**Activation Functions Derivatives Logic**

[Here](https://www.tensorflow.org/api_docs/python/tf/keras/activations) is the list of all the activation functions of Tensorflow.

Common math for all the developments that follow:

is the input sent to the activation function.

is the output obtained from the activation function.

Once we have the slope of the tangent function (u), we then proceed to get the independent term. So that:

Thus:

[**elu(...)**](https://www.tensorflow.org/api_docs/python/tf/keras/activations/elu)**: Exponential Linear Unit**

**Definition:**

**Derivative:**

**Code:**

# Elu:

def af\_elu(af\_in, af\_out, alpha=1):

if af\_in<0:

u = af\_out + alpha

v = af\_out - u\*af\_in

else:

u, v = 1, 0

return u, v

[**exponential(...)**](https://www.tensorflow.org/api_docs/python/tf/keras/activations/exponential)**: Exponential activation function.**

**Definition:**

**Derivative:**

**Code:**

# Exponential:

def af\_exp(af\_in, af\_out):

u = af\_out

v = af\_out - u\*af\_in

return u, v

[**gelu(...)**](https://www.tensorflow.org/api_docs/python/tf/keras/activations/gelu)**: Applies the Gaussian error linear unit (GELU) activation function.**

**Definition:**

**Derivative:**

Luckily, we can obtain the derivative of the erf() directly through the  [Fundamental Theorem of Calculus (First Part)](https://proofwiki.org/wiki/Derivative_of_Error_Function). Also see [Wikipedia](https://en.wikipedia.org/wiki/Error_function):

Our argument of the erf() has a factor , one can apply the chain rule to obtain the derivative of this calling :

Let us differentiate with respect to :

Regardless of the value of approximate, we can replace the first term of u by :

Then:

**Code:**

# Gelu:

def af\_gelu(af\_in, af\_out):

u = af\_out/af\_in + (af\_in/(np.sqrt(2\*np.pi()))) \* np.exp( - (af\_in\*\*2)/2 )

v = af\_out - u\*af\_in

return u, v

[**hard\_sigmoid(...)**](https://www.tensorflow.org/api_docs/python/tf/keras/activations/hard_sigmoid)**: Hard sigmoid activation function.**

**Definition:**

**Derivative:**

**Code:**

# Hard Sigmoid:

def af\_hardsigmoid(af\_in, af\_out):

if af\_in<-2.5:

u, v = 0, 0

elif 2.5<af\_in:

u, v = 0, 1

else:

u, v = 0.2, 0.5

return u, v

[**linear(...)**](https://www.tensorflow.org/api_docs/python/tf/keras/activations/linear)**: Linear activation function (pass-through).**

**Definition:**

**Derivative:**

**Code:**

# Linear:

def af\_linear(af\_in, af\_out):

u, v = 1, 0

return u, v

[**relu(...)**](https://www.tensorflow.org/api_docs/python/tf/keras/activations/relu)**: Applies the rectified linear unit activation function.**

**Definition:**

**Derivative:**

**Code:**

# Relu:

def af\_relu(af\_in, af\_out, alpha=0, max\_value=np.inf, threshold=0):

if af\_out >= max\_value:

return 0, max\_value

if threshold < af\_in:

u, v = 1, 0

else:

u, v= alpha, -alpha\*threshold

return u, v

[**selu(...)**](https://www.tensorflow.org/api_docs/python/tf/keras/activations/selu)**: Scaled Exponential Linear Unit (SELU).**

**Definition:**

**Derivative:**

**Code:**

# Selu:

def af\_selu(af\_in, af\_out):

# alpha and scale are pre-defined constants:

alpha=1.67326324

scale=1.05070098

if af\_in<0:

u = scale\*alpha\*np.exp(af\_in)

v = af\_out - u\*af\_in

else:

u, v = scale, 0

return u, v

[**sigmoid(...)**](https://www.tensorflow.org/api_docs/python/tf/keras/activations/sigmoid)**: Sigmoid activation function, sigmoid(x) = 1 / (1 + exp(-x)).**

