1 Activation and Initialization of Convolution Neural Network

1.1 Activation Function

1.1.1 The meaning of the Activation Function

Activation function is a function that is added into an neural network layer in order to help the network learn complex (non-linear) patterns in the data.

For one layer neural network:

$$\hat{y} = f(W^T X),$$

where

- $f(\cdot)$ is the activation function - any arbitrary function (with non-linear behavior);
- $\hat{y}$ is the network output;
- W, X are the weight and input.

The meaning of the activation function is to describe some non-linear relations for the input-output.

Note

- there are two popular activation functions:

- **sigmoid/softmax** for output;
- **ReLU family** (including non-linear relu) for intermediate (hidden) layers.
- Out-put without activation function is the so-called **Linear Activation**.
- If you also have **Batch-Norm layers** in your network, that is added before the activation function making the order CNN-Batch Norm-Act. Although the order of Batch-Norm and Activation function is a topic of debate and some say that the order doesn't matter.

$$\hat{y} = f(BN(W^T X)),$$

where BN is the Batch-Norm.

There are the following requirements for activation functions:

- **Differentiable equation** - due to the method of training it is required for activation function to have certain derivative.
- **Low computation complexity** - it is not the requirement, only recommendation due to reducing all network inference time.

  This includes:

  - **self-computation complexity**

  - and **sparsity potential** - a lot of zero values allow to use specific low-complexity algorithm for operations with matrix (see the pruning of neural networks).

- **Low potential ability to gradient vanishing and explosion**.

  Intuitively a lot of activation functions with saturation in the lowest and highest values lead to the problem of when the network stopping its training. It will be due to layers produce outputs of activation function in saturation ranges and error value will give zero gradient and not upgrade the values of the weights.

  Note

  - Main reason of the **Gradient Vanishing** is the saturation of activation function derivative.
    - When the derivative of activation function is small or converge to 0 the learning is minimal (or stop for 0).

      When slow learning occurs, the optimization algorithm that minimizes error can be attached to local minimum values and cannot provide maximum performance.
  - Main reason of the **Gradient Explosion** is the lack of saturation of activation function (too high value of the activation

function output).

In this case we will have too coarse search of minimum of loss, and will not obtain high accuracy.

Essentially, **we want to have the function without of its value saturation, but with saturation of its derivative to 1, and with 0 only when it is necessary.**

1.1.2 Logistic Activation Function

The meaning of the activation function lead analytical solution of the problem of non-linear regression.

In particular, in the case of **Logistic regression** the problem can be given as to find the best hyperplane (line in 2d) to divide to classes.

In this problem we can write it as follows:

$$y_i = f(\sum_{j=0}^{M} w_j x_{ij}) = f(w^T x_i),$$

where:

- w is the weight vector,
- $x_j = x_{0j}, \ldots, x_{ij}, \ldots, x_{Nj}$ is the j-th input vector of length N of the training data set with M data;
- y is the labels of classes in form 0 and 1 - thus we want make a decision by the rule

$$\begin{cases} y \geq 1/2 \rightarrow \text{class } 1 \\ y \leq 1/2 \rightarrow \text{class } 0 \end{cases}.$$

The simplest solution of this problem is the **step function (Heaviside function)**,

$$\textbf{step} = \begin{cases} y \geq 0 \rightarrow \text{class } 1 \\ y \leq 0 \rightarrow \text{class } 0 \end{cases}.$$

However, step function if not differential and can not be optimized analytically, moreover it has uncertainty value for inflection point $y = 0$

The analytical solution of the Logistic regression problem is the **Maximization of Log-Likelihood function for binominal distribution (Bernoulli distribution)**

$$L(y|w^T x) = \prod_{i=1}^{N} P[y_i|w^T x_i] = \prod_{i=1}^{N} a_i^{y_i}(1 - a_i)^{1-y_i}$$

$$log(L(y|w^T x)) = \sum_{i=1}^{N} a_i^{y_i} + \sum_{i=1}^{N}(1 - a_i)^{1-y_i} \rightarrow max_w$$

where $a = f[w^T x_i]$. Thus,

$$\frac{\partial log(L(y|w^T x))}{\partial w} \rightarrow 0 \implies \frac{\partial P[y_i|w^T x_i]}{\partial y} = P[y_i|w^T x_i](1 - P[y_i|w^T$$

The corresponding function to the expression is the **Sigmoid**:

$$P[y_i|w^T x_i] = \sigma(w^T x) = P[y_i|w^T x_i] = \frac{1}{1 + exp[-w^T x]},$$

$$\frac{\partial \sigma(w^T x)}{\partial w} = \sigma(w^T x)(1 - \sigma(w^T x))$$

So we have solution that the activation function $f(\cdot)$ is the *sigmoid* function.

For the several classes the extension of the sigmoid is the **Softmax** activation function.

