Activation Functions in Neural Networks | Analytics Vidhya

Introduction

In the domain of deep learning, a neural network absent of an activation function resembles a linear regression model. These functions drive a neural network's ability to handle intricate tasks by performing crucial non-linear computations. This article aims to delve into the derivatives, implementations, strengths, and limitations of various activation functions in neural network to help you make an informed choice and infuse non-linearity and precision into your neural network models.

This article was published as a part of the Data Science Blogathon.

Table of contents

- Introduction
- What is Activation Functions in Neural Network?
- Activation Functions in Linear Models
- Nonlinear Activation Functions
- Why Activation Functions?
- How to use it?
- What is a Good Activation Function ?
- Types of Activation Functions
  - Linear Activation Functions
  - Non-Linear Activation Functions
- When to use which Activation Function in a Neural Network?
  - Elements of Neural Networks
- Activation Functions and their Derivatives
- Implementation using Python
  - Sigmoid Activation Function
  - Tanh Activation Function
- Pros and Cons of Activation Functions
- Conclusion
- Frequently Asked Questions

What is Activation Functions in Neural Network?

Activation functions in Neural Network are used in a neural network to compute the weighted sum of inputs and biases, which is in turn used to decide whether a neuron can be activated or not. It manipulates the presented data and produces an output for the neural network that contains the parameters in the data. The activation functions are also referred to as *transfer functions* in some literature. These can either be linear or nonlinear depending on the function it represents and is used to control the output of neural networks across different domains.

Activation Functions in Linear Models

For a linear model, a linear mapping of an input function to output is performed in the hidden layers before the final prediction for each label is given. The input vector *x* transformation is given by:

f(x) = wT . x + b

where, *x* = input, *w* = weight, and *b* = bias

Linear results are produced from the mappings of the above equation and the need for the activation function arises here, first to convert these linear outputs into non-linear output for further computation, and then to learn the patterns in the data. The output of these models are given by:

y = (w1 x1 + w2 x2 + … + wn xn + b)

These outputs of each layer are fed into the next subsequent layer for multilayered networks until the final output is obtained, but they are linear by default. The expected output is said to determine the type of activation function that has to be deployed in a given network.

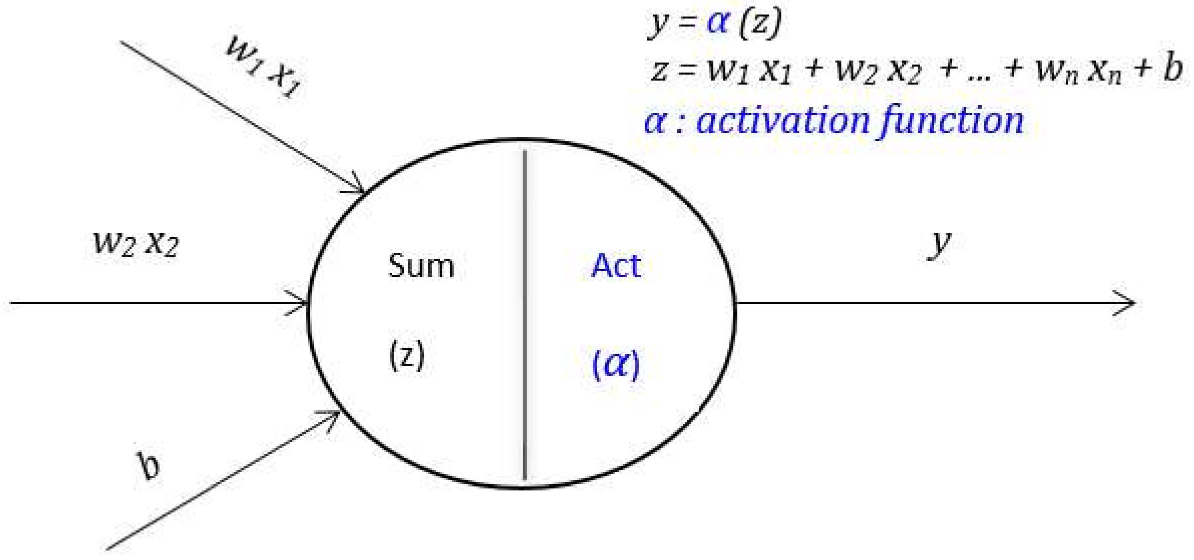
Nonlinear Activation Functions

However, since the outputs are linear in nature, the nonlinear activation functions are required to convert these linear inputs to non-linear outputs. These transfer functions, applied to the outputs of the linear models to produce the transformed non-linear outputs are ready for further processing. The non-linear output after the application of the activation function is given by:

y = α (w1 x1 + w2 x2 + … + wn xn + b)

where *α* is the activation function

Why Activation Functions?


