Deep Feedforward Networks

 Romeo Kienzler

Let’s get started with Deep Feedforward Networks

Example

Part No.	Max Temp.	Min Temp.	Max Vibration	Asperity
100	35	35	12	0.32
101	46	35	21	0.34
130	56	46	3412	12.42
131	58	48	3542	13.43

This is an example of a hypothetical sensor data of a production machine.

Example

Part No.	Max Temp.	Min Temp.	Max Vibration	Asperity
100	35	35	12	0.32
101	46	35	21	0.34
130	56	46	3412	12.42
131	58	48	3542	13.43

Each row contains information about sensor values during production of a particular part number

Example

Part No.	Max Temp.	Min Temp.	Max Vibration	Asperity
100	35	35	12	0.32
101	46	35	21	0.34
130	56	46	3412	12.42
131	58	48	3542	13.43

Example

Part No.	Max Temp.	Min Temp.	Max Vibration	Asperity
100	35	35	12	0.32
101	46	35	21	0.34
130	56	46	3412	12.42
131	58	48	3542	13.43

Example

Part No.	Max Temp.	Min Temp.	Max Vibration	Asperity
100	35	35	12	0.32
101	46	35	21	0.34
130	56	46	3412	12.42
131	58	48	3542	13.43

Example

Part No.	Max Temp.	Min Temp.	Max Vibration	Asperity
100	35	35	12	0.32
101	46	35	21	0.34
130	56	46	3412	12.42
131	58	48	3542	13.43

Example

Part No.	Max Temp.	Min Temp.	Max Vibration	Asperity
100	35	35	12	0.32
101	46	35	21	0.34
130	56	46	3412	12.42
131	58	48	3542	13.43

Example

Part No.	Max Temp.	Min Temp.	Max Vibration	Asperity
100	35	35	12	0.32
101	46	35	21	0.34
130	56	46	3412	12.42
131	58	48	3542	13.43

Example

Part No.	Max Temp.	Min Temp.	Max Vibration	Asperity
100	35	35	12	0.32
101	46	35	21	0.34
130	56	46	3412	12.42
131	58	48	3542	13.43

Example

Part No.	Max Temp.	Min Temp.	Max Vibration	Asperity
100	35	35	12	0.32
101	46	35	21	0.34
130	56	46	3412	12.42
131	58	48	3542	13.43

Example

Part No.	Max Temp.	Min Temp.	Max Vibration	Asperity
100	35	35	12	0.32
101	46	35	21	0.34
130	56	46	3412	12.42
131	58	48	3542	13.43

So our goal is to predict Asperity, which is highly correlated with the fact of the part is healthy or broken based on the observed sensor values

Quiz

Which column can be used to create a simple rule for this?

Which column can be used to create a simple rule for this?

Example

Part No.	Max Temp.	Min Temp.	Max Vibration	Asperity
100	35	35	12	0.32
101	46	35	21	0.34
130	56	46	3412	12.42
131	58	48	3542	13.43

First of all we observe that all three sensor columns are aggregations of a time series. So we are loosing information. But this is fine for this example. We notice that using the vibration aggregation will give us the simplest and clearest rule, so let’s code it

a3_m1_v2_deep_feedforward_neural_networks_ex1_1

 Romeo Kienzler

So let’s start with creating a data point. The first field is part number. Let’s take 100. The next field is max temp which we set to 35. Then min temp which we set to 35 as well. And finally let’s set max vibration to 12. After producing the part asperity is measured. So we set it to 0.32. Now let’s copy this four times to reflect our four measurements. As in the examples of the slides I’ll now update all the values accordingly. No let’s implement our simple rule to predict asperity based on a datapoint. Obviously, max vibration has the largest impact on the outcome. Therefore if max vibration is greater than a100 we return 13, and 0.33 otherwise. If we now test this function we’ll get quite good results already.

Machine Learning

$$y = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3$$

So now it turns out that we can achieve this and even do better without hardcoding a rule. This formula here is called a linear regression model. Regression because it predicts a continuous value y based on a linear combination between parameters w and data points x

a3_m1_v2_deep_feedforward_neural_networks_ex1_2

 Romeo Kienzler

So now let’s create our first machine learning algorithm called linear regression in python. So remember that we have to return a linear combination between your input fields and some parameters double u. And that double u one is the offset of the line if we plot this. So if we run this now, we obviously run into an error because we haven’t defined double u yet. So let’s do this now. And let’s try to choose those parameters in a way that it resembles our manually created rule from before. So if we set everything to zero of course we are getting zero as a result. Now let’s try to do better by cheating and taking the values from the data set itself in order to come up with some good values for double u. So as you can see we are getting relatively close here. Of course, in a real world scenario a learning algorithm will choose an optimal assignment for the weights double u for us.