**Definition:**

**Derivative:**

Then:

**Code:**

# Sigmoid:

def af\_sigmoid(af\_in, af\_out):

u = af\_out\*(1-af\_out)

v = af\_out - u\*af\_in

return u, v

[**softmax(...)**](https://www.tensorflow.org/api_docs/python/tf/keras/activations/softmax)**: Softmax converts a vector of values to a probability distribution.**

**Definition:**

Assuming we have -dimensional input vector in the problem, and number of neurons in this softmax layer. Then, for each neuron of the layer we have a given input and an output . We know that the input can be represented as a linear function as , where are the coefficients of the linear function. To simplify, we are considering the independent term as the d+1 dimension, where . Our goal is to find the new linear function that represents the output of the neuron , which we will define as . In other words, we need to find the coefficients.

Then,

**Derivative:**

Let us break the problem in two parts. First,

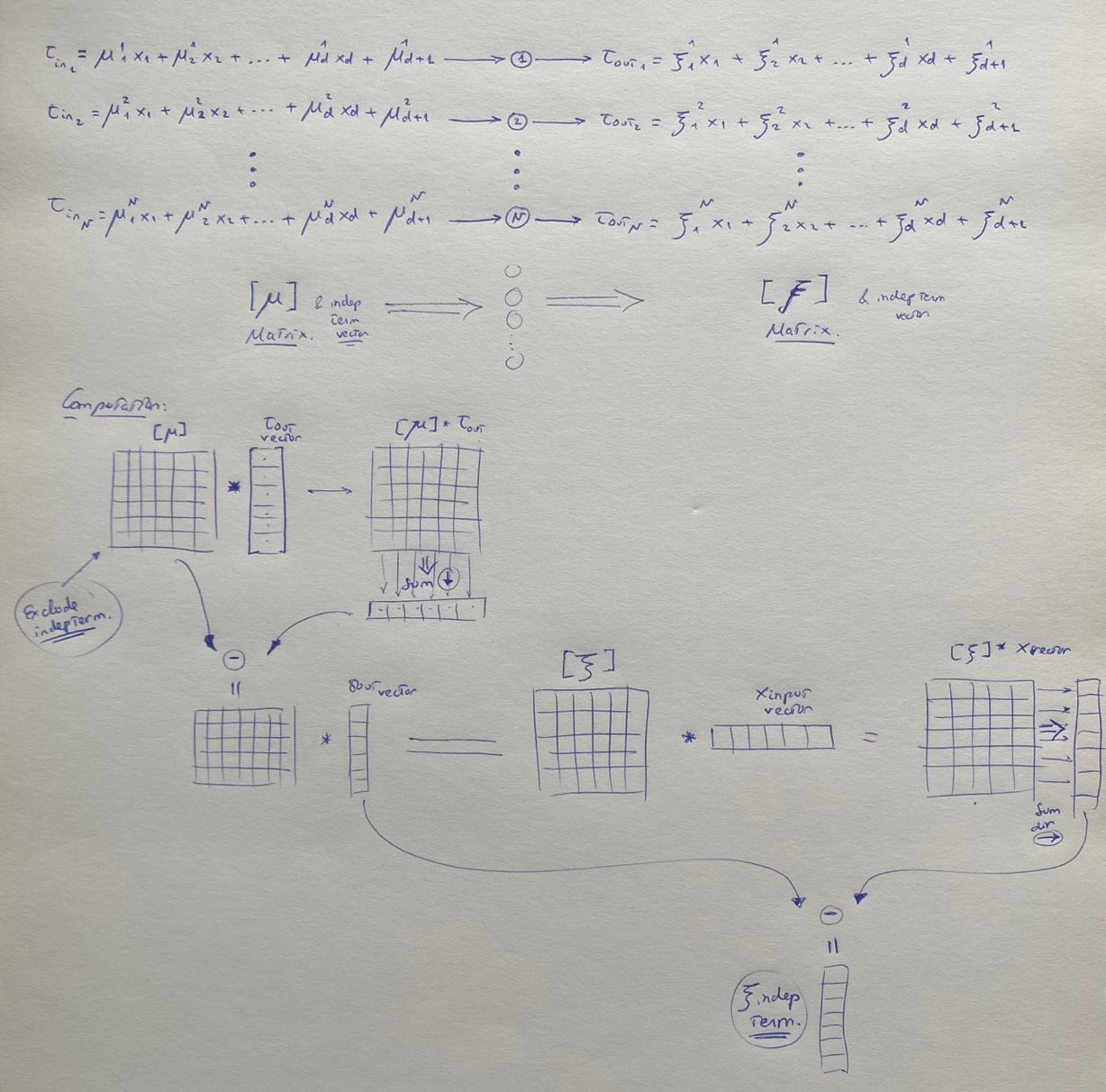
Second,

So, adding both terms,

Thus, the th dependent term of the neuron is:

Then the independent term can be calculated simply by substituting the input values of the instance in the following formula,

**Architecture:**

****

**Code:**

# Softmax (not normal, uses info of all the activation functions in the layer):

def afs\_softmax(in\_slopes\_mat, afs\_out\_vec, x\_in\_vec):

# afs\_out\_vec must be a column vector, x\_in\_vec must be a row vector:

afs\_out\_vec = afs\_out\_vec.reshape((-1, 1))

x\_in\_vec = x\_in\_vec.reshape((1, -1))

# Get the matrix of slopes for the outputs:

sum\_of\_prod\_1 = np.sum(np.multiply(in\_slopes\_mat, afs\_out\_vec), axis =0)