By analogy with the above we have

$$\text{For two classes}: \sigma(w^T x) \rightarrow P[y_i|w^T x_i] = \frac{P[y_i = 1|w^T x_i]}{P[y_i = 0|w^T x_i] + P[y_i =}$$

$$\text{For several classes}: P[y_i|w^T x_i] = \frac{P[y_i = class_c|w^T x_i]}{\sum_{k=0}^{K-1} P[y_i = class_k|w^T x_i]} \rightarrow$$

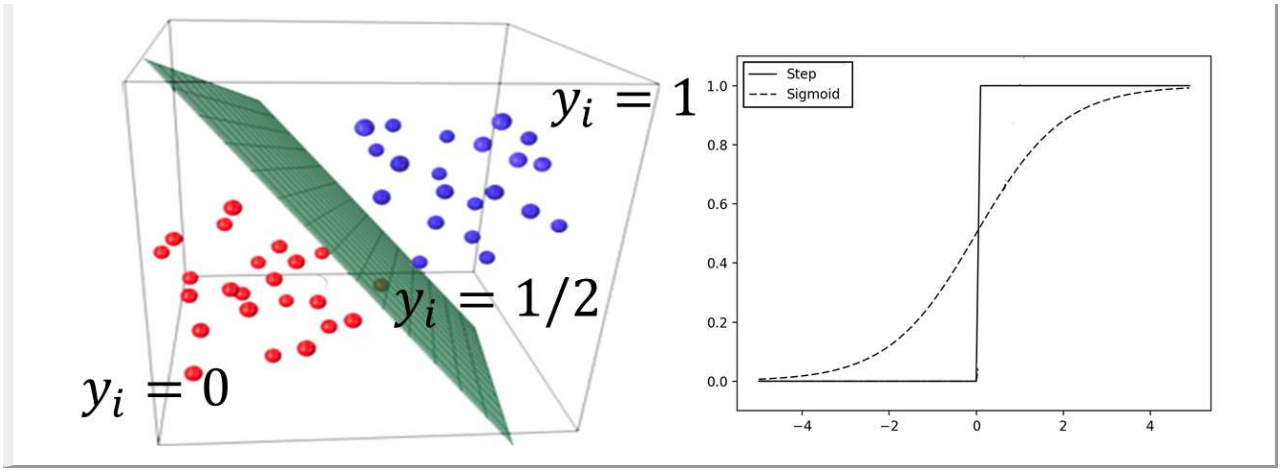
$$\rightarrow \textbf{softmax} = \frac{exp[w_c^T x]}{\sum_{k=0}^{K-1} exp[w_k^T x]},$$

where c is the current class and K is number of classes.

$$\frac{\partial(\text{softmax}(x))}{\partial x} = \text{softmax}(x)(1 - \text{softmax}(x))$$

Now we can imagine that the neural network is the multilayer combination of a set of logistic-like estimators. Thus we can propose that the analytical solution for us is the **sigmoid/softmax** function also.

Example of the decision plane made by sigmoid (logistic regression)

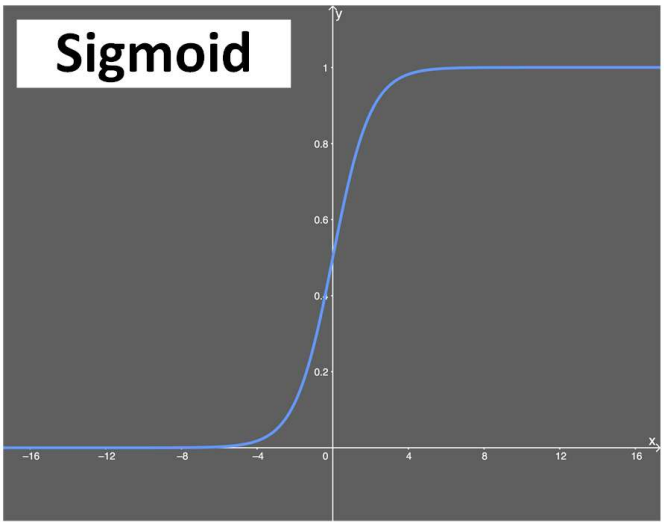


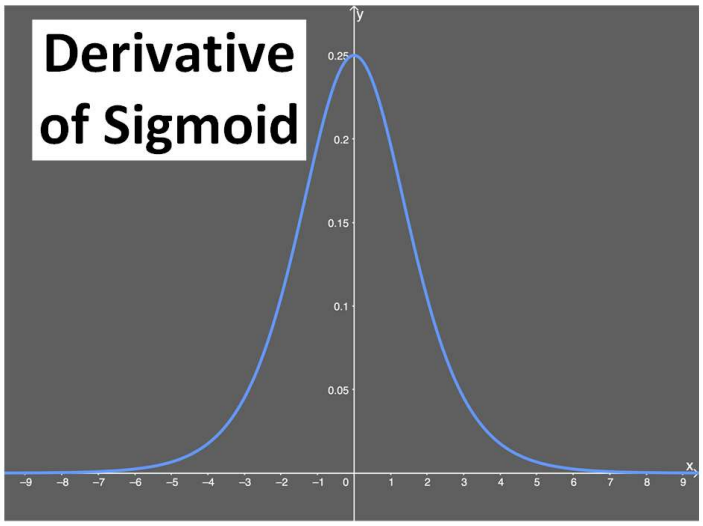
Let's rewrite **Sigmoid**

$$\text{sigmoid}(x) = \frac{1}{1 + exp(-x)}, \quad \text{sigmoid}'(x) = \text{sigmoid}(x)(1 - \text{sigmoid}(x))$$

and **Softmax**:

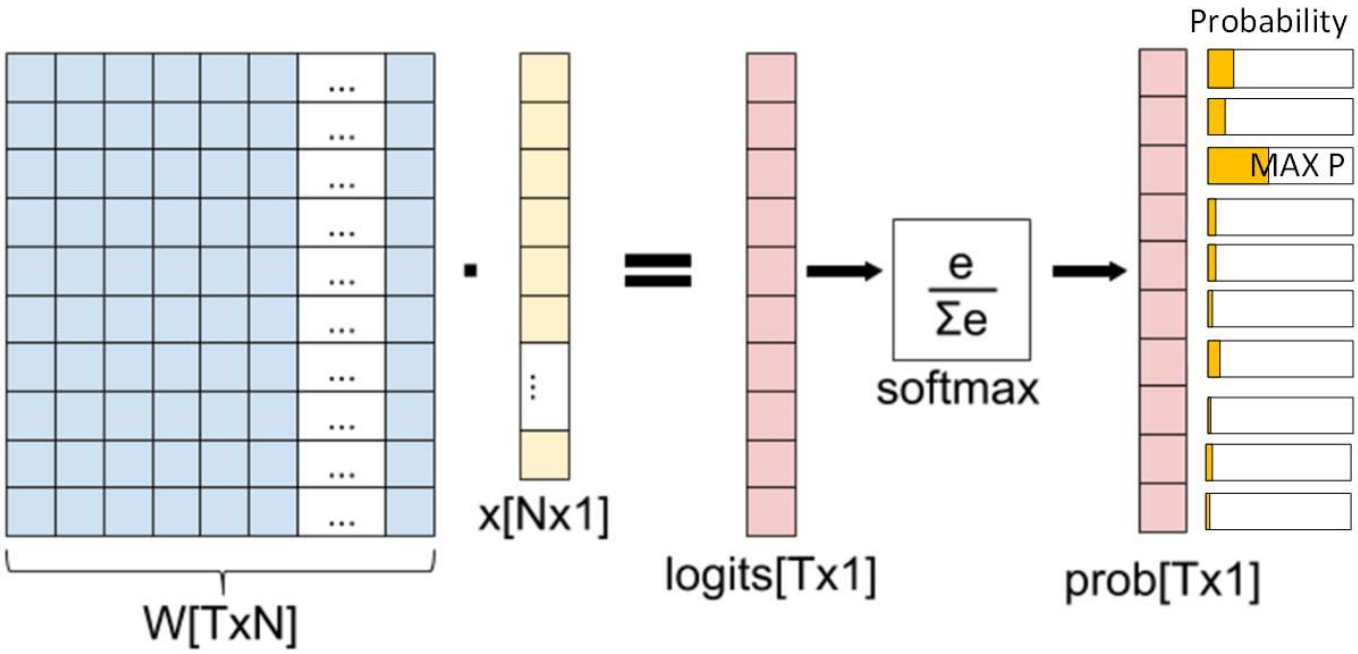
$$\text{softmax}(x_i) = \frac{exp(x_i)}{\sum_j^K exp(x_j)}, \quad \text{softmax}'(x_i) = \text{softmax}(x_i)(1 - \text{softmax}(x_i))$$





Note The softmax allow to obtain probabilities of all classes (i.e. vector).

If you want to get one result you will need to take maximum of the softmax output. $$ \text{result} = \text{max}_i(\text{softmax}(x_i))$$



Note

- you can replace the sigmoid with so-called **hard-sigmoid**, $$ \text{hard-σ} = \text{ReLU}6(x+3)/6, $$

  where $\text{ReLU}6 = \min(6,\max(0,x))$.

- If you need to have values below 0 you may use **tanh** instead of $sigmoid$: $$ \text{tanh}(x) = \frac{exp(x)+exp(-x)}{exp(x)-exp(-x)}, \ \ \ \text{tanh}'(x) = 1-\text{tanh}^2(x)$$ This function allows to have gradient below zero and do not better than sigmoid in any other case.

  Hyperbolic tangent can be learned a little slower because of its wide range of activating values.

Actually, the sigmoid function have a several drawbacks.

The main problem with sigmoid is its saturation- i.e. vanishing of its derivative.

But it is also problem with complexity in some cases.

As a rule the sigmoid and softmax are used only in the last layer for make a decision in classification task.

In other layers these functions can be replaced by its analogue.

The one of the solutions of the gradient vanishing/explosion problem is the

- **Gradient Clipping or Normalization**.

  Set some threshold values of the activation function output. For instance $[0.25,0.75]$ and clip or normalize all values exceeding that range.

- **The other solution is to use activation functions without saturations.**

  - The simples case is to use **softplus** function: $$\begin{aligned} &\text{softplus}(x)= log(1+exp(x)), \ \ \ \text{softplus}'(x)= \frac{1}{1+exp(-x)}\\ &\text{or} \\ &\text{shift_softplus}(x)=

log(0.5+0.5exp(x)), \ \ \ \text{shift_softplus}'(x)= \frac{0.5}{1+exp(-x)} \end{aligned}$$

- the other empirical modification is **soft-sign** function $$ \text{softsign}(x) = \frac{x}{1 + |x|}, \ \ \ \text{softsign}'(x)=\frac{1}{(1 + |x|)^{2}} $$

  But these solution is not popular due to presence of all the drawbacks of sigmoid and in the case of softplus it also require addition operation of taking log value; in the case of softsign it does not allow sparsity and regularization of negative gradient.

1.1.3 Rectified Linear Unit (ReLU)

- The one of common chose for overcome the sigmoid drawbacks is to use **Rectified Linear Unit (ReLU)**. $$ \text{ReLU}(x)=\text{max$(0,x)$}, \ \ \ \text{ReLU}'(x) = \begin{cases} \mbox{1} & \mbox{if } x > 0\\ \mbox{0} & \mbox{if } x \leq 0 \end{cases} $$ Here we reduce the probability of the vanishing problem for values above zero.
  - ReLU provide
    - low complexity

- high sparsity (a lot of zero values allow to use specific low-complexity algorithm for operations with matrix).
- ReLU is Scale-invariant: $\text{ReLU}(x)=\text{max$(0,ax)$}=a\text{max$(0,x)$}$.
- ReLU have some regularization effect due to restriction of negative values.
- Actually, having a value of 0 on the negative axis means that the network will run faster,

  however it also mean that we does not learn the weights here.
- However, if too many values are below 0, We get a bunch of weights and biases that is not updated, since the update is equal to zero.