Quiz

$$y = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3$$

Write this formula as dot product of two vectors

Quiz

$$x_0 = 1$$

 Romeo Kienzler

So maybe you have noticed that there is one x value missing in order to have equal length vectors, so let’s define x zero as one

Quiz

$$x_0 = 1$$

$$w * x$$

so now we can multiple w by x because both vectors have the same size

Quiz

$$x_0 = 1$$

$$w * x = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \end{bmatrix} * \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

if we now just write down what we've learned previously we'll come up with the following

Quiz

$$x_0 = 1$$

$$w * x = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \end{bmatrix} * \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = w_0 * 1 + w_1 x_1 + w_2 x_2 + w_3 x_3$$

so doesn’t this look like linear regression?

Quiz

$$x_0 = 1$$

$$w * x = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \end{bmatrix} * \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = w_0 * 1 + w_1 x_1 + w_2 x_2 + w_3 x_3 = y$$

just put the y at the end and you can see it. so we can express linear regression as a single vector dot product operation, isn’t that cool?

Example

Part No.	Max Temp.	Min Temp.	Max Vibration	Asperity
100	35	35	12	0.32
101	46	35	21	0.34
130	56	46	3412	12.42
131	58	48	3542	13.43

Now it turns out that parts with an Asperity greater than one are not usable, so they are broken.

Example

Part No.	Max Temp.	Min Temp.	Max Vibration	Broken
100	35	35	12	0
101	46	35	21	0
130	56	46	3412	1
131	58	48	3542	1

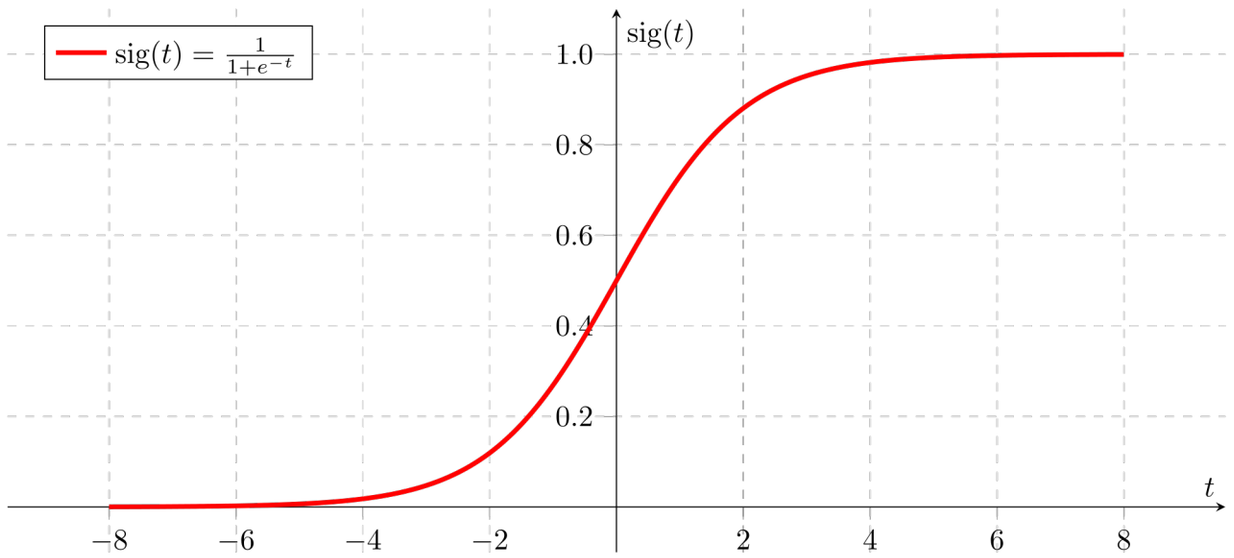
Let’s change our table to reflect this. And let’s code it…

a3_m1_v2_deep_feedforward_neural_networks_ex1_3

 Romeo Kienzler

So now let’s change this regression data set into a binary classification one. Then our rule get’s much more simple and also much more precise. And now let’s change our linear regression model to a logistic regression model. In order to do this we just need to ad a sigmoid computation step to our result. Now the result is in the range between zero and one and by setting a good threshold we end up with a binary classification model. So this looks brilliant.

