ACTIVATION FUNCTIONS

SHAYAN JAFARI

October 24, 2024

Thank you to Professor Vahid Ghorbani.

1 History of artificial intelligence

2 perceptron

3 activation functions

Figure 1: Create a visually stunning image that illustrates the impact of artificial intelligence intertwined with mathematical functions and step functions in a natural setting. The scene should blend elements of nature, such as trees and rivers, with abstract representations of mathematical equations and AI components, like neural networks. Use vibrant colors and dynamic shapes to convey the integration of technology and nature, showcasing harmony between the two. –ar 16:9 (Flux-schnell).

History of artificial intelligence

1 Introduction to Artificial Neural Networks

Birds inspired us to fly, burdock plants inspired Velcro, and nature has inspired countless more inventions. It seems only logical, then, to look at the brain's architec- ture for inspiration on how to build an intelligent machine. This is the logic that sparked artificial neural networks (ANNs), machine learning models inspired by the networks of biological neurons found in our brains. However, although planes were inspired by birds, they don't have to flap their wings to fly. Similarly, ANNs have gradually become quite different from their biological cousins. Some researchers even argue that we should drop the biological analogy altogether (e.g., by saying "units" rather than "neurons"), lest we restrict our creativity to biologically plausible systems.1 ANNs are at the very core of deep learning. They are versatile, powerful, and scalable, making them ideal to tackle large and highly complex machine learning tasks such as classifying billions of images (e.g., Google Images), powering speech recognition services (e.g., Apple's Siri), recommending the best videos to watch to hundreds of millions of users every day (e.g., YouTube), or learning to beat the world champion at the game of Go (DeepMind's AlphaGo). The first part of this chapter introduces artificial neural networks, starting with a quick tour of the very first ANN architectures and leading up to multilayer percep- trons, which are heavily used today (other architectures will be explored in the next chapters). In the second part, we will look at how to implement neural networks using TensorFlow's Keras API. This is a beautifully designed and simple high-level API for building, training, evaluating, and running neural networks. But don't be fooled by its simplicity: it is expressive and flexible enough to let you build a wide variety of neural network architectures. In fact, it will probably be sufficient for most of your use cases. And should you ever need extra flexibility, you can always write custom Keras components using its lower-level API, or even use TensorFlow directly

1.1 From Biological to Artificial Neurons

Surprisingly, ANNs have been around for quite a while: they were first introduced back in 1943 by the neurophysiologist Warren McCulloch and the mathematician Walter Pitts. In their landmark paper2 "A Logical Calculus of Ideas Immanent in Nervous Activity", McCulloch and Pitts presented a simplified computational model of how biological neurons might work together in animal brains to perform complex computations using propositional logic. This was the first artificial neural network architecture. Since then many other architectures have been invented, as you will see. The early successes of ANNs led to the widespread belief that we would soon be conversing with truly intelligent machines. When it became clear in the 1960s that this promise would go unfulfilled (at least for quite a while), funding flew elsewhere, and ANNs entered a long winter. In the early 1980s, new architectures were invented and better training techniques were developed, sparking a revival of interest in connectionism, the study of neural networks. But progress was slow, and by the 1990s other powerful machine learning techniques had been invented, such as support vector machines (see Chapter 5). These techniques seemed to offer better results and stronger theoretical foundations than ANNs, so once again the study of neural networks was put on hold. We are now witnessing yet another wave of interest in ANNs. Will this wave die out like the previous ones did? Well, here are a few good reasons to believe that this time is different and that the renewed interest in ANNs will have a much more profound impact on our lives:

• There is now a huge quantity of data available to train neural networks, and ANNs frequently outperform other ML techniques on very large and complex problems.

• The tremendous increase in computing power since the 1990s now makes it possible to train large neural networks in a reasonable amount of time. This isin part due to Moore's law (the number of components in integrated circuits has doubled about every 2 years over the last 50 years), but also thanks to the gaming industry, which has stimulated the production of powerful GPU cards by the millions. Moreover, cloud platforms have made this power accessible to everyone.

• The training algorithms have been improved. To be fair they are only slightly different from the ones used in the 1990s, but these relatively small tweaks have had a huge positive impact.

• Some theoretical limitations of ANNs have turned out to be benign in practice. For example, many people thought that ANN training algorithms were doomed because they were likely to get stuck in local optima, but it turns out that this is not a big problem in practice, especially for larger neural networks: the local optima often perform almost as well as the global optimum.

