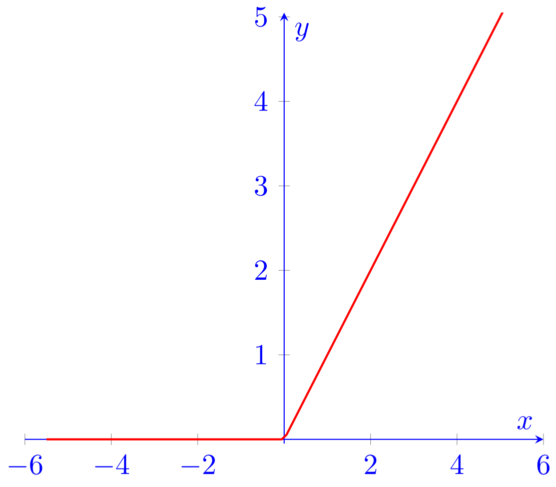
Machine Learning

Assignment 6.4

Submitted By: Ranji Raj

December 5, 2020

Rectified Linear Unit (ReLU)



- Mathematically,

$$ReLU(x) = max(0, x)$$

  or

$$ReLU(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

- A unit employing a rectifier is called as ReLU.
- Linearity signifies that the slope isn't plateau or saturate when x gets large.
- It is **non-linear** and **differentiable**.

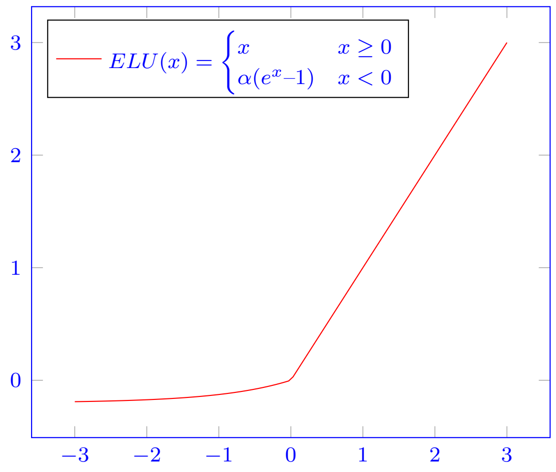
Pros

- It does not suffer from the phenomena of *vanishing gradient*. Thus a neural net employed with a relu activation function will not learn slowly and poorly.

Cons

- ReLU, however, suffers from another problem called the **dying neuron**. Meaning, for any negative value of x, the derivative is 0 and due this a bunch of weights and biases are incurred which are not updated eventually causing the neurons to die.
- It does not avoid the *exploding gradient* problem.

1 Exponential Linear Unit (ELU)-Variant of ReLU



- Mathematically,

$$ELU(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha(e^x - 1) & \text{if } x < 0 \end{cases}$$

- In the latter case the output not only depends on x-value, but also on a parameter α, which can be tuned further. Common pick for α value is $0.1 < \alpha < 0.3$.
- Furthermore, an exponential operation e^x is introduced, meaning it adds more to the computational effort.

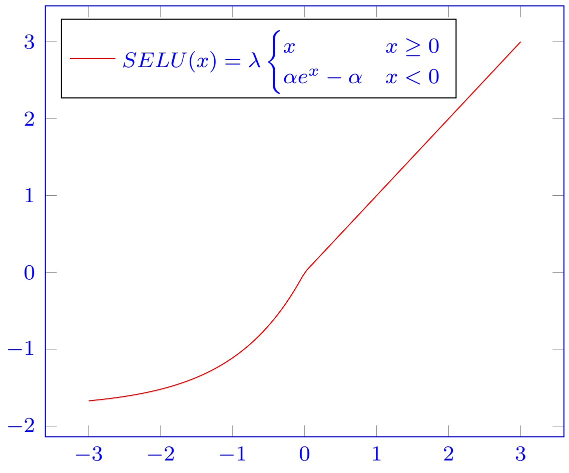
Pros

- Avoids the **dying relu** problem to some extent.
- Produce activations instead of letting them to be zero, when calculating the gradient.