Sigmoid



By MartinThoma (Own work) [CC0], via Wikimedia Commons

So the logistic sigmoid function squashes any range between minus and plus infinity to a range between zero and one.

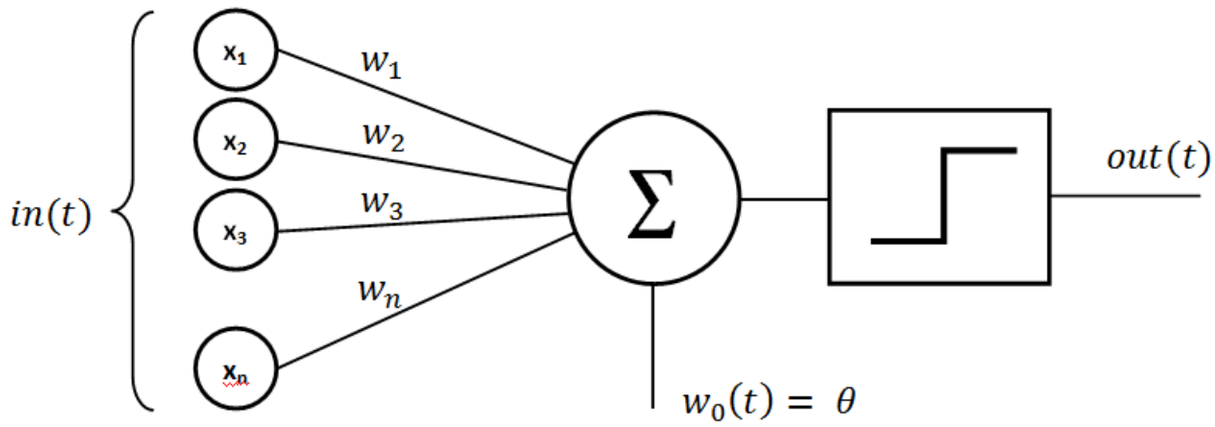
Logistic Regression

$$sigmoid(x) = \frac{1}{1+e^{-x}}$$

$$y = sigmoid(w_0 + w_1x_1 + w_2x_2 + w_3x_3)$$

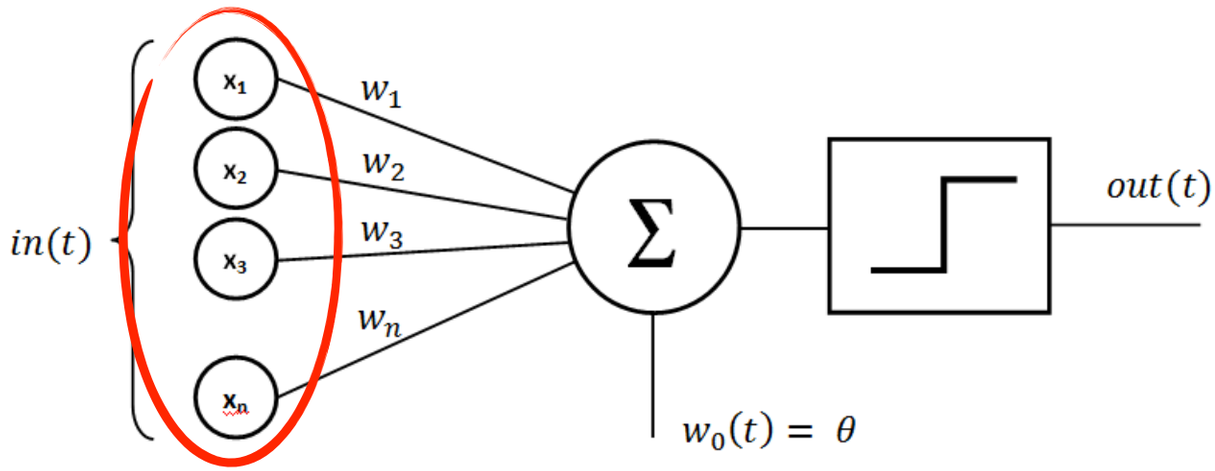
This means we can easily turn our linear regression model into a logistic regression model and by doing so we created a binary classifier. ok wait, not yet, we have do define a threshold between zero and one in order to get a clear separation between classes instead of a continuous prediction of values. So why am I telling you all this in a lecture on neural networks? so let’s have a look at the most simple neural network

Perceptron



The so-called perceptron.

