Activation Functions

Activation functions provide non-linearity to the system and act like a switch to the system.

Activation functions are classified into single kink and double kinks on their graph. The shape of the kink e.g. smooth vs hard kink is depend on scaling value equivalence hard kink does not change scaling factor, while the smooth kink will change the the output value scaling factor

Hard kink vs Soft kink it matter of scaling equivalent

Hard kink it provide scaling of output

when the input multiply by a scalar value the output also have the effect with multiply by same scalar

Soft Kink it change the behavior of output

When the input value e.g. multiply by 100 the output look like it have hard kink

When divide the input with the same value the output look smooth

## Activation that have single kink in the graph

which does not change signal only the scale

Rectilinear Unit (ReLU)

There are multiple variation of ReLU activation functions

* ReLU(x) = (x)+ = max(0,x) has 0 at bottom and sharp kink for all positive value
* Leaky ReLU (x) = {x if x ≥ 0, negative slope of x otherwise

Allow bottom (horizontal asymptote) to have negative value so there still chance to have gradient

* PReLU (x) {x, if x ≥ 0 , ax otherwise a = alpha is a leakable parameter

The value does not change other than the scale of value a, which different input units

Value alpha may be fixed or learned. It can have different values or a may be shared with different units. A value may not have to be learn but preset to ensure the gradient value is non zero

* SoftPlus(x) = 1/beta \* log(1 + exp(beta\* x))

A smooth approximation of ReLU can be used to constrain the output to always be positive where the beta get large it closer to ReLU

* RReLU(x) = {x if x≥ 0, ax otherwise randomize leaky ReLU the same as leaky ReLU
* ELU(x) = max(0,x) + min(0,a \* (exp(x) - 1) a softer version of ReLU by adding a small constant to smoothing it. And the bottom(horizontal asymptote) can be negative it allow for system to have average output value to be 0 which is useful for certain situation which have both positive and negative value which allow for faster convergent
* CELU(x) = max(0,x) + min(0,a \* (exp(x/a) - 1) have the same effect as ELU
* GELU(x) = x \* phi(x) where phi(x) is the cumulative distribution function for Gaussian Distribution
* ReLU6(x) = min(max(0,x),6) has two kink point on the graph at 0 and at saturation point at 6

## Activation that have 2 kink in the graphs

## Double kink has built in scale where input signal is very different from output signal

* Sigmoid(x) = sigma(x) = 1/1+exp(-x) has smooth curve for value between 0 and 1 problem with gradient go to 0 fairly quickly as it get closer to asymptote so the weight of the unit become too large which saturate the unit so the gradient drop fast the learning is not efficient. Sigmoid doesn’t work well when the value normalizes (with batch/group norm) just before it. Since normalization fix the scaling make the mean 0 and variance constance so the system doesn’t depend on the weight but the system now loss the ability to choose with kink to use in the 2 kink like sigmoid
* tanh(x) = tanh(x) = exp(x) - exp(-x)/exp(x) + exp(-x) has graph that identical to sigmoid except it center at 0 and go from -1 to 1 has the output to be closer to 0 mean but not 0 which allow the weight value after it to see both + and - which converge faster. Unfortunately, for deep stacking layers they are not learning efficiently and must be careful in normalizing the value. Which is why the single kink function tend to do better with deep layer network
* Softsign(x) = x/1+|x| similar to sigmoid except it doesn’t get to the asymptote as fast and doesn’t get stuck toward asymptote as quickly as sigmoid
* HardTanh(x) = {1 if x ≥ 1, -1 if x ≤ -1,x otherwise the linear region range is -1,1. It has a sharp kink both at negative value at bottom and positive value at top. Work fairly well for small network where the weight is not saturate too much

## Use in sparse coding

rarely use in neural network

* Threshold y = { x, if x threshold, value otherwise

rarely use since gradient can not propagate through it

* Tanhshrink(x) = x - tanh(x) use mostly on sparse coding to shrinking the latent variable
* softShrinkate(x) = { x - lambda, if x > lambda, x+ lambda, if x < -lambda, 0 otherwise it has sharp kink at both bottom and top of graph
* HardShrink(x) = {x - lambda, if x > lambda, x+ lambda, if x < -lambda, 0 otherwise where value between -lambda and +lambda are set to 0
* LogSigmoid(x) = log(1/1+exp(-x)) mostly use for loss function instead
* softmin(xi) = exp(-xi)/sum(exp(-xj)) when apply to input Tensor it rescaling the element of vector to the range of [0,1] which the sum to 1 similar to softmax with minus sign in front xi value
* Softmax(xi) = exp(xi)/sum(exp(xj)) same with softmin

Temperature is redundant with incoming weight (beta value) matter when the input size is fixed due to normalization. It uses to control how hard(the different between large and small value in the output vector) the distribution of the output value will be it the system to make hard decision even though it doesn’t learn as well as before. It used in case of mixture of expert system when multiple subnetwork output are linear combine with coefficients from the output of the softmax which may be controlled by another NN with soft mixture with low beta then gradually increase to infinity the will force NN to choose one and ignore the rest this may safe compute time (call annealing)

* LogSoftmax(xi) = log(exp(xi)/sum(exp(xj))) use at the output as part of the loss function