Homework #2 – Neural Network Overview – Activation Functions and Back Propagation - Solutions
CAP 5619, Deep & Reinforcement Learning (Spring 2020), Department of Computer Science, Florida State University

---

Points: 75
Due: Beginning of the class (11:00am) on Thursday, February 6, 2020

Problem 1 (20 points) 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 0.9, 2) active learning region if the magnitude of the gradient is between 0.1 and 0.9 (inclusive), 3) slow learning region if the magnitude of the gradient is larger than 0 but smaller than 0.1, 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 -10 to 10 and then list the four types of regions. If the gradient is not well defined for an input value, any reasonable value is acceptable.

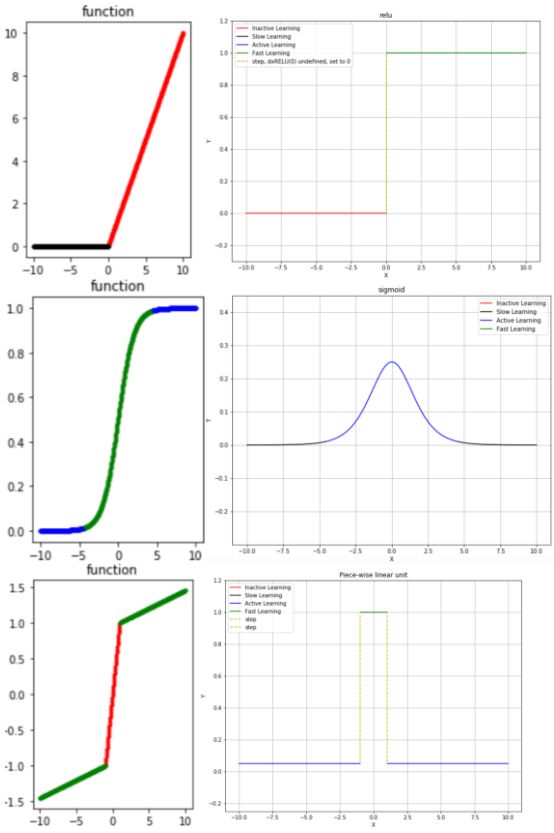
(1) Rectified linear unit $f(z) = \begin{cases} z & if\ z \geq 0 \\ 0 & otherwise \end{cases}$.

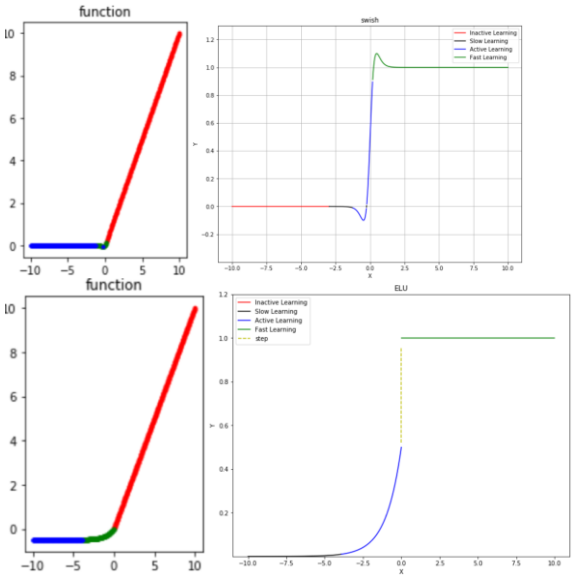
(2) Logistic sigmoid activation function $f(z) = \sigma(z) = \frac{1}{1+e^{-z}}$.

(3) Piece-wise linear unit $f(z) = \begin{cases} 0.05z + 0.95 & if\ z > 1 \\ z & if\ 1 \geq z \geq -1 \\ 0.05z - 0.95 & Otherwise \end{cases}$.

(4) Swish $f(z) = z\sigma(5z)$, where $\sigma(z) = \frac{1}{1+e^{-z}}$.

Thanks to Zezhong and Daniel for the following solution:





Problem 2 (15 points) Answer the following questions regarding the back propagation algorithm.

(1) (**10 points**) Describe in your words Algorithm 6.1 and Algorithm 6.2 in the Deep textbook (on pages 202 and 204). For each algorithm, you need to describe what each line is doing and why the algorithm would work correctly.

(2) (**5 points**) Explain in your own words why the two algorithms require "the same order of the number of computations". (See page 204 of the textbook for more information.)
Note that each edge is used once in the forward computation as well as in the backpropagation computation. So the complexity is on the order of the number of edges for both of the algorithms.