  That can led to the vanishing problem again (so-called **dead relu problem)**.

- *There are a lot of modifications of ReLU, including*:
  - ReLU with saturation: **ReLU6**: $$ \text{ReLU6}(x)=\text{min(max$(0,x)$,6)}, \ \ \ \text{ReLU6'}(x)=\begin{cases} \mbox{1} & \mbox{if } x \in (0 , 6) \\ \mbox{0} & \mbox{if } x

\leq 0 \text{ and }x\geq 6 \end{cases}. $$ This helps to stop blowing up the activation out growing, thereby stopping the gradients to explode(going to inf).

- **Leaky ReLU**: $$ \text{LReLU}(x)=\begin{cases} \mbox{x} & \mbox{if } x > 0\\ \mbox{$0.01x$} & \mbox{if } x \leq 0 \end{cases} , \ \ \ \text{LReLU}'(x) = \begin{cases} \mbox{1} & \mbox{if } x > 0\\ \mbox{0.01} & \mbox{if } x \leq 0 \end{cases}. $$ Leaky ReLU can be used with different coefficients for values below zero, but 0.01 is traditional value.
- **Parametric ReLU**: $$ \text{PReLU}(x)=\begin{cases} \mbox{x} & \mbox{if } x > 0\\ \mbox{αx} & \mbox{if } x \leq 0 \end{cases}, \ \ \ \text{PReLU}'(x) = \begin{cases} \mbox{1} & \mbox{if } x > 0\\ \mbox{α} & \mbox{if } x \leq 0 \end{cases}, $$ where α is the learnable-parameter, which can be set for each neuron.

  for the case if $\alpha \le 1 \text{PReLU}(x)=\text{max(ax,x)}$.

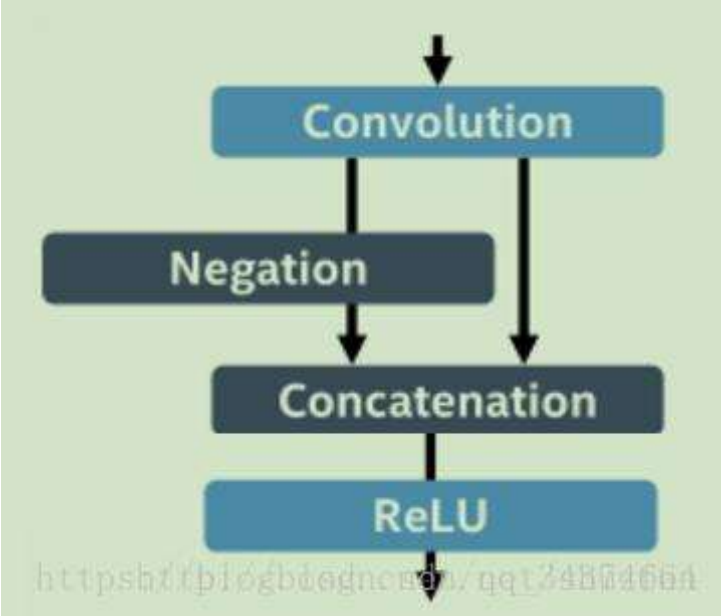
  If $\alpha=1$ we will have linear activation, thus do not use α values near 1.

Leaky ReLU and PReLU in differs with conventional relu allow to slightly learn negative values. But it cancel the sparsity effect.

- **Maxout**:

  In essence, it is not the activation function, but from the other point of view it is the generalization of ReLU. $$ \text{maxout}(x)=\text{max$(xw_2+b_2, \ xw_1+b_1)$}, $$ where w_2, w_1, b_2, b_1 are a several coefficients.

  - For the case $w_2=1, w_1=0, b_2=0, b_1=0$ we will have ReLU.
  - For the case $w_2=1, w_1=0, b_2=0.01, b_1=0$ we will have LReLU.
  - **Concatenated ReLU** $$\text{CReLU} = concat(max(0,x),max(0,-x)) $$ The idea is to overcome the problem of relu by combining negative and positive relu together.



