Homework #1 – Neural Network Overview – Activation Functions and Back Propagation

CAP4613/CAP 5619, Deep & Reinforcement Learning (Spring 2025), Department of Computer Science, Florida State University

---

Points: 60
Due: 11:59pm on Monday, February 10th, 2025

Submission: You need to submit electronically via Canvas by uploading separately a) a pdf file (named "**hw1-Firstname-Lastname.pdf**") for your answers to the questions, and b) the program(s) you have created (named as "**hw1-prog-Firstname-Lastname.???**"); if there are multiple program files, please zip them as a single archive. Here replace "Firstname" using your first name and replace "Lastname" using your last name in the file names. Note that you must upload the pdf file for your answers as a separate file and do not include the pdf in your archive file.

The main purpose of this assignment is to let you be familiar with neural network architecture elements including activation functions and how to define and train a simple neural network.

Problem 1 (24 points, 4 point each) As neural networks are typically trained using (stochastic) gradient descent optimization algorithms, properties of the activation functions affect the learning. Here we divide the domain of an activation function into: 1) **fast learning region** if the magnitude of the gradient is larger than 0.99, 2) **active learning region** if the magnitude of the gradient is between 0.01 and 0.99 (inclusive), 3) slow learning region if the magnitude of the gradient is larger than 0 but smaller than 0.01, and 4) **inactive learning region** if the magnitude of the gradient is 0. For each of the following activation functions, **plot** its gradient in the range from -5 to 5 of the input and then **list** the four types of regions. If the gradient is not well defined for an input value, indicate so and then use any reasonable value.

(1) Rectified linear unit $f(z) = \begin{cases} z & if\ z \geq 0 \\ 0 & otherwise \end{cases}$.

(2) Logistic sigmoid activation function $f(z) = \sigma(z) = \frac{1}{1+e^{-z}}$.

(3) Piece-wise linear unit $f(z) = \begin{cases} 0.2z + 0.8 & if\ z > 1 \\ z & if\ 1 \geq z \geq -1 \\ 0.2z - 0.8 & Otherwise \end{cases}$.

(4) Swish $f(z) = z\sigma(3z)$, where $\sigma(z) = \frac{1}{1+e^{-z}}$.

(5) Exponential Linear Unit (ELU) $f(z) = \begin{cases} z & if\ z \geq 0 \\ 0.1(e^{z} - 1) & otherwise \end{cases}$. (Note here is a special case of the general ELU function with α=0.1.)

(6) Gaussian Error Linear Unit (GELU), $\mathrm{f(x)} = \frac{\mathrm{x}}{2}\left(1 + \mathrm{erf}\left(\frac{\mathrm{x}}{\sqrt{2}}\right)\right)$, where erf is the error function, (also known as the Gauss error function), given by $erf(x) = \frac{2}{\sqrt{\pi}}\int_0^{\mathrm{x}} \mathrm{e}^{-\mathrm{t}^2}\,\mathrm{dt}$. (Note that there is an approximation using $\mathrm{x}\sigma(1.702\mathrm{x})$ and there are approximations for the error function one could use; however, typical deep learning frameworks provide an efficient implementation of the error function.)

Problem 2 (16 points) Here we use a simple neural network for solving the XOR problem given in the textbook (Section 6.1) but with two changes: we classify the input for being class 1 by adding a sigmoid activation function and initialize *b* as -0.5 instead of 0. In other words, the neural network will be given as

$$\mathrm{f(x; W, c, w, b)} = \sigma(\mathrm{w^T max\{0, W^T x + c\} + b}),$$

where σ is the sigmoid activation function. The parameters are initialized as follows:

$$W = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \quad c = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \quad w = \begin{bmatrix} 1 \\ -2 \end{bmatrix}, \quad b = -0.5.$$

The output of the neural network is the probability that the input belongs to class 1, which implies the probability for belonging to class 0 is 1 - the output. We will use the cross-entropy loss and the training set consists of all the four samples as given in the textbook.

(1) (**8 points**) Compute the output and its loss for the given network for each of the four training samples using Algorithm 6.3 in the textbook.

(2) (**8 points**) Compute the gradients for all the parameters for each of the four training samples using Algorithm 6.4 in the textbook.

Problem 3 (20 points) Using a deep learning framework you have established, implement the neural network in the previous problem.

(1) (**10 points**) Verify the results from your implementation is the same as you have for the previous problem.

(2) (**6 points**) Train your network for 100 epochs using stochastic gradient descent (with a batch size of 1 and learning rate of 0.1). Plot the training loss with respect to the epoch at the end of each epoch, and then comment on the effectiveness of gradient descent.

(3) (**4 points**) For a neural network, we define an optimal adversarial example for a set of samples as the input with the smallest distance to any sample in the set with a different classification compared to that sample. Here a sample is classified as class 1 if the output is higher than 0.5 and as class 0 if the ouput is lower than 0.5; when the output is exactly 0.5, it is ambiguous. Find an optimal adversarial example for the initial network and your trained network. (To be more precise for this problem, OAE (optimal adversarial example) for a model f and a set T is given as $\mathrm{OAE}(f, T) = \underset{x}{\operatorname{argmin}}\{\| x\text{-}t \|_2, \text{where } t \in T \text{ and } (f(x)\text{-}0.5) \times (f(t)\text{-}0.5) < 0\}$; please note the OAE is not necessarily unique.)

Extra Credit Problem

Problem 4 (6 points) JumpReLU is a variant of ReLU, defined as $\mathrm{JumpReLU}(x) = \begin{cases} x & \text{if } x \geq \theta \\ 0 & \text{otherwise} \end{cases}$, where θ is a parameter (see https://www.stat.berkeley.edu/~mmahoney/pubs/ICPRAM_2020_100.pdf for more details).

(1) (**2 points**) Explain why JumpReLU could improve the robustness of a ReLU neural network by replacing all the ReLU activation functions using JumpReLU functions, assuming the parameter for each JumpReLU is chosen optimally.

(2) (**4 points**) For the neural network you have trained for the previous problem, replace the ReLU using a JumpReLU with an optimized parameter and recompute the optimal adversarial example. Describe your findings and give explanations.