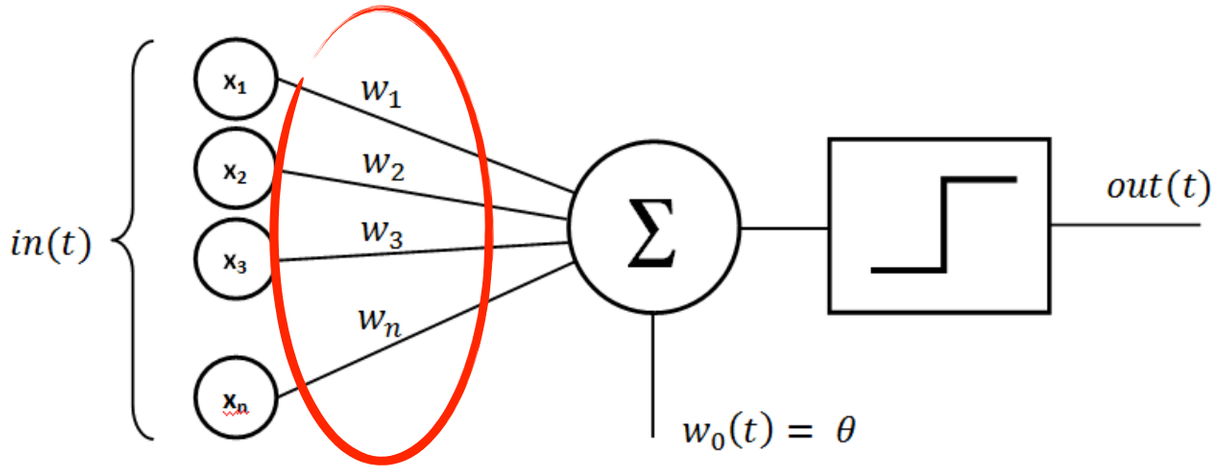
Perceptron



By Mayranna (Own work) [CC BY-SA 3.0 (https://creativecommons.org/licenses/by-sa/3.0)], via Wikimedia Commons

The idea is that your input vector gets multiplied with …

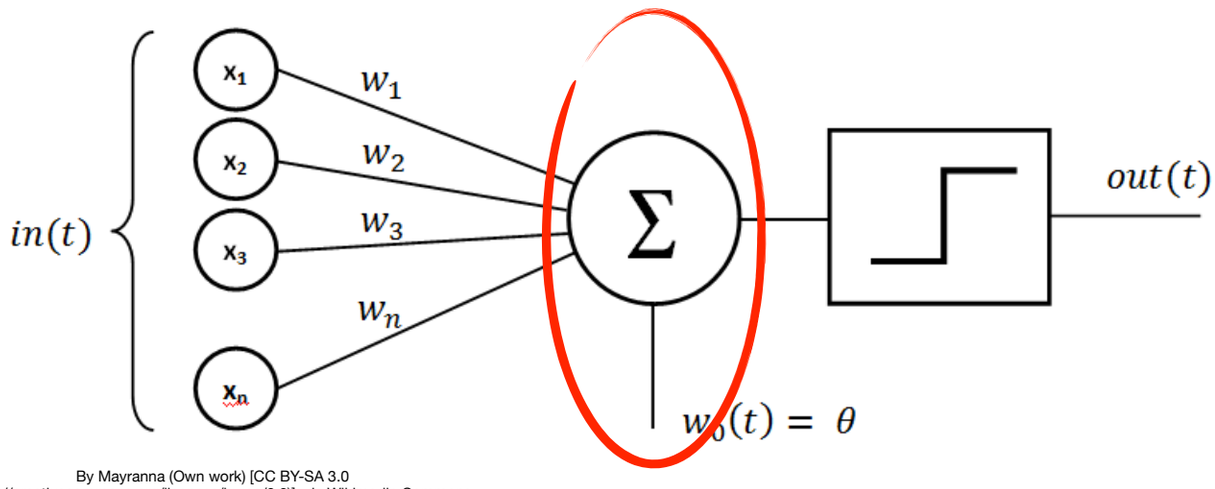
Perceptron



By Mayranna (Own work) [CC BY-SA 3.0 (https://creativecommons.org/licenses/by-sa/3.0)], via Wikimedia Commons

…a weights vector..