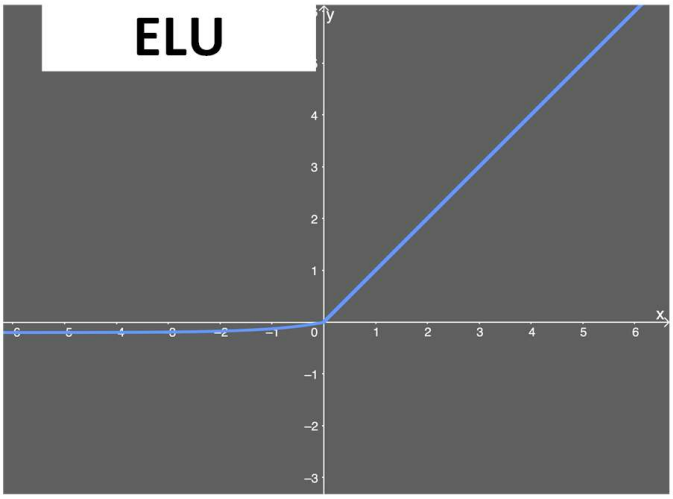
1.1.4 Non-Linear ReLU

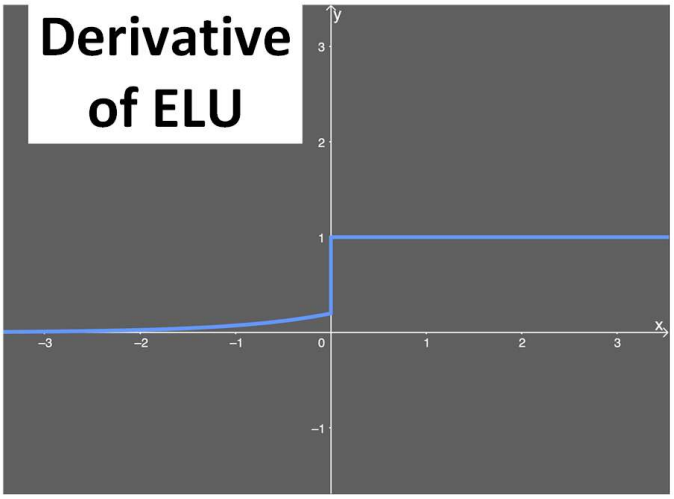
- **Exponential Linear Unit, ELU**: $$ \text{ELU}(x) = \begin{cases} \mbox{x} & \mbox{if } x > 0\\ \mbox{$\alpha (e^x-1)$} & \mbox{if } x < 0 \end{cases}, \ \ \ \text{ELU}'(x) = \begin{cases} \mbox{1} & \mbox{if } x > 0\\ \mbox{αe^x} & \mbox{if } x \leq 0 \end{cases} $$ where α is the hyper-parameter.

  The features of non-linear RELU

  - This function is allow to avoid the dead relu problem, while still keeping some of the computational speed gained by the ReLU activation function (sparsity of output).
  - Also small values in the negative direction allow to correct vanishing problem in some cases.
  - However, exponent here lead to increase the complexity.

- it is also bad to use α as hyper-parameter instead of learning it or determine by some recommendations.





- **Scaled Exponential Linear Unit. SELU**: $$\begin{aligned} \text{SELU}(x) = \lambda\cdot\text{ELU}(x)=\lambda \begin{cases} \mbox{x} & \mbox{if } x > 0\\ \mbox{$\alpha e^x-\alpha$} & \mbox{if } x \leq 0 \end{cases}, \ \ \ \text{SELU}'(x) = \lambda \begin{cases} \mbox{1} & \mbox{if } x > 0\\ \mbox{αe^x} & \mbox{if } x \leq 0 \end{cases} \end{aligned} $$ The authors are recommend to use: $$\begin{aligned} &a \approx 1.673263242354377284817042991671 7 \\ & \lambda \approx 1.050700987355480493419334985294 6 \end{aligned} $$ It is shown that:

- The output of a SELU is normalized to the normal distribution (potentially with zero mean value), which could be called **internal normalization**.
- If all activation is SELU, then all network will be normalized (**external normalization**).
- SELU Function has faster internal normalization then external, which means the network converges fast.
- For SELU Variance of normalization decreases when the input is less than zero, and variance increases when the input is greater than zero.
- Vanishing and exploding gradient problem is impossible, (shown in the original paper).
- SELU is similar to the idea and principles of self-normalizing neural networks

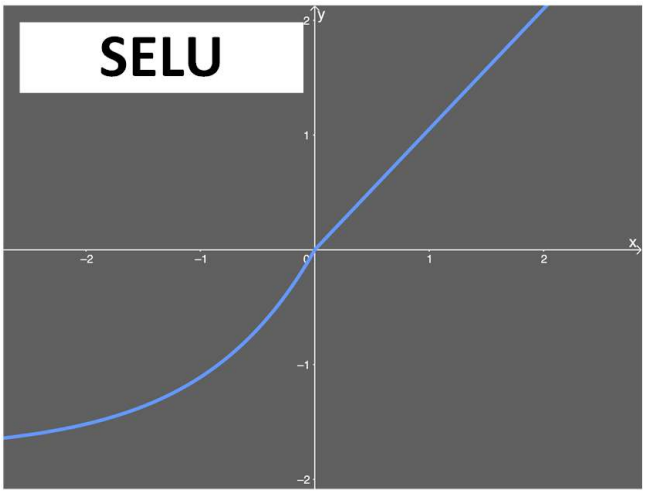
Note

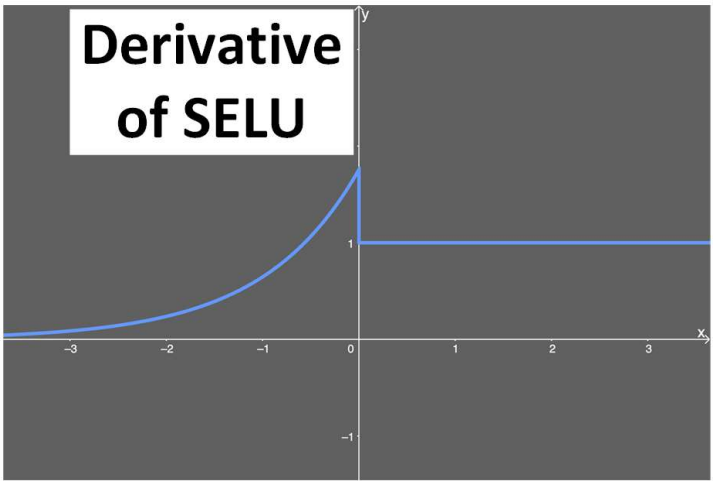
The SELU designed with for satisfy the following

- normal distribution of activation function output,
- non-zero in the negative and positive values for controlling the mean,
- saturation regions (derivatives approaching zero) to dampen the variance if it is too large in the lower layer,
- a slope larger than one to increase the variance if it would be too small in the lower layers,
- a continuous curve to ensures a fixed point, where variance damping is equalized by variance increasing.