• ANNs seem to have entered a virtuous circle of funding and progress. Amazing products based on ANNs regularly make the headline news, which pulls more and more attention and funding toward them, resulting in more and more progress and even more amazing products.

1.2 Biological Neurons

Before we discuss artificial neurons, let's take a quick look at a biological neuron (represented in Figure 10-1). It is an unusual-looking cell mostly found in animal brains. It's composed of a cell body containing the nucleus and most of the cell's complex components, many branching extensions called dendrites, plus one very long extension called the axon. The axon's length may be just a few times longer than the cell body, or up to tens of thousands of times longer. Near its extremity the axon splits off into many branches called telodendria, and at the tip of these branches are minus- cule structures called synaptic terminals (or simply synapses), which are connected to the dendrites or cell bodies of other neurons.3 Biological neurons produce short electrical impulses called action potentials (APs, or just signals), which travel along the axons and make the synapses release chemical signals called neurotransmitters. When a neuron receives a sufficient amount of these neurotransmitters within a few milliseconds, it fires its own electrical impulses (actually, it depends on the neurotransmitters, as some of them inhibit the neuron from firing).

Thus, individual biological neurons seem to behave in a simple way, but they're organized in a vast network of billions, with each neuron typically connected to thousands of other neurons. Highly complex computations can be performed by a network of fairly simple neurons, much like a complex anthill can emerge from the combined efforts of simple ants. The architecture of biological neural

Figure 2: Create a captivating image that illustrates how humans draw inspiration from nature to invent various tools. The scene should depict a harmonious blend of natural elements, such as plants and animals, with human figures working creatively to design and fabricate tools. This could include individuals observing animal behaviors, studying plant structures, and crafting tools made from natural materials. The background should showcase a vibrant, lush landscape filled with diverse flora and fauna, emphasizing the connection between nature and human innovation. –ar 16:9 (Flux-schnell).

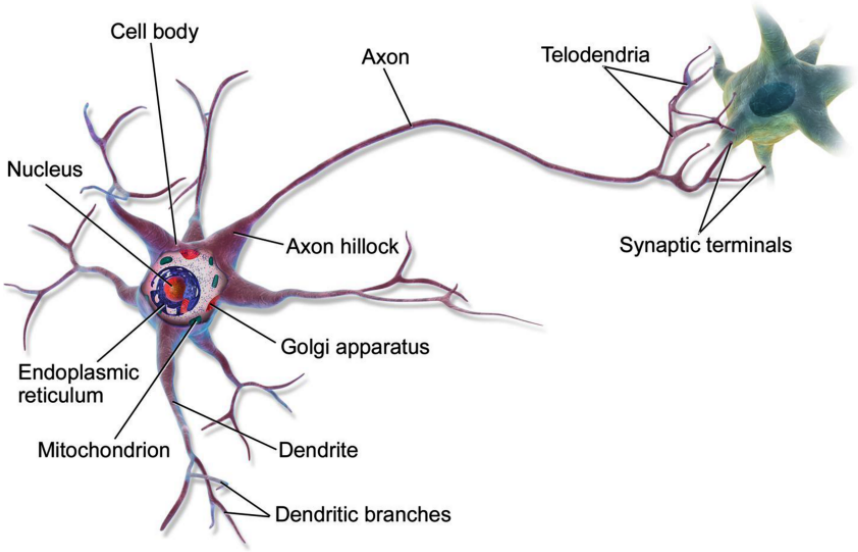


Figure 3: A biological neuron

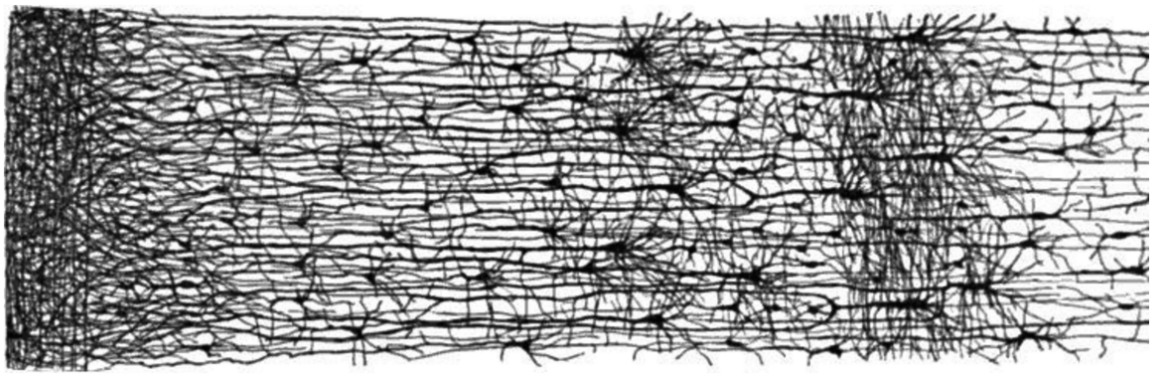


Figure 4: Multiple layers in a biological neural network (human cortex)