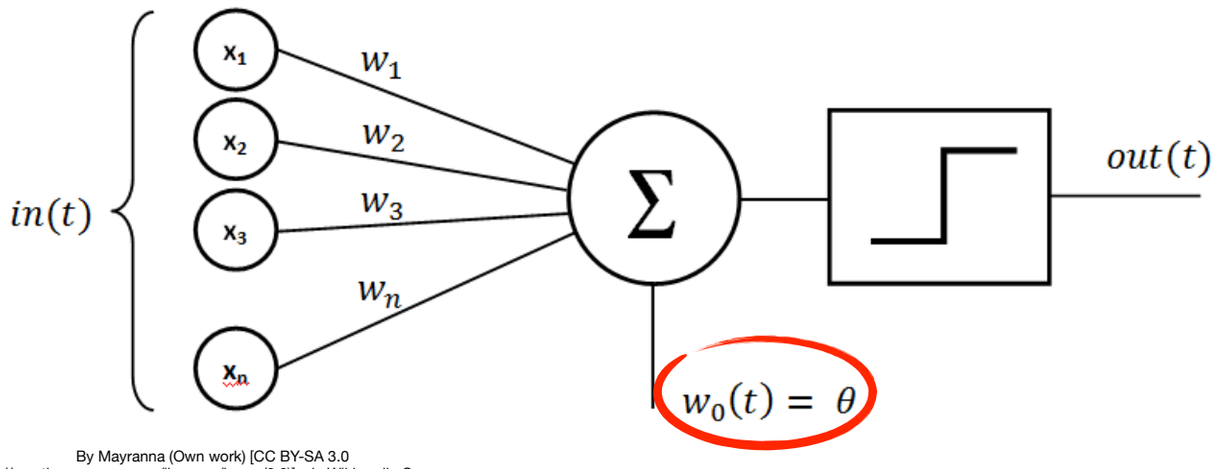
Perceptron



By Mayranna (Own work) [CC BY-SA 3.0 (https://creativecommons.org/licenses/by-sa/3.0)], via Wikimedia Commons

…and finally the sum is taken.

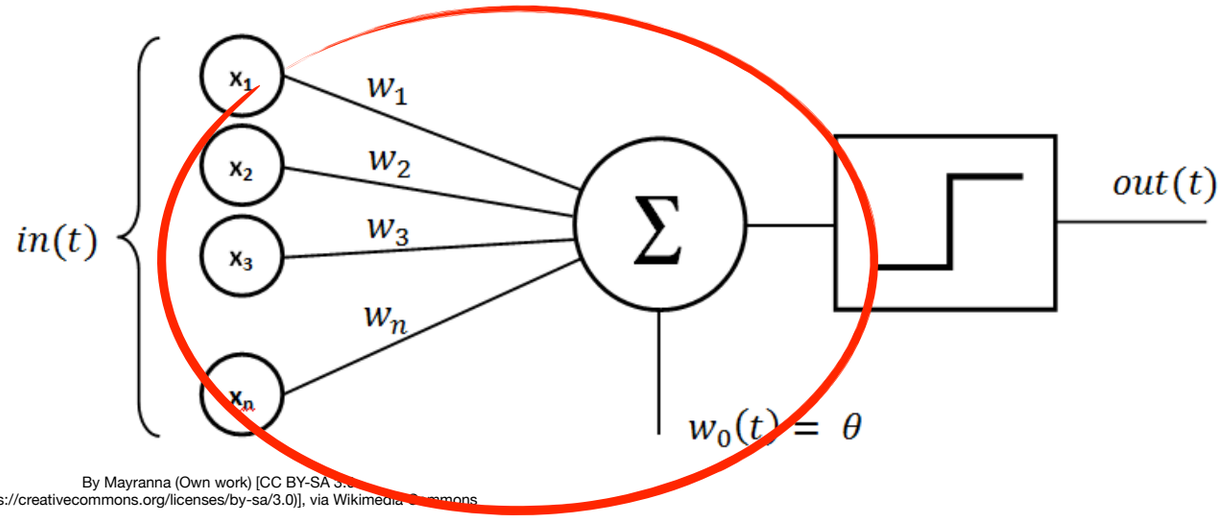
Perceptron



By Mayranna (Own work) [CC BY-SA 3.0 (https://creativecommons.org/licenses/by-sa/3.0)], via Wikimedia Commons

Note that even the bias term is taken into the sum.