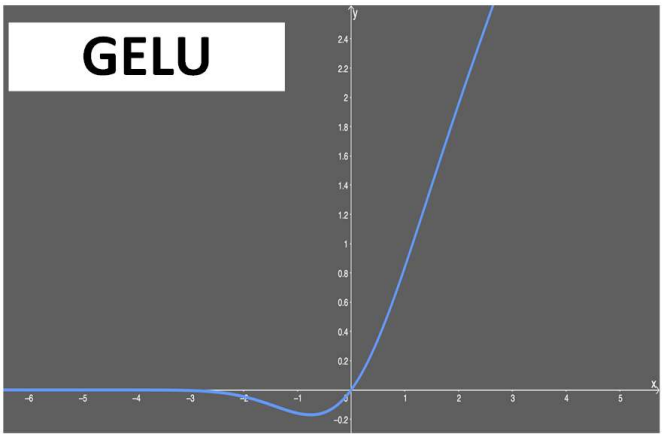
Note When using this activation function in practice, one must use **lecun_normal** for weight initialization (with normal Gaussian distribution with std 1/n and mean 0,where n is the number of parameters

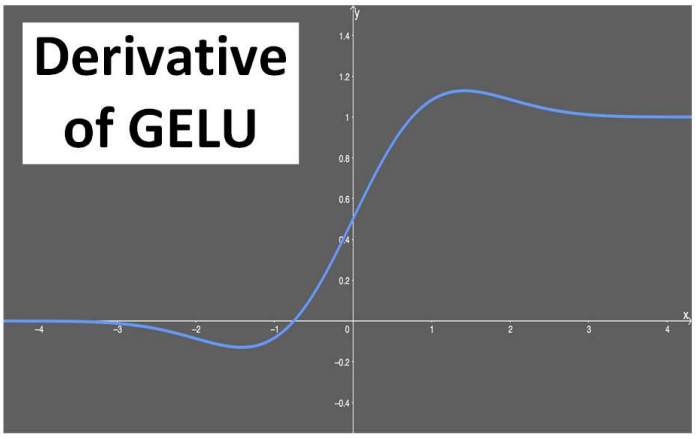
in the layer), and if dropout wants to be applied, one should use **AlphaDropout**.



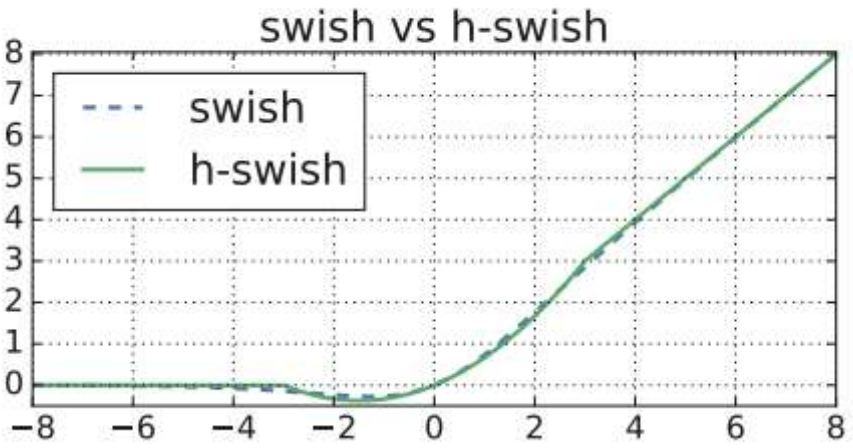


- **Gaussian Exponential Linear Unit. GELU**: $$ \text{GELU}(x) = 0.5x\left(1+\text{tanh}\left(\sqrt{2/\pi}(x+0.044715x^3)\right)\right) $$ $$ \text{GELU}'(x) = 0.5\text{tanh}(0.0356774x^3 + 0.797885 x) + (0.0535161 x^3 + 0.398942 x) \text{sech}^2(0.0356774x^3+0.797885x)+0.5$$ This type of activation functions is popular in such modern architectures as Transformer, GPT and so-on. This function allow to avoid the vanishing problem. However function has high computation complexity and in CNN this chose is not popular.