networks (BNNs)5 is the subject of active research, but some parts of the brain have been map- ped. These efforts show that neurons are often organized in consecutive layers, espe- cially in the cerebral cortex (the outer layer of the brain), as shown in Figure 4.

1.3 Logical Computations with Neurons

McCulloch and Pitts proposed a very simple model of the biological neuron, which later became known as an artificial neuron: it has one or more binary (on/off) inputs and one binary output. The artificial neuron activates its output when more than a certain number of its inputs are active. In their paper, McCulloch and Pitts showed that even with such a simplified model it is possible to build a network of artificial neurons that can compute any logical proposition you want. To see how such a network works, let's build a few ANNs that perform various logical computations (see Figure 5), assuming that a neuron is activated when at least two of its input connections are active.

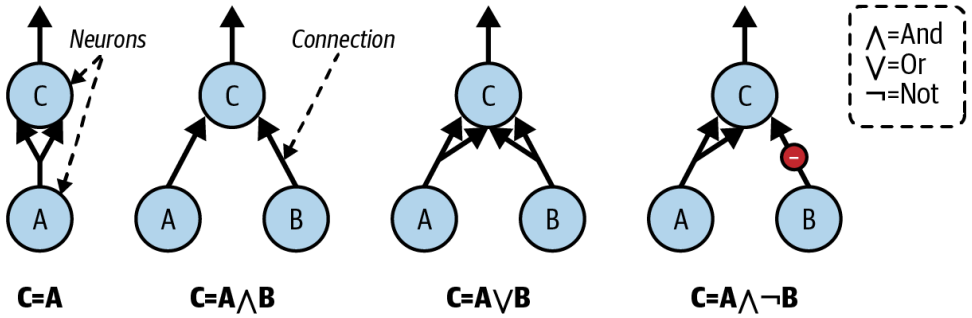
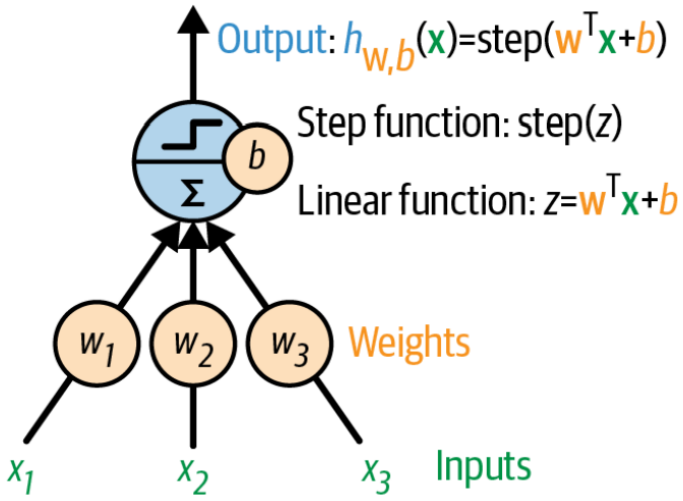


Figure 5: ANNs performing simple logical computations

1 percpetron

The perceptron is one of the simplest ANN architectures, invented in 1957 by Frank Rosenblatt. It is based on a slightly different artificial neuron (see Figure 6) called a threshold logic unit (TLU), or sometimes a linear threshold unit (LTU). The inputs and output are numbers (instead of binary on/off values), and each input connection is associated with a weight. The TLU first computes a linear function of its inputs: z = 'w1 x1 + w2 x2 + + wn xn + b = w x + b'. Then it applies a step function to the result: hw(x) = step(z). So it's almost like logistic regression, except it uses a step function instead of the logistic function . Just like in logistic regression, the model parameters are the input weights w and the bias term b.