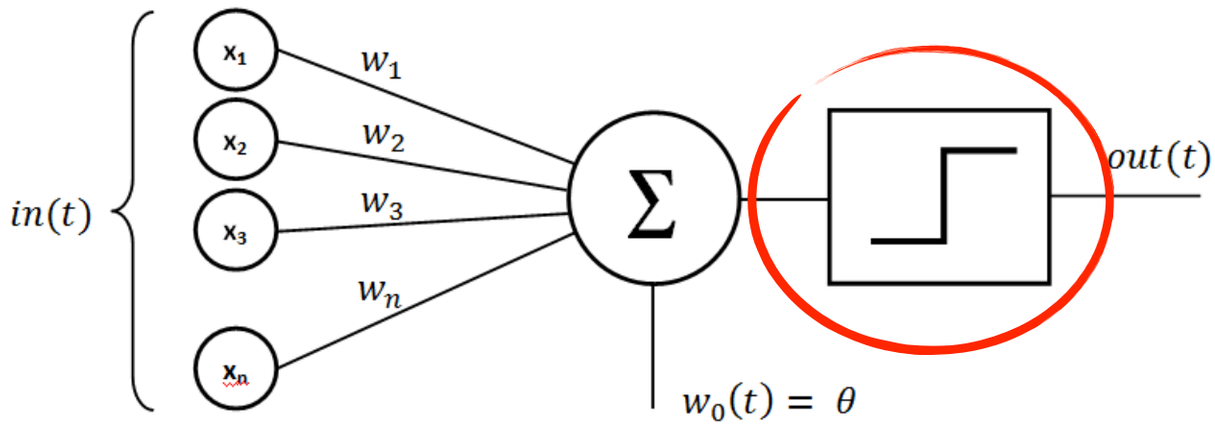
Perceptron



By Mayranna (Own work) [CC BY-SA 3.0 (https://creativecommons.org/licenses/by-sa/3.0)], via Wikimedia Commons

So you should be aware that this part here is nothing else than the dot product of two vectors, or in other words a linear combination of two vectors, or just in other words, linear regression.

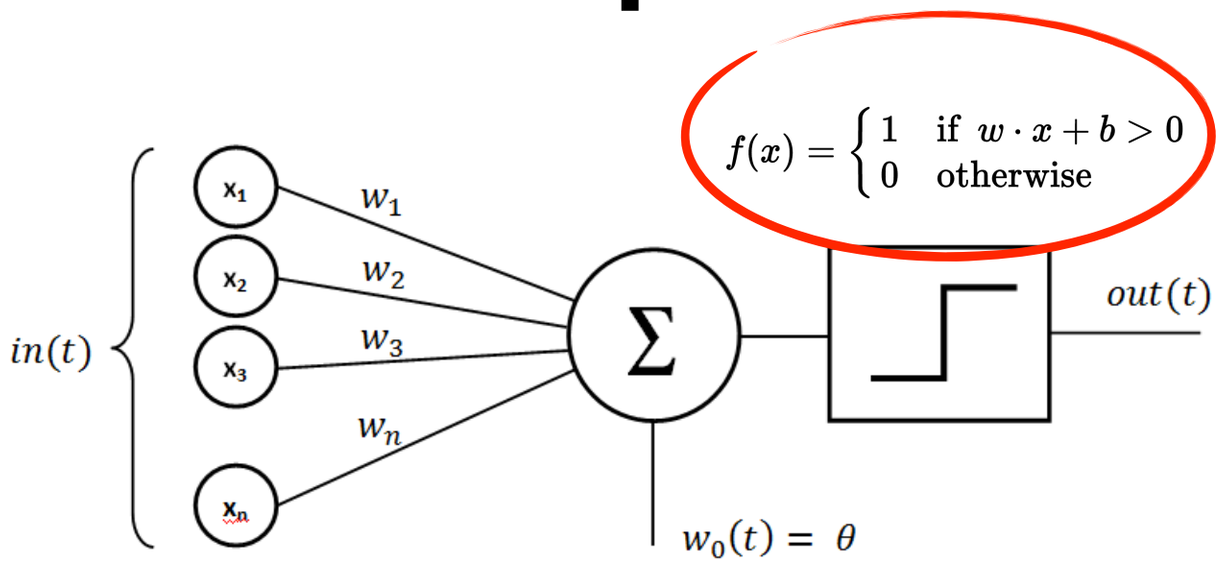
Perceptron



By Mayranna (Own work) [CC BY-SA 3.0 (https://creativecommons.org/licenses/by-sa/3.0)], via Wikimedia Commons