$y = \alpha(z)$
$z = w_1 x_1 + w_2 x_2 + \dots + w_n x_n + b$
α : activation function
$w_1 x_1$
$w_2 x_2$
b
Sum
(z)
Act
(α)
y

The need for these activation functions includes converting the linear input signals and models into non-linear output signals, which aids the learning of high order polynomials for deeper networks.

How to use it?

In a neural network every neuron will do two computations:

- *Linear summation of inputs:* In the above diagram, it has two inputs *x1, x2* with weights *w1, w2,* and bias *b*. And the linear sum *z = w1 x1 + w2 x2 + ... + wn xn + b*
- *Activation computation:* This computation decides, whether a neuron should be activated or not, by calculating the weighted sum and further adding bias with it. The purpose of the activation function is to introduce non-linearity into the output of a neuron.

Most neural networks begin by computing the weighted sum of the inputs. Each node in the layer can have its own unique weighting. However, the activation function is the same across all nodes in the layer. They are typical of a fixed form whereas the weights are considered to be the learning parameters.

What is a Good Activation Function_?_

A proper choice has to be made in choosing the activation function to improve the results in neural network computing. All activation functions must be monotonic, differentiable, and quickly converging with respect to the weights for optimization purposes.

Types of Activation Functions

The different kinds of activation functions include:

Linear Activation Functions

A linear function is also known as a straight-line function where the activation is proportional to the input i.e. the weighted sum from neurons. It has a simple function with the equation:

f(x) = ax + c

The problem with this activation is that it cannot be defined in a specific range. Applying this function in all the nodes makes the activation function work like linear regression. The final layer of the Neural Network will be working as a linear function of the first layer. Another issue is the gradient descent when differentiation is done, it has a constant output which is not good because during backpropagation the rate of change of error is constant that can ruin the output and the logic of backpropagation.

Non-Linear Activation Functions

The non-linear functions are known to be the most used activation functions. It makes it easy for a neural network model to adapt with a variety of data and to differentiate between the outcomes.

These functions are mainly divided basis on their range or curves:

a) Sigmoid Activation Functions

Sigmoid takes a real value as the input and outputs another value between 0 and 1. The sigmoid activation function translates the input ranged in $(-\infty,\infty)$ to the range in $(0,1)$

b) Tanh Activation Functions

The tanh function is just another possible function that can be used as a non-linear activation function between layers of a neural network. It shares a few things in common with the sigmoid activation function. Unlike a sigmoid function that will map input values between 0 and 1, the Tanh will map values between -1 and 1. Similar to the sigmoid function, one of the interesting properties of the tanh function is that the derivative of tanh can be expressed in terms of the function itself.

c) ReLU Activation Functions

The formula is deceptively simple: *max(0,z)*. Despite its name, Rectified Linear Units, it's not linear and provides the same benefits as Sigmoid but with better performance.

Leaky Relu

Leaky Relu is a variant of ReLU. Instead of being 0 when $z<0$, a leaky ReLU allows a small, non-zero, constant gradient α (normally, $\alpha=0.01$). However, the consistency of the benefit across tasks is presently unclear. Leaky ReLUs attempt to fix the “dying ReLU” problem.

