Department of Computing

Imperial College of Science, Technology and Medicine

Activation Functions

Lecture regarding Activation Layers and different types of normalization

Author: Anton Zhitomirsky

Contents

Definition 0.1 (BLANK Activation)**.**

$$f(X)$$

1 Activation Functions

They determine how neurons in the network respond, or "activate", when they receive a set of inputs. Activation functions introduce non-linear properties into the system, allowing the network to learn from complex data.

Activation functions are mathematical formulas that dictate the output of a neuron given a certain input. In essence, they act as the "gatekeepers" of each node, deciding how much signal should pass through to the next layer. A key point to remember is that activation functions introduce non-linearity into the network. This non-linearity is crucial because it allows the neural network to learn from complex and varied data. Without non-linear activation functions, your neural network would essentially become a simple linear regression model, incapable of learning complex functions.

We aim to move beyond binary "activated" or "not activated" outputs, and isntead seek a more nuanced, continuous range of outputs.

$$Input = \sum (w_i \cdot input) + b_i$$

A neuron applies weights and a bias to inputs it receives. This can vary between negative infinity and infinity.

We cannot simply truncate values because this isn't differentiable

1.1 Linear

Definition 1.1 (Linear Activation)**.**

$$f(x) = c \cdot x$$

- This produces a constant gradient, meaning that during backpropagation, the updates applied to weights are constant and independent of the change in input, denoted by Δx.
- If each layer in a multi-layered network employs a linear activation function, the output of one layer becomes the input to the next, perpetuating linearity throughout the network; no matter how many layers you have, the entire network behaves like a single-layer linear model. This means you could replace all N linear layers with just a single linear layer and achieve the same output. It renders the "depth" of the network irrelevant.

Therefore, while linear activation functions may have some use-cases, they aren't typically chosen for complex machine learning tasks that require the network to capture more complex, non-linear relationships in the data.

1.2 Sigmoid

Definition 1.2 (Sigmoid Activation)**.**

$$f(x) = \frac{1}{1 + e^{-x}}$$

- This function is non-linear, allowing us to stack layers in a neural network, thereby facilitating the learning of more complex representations.
- gives a more analog or continuous output w.r.t binary step function.

- Smooth gradient which is crucial for gradient descent algorithms. One notable characteristic is that between the X values of -2 and 2, the curve is especially steep. This implies that small changes in the input within this region result in significant shifts in output, facilitating rapid learning during the training phase.
- Towards the tails of the function, the curve flattens out, and the output values become less sensitive to changes in input. This results in a vanishing gradient problem, where gradients become too small for the network to learn effectively, leading to slow or stalled training.

1.3 Tanh

Definition 1.3 (Tanh Activation)**.**

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

- Scaled version of sigmoid but from -1 to 1 instead of 0 to 1
- non-linear, we can stack it
- since $(-1, 1)$ less concern about activations becoming too large and dominating the learning process
- One key benefit of tanh over sigmoid is that its gradient is stronger; that is, the derivatives are steeper. This can make it a better choice for certain problems where faster convergence is desired.
- its outputs are zero-centered, meaning the average output is close to zero. This is beneficial for the learning process of subsequent layers, as it tends to speed up convergence by allowing for a balanced distribution of outputs and gradients.

However, like the sigmoid function, tanh also suffers from the vanishing gradient problem when you stack many layers, which can slow down learning.

Careful normalization of the inputs is also essential when using tanh to ensure effective learning.

1.4 ReLU

Definition 1.4 (ReLU Activation)**.**

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

- piecewise linear that outputs the input directly if it is positive, otherwise, it outputs zero
- ReLU is inherently non-linear when considered as a whole, particularly due to the sharp corner at the origin.
- combinations of ReLU functions are also non-linear, enabling us to stack layers in neural networks effectively. Because it is a universal approximator.
- unbounded as $[0, \infty)$
- unboundedness can cause explosions of activations if not managed properly
- tends to produce sparse activations
- In a neural network with many neurons, using activation functions like sigmoid or tanh would cause almost all neurons to activate to some degree, leading to dense activations. ReLU, on the other hand, will often output zero, effectively ignoring some neurons, which can make the network more computationally efficient.

- Dying ReLU problem caused by often outputting zero (If a neuron's output is always zero (perhaps due to poor initialization), the gradient for that neuron will also be zero.) As a result, during backpropagation, the weights of that neuron remain unchanged, effectively "killing" the neuron. This can result in a portion of the neural network becoming inactive, thereby limiting its capacity to model complex functions.

1.5 Leaky ReLU

Definition 1.5 (Leaky ReLU Activation)**.**

$$f(x) = \begin{cases} x & for\ x \geq 0 \\ 0.01 \cdot x & for\ x < 0 \end{cases}$$

- addresses the "Dying ReLU problem"
- Leaky ReLU attempts to solve this by introducing a small slope for negative values, typically 0.01, to ensure the gradient is non-zero. This small slope allows "dead" neurons to reactivate during the course of training. In other words, it provides a pathway for gradients to flow, even when the neuron is not active.
- computationally efficient (simple math operations)
- maintains benefit of ReLU such as sparsity and ability to approximate a wide range of functions

1.6 PReLU

Definition 1.6 (Parametric ReLU Activation)**.**

$$f(x) = \begin{cases} x & for\ x \geq 0 \\ a \cdot x & for\ x < 0 \end{cases}$$

Where a is learnable.

- negative slope becomes a learnable parameter
- flexibility allows to adapt during training, potentially leading to better performance than Leaky ReLU in some scenarios.
- scale invariant - if you multiply the input by a scalar, the shape of the output remains the same, just scaled. In the context of CNN architectures, where scale invariance can be valuable, PReLU and its variants can be especially useful

1.7 SoftPlus

Definition 1.7 (SoftPlus Activation)**.**

$$f(x) = \frac{1}{\beta} \cdot \log(1 + e^{\beta \cdot x})$$

- smooth and (easier) differentiable approximation to ReLU
- parameterized by a scale factor β which controls how closely the function approximates the ReLU (highest meaning closest)
- outputs only positive values, suitable for layers where you specifically require positive activations

- numerical stability issues for large input values PyTorch implementation switches to a linear function when the condition $\beta \times x$ exceeds a predefined threshold
- non-linear across its entire domain
- SoftPlus is sensitive to the amplitude of the input signal, which means that it's non-linear regardless of the input size. That's beneficial for models where amplitude variation is a significant feature.

1.8 Exponential Linear Unit

Definition 1.8 (ELU Activation)**.**

$$f(x) = \max(0, x) + \min(0, \alpha \cdot (e^x - 1))$$

- extension to the ReLU, designed to be element-wise, operating on each element of the input independently.
- One thing that sets ELU apart is its ability to output negative values. Unlike ReLU, which only outputs positive values, ELU can go below zero.
- Being able to output negative values allows ELU to push the mean activation closer to zero. A zero-centered mean can help the network converge faster, a useful property in deep learning models.
- soft and smooth version of ReLU while still being positive and negative
- ELU is a strong candidate for scenarios where you want a balance of smoothness, differentiability, and the ability to have a mean activation around zero
- α can change everything

2 Loss

2.1 Why?

These functions measure how well your network is doing, quantifying the difference between the predicted outputs and the actual ground truth. Backpropagation uses this error measurement to update the model parameters, aiming to minimize this error.

3 Normalization