Now let’s replace sigmoid of logistic regression…

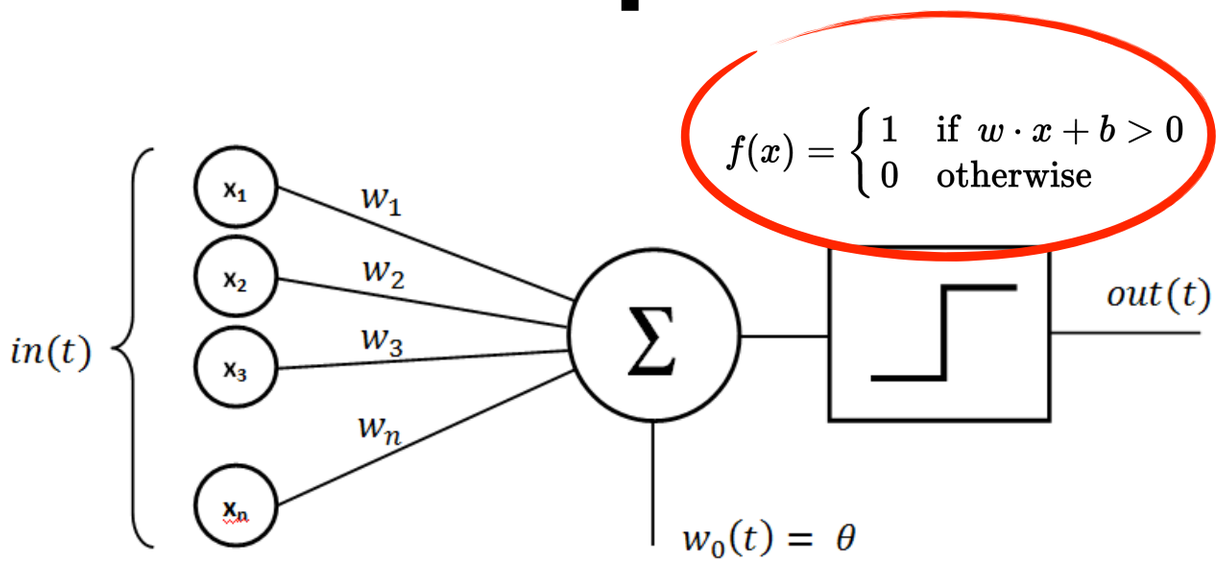
Perceptron



By Mayranna (Own work) [CC BY-SA 3.0 (https://creativecommons.org/licenses/by-sa/3.0)], via Wikimedia Commons

…with this simple step function and you are done.

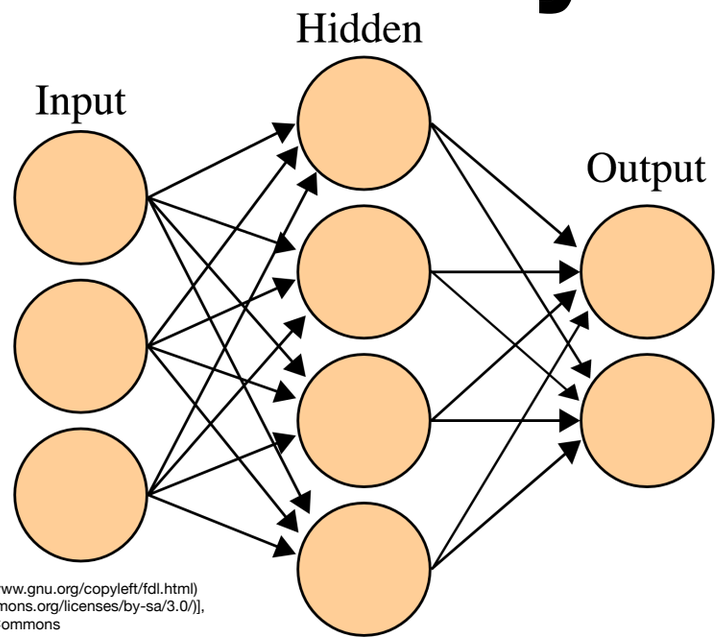
Perceptron



By Mayranna (Own work) [CC BY-SA 3.0 (https://creativecommons.org/licenses/by-sa/3.0)], via Wikimedia Commons

…with this simple step function and you are done. A perceptron is a binary, linear classifier. So let's see how we can improve it. By the way, the first software perceptron was build on the IBM 704, in 1954, which was the first mass-produced computer with floating-point arithmetic hardware