Parametric Relu

PReLU gives the neurons the ability to choose what slope is best in the negative region. They can become ReLU or leaky ReLU with certain values of α.

d) Maxout

The Maxout activation is a generalization of the ReLU and the leaky ReLU functions. It is a piecewise linear function that returns the maximum of inputs, designed to be used in conjunction with the dropout regularization technique. Both ReLU and leaky ReLU are special cases of Maxout. The Maxout neuron, therefore, enjoys all the benefits of a ReLU unit and does not have any drawbacks like dying ReLU. However, it doubles the total number of parameters for each neuron, and hence, a higher total number of parameters need to be trained.

e) ELU

The Exponential Linear Unit or ELU is a function that tends to converge faster and produce more accurate results. Unlike other activation functions, ELU has an extra alpha constant which should be a positive number. ELU is very similar to ReLU except for negative inputs. They are both in the identity function form for non-negative inputs. On the other hand, ELU becomes smooth slowly until its output equal to -α whereas ReLU sharply smoothes.

f) Softmax Activation Functions

Softmax function calculates the probabilities distribution of the event over 'n' different events. In a general way, this function will calculate the probabilities of each target class over all possible target classes. Later the calculated probabilities will help determine the target class for the given inputs.

When to use which Activation Function in a Neural Network?

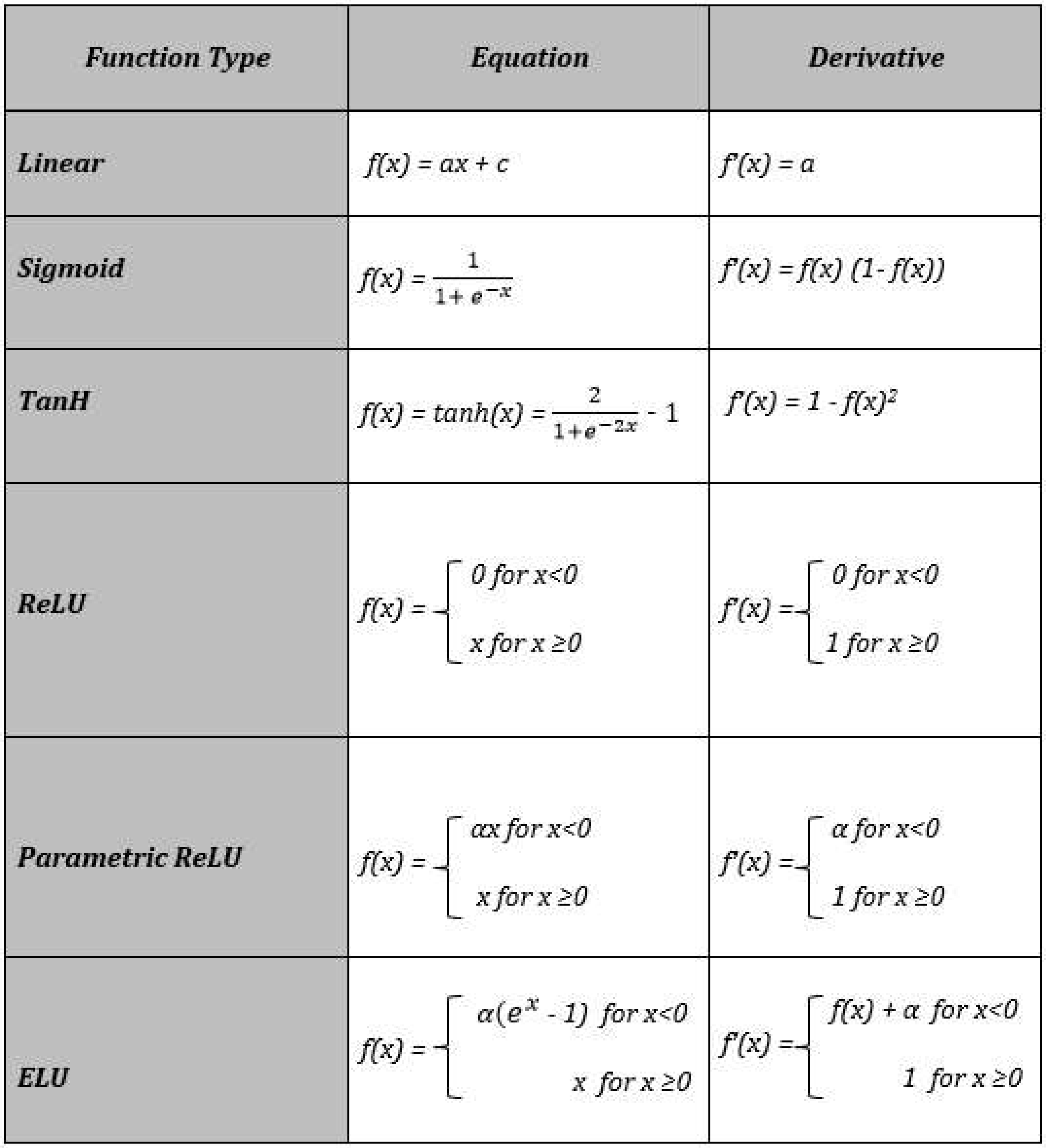
Specifically, it depends on the problem type and the value range of the expected output. For example, to predict values that are larger than 1, tanh or sigmoid are not suitable to be used in the output layer, instead, ReLU can be used. On the other hand, if the output values have to be in the range (0,1) or (-1, 1) then ReLU is not a good choice, and sigmoid or tanh can be used here. While performing a classification task and using the neural network to predict a probability distribution over the mutually exclusive class labels, the softmax activation function should be used in the last layer. However, regarding the hidden layers, as a rule of thumb, use ReLU as an activation for these layers.

