ReLU and Softmax Activation Functions

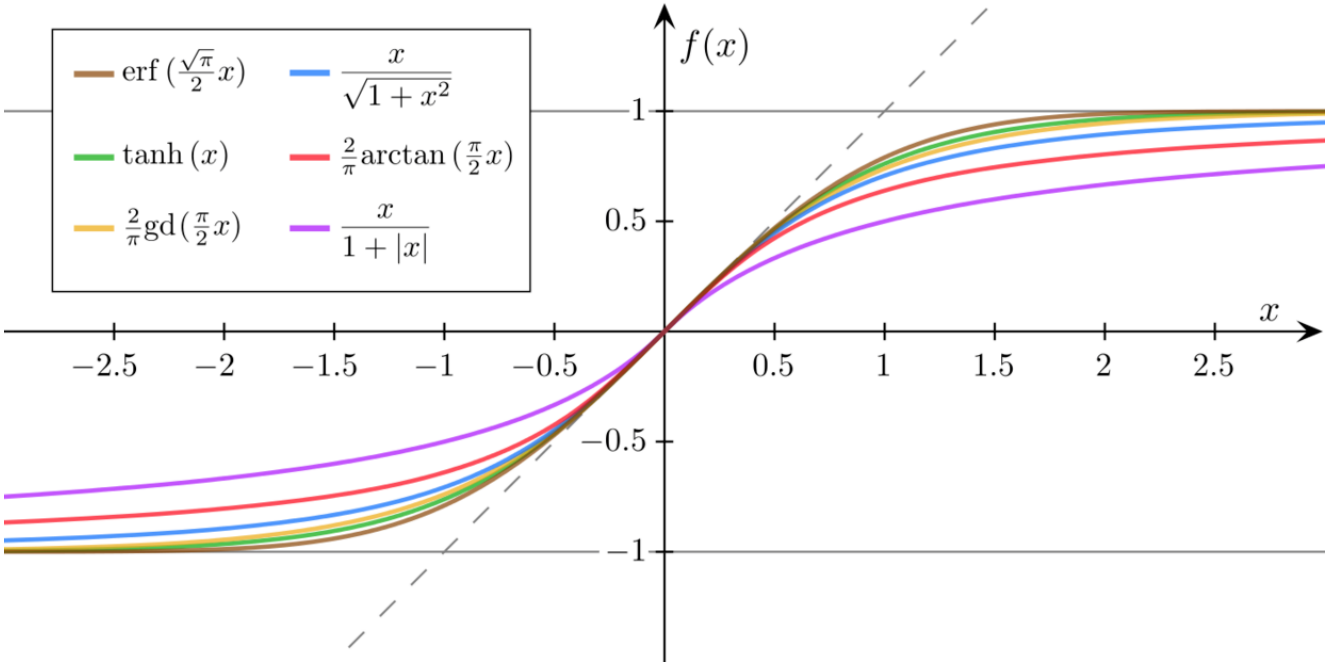
Jump to bottom

Ji Yang edited this page on Feb 11, 2017 · 2 revisions

You can find this article and source code at my GitHub

Refresher: The Sigmoid Function

The sigmoid function has been widely used in machine learning intro materials, especially for the logistic regression and some basic neural network implementations. However, you may need to know that the sigmoid function is not your only choice for the activation function and it does have drawbacks.



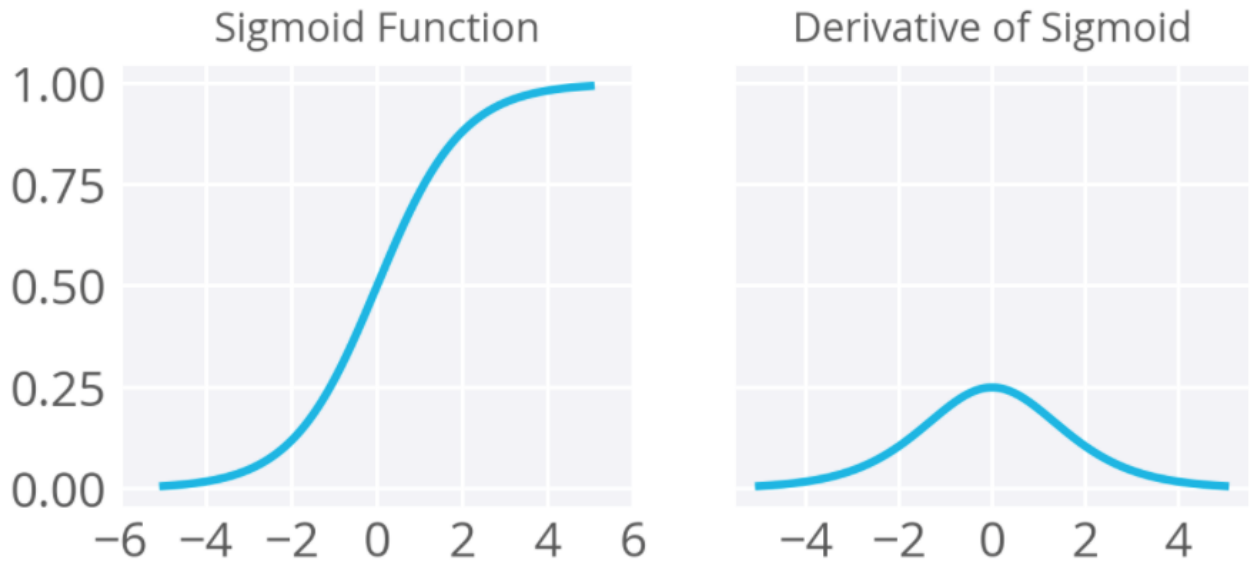
Let's see what a sigmoid function could benefit us.

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

By taking this formula, we can get the derivative of the sigmoid function, note that for shortening the formula, here f(x) is the sigmoid function.

$$f'(x) = f(x)(1 - f(x))$$

Despite that, such a result from the derivate is easy to calculate and save times for building models, the sigmoid function actually forces your model "losing" knowledge from the data. Why? Think about the possible maximum value of the derivative of a sigmoid function.



For the backpropagation process in a neural network, it means that your errors will be squeezed by (at least) a quarter at each layer. Therefore, deeper your network is, more knowledge from the data will be "lost". Some "big" errors we get from the output layer might not be able to affect the synapses weight of a neuron in a relatively shallow layer much ("shallow" means it's close to the input layer).

Due to this, sigmoids have fallen out of favor as activations on hidden units.

Addition notes from Wikipedia (just FYI, in case you are interested in):

> Besides the logistic function, sigmoid functions include the ordinary arctangent, the hyperbolic tangent, the Gudermannian function, and the error function, but also the generalised logistic function and algebraic functions

Rectified Linear Units

Instead of sigmoids, most recent deep learning networks use rectified linear units (ReLUs) for the hidden layers. A rectified linear unit has output 0 if the input is less than 0, and raw output otherwise. That is, if the input is greater than 0, the output is equal to the input. ReLUs' machinery is more like a real neuron in your body.

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

ReLU activations are the simplest non-linear activation function you can use, obviously. When you get the input is positive, the derivative is just 1, so there isn't the squeezing effect you meet on backpropagated errors from the sigmoid function. Research has shown that ReLUs result in much faster training for large networks. Most frameworks like TensorFlow and TFLearn make it simple to use ReLUs on the the hidden layers, so you won't need to implement them yourself.

However, such a simple solution is not always a perfect solution. From Andrej Karpathy's CS231n course:

> Unfortunately, ReLU units can be fragile during training and can "die". For example, a large gradient flowing through a ReLU neuron could cause the weights to update in such a way that the neuron will never activate on any datapoint again. If this happens, then the gradient flowing through the unit will forever be zero from that point on. That is, the ReLU units can irreversibly die during training since they can get knocked off the data manifold. For example, you may find that as much as 40% of your network can be "dead" (i.e. neurons that never activate across the entire training dataset) if the learning rate is set too high. With a proper setting of the learning rate this is less frequently an issue.

Softmax

The sigmoid function can be applied easily, the ReLUs will not vanish the effect during your training process. However, when you want to deal with classification problems, they cannot help much. Simply speaking, the sigmoid function can only handle two classes, which is not what we expect.

The softmax function squashes the outputs of each unit to be between 0 and 1, just like a sigmoid function. But it also divides each output such that the total sum of the outputs is equal to 1 (check it on the figure above).

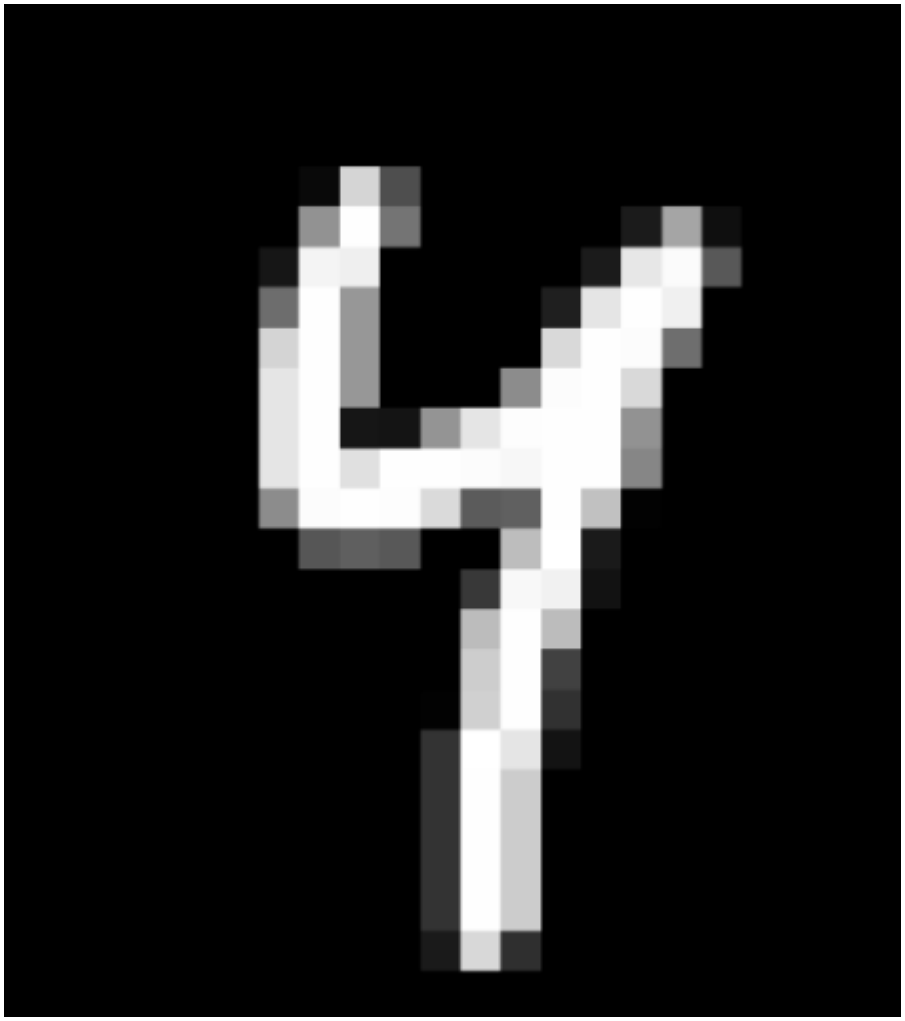
The output of the softmax function is equivalent to a categorical probability distribution, it tells you the probability that any of the classes are true.

Mathematically the softmax function is shown below, where z is a vector of the inputs to the output layer (if you have 10 output units, then there are 10 elements in z). And again, j indexes the output units, so j = 1, 2, ..., K.

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

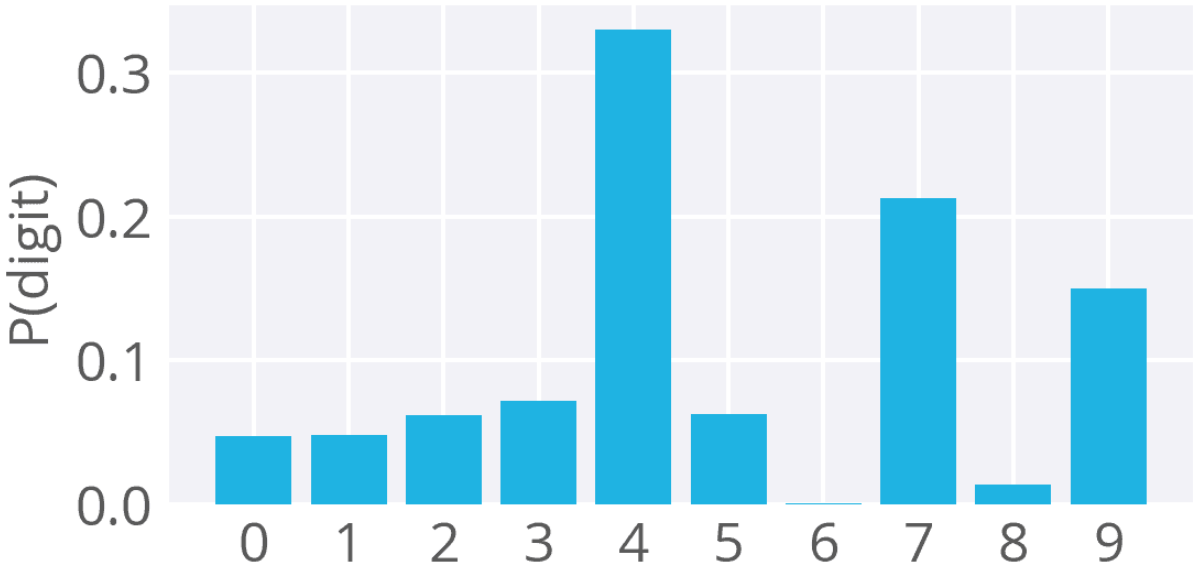
The material below is taken from a Udacity course.

To understand this better, think about training a network to recognize and classify handwritten digits from images. The network would have ten output units, one for each digit 0 to 9. Then if you fed it an image of a number 4 (see below), the output unit corresponding to the digit 4 would be activated.



Building a network like this requires 10 output units, one for each digit. Each training image is labeled with the true digit and the goal of the network is to predict the correct label. So, if the input is an image of the digit 4, the output unit corresponding to 4 would be activated, and so on for the rest of the units.

For the example image above, the output of the softmax function might look like:



The image looks the most like the digit 4, so you get a lot of probability there. However, this digit also looks somewhat like a 7 and a little bit like a 9 without the loop completed. So, you get the most probability that it's a 4, but also some probability that it's a 7 or a 9.

The softmax can be used for any number of classes. It's also used for hundreds and thousands of classes, for example in object recognition problems where there are hundreds of different possible objects.

Reference

- All formulas are generated by HostMath
- Sigmoid function - Wikipedia
- ImageNet Classification with Deep Convolutional Neural Networks
- The MNIST database by Yann LeCun

Thanks for reading. If you find any mistake / typo in this blog, please don't hesitate to let me know, you can reach me by email: jyang7[at]ualberta.ca

Pages 3

Find a Page...

Home

Introduction to Neural Network: Feedforward

ReLU and Softmax Activation Functions

Clone this wiki locally

```
https://github.com/Kulbear/deep-learning-nano-foundation.wiki.git
```