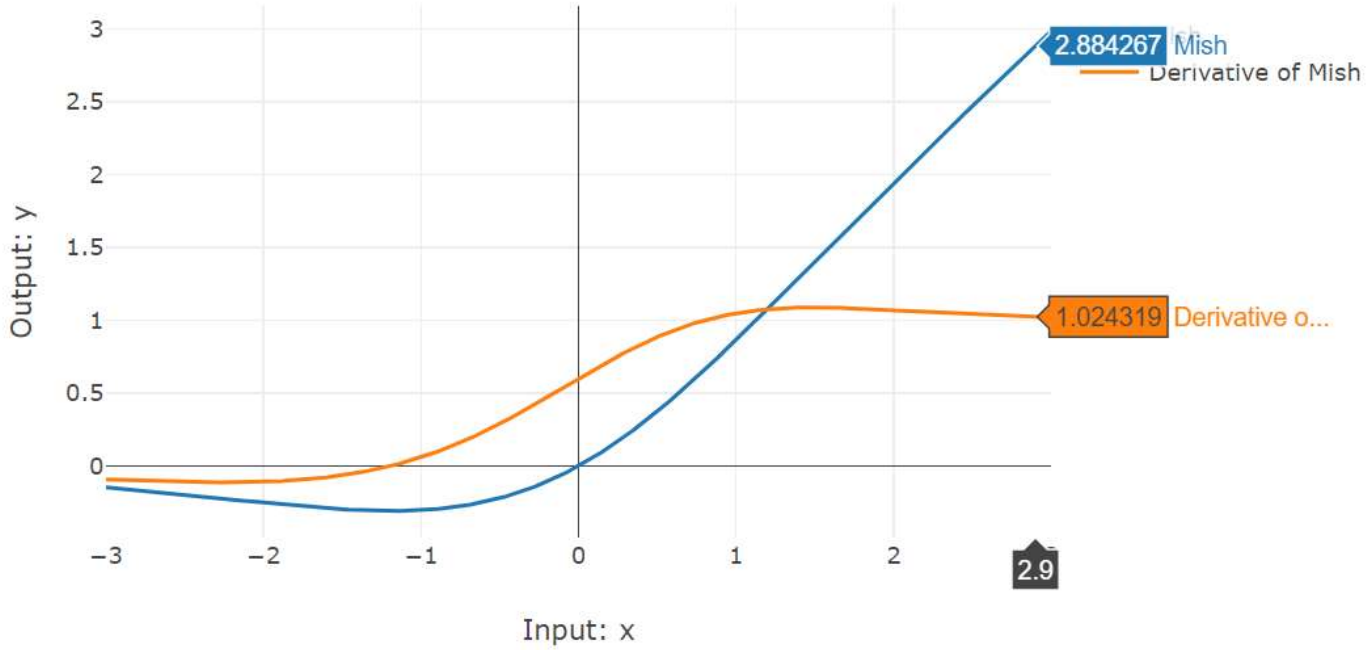




- **Hard Swish, H-Swish** and **Swish**: The other modern idea of some approximation of ReLU is Swish function: $$ swish = x\cdot\text{sigmoid}(x), \ \ \ swish'=\sigma(x)+x\sigma(x)(1-\sigma(x))$$ It could also be parametric swish with learnable parameter β: $$pswish(x)=x\cdot\text{sigmoid}(\beta x)$$ and Hard-Swish $$ hswish = x\frac{\text{ReLU6}(x+3)}{6}, \ \ \ hswish' = \frac{\text{ReLU6}(x+3)}{6}+x \cdot \begin{cases} \mbox{1} & \mbox{if } x \in (0 , 6) \\ \mbox{0} & \mbox{if } x \leq 0 \text{ and }x\geq 6 \end{cases}. $$ it is almost similar to swish but it is less expensive computationally since it replaces sigmoid (exponential function) with a ReLU (linear type).



The other modification of swish is the **mish**: $$ \text{Mish}(x) = x \tanh(\ln(\mathrm{e}^x+1)) \ \ \ \text{Mish}'(x) = \tanh(\ln(\mathrm{e}^x+1))+\dfrac{x\mathrm{e}^x\operatorname{sech}^2(\ln(\mathrm{e}^x+1))}{\mathrm{e}^x+1}$$



1.2 Weights initialization

1.2.1 Popular Initialization functions

In the case if you have not pre-trained neural network it is necessary to initialize weights.

There are following recommendation for weights initialization:

- Never initialize all the weights to zero or to the same value.

  If all weights have the same values, in all subsequent iterations the weights are going to remain the same (they will move away from value but they will be equal), this symmetry will be hard to break during the training. Hence weights connected the same neuron should never be initialized to the same value. This kind of phenomenon is known as **symmetry breaking problem**.

- Use some random initializations (less then 1).
  - using too small values or too high values lead to the gradient explosion and vanishing.
  - it is need to note, that it is bad to use values near 0.5 for sigmoid and about 0 for tanh due to the high probability of gradient vanishing.
- **Standard Normal initialization**

The initialization with same normal distribution with fixed deviation value: $$w_i \sim N\big[0,\sigma\big],\sigma\leq 1$$ where $N = N(\mu,\sigma)$ is the normal distribution with mean value μ and variance σ.

- **Standard Uniform initialization**

  The initialization with same uniform distribution with fixed deviation value: $$w_i \sim U\big[-\epsilon,\epsilon\big],\epsilon\leq 1$$ where $U = U(-\epsilon,\epsilon)$ is the uniform distribution in the range from $-\epsilon$ to ϵ.

- **Lecun Weights Initialization**

  If variance or dispersion of values is predefined then with increasing the amount of values the dispersion of out put increase also, thus use dispersion value inversely to the amount of layer input and/or outputs or the amount of layer parameters.

  The normal distribution with variance inver $$\text{variance} = \sigma^2 = \frac{1}{n_i}, w_i \sim N\big[0,\frac{1}{\sqrt{n_i}}\big] $$ where n_i is the number of layer inputs is the **Lecun Weights Initialization**.

Note

The Lecun derivation is based in the idea, that $$ \begin{align} &\text{Var}(s) = \text{Var}(\sum_i^n w_ix_i) = \sum_i^n \text{Var}(w_ix_i) \\\\ &= \sum_i^n [E(w_i)]^2\text{Var}(x_i) + [E(x_i)]^2\text{Var}(w_i) + \text{Var}(x_i)\text{Var}(w_i) \\\\ &= \sum_i^n \text{Var}(x_i)\text{Var}(w_i) = \left(n \text{Var}(w) \right) \text{Var}(x) \\\\ &\text{Thus, due to }\text{Var}(aX) = a^2\text{Var}(X),\text{ we need to have }a = \frac{1}{\sqrt{n}} \end{align}$$

- **Xavier Initialization** (or also **Glorot initialization**)

  The method is based on the supposition that in the classification task the variance of each result should be 1 in average.

  This case allow to avoiding gradient vanishing and explosion.