In the case of a binary classifier, the Sigmoid activation function should be used. The sigmoid activation function and the tanh activation function work terribly for the hidden layer. For hidden layers, ReLU or its better version leaky ReLU should be used. For a multiclass classifier, Softmax is the best-used activation function. Though there are more activation functions known, these are known to be the most used activation functions.

Elements of Neural Networks

- **Input Layer:** This is the first layer of the neural network, and it takes in the data that the network is going to learn from. For example, if you are training a neural network to recognize handwritten digits, the input layer would receive an image of a handwritten digit.
- **Output Layer:** This is the last layer of the neural network, and it produces the output of the network. For example, if you are training a neural network to recognize handwritten digits, the output layer would produce a prediction of which digit is in the image.
- **Hidden Layers:** These layers are in between the input layer and the output layer, and they do all the hard work of learning the patterns in the data. Neural networks with more hidden layers are better at learning complex patterns, like the ones that tell a cat from a dog.
- **Neurons:** Neural networks are made up of tiny processing units called neurons. They are connected to each other by weights, and they process the data that is passed to them from other neurons. The weights determine how much influence one neuron has on another neuron.

Activation Functions and their Derivatives



Function Type	Equation	Derivative
Linear	$f(x) = ax + c$	$f'(x) = a$
Sigmoid	$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1- f(x))$
TanH	$f(x) = tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ReLU	$f(x) = \begin{cases} 0 \text{ for } x<0 \\ x \text{ for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 \text{ for } x<0 \\ 1 \text{ for } x \geq 0 \end{cases}$
Parametric ReLU	$f(x) = \begin{cases} \alpha x \text{ for } x<0 \\ x \text{ for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha \text{ for } x<0 \\ 1 \text{ for } x \geq 0 \end{cases}$
ELU	$f(x) = \begin{cases} \alpha(e^x - 1) \text{ for } x<0 \\ x \text{ for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha \text{ for } x<0 \\ 1 \text{ for } x \geq 0 \end{cases}$