The most common step function used in perceptrons is the Heaviside step functionSometimes the



sign function is used instead.

$$\text{heaviside}(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases} \qquad \text{sgn}(z) = \begin{cases} -1 & \text{if } z < 0 \\ 0 & \text{if } z = 0 \\ +1 & \text{if } z > 0 \end{cases}$$

A single TLU can be used for simple linear binary classification. It computes a linear function of its inputs, and if the result exceeds a threshold, it outputs the positive class. Otherwise, it outputs the negative class. This may remind you of logistic regres- sion or linear SVM classification . You could, for example, use a single TLU to classify iris flowers based on petal length and width. Training such a TLU would require finding the right values for w1 , w2 , and b (the training algorithm is discussed shortly). A perceptron is composed of one or more TLUs organized in a single layer, where every TLU is connected to every input. Such a layer is called a fully connected layer, or a dense layer. The inputs constitute the input layer. And since the layer of TLUs produces the final outputs, it is called the output layer. For example, a perceptron with two inputs and three outputs is represented in Figure 6

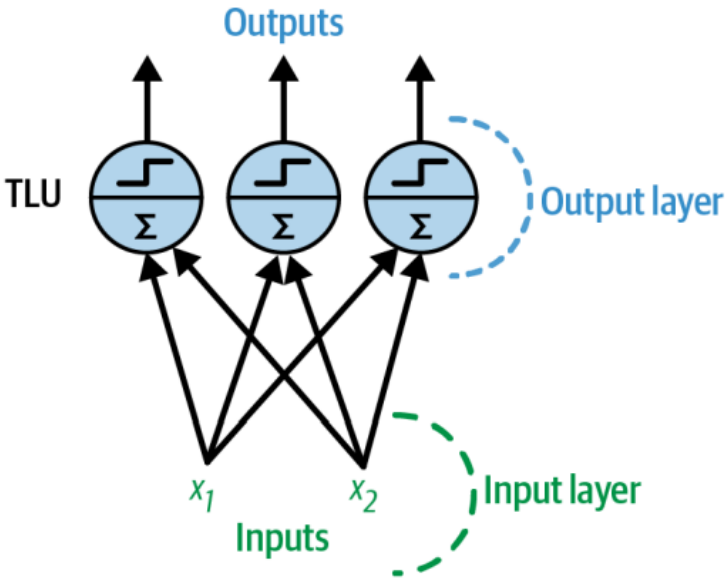


Figure 6: Architecture of a perceptron with two inputs and three output neurons

1.1 How to make a PRECPETRON

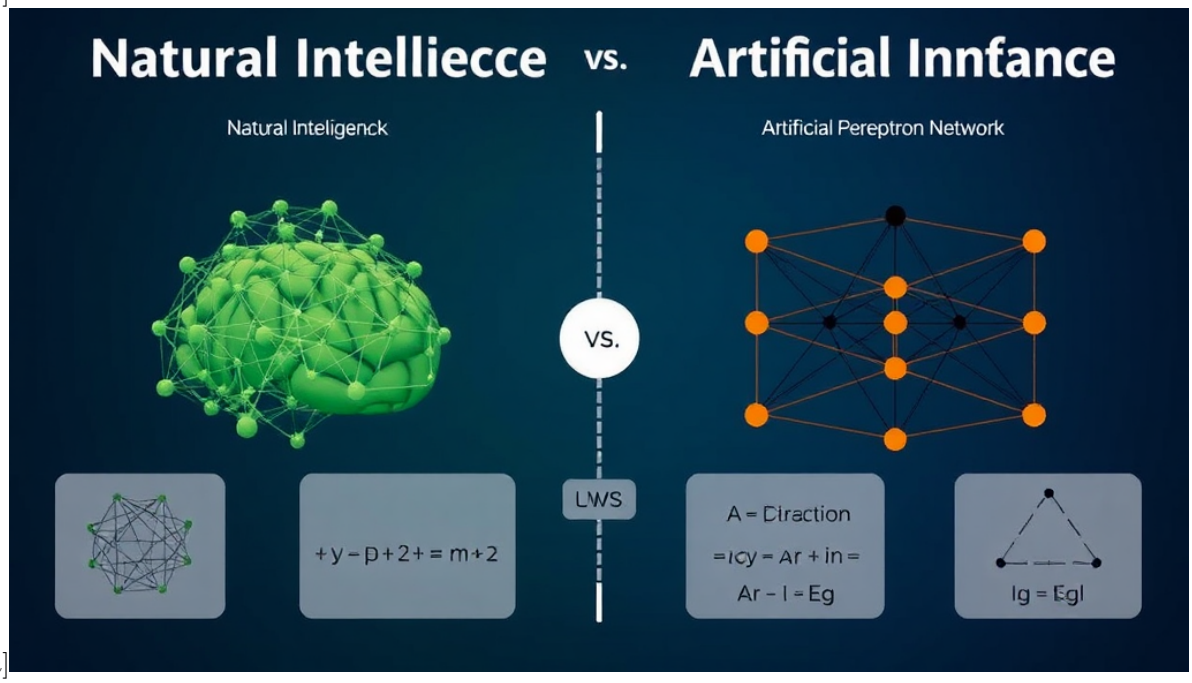
```
1  import numpy as np
2  from sklearn.datasets import load_iris
3  from sklearn.linear_model import Perceptron
4  iris = load_iris(as_frame=True)
5  X = iris.data[["petal length (cm)", "petal width (cm)"]].values
6  y = (iris.target == 0) # Iris setosa
7  per_clf = Perceptron(random_state=42)
8  per_clf.fit(X, y)
9  X_new = [[2, 0.5], [3, 1]]
10 y_pred = per_clf.predict(X_new)

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS        Python

guts@shayan:~$ /bin/python3 /home/guts/precpetron.py
guts@shayan:~$
```