diff\_1 = np.subtract(in\_slopes\_mat, sum\_of\_prod\_1)

out\_slopes\_mat = np.multiply(diff\_1, afs\_out\_vec)

# Get the vector of independent terms for the outputs:

sum\_of\_prod\_2 = np.sum(np.multiply(out\_slopes\_mat, x\_in\_vec), axis=1).reshape((-1, 1))

out\_indeps\_vec = np.subtract(afs\_out\_vec, sum\_of\_prod\_2)

return out\_slopes\_mat, out\_indeps\_vec

[**softplus(...)**](https://www.tensorflow.org/api_docs/python/tf/keras/activations/softplus)**: Softplus activation function, softplus(x) = log(exp(x) + 1).**

**Definition:**

**Derivative:**

**Code:**

# Softplus:

def af\_softplus(af\_in, af\_out):

u = 1/(1+np.exp(-af\_in))

v = af\_out - u\*af\_in

return u, v

[**softsign(...)**](https://www.tensorflow.org/api_docs/python/tf/keras/activations/softsign)**: Softsign activation function, softsign(x) = x / (abs(x) + 1).**

**Definition:**

**Derivative:**

So, we can group as:

**Code:**

# Softsign:

def af\_softsign(af\_in, af\_out):

u = (1+np.abs(af\_in))\*\*(-2)

v = af\_out - u\*af\_in

return u, v

[**swish(...)**](https://www.tensorflow.org/api_docs/python/tf/keras/activations/swish)**: Swish activation function, swish(x) = x \* sigmoid(x).**

**Definition:**

**Derivative:**

Note that

So

Then:

**Code:**

# Swish:

def af\_swish(af\_in, af\_out):

u = (af\_out/af\_in) \* (1+af\_in-af\_out)

v = af\_out - u\*af\_in

return u, v

[**tanh(...)**](https://www.tensorflow.org/api_docs/python/tf/keras/activations/tanh)**: Hyperbolic tangent activation function.**

**Definition:**

**Derivative:**

**Code:**

# Tanh:

def af\_tanh(af\_in, af\_out):

u = (np.cosh(af\_in))\*\*(-2)

v = af\_out - u\*af\_in

return u, v