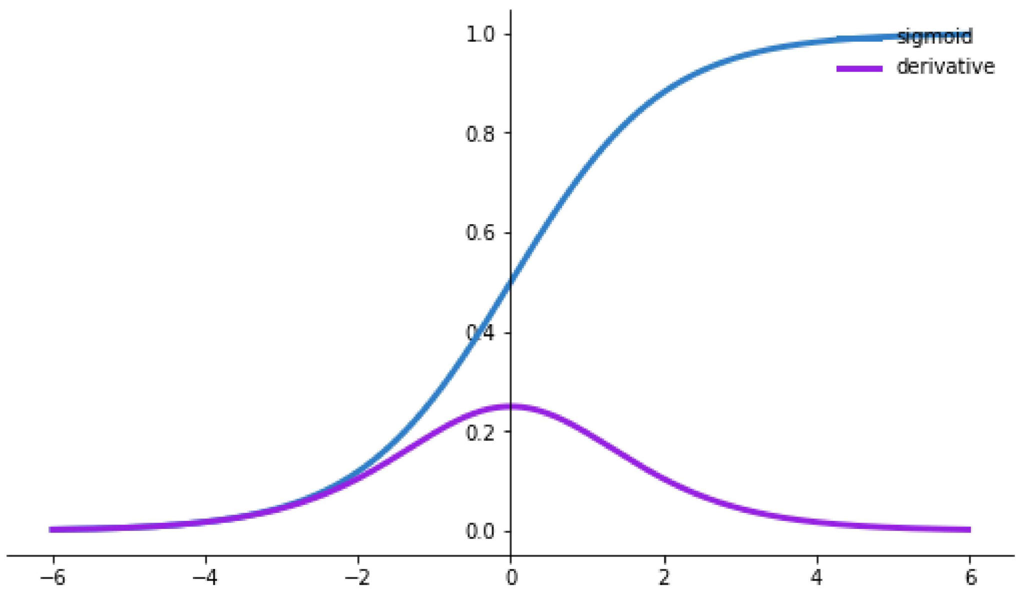
Implementation using Python

Having learned the types and significance of each activation function, it is also essential to implement some basic (non-linear) activation
functions using python code and observe the output for more clear understanding of the concepts:

Sigmoid Activation Function

```
import matplotlib.pyplot as plt
import numpy as np
def sigmoid(x):
    s=1/(1+np.exp(-x))
    ds=s*(1-s)
    return s,ds
x=np.arange(-6,6,0.01)
```

```
sigmoid(x)
fig, ax = plt.subplots(figsize=(9, 5))
ax.spines['left'].set_position('center')
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.xaxis.set_ticks_position('bottom')
ax.yaxis.set_ticks_position('left')
ax.plot(x,sigmoid(x)[0], color="#307EC7", linewidth=3, label="sigmoid")
ax.plot(x,sigmoid(x)[1], color="#9621E2", linewidth=3, label="derivative")
ax.legend(loc="upper right", frameon=False)
fig.show()
```



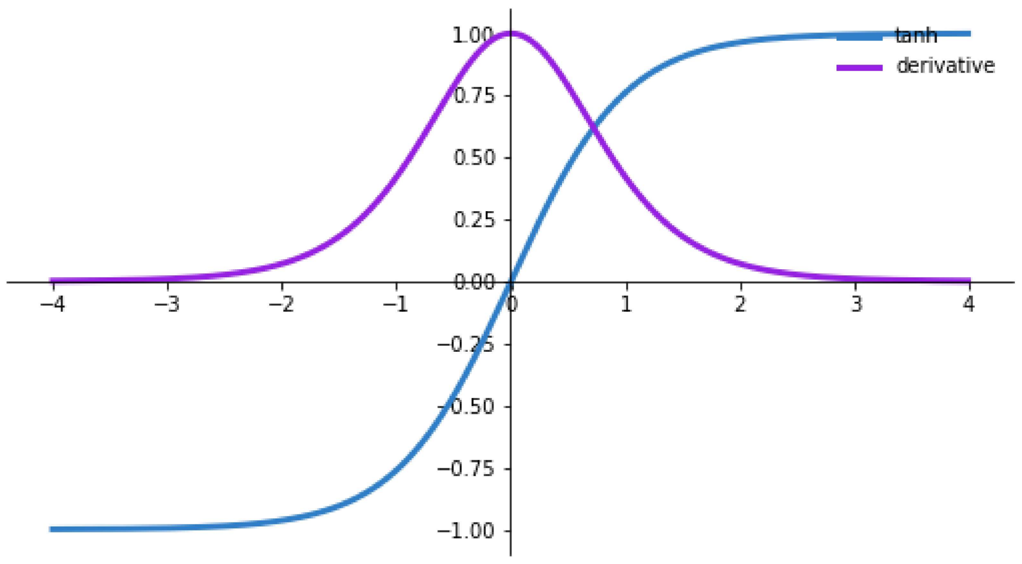
Observations:

- The sigmoid function has values between 0 to 1.
- The output is not zero-centered.
- Sigmoids saturate and kill gradients.
- At the top and bottom level of sigmoid functions, the curve changes slowly, the derivative curve above shows that the slope or gradient it is zero.