Scikit-Learn provides a Perceptron class that can be used pretty much as you would expect—for example, on the iris dataset

You may have noticed that the perceptron learning algorithm strongly resembles sto- chastic gradient descent . In fact, Scikit-Learn's Perceptron class is equivalent to using an SGDClassifier with the following hyperparameters:
loss="perceptron", learning$_rate$ = "$constant$", $eta0 = 1(thelearningrate), andpenalty = None(noregularization)$.

[h]

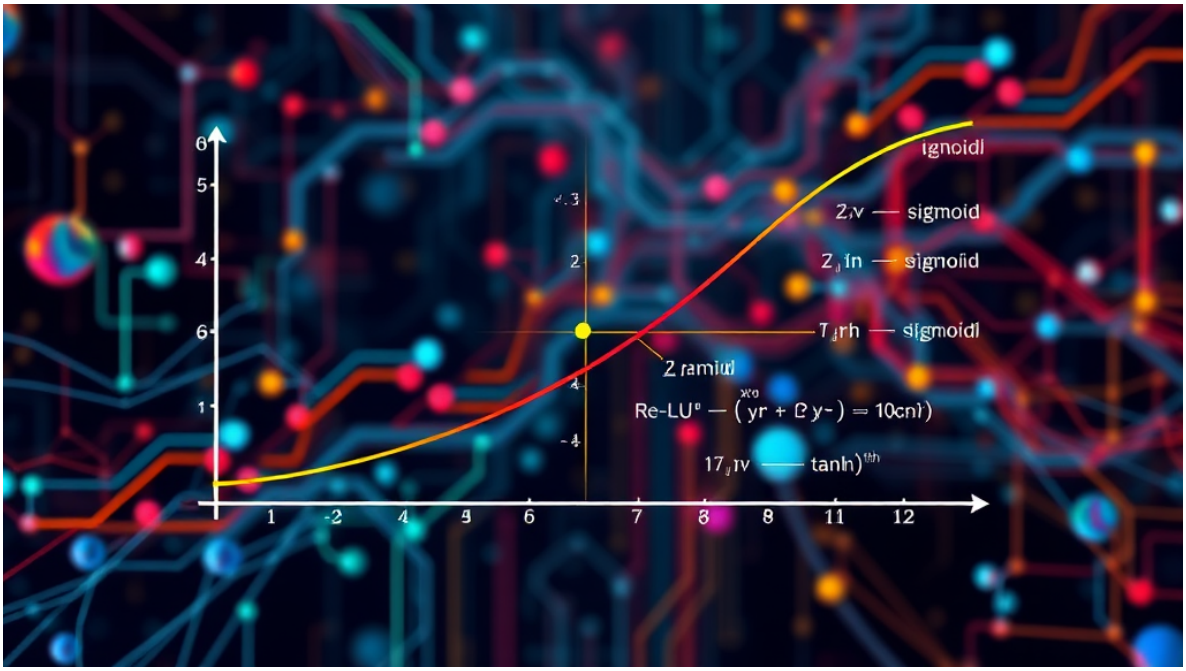
[h]



Activation functions

1 Introduction to activation functions

The activation function of a node in an artificial neural network is a function that calculates the output of the node based on its individual inputs and their weights. Nontrivial problems can be solved using only a few nodes if the activation function is nonlinear.
The function is called the activation function: when the artificial neurons are TLUs, it is a step function (we will discuss other activation functions shortly).
Modern activation functions include the smooth version of the ReLU, the GELU, which was used in the 2018 BERT model the logistic (sigmoid) function used in the 2012 speech recognition model developed by Hinton et al, the ReLU used in the 2012 AlexNet computer vision model and in the 2015 ResNet model.



