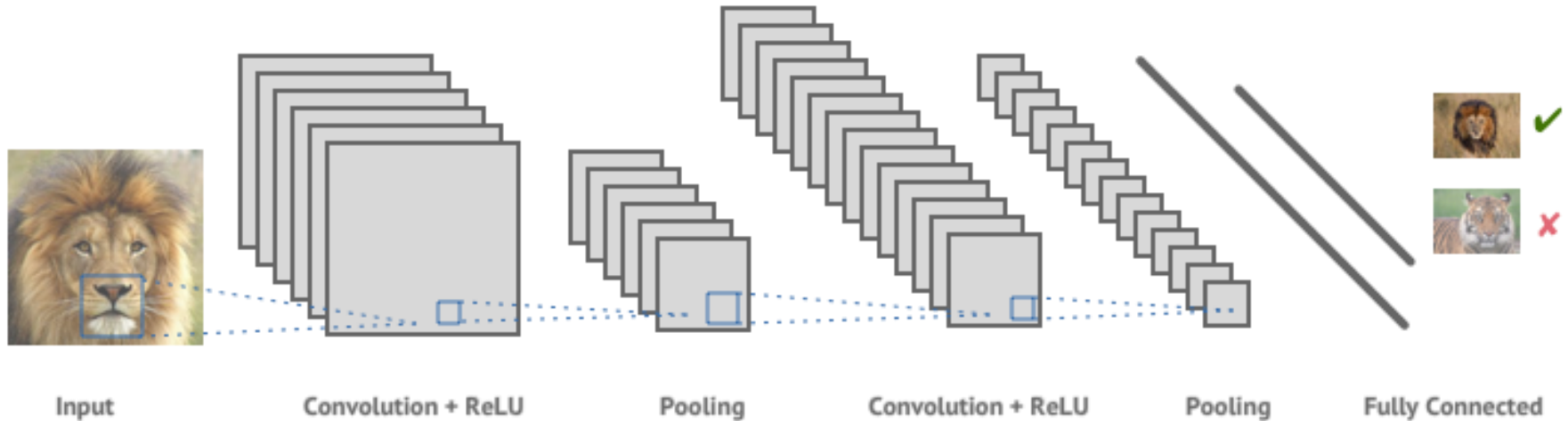


## ANN Example: Convolutional NN (CNN)

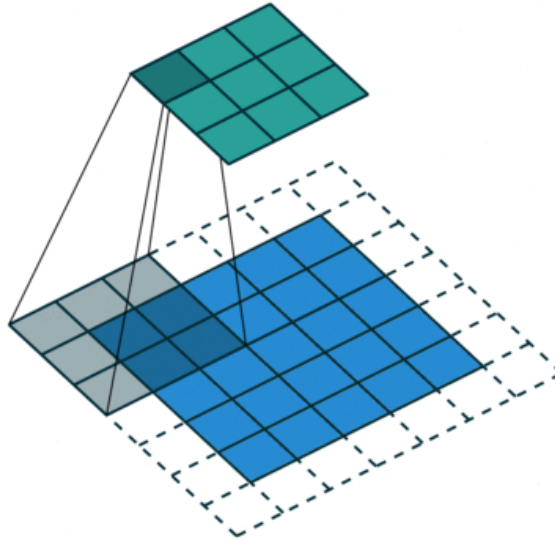


<https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>

## ANN Example: Convolutional NN (CNN)



## ANN Example: Convolutional NN (CNN)

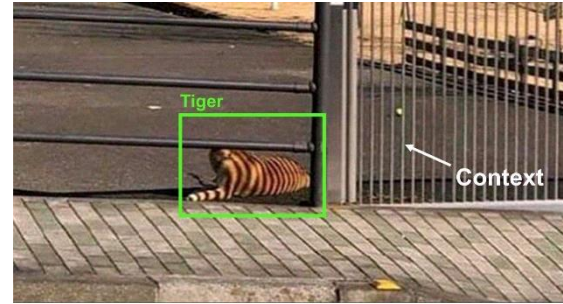


<https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>

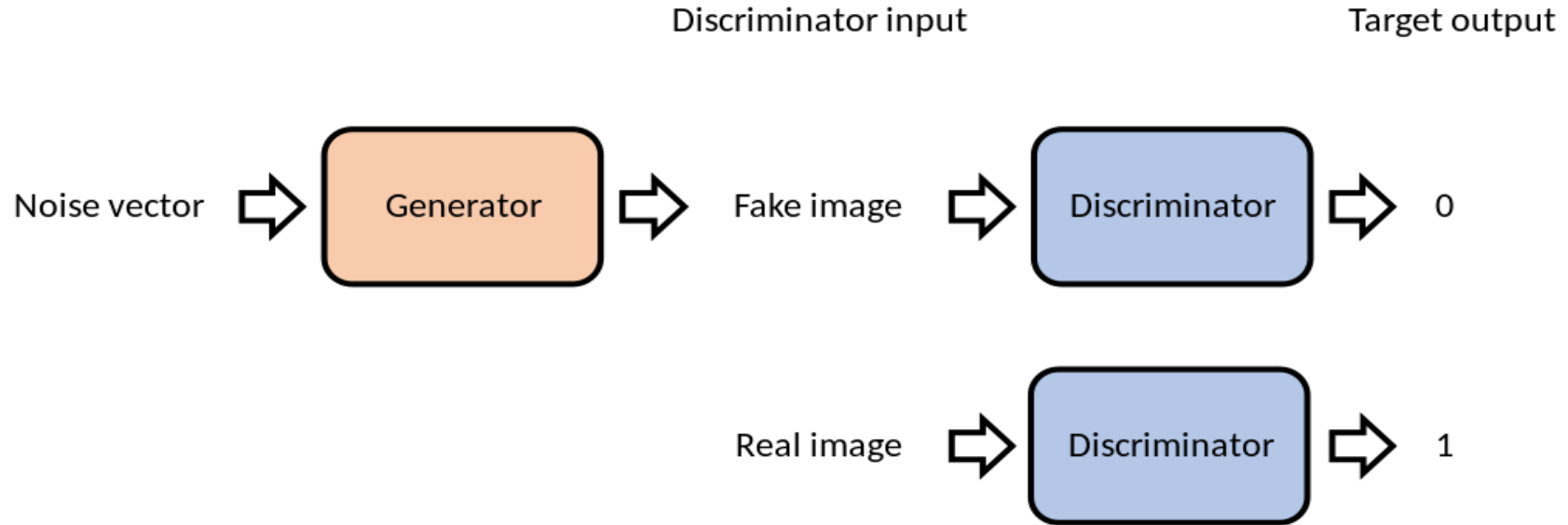


## CNN Limitations

Chihuahua or Muffin?

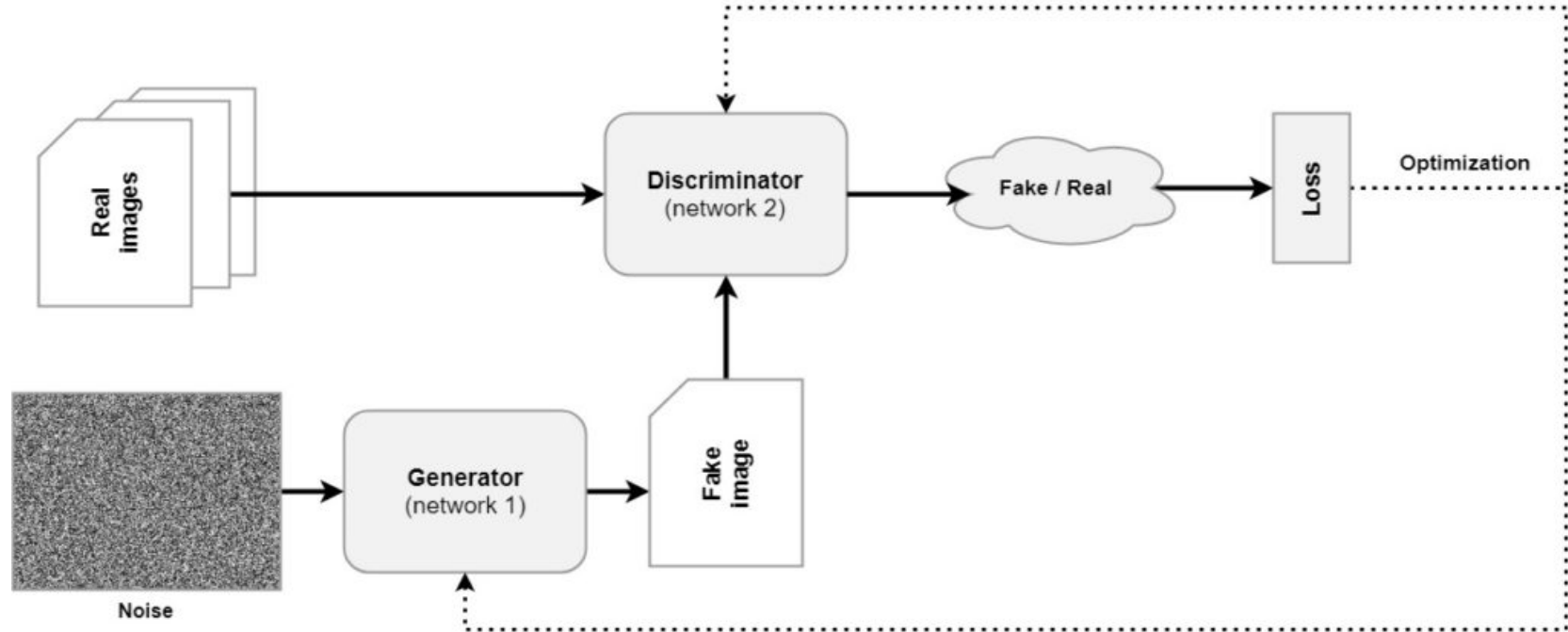


## Generative Adversarial Networks (GAN)



[https://commons.wikimedia.org/wiki/File:Generative\\_Adversarial\\_Network\\_illustration.svg](https://commons.wikimedia.org/wiki/File:Generative_Adversarial_Network_illustration.svg)

## Generative Adversarial Networks (GAN)



## GAN Deep Fakes



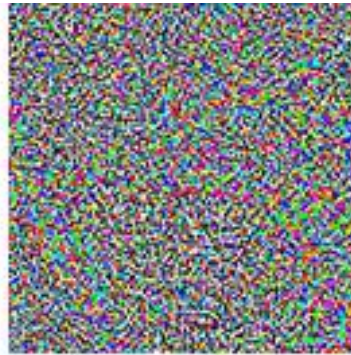
## GAN Data Poisoning attack



"panda"

57.7% confidence

+  $\epsilon$



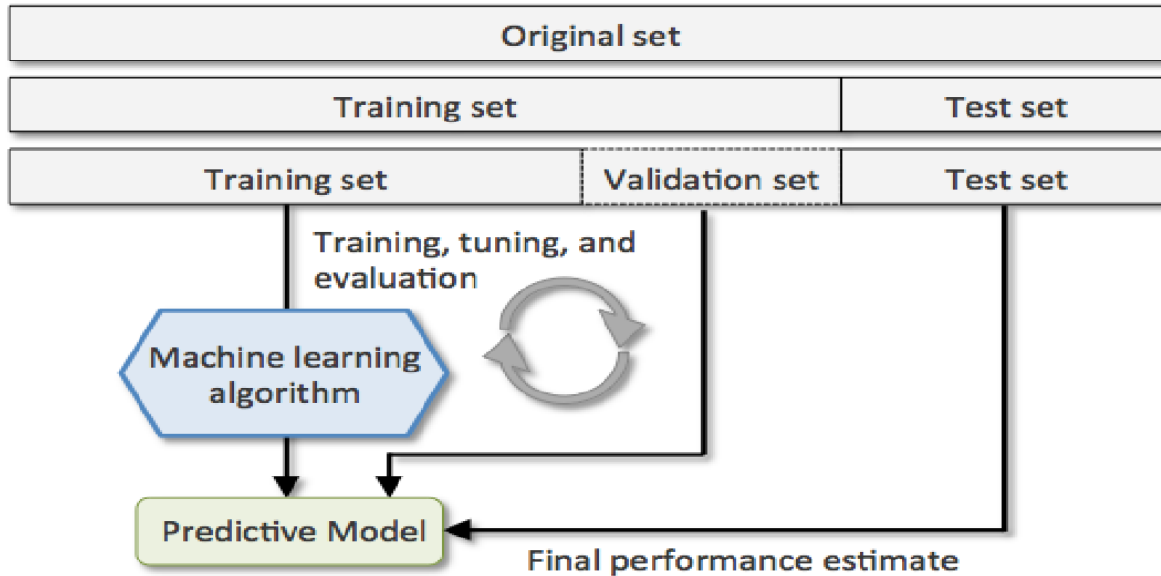
=



"gibbon"

99.3% confidence

## Model evaluation



# AI for Cybersecurity

		Predicted class	
		$P$	$N$
Actual Class	$P$	True Positives (TP)	False Negatives (FN)
	$N$	False Positives (FP)	True Negatives (TN)

## Prediction errors & Model accuracy

$$Error = \frac{FP + FN}{FP + FN + TP + TN}$$

$$Accuracy = \frac{TP + TN}{FP + FN + TP + TN} = 1 - Error$$



## Precision, Recall, F1 score

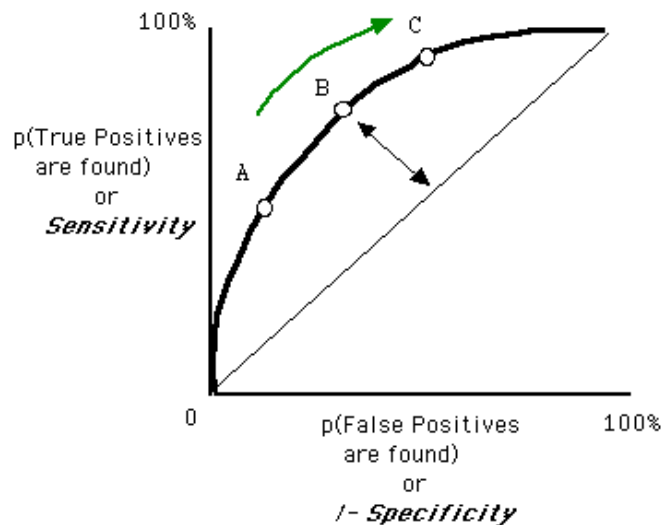
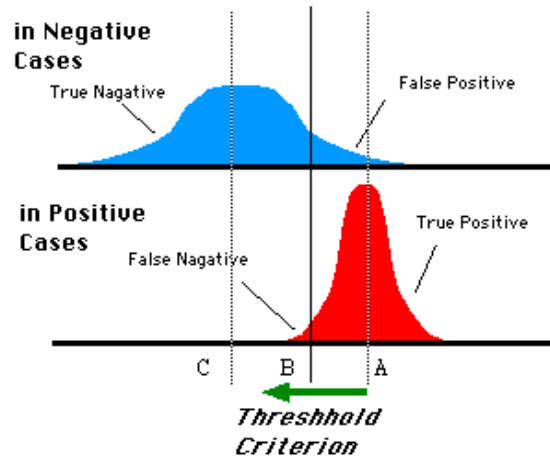
$$P = \frac{TP}{TP + FP}$$

$$R = TPR = \frac{TP}{P} = \frac{TP}{FN + TP}$$

$$F1 = 2 \times \frac{P \times R}{P + R}$$

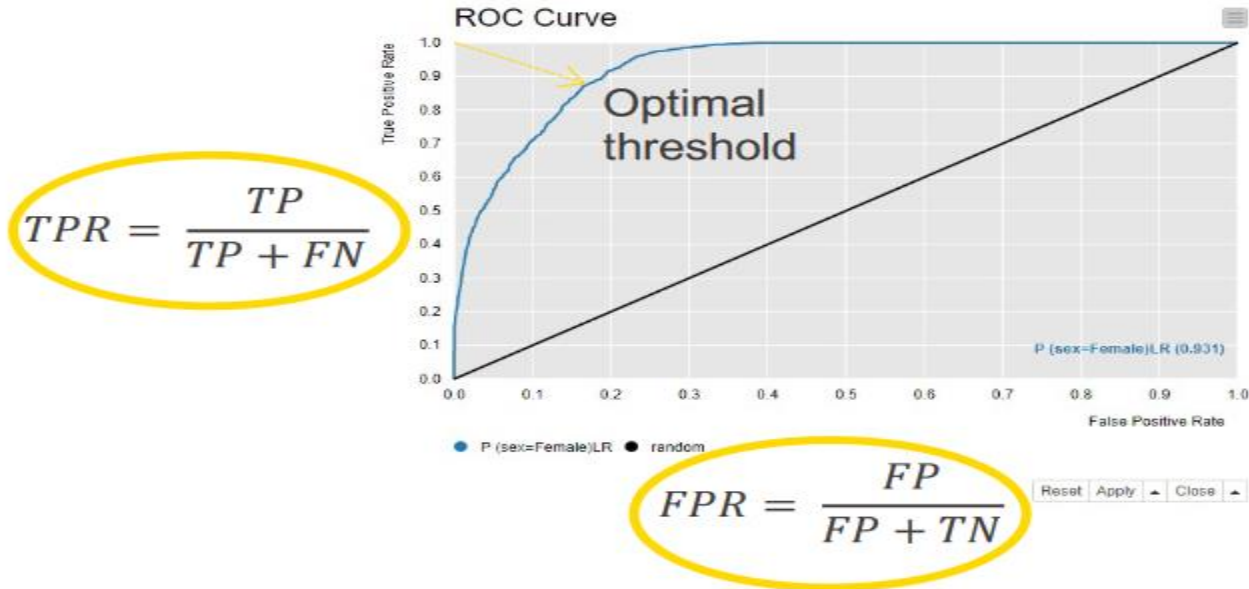
## ROC Curve

Distributions of the Observed signal strength

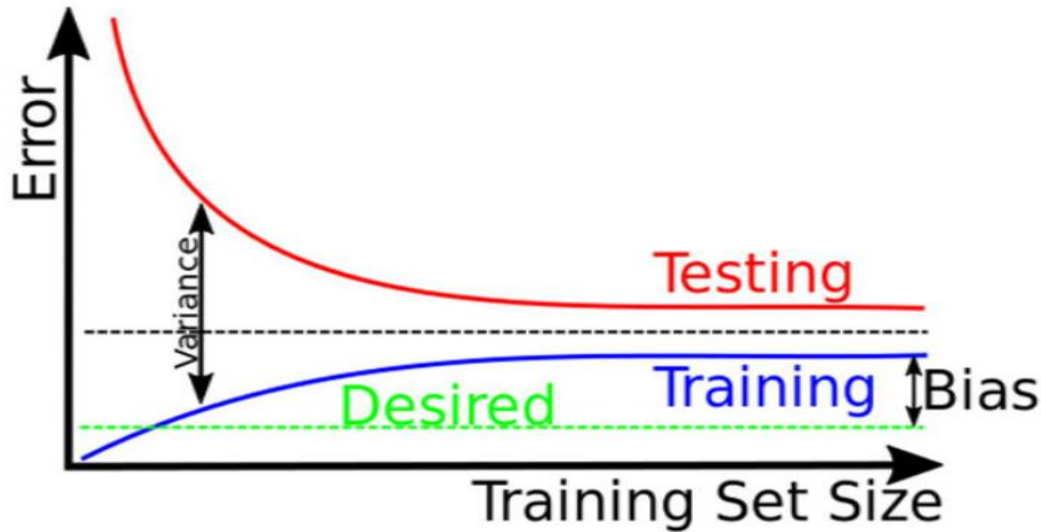


<https://upload.wikimedia.org/wikipedia/commons/5/5c/ROCfig.PNG>

## ROC Curve



## Bias – Variance tradeoff again



## Case studies

## Malware Detection

With the almost exponential increase in the number of threats associated with the daily spread of new malware, it is practically impossible to think of dealing with these threats effectively using only the analysis conducted by **human** operators.

Therefore, it is necessary to introduce algorithms that allow us to at least automate the preparatory phase of malware analysis (known as triage, deriving from the same practice adopted by doctors during the First World War, and consists of selecting for treatment the wounded that are most likely to survive). That is to say, conducting a preliminary screening of the malware to be analyzed by the malware analyst allows them to respond in a timely and effective manner to real cyber threats.

These algorithms actually take the form of the adoption of AI tools, given the dynamism that—by definition—characterizes cybersecurity. In fact, it is necessary that the machines can respond effectively, adapting themselves to the contextual changes related to the spread of unprecedented threats.

## Malware Detection

In particular, it is increasingly important that the similarities in malware behavior are correctly identified, which means that malware samples must be associated to classes or families of the same type, even if the individual malware signatures are not comparable to each other, due to, for example, the presence of polymorphic codes that alter the hash checksums accordingly.

The analysis of similarities can be carried out in an automated form, by using **clustering algorithms**.

## Malware Detection

The clustering process therefore consists of classifying together elements that show certain similarities between them.

Having defined the concept of similarity using some mathematical definitions of distance, the clustering process is thus reduced in the exploration of the various dimensions of a given data space in every direction, starting from a given point, and then aggregating together the samples that fall into a certain distance.



## Malware Detection with CNN

In the description that follows, we will show an alternative approach to malware detection that takes advantage of the typical skills of CNNs in image recognition. But in order to do this, it is first necessary to represent the executable code of the malware in the form of an image to be fed to the CNN.

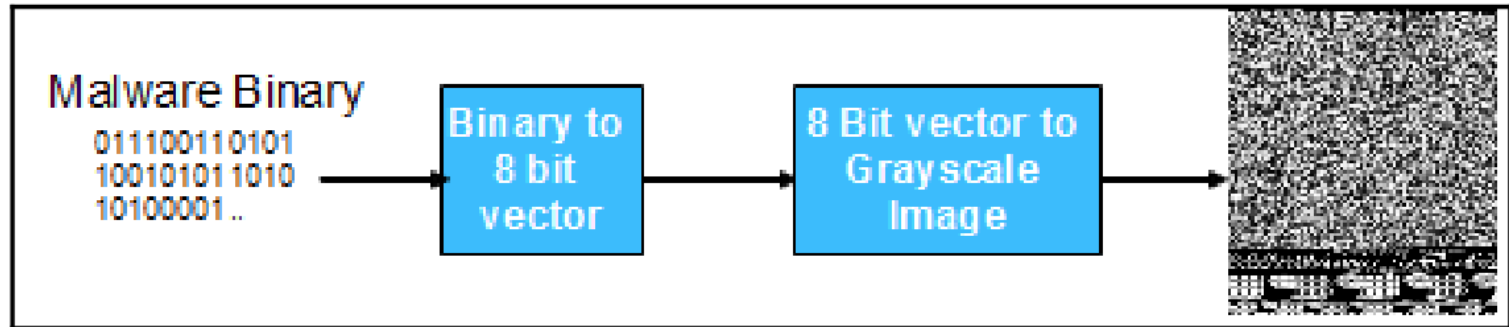
This approach was described in the paper entitled *Towards Building an Intelligent Anti-Malware System: A Deep Learning Approach using Support Vector Machine (SVM) for Malware Classification* by Abien Fred M. Agarap, in which each executable malware is treated as a binary sequence of zeros and ones, which is then translated into a gray-scale image.

In this way, it is possible to recognize the malware families based on the similarities in terms of layouts and textures existing in the images that represent them.

## Malware Detection with CNN

To perform the classification of the images, a k-NN clustering algorithm was used, in which the Euclidean distance was adopted as the metric used to represent the distance.

The experimental results obtained showed a classification rate of 99.29%, with extremely reduced computational loads:



## Botnet Detection with Gaussian distribution

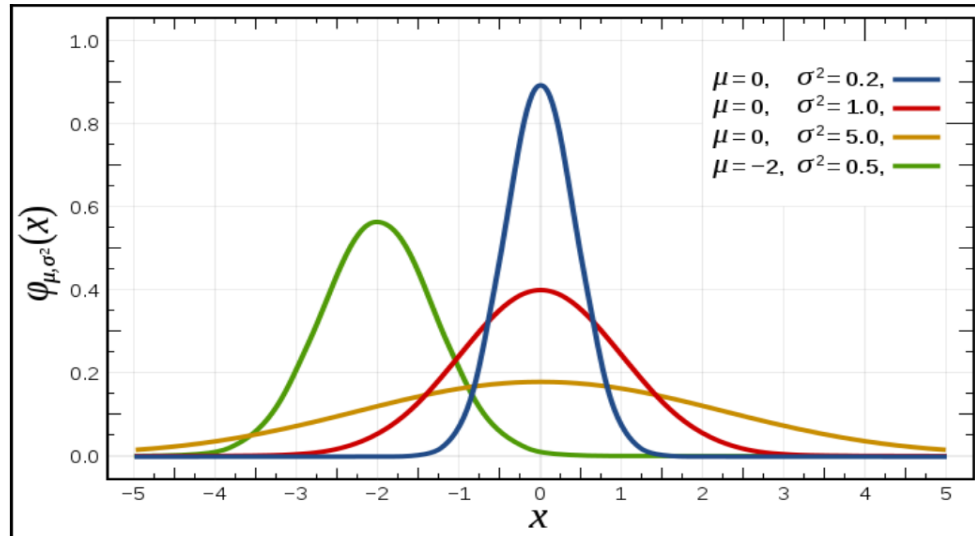
One of the most widespread approaches for detecting regularity within data distribution makes use of the Gaussian distribution of probabilities.

As we shall see, this statistical distribution presents a series of interesting characteristics that help to adequately model many natural, social, and economic phenomena.

Obviously, not all the phenomena under investigation can be represented by the Gaussian distribution (very often, as we have seen, the underlying distribution of the analyzed phenomena is unknown); however, it constitutes a reliable reference point in many cases of anomaly detection.

## Botnet Detection with Gaussian distribution

In other words, the observations, as their number increases, are distributed symmetrically (and with greater probability) around the mean,  $\mu$ :

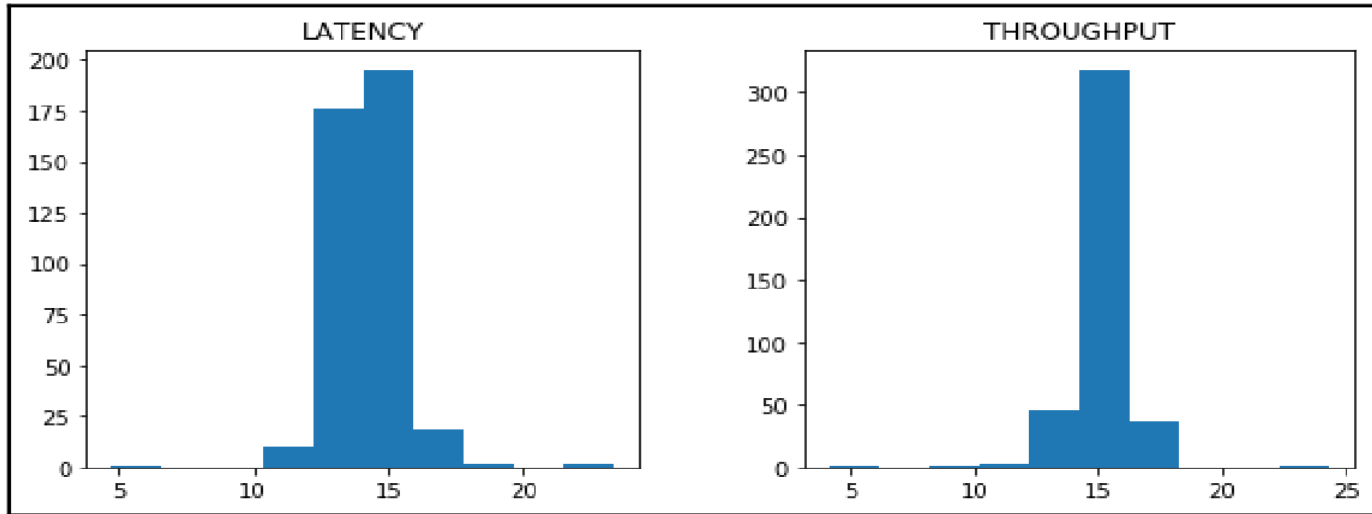


## Botnet Detection with Gaussian distribution

To use the Gaussian distribution in anomaly detection, we will have to perform the following steps:

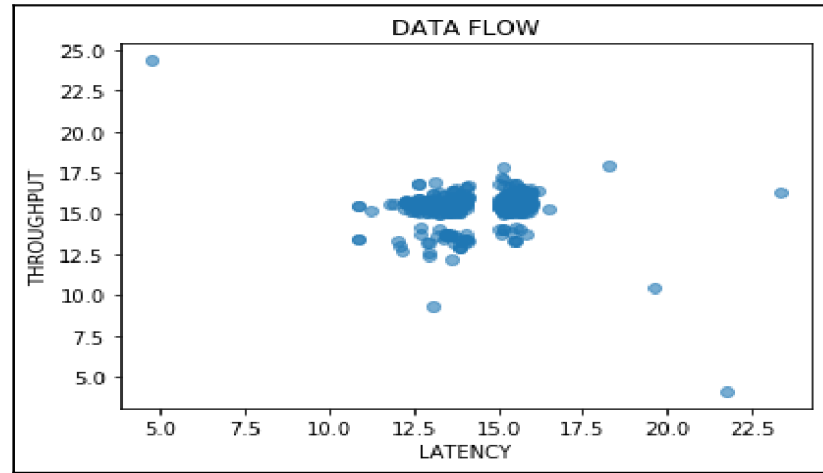
1. Assume that the features of the training set are normally distributed (this can also be verified intuitively from the visual analysis of plotted data)
2. Estimate the  $\mu$  and  $\sigma$  values, representative of the distribution
3. Choose an adequate threshold, representative of the probability that the observations are anomalous
4. Assess the reliability of the algorithm

## Botnet Detection with Gaussian distribution



At this point, we perform the data plotting on a scatter diagram, visually identifying the possible outliers:

## Botnet Detection with Gaussian distribution



As can also be seen visually, most of the observations are concentrated around the average values, except for some cases. We therefore want to verify whether the anomalous cases are real, and then we proceed to estimate the representative values,  $\mu$  and  $\sigma$ , of the underlying Gaussian distribution:

# Learn more