Cons

- Introduces longer computations due to exponential operation.
- Does not avoid the *exploding gradient problem.*

2 Scaled Exponential Linear Unit (SELU)-Variant of ReLU



- Mathematically,

$$SELU(x) = \lambda \begin{cases} x & \text{if } x \geq 0 \\ \alpha e^x - \alpha & \text{if } x < 0 \end{cases}$$

- SELU is a self-normalizing activation function. For each input it is subtracted from the **mean**, then divided by the **standard deviation** i.e. computing standard scores.
$$Z = \frac{x_i - \mu}{\sigma}$$
- Thereby, the network components (weights, biases and activations) will have a mean, μ of zero and a standard deviation, σ of 1 after normalization.
- It has two hyper parameters,
$$\alpha \approx 1.6732632423543772848170429916717$$
$$\lambda \approx 1.0507009873554804934193349852946$$
the granuralities are for absolute precision, and are predetermined.
- The output of a SELU is normalized, called as *internal normalization*, hence the fact that all outputs are with μ=0 and σ=1, which is different from *external normalization*, where methods like batch normalization are used.

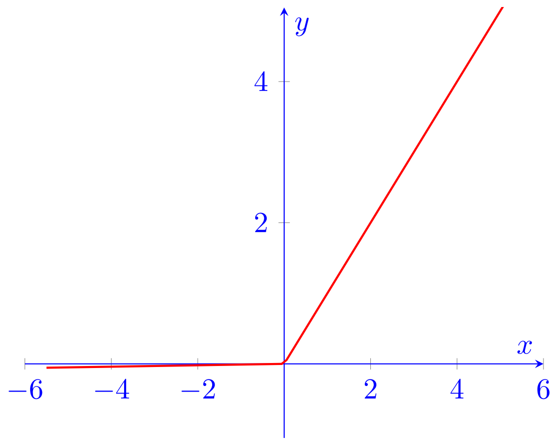
Pros

- Internal normalization is faster than external normalization, meaning the network converges **faster**.
- Free from vanishing and exploding gradient problem.

Cons

- Relatively new activation function hence, requires more research.

Leaky Rectified Linear Unit (Leaky ReLU)



- Mathematically,

$$LReLU(x) = \begin{cases} x & \text{if } x > 0 \\ 0.01x & \text{if } x \leq 0 \end{cases}$$

- If the input x-value is **greater** than zero, then it's the same as the ReLU i.e. x.
- If the input x-value is **lesser** than 0, then the output will be 0.01 times the input.
- Parametric ReLU (PReLU), a type of this where instead of having a predetermined slope like 0.01 makes it a **parameter** for the neural network to figure out itself.

Pros

- To some extent fixes the **dying relu** problem, by allowing a small gradient, when computing the derivative.
- Faster than ELU, no exponential operations involved.

Cons

- Becomes a linear function, when it is differentiated, whereas ELU is partially linear and non-linear.
- Does not escape from the *exploding gradient problem.*

3 Randomized Leaky Rectified Linear Unit (RReLU)-Variant of Leaky ReLU

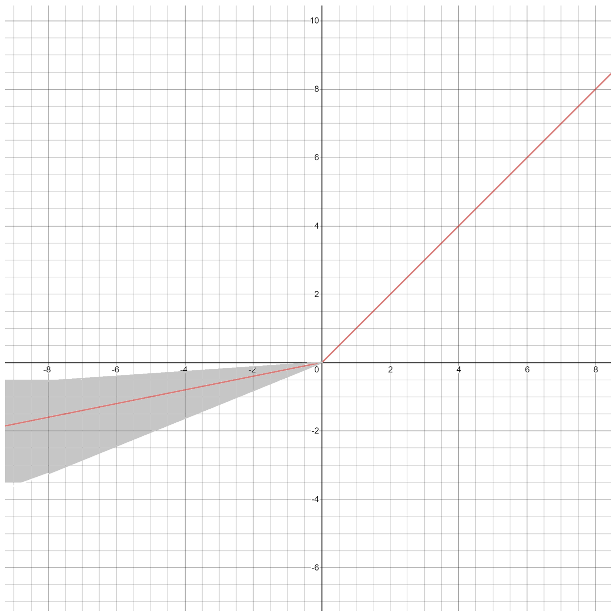


Figure 1: Plot of RReLU

- Mathematically,

$$y_{ji} = \begin{cases} x_{ji} & \text{if } x_{ji} \geq 0 \\ \alpha_{ji} * x_{ji} & \text{if } x_{ji} < 0 \end{cases}$$

  where,

$$\alpha_{ji} = U(l, u), \quad l < u \quad and \quad l, u \in [0, 1]$$

- The negative slope of RReLU is randomly calculated from a **uniform distribution** $U(l, u)$ and sets this slope to be a random value between an upper(u) and lower(l) bound during the training and an average, $(\frac{x_{ji}}{\frac{l+u}{2}})$ of these bounds during the testing.

Pros

- Outperforms other variants of ReLU.
- The motivation to introduce a random negative slope is to reduce **overfitting**.

Cons

- Becomes a linear function, when it is differentiated.