1.1 Types of activation functions(tanh)

The hyperbolic tangent function: tanh(z) = 2(2z) – 1 Just like the sigmoid function, this activation function is S-shaped, continuous, and differentiable, but its output value ranges from –1 to 1 (instead of 0 to 1 in the case of the sigmoid function). That range tends to make each layer's output more or less centered around 0 at the beginning of training, which often helps speed up convergence.

1.2 Types of activation functions(ReLU)

The rectified linear unit function: ReLU(z) = max(0, z) The ReLU function is continuous but unfortunately not differentiable at z = 0 (the slope changes abruptly, which can make gradient descent bounce around), and its derivative is 0 for z ¡ 0. In practice, however, it works very well and has the advantage of being fast to compute, so it has become the default.11 Importantly, the fact that it does not have a maximum output value helps reduce some issues during gradient descent

1.3 Comparison of activation functions

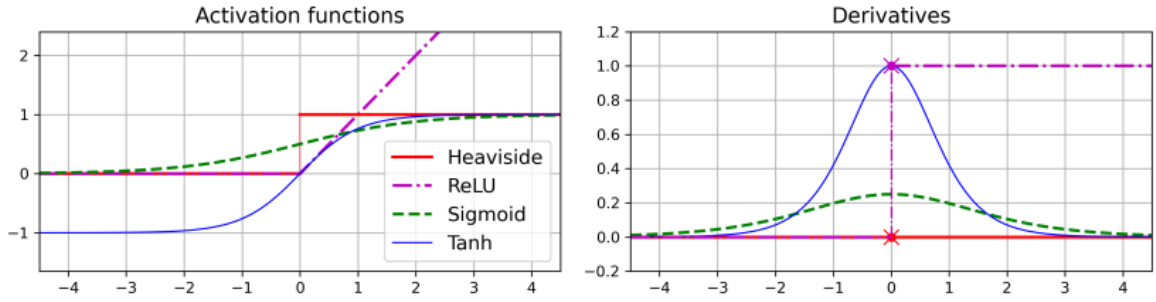


Table 5. Final ratio of correctly classified images

	sigmoid	tanh	relu	leaky_relu	swish	softsign	softplus
Final accuracy	0.6166	0.6757	0.7179	0.7295	0.6989	0.6901	0.6598

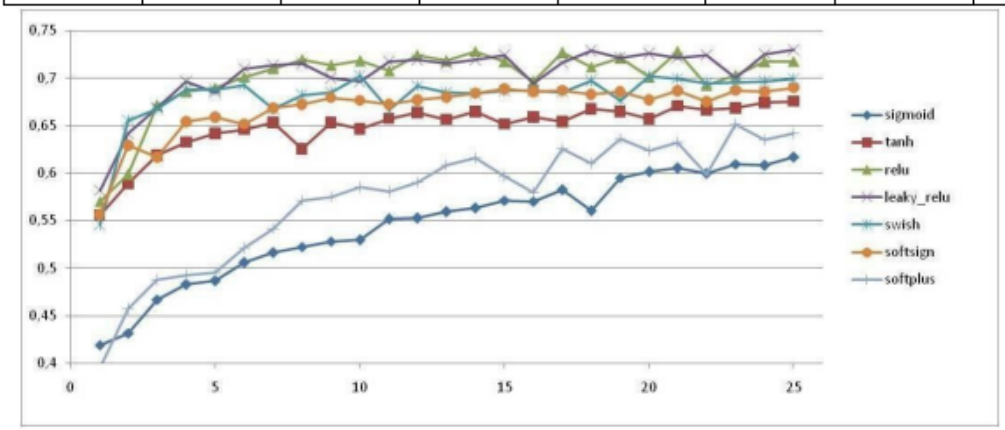


Fig. 12. Network accuracy over training for each activation function

References

1.Aurélien$_G$éron$_H$ands$_O$n$_M$achine$_L$earning$_w$ith$_S$cikit$_L$earn,$_K$ eras

2.Review and Comparison of Commonly Used Activation Functions for Deep Neural Networks Tomasz Szandała 1 1Wroclaw University of Science and Technology Wroclaw, Poland

3.chatgpt(4-o,flux)

hope my words were useful.
Thank you for your kindness.

THE END.