  For network it can be considering that the result of each layer is liner

independent with other ones, due to this it is sufficient to corresponding each one layer differently, for whom the dispersion can be proposed to be geometric average of inputs and outputs :

- **Xavier uniform initialization**: $$w_i\sim U\big[-\frac{\sqrt{6}}{\sqrt{n_{i}+n_{i+1}}},\frac{\sqrt{6}}{\sqrt{n_{i}+n_{i+1}}}\big]$$ Where U - is the uniform distribution, w_i is the weights of i-th layer, n_{i}, n_{i+1} are the number of inputs and outputs (inputs of the next layer).
- **Xavier normal initialization**: $$w_i\sim N\big[0,\frac{\sqrt{2}}{\sqrt{n_{i}+n_{i+1}}}\big]$$ Where N - is the normal distribution, w_i is the weights of i-th layer, n_{i}, n_{i+1} are the number of inputs and outputs.
- **On the practice**: $$\begin{aligned} &w_i\sim U\big[-\frac{2}{n_i},\frac{2}{n_i}\big], \text{ for Xavier uniform initialization} \\ \end{aligned} $$ where n_i - is the number of weights in layer.

- **Xavier uniform initialization are recommended for sigmoid layer.**

  - in some cases of **tanh layer** it is recommended to use $w_i\sim N\big[0,\frac{1}{{n_{i}}}\big]$

- **He Initialization** (or also Kaiming Initialization)

  The method is taking into account that **ReLU** has non-symmetric output. $$ w_i\sim N\big[0 ,\frac{2}{n_i} \big]$$ The He method is most popular choose for ReLU and based on it functions.

  In some cases it is recommended to use for LRELU and PReLU $$ w_i\sim N\big[0 ,\frac{2}{(1+\alpha^2)n_i} \big]$$, where α is 0.01 for LReLU and hyper parameters for PReLU.

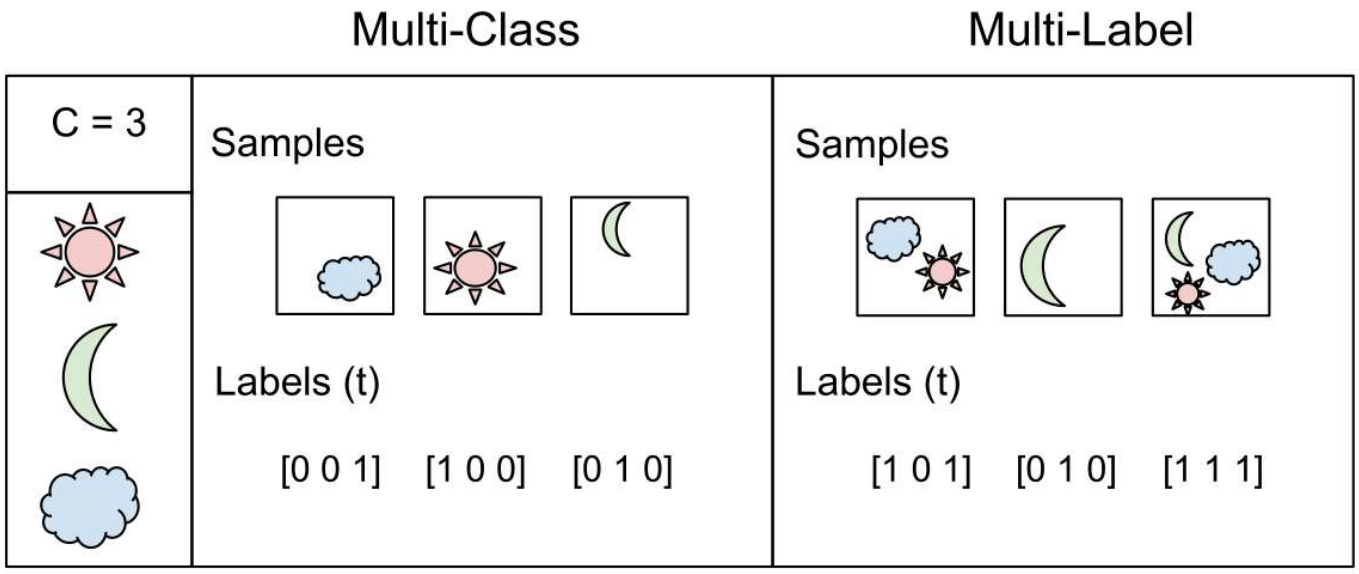
- **Nguyen – Widrow Initialization**

  The idea is to chooses values in order to distribute the active region of each layer output approximately evenly across as the layer's input space. $$ w_i=\beta\frac{ w_i (RND)}{||w_i (RND))||_2} , \ \ \ \beta = 0.7 n_i^{1/M}, \ \ RND\sim U(−0.5,0.5) $$ where M is the number of layers; $||w_i (RND))||_2=\sqrt{\sum_j w_{ij}^2}$.

1.2.2 Recommendations for using activation functions

The main recommendations for using activation functions and weights initialization are:

- Start from ReLU for the hidden layers and softmax / sigmoid / linear for the output depends on the task:
  - for binary classification use sigmoid.
  - for multi-class classification use softmax (for mutually exclusive cases such as dog/cat/goose).
  - for multi-label binary classification use sigmoid (for mutually non-exclusive cases, such as dog or cat and seat or stay).
  - for regression task use linear activation.
- Leaky ReLU can be the first solution to the problem of the gradients' vanish.
- All hidden layers typically use the same activation function.
- Most common that output layer does not have an activation function if it is include in the loss function.
- use Xavier uniform initialization for sigmoid/softmax layers.
- use He normal initialization for ReLU layers.
- use LeCun normal initialization for SELU layers.


Multi-Class
Multi-Label
C = 3
Samples
Labels (t)
[0 0 1] [1 0 0] [0 1 0]
Samples
Labels (t)
[1 0 1] [0 1 0] [1 1 1]

In [ ]: