Programming Assignment 2: Universal Approximation

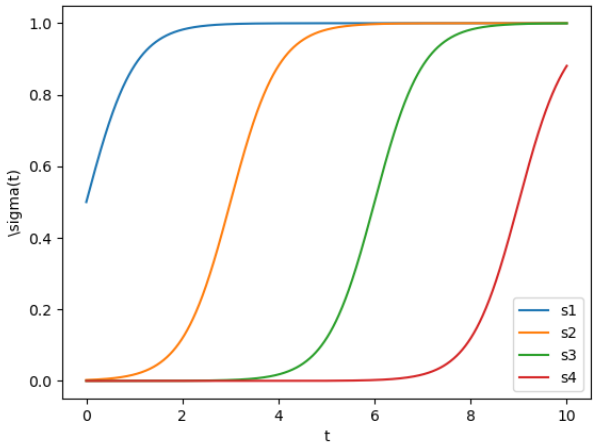
Due Feburary 17, 2023

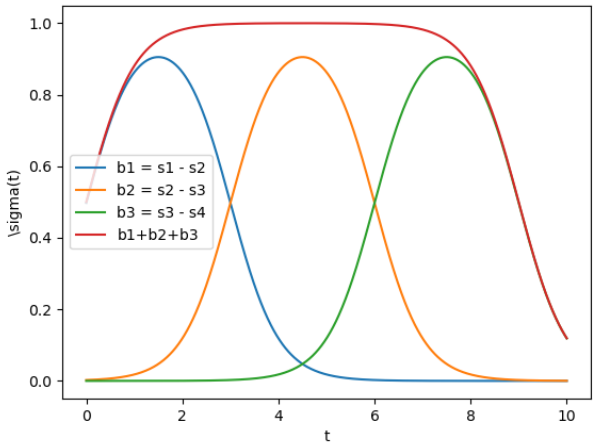
In the repository you have starter code for the second programming assignment, 'pa2.py.' In this assignment, you will convince yourself empirically of the Universal Approximation Theorem, which states that any continuous function on a bounded (compact) domain can be arbitrarily well approximated by a one(-hidden)-layer neural network. The code contains two classes, 'UniversalApprox,' and 'UANet;' the former you will use to construct smooth bump functions and generate neural network parameters, and the latter you will use to implement (only) the forward pass for the network with these parameters.

In the first part of this assignment you will fill out methods in UniversalApprox to generate a partition of unity approximation. The ingredients are the smooth sigmoidal of the form

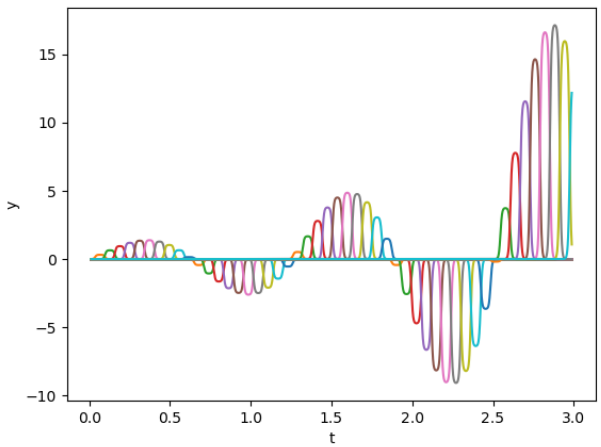
$$\sigma_{\alpha,\beta}(t) := \frac{1}{1+\exp\left(\frac{-1}{\alpha}(t-\beta)\right)},$$

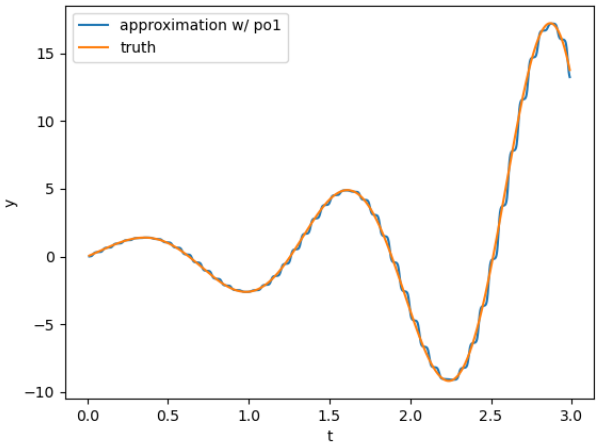
as shown in the figure below.





Observe that by offsetting successive sigmoidals and taking differences you can generate (smooth) bump functions whose apex you may multiply by the value of a function you are trying to approximate. Of course, this will only provide a local approximation, and the point is that by adding the collection of local approximations, you can arrive at a global approximation (see third figure).





You will generate a smooth grid and identify parameters of α, β so that you can approximate a function at grid lattice points. These parameters will feed into the constructor of a one-layer neural network. In the second part, therefore, you will manually instantiate a one-layer network (with sigmoidal activation in the hidden layer) and replicate the function approximation you constructed in part one with the network.

Outputs for this assignment should be plots. The starter code already provides the code for (some of) these plots, and your job is to fill in methods so that plots may successfully generate. You should, of course, experiment as your curiosity drives you.

Challenges in this assignment will likely concentrate at either understanding of the mathematics used for approximating functions and/or indexing and size issues with array operations. This will provide good practice, and if you find yourself very stuck don't hesitate to read through solution code and understand what it is doing. Then see if you can implement its ideas differently, e.g. by iterating with for loops (instead of using native numpy functionality). Note that we are doing no "training" in the typical sense. You are manually finding good weights from the get-go.