1. **Main activation functions**

Let’s present some common non-linear activation functions, their characteristics, with the pros and cons.

1. **The sigmoid function**

We know that one! A reminder of its definition

**Pros**

* It is a very popular choice, mostly due to the output range from 0 to 1, convenient to generate probabilities as output.
* The function is differentiable and the gradient is smooth, i.e. no jumps in the ouput values.

**Cons**

* The sigmoid’s derivative vanishes at its extreme input values (z→−∞ and z→+∞) and is thus proned to the issue called Vanishing Gradient problem (see [The risk of vanishing or exploding gradients](https://clairedavid.github.io/intro_to_ml/week2/NN1_activationF.html#nn1-activationf-risksgradient)).

1. **Hyperbolic Tangent**

Alike the sigmoid, the hyperbolic tangent is S-shaped and continously differentiable. The output values range is different from the sigmoid, as it goes from -1 to 1.

**Pros**

* It is zero-centered. Unlike the sigmoid when we had to have a decision boundary of 0.5 (half the output range), here the mapping is more straightforward: negative input values gets negative output, and positive input values will be positive, with one point (z=0) returning a neutral output of zero.
* That fact the mean of the ouput values is close to zero (middle of the output range) makes the learning easier.

**Cons**

* The gradient is much steeper than for the sigmoid (risk of jumps while descending)
* There is also a Vanishing Gradient problem due to the derivative cancelling for z→−∞ and z→+∞.

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

Welcome to the family of rectifiers, three most popular activation function for deep neural networks. The ReLU is defined as:

**Pros**

* Huge gain in computational efficiency (much faster to compute than the sigmoid or tanh)
* Only 50% of hidden activation units are activated on average (it is called sparse activation), further improving the computational speed
* Better gradient descent as the function does not saturate in both directions like the sigmoid and tanh. In other words, the Vanishing Gradient problem is half reduced

**Cons**

* Unlike the hyperbolic tangent, it is not zero-centered
* The range is infinite for positive input value (not bounded)
* ReLU is not differentiable at zero (but this can be solved by choosing arbitrarily a value for the derivative of either 0 or 1 for z=0)
* The “Dying ReLU problem”

1. **Leaky ReLU**

It is a ReLU with a small positive slope for negative input values:

**Pros**

* All advantages of the ReLU mentioned above (fast computation, no saturation for positive input values)
* The small positive gradient when units are not active makes it possible for backpropagation to work, even for negative input values
* The non-zero gradient mitigate the Dying ReLU problem

**Cons**

* The slope coefficient is determined before training, i.e. it is not learnt during training
* The small gradient for negative input value requires a lot of iterations during training: the learning is thus time-consuming

1. **Parametric ReLU (PReLU)**

The caveat of the Leaky ReLU is addressed by the Parametric ReLU (PReLU), where the small slope of the negative part is tuned with a parameter that is learnt during the backpropagation algorithm. Think of it as an extra hyper-parameter of the network.

**Pros**

* The parametric ReLU collects all advantages of the ReLU and takes over when the Leaky ReLU still fails too reduce the number of dead neurons

**Cons**

* There is an extra parameter to tweak in the network, the slope value a, which is not trivial to get as its optimized value is different depending on the data to fit

1. **Exponential Linear Units (ELUs)**

It does not have Rectifier in the name but the Exponential Linear Unit is another variant of ReLU.

with a a hyper-parameter to be tuned.

**Pros**

* From high to low input values, the ELU smoothly decreases until it outputs the negative value −a. There is no more a ‘kick’ like in ReLU
* ELU functions have shown to converge cost to zero faster and produce more accurate results

**Cons**

* The parameter a needs to be tuned; it is not learnt
* For positive inputs, there is a risk of experiencing the Exploding Gradient problem (explanations further below in [The risk of vanishing or exploding gradients](https://clairedavid.github.io/intro_to_ml/week2/NN1_activationF.html#nn1-activationf-risksgradient))

1. **Scaled Exponential Linear Unit (SELU)**

The Scaled Exponential Linear Unit (SELU) is defined as:

**Pros**

* All the rectifier’s advantages are at play
* Thanks to internal normalization, the network converges faster

**Cons**

* Not really a caveat in itself, but the SELU is outperforming other activation functions only for very deep networks

1. **Gaussian Error Linear Unit (GELU)**

Another modification of ReLU is the Gaussian Error Linear Unit. It can be thought of as a smoother ReLU.

The definition is:

is the cumulative distribution function of the standard normal distribution.

**Pros**

* Differentiable for all input values z
* Avoids the Vanishing Gradient problem
* As seen above, the function is non-convex, non-monotonic and not linear in the positive domain: it has thus curvature at all points. This actually allowed GELUs to approximate better complicated functions that ReLUs or ELUs can as it weights inputs by their value and not their sign (like ReLu and ELU do)
* The GELU, by construction, has a probabilistic interpretation (it is the expectaction of a stochastic regularizer)

**Cons**

* GELU is time-consuming to compute

1. **Sigmoid Linear Unit (SiLU) and Swish**

The SiLU and Swish are the same function, just introduced by different authors (the Swish authors are from Google Brain). It is a state-of-the-art function aiming at superceeding the hegemonic ReLU. The Swish is defined as a sigmoid multiplied with the identity:

**Pros**

* It is differentiable on the whole range
* The function is smooth and non-monotonic (like GELU), which is an advantage to enhance input data during learning
* Unlike the ReLU function, small negative values are not zeroed, allowing for a better modeling of the data. And large negative values are zeroed out (in other words, the node will die only if it needs to die)

**Cons**

* More a warning than a con: the Swish function is only relevant if it is used in neural networks having a depth greater than 40 layers