Tanh Activation Function

```
import matplotlib.pyplot as plt
import numpy as np
def tanh(x):
    t=(np.exp(x)-np.exp(-x))/(np.exp(x)+np.exp(-x))
    dt=1-t**2
    return t,dt
```

```
z=np.arange(-4,4,0.01)
tanh(z)[0].size,tanh(z)[1].size
fig, ax = plt.subplots(figsize=(9, 5))
ax.spines['left'].set_position('center')
ax.spines['bottom'].set_position('center')
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.xaxis.set_ticks_position('bottom')
ax.yaxis.set_ticks_position('left')
ax.plot(z,tanh(z)[0], color="#307EC7", linewidth=3, label="tanh")
ax.plot(z,tanh(z)[1], color="#9621E2", linewidth=3, label="derivative")
ax.legend(loc="upper right", frameon=False)
fig.show()
```



Observations:

- Its output is zero-centered because its range is between -1 to 1. i.e. -1 < output < 1.
- Optimization is easier in this method hence in practice it is always preferred over the Sigmoid function.

Pros and Cons of Activation Functions

- Type of Function: Linear
  - Pros: – Provides a range of activations, not binary. – Can connect multiple neurons and make decisions based on max activation. – Constant gradient for stable descent. – Changes are constant for error correction.
  - Cons: – Limited modeling capacity due to linearity.
- Type of Function: Sigmoid

  - Pros: – Nonlinear, allowing complex combinations. – Produces analog activation, not binary.
  - Cons: – Suffers from the “vanishing gradients” problem, making it slow to learn.
- Type of Function: Tanh
  - Pros: – Stronger gradient compared to sigmoid. – Addresses the vanishing gradient issue to some extent.
  - Cons: – Still prone to vanishing gradient problems.
- Type of Function: ReLU
  - Pros: – Solves the vanishing gradient problem. – Computationally efficient.
  - Cons: – Can lead to “Dead Neurons” due to fragile gradients. Should be used only in hidden layers.
- Type of Function: Leaky ReLU
  - Pros: – Mitigates the “dying ReLU” problem with a small negative slope.
  - Cons: – Lacks complexity for some classification tasks.
- Type of Function: ELU
  - Pros: – Can produce negative outputs for $x>0$.
  - Cons: – May lead to unbounded activations, resulting in a wide output range.

Conclusion

Activation functions in neural Networks serve to introduce non-linear properties to neural networks. Without them, neural networks perform linear mappings between inputs and outputs, essentially computing dot-products between input vectors and weight matrices. These successive dot-products result in repeated linear operations, which, in essence, are individual learn operations. To handle complex data, neural networks must approximate nonlinear relationships from input features to output labels. In the absence of these functions, neural networks cannot mathematically comprehend such intricate mappings, limiting their ability to tackle their intended tasks.

Frequently Asked Questions

Q1. What is activation function in neural networks?

a. In deep learning, an activation function in neural networks is like a switch. It decides if a neuron should be turned on or off based on the input it gets. This switch adds twists and turns to the network’s thinking, letting it understand and work with complicated patterns in data. This article talks about different activation functions in machine learning to help you choose the best one for your neural network.

Q2. What are activation functions for?

a. Neural networks can be made non-linear by utilizing activation functions. They enable the network to learn and comprehend intricate patterns in data, enabling it to handle more difficult tasks like language processing and image identification.

Q3. What is the best activation function in neural networks?

a. No single activation function is “best” in every circumstance. It relies on the particular issue you’re attempting to resolve as well as the features of your data. Sigmoid, tanh, and ReLU (Rectified Linear Activation) are a few common activation functions available. Every option has advantages and disadvantages, and the optimum decision is frequently reached through trial and experimentation.

Q4. What is the ELU activation function?

a. Exponential Linear Unit is referred to as ELU. It’s an activation function that resembles ReLU but differs in a few ways. Negative values are permitted in ELU, which helps address the “dying ReLU” issue, in which neurons may become immobile. When working with deep neural networks, in particular, ELU can be helpful in some circumstances.