Hidden Layers



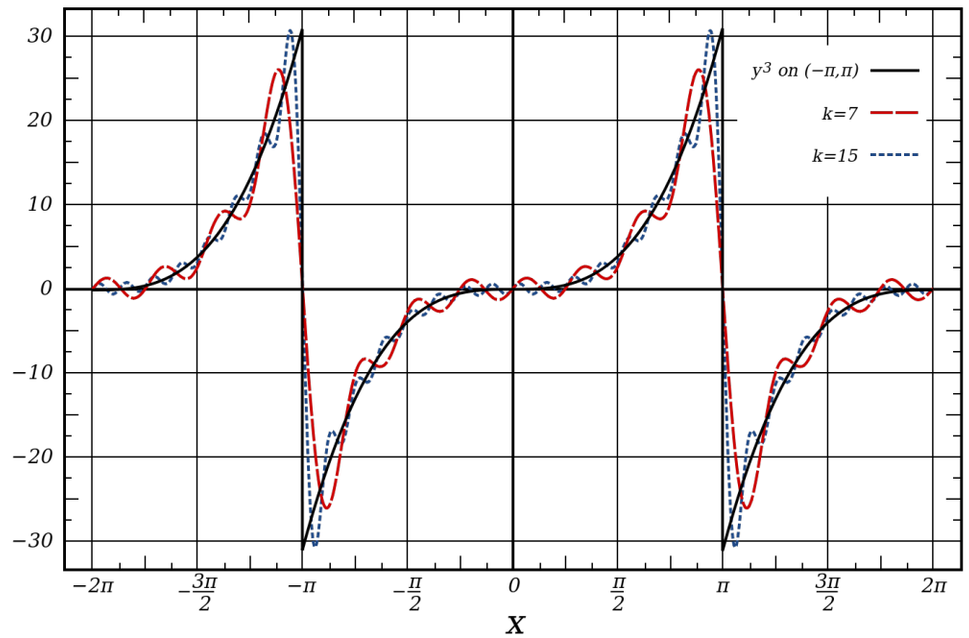
By en:User:Cburnett [GFDL (http://www.gnu.org/copyleft/fdl.html) or CC-BY-SA-3.0 (http://creativecommons.org/licenses/by-sa/3.0/)], via Wikimedia Commons

So this guy here is a feed forward network with a so called hidden layer. In order to understand this better, let’s code it first.

a3_m1_v2_deep_feedforward_neural_networks_ex1_4a and b

 Romeo Kienzler

So python dictionaries are of course not the best way to do data science. So let’s convert our data points to a matrix. First we have to get rid of the labels which are called keys in python dictionaries, and turn each data point into an array. Neither part number nor asperity are required here so let’s get rid of those. But we need to align this array to the size of the weight vectors so we add a one at the beginning in order to support the bias parameter. So this is what we want. Native python arrays are slow and also not supporting linear algebra operations. So let’s wrap it into a numpy array. And then let’s do it for all four data points. Now lets create a function called neuron which computes the logistic function for just one data point. We define a weight vector double u which we initialise randomly and use the dot product between the data point vector and the weight vector in order to compute the linear regression function. Then we apply sigmoid as activation function and we are done. So at lest it compiles and runs. Note that we won’t get any meaningful result since we’ve initialised double u randomly and didn’t train the model, meaning updating the double u accordingly. No let’s pimp this function a bit in order to compute all data points in parallel. Therefore we first have to improve the sigmoid function that it takes vectors instead of only scalars. Fortunately, numpy does the job here. As this function now takes a matrix as argument we also have to change our weights double u from vector to matrix. And of course, finally we have to create a matrix out of our input data. We can use numpy again for this since a nested array is nothing else than a matrix. So now we can compute our layer in one go. Now let’s create the 2nd layer. All we need to change is the weight matrix double u since the 2nd layer is computed independently from the 1st. And as a last step we just stack those computations on each other and we are done with our first neural network. Of course the predicted values are incorrect since we are using random values for the weight matrices double u. So this is now a untrained neural network.



By Inductiveload [Public domain], via Wikimedia Commons

Note that a single hidden layer neural network is capable of representing any mathematical function. This is know as the universal function approximation theorem. Therefore you might ask yourself why there are multiple hidden layers and other neural network topologies used at all. So this is because of training. Even if you can represent any mathematical function by choosing appropriate values for the weight matrix double u there is no efficient way to find them so easily. But let’s talk about this later.

Recurrent Neural Networks

 Romeo Kienzler

Let’s get started with Recurrent Neural Networks.