Problem 3 (20 points) Using the framework you have established, design and train a neural network to learn how to approximate

$$g(x) = \sin\left(\frac{3\pi}{4}x\right).$$

Note that your neural network should have as few as possible neurons in the hidden layer(s) and the largest error (i.e., the absolute difference between g(x) and the output of your neural network) should be no more than 0.05 in the range of inputs from -1 to 1. You can generate and use no more than 200 training samples.

There are many acceptable solutions. For example, your network may have two hidden layers and at least ten neurons in each layer if you use ReLU or sigmoid activation functions in the hidden layer to approximate the nonlinearity of the sinusoid function.

Problem 4 (**20 points**) The weights and biases of a softmax layer (implemented using equations (6.28) and (6.29) in tbe Deep textbook) can be downloaded from http://www.cs.fsu.edu/~liux/courses/deepRL/assignments/hw2_softmax_weights.m. (Note that they are given as a Matlab program and you should be able to extract the numbers easily from the file.)

(1) (**2 points**) What is the architecture of the softmax layer? In other words, how many inputs, how many neurons, and how many connections are there in the layer?
There are 100 inputs, 20 neurons, and 20x100+20=2020 connections.

(2) **(4 points)** Classify the following example, which can be downloaded from http://www.cs.fsu.edu/~liux/courses/deepRL/assignments/hw2_softmax_sample.txt .
The output from the softmax unit is
[[1.30782995e-01 1.23871563e-03 1.21610559e-01 1.46294122e-01
8.47877610e-03 2.13691213e-02 1.51689880e-02 2.00111944e-02
1.33690762e-01 2.57447239e-02 8.72133367e-03 4.24674462e-02
4.01887446e-09 8.06923030e-02 6.63685798e-02 8.49256060e-02
4.70475825e-02 2.70795867e-02 8.18395274e-03 1.01236478e-02]]

Therefore, it is classified as from the fourth class. (In Python, C/C++, and Java, the index is 3.)

(3) **(8 points)** Suppose that we use cross entropy loss, the learning rate is 0.1, and the correct label for the sample is the first class, after we apply one step gradient decent optimization using the given sample in (2), how many of the weights and biases will be increased, decreased, or remain the same respectively? For numerical stability, we consider that any change smaller than 0.0005 is the same.

In this sample, the loss is given by

$L = -\ln(y_1) = -\ln\left(\frac{e^{z_1}}{\sum_{i=1}^{20} e^{z_i}}\right)$=2.03421586, $\frac{\partial L}{\partial z_1} = -1 + y_1$, and $\frac{\partial L}{\partial z_k} = y_k$, for k≠1.

$\frac{\partial L}{\partial W} = h \times \left(\frac{\partial L}{\partial z}\right)^T$ and $\frac{\partial L}{\partial b} = \frac{\partial L}{\partial z}$.

Among all the weights, 10 will increase, 125 will decrease, and 1865 will remain roughly the same; among all the biases, 1 will increase, 17 will decrease, and 2 will remain roughly the same.

Note that a very common mistake is to assume $\frac{\partial L}{\partial z_k} = 0$, for k≠1.

(4) **(6 points)** Compute a small vector so that the new sample created by adding the vector to the given sample will be classified as from the second class (i.e., class 1 if you labels the classes as 0, 1, …, 19). The length of the vector (in L_2 norm) should be as small as possible. Explain how you have obtained your solution and confirm that the resulting sample is indeed classified as from the second class.

Note that in order to classify the new sample to be from the second class, all we need to do is to make sure the second component is the largest.

Let $z_0 = W^T \times h_0 + b$

$z_{new} = W^T \times (h_0 + \Delta h) + b = z_0 + W^T \times \Delta h$

Here we like to minimize $\Delta h^T \times \Delta h$ subject to $z_{new}(2) \geq z_{new}(i)$, where $i \neq 2$ (i starts from 1). This is a quadratic optimization problem with linear inequality constraints. You can set up to solve using any stand quadratic optimization solver.

I solved the problem in Octave using its quadratic solver qp. The solution we got is

0.020836 -0.114083 -0.010279 -0.039335 -0.002464
0.019965 -0.014249 0.001381 -0.032855 0.015095
-0.003797 0.016129 -0.030553 -0.078239 0.061061
-0.032860 -0.209327 -0.098069 -0.015851 -0.020002
-0.000650 0.005107 0.054764 -0.011149 0.001317
-0.008060 0.074133 -0.014791 0.003902 0.030253
-0.116441 -0.042089 -0.035096 0.138119 0.010355
-0.025982 0.159807 -0.231517 0.016665 -0.028942
-0.023300 0.004138 -0.060562 -0.011834 -0.005384
0.024927 0.041678 -0.029999 -0.108009 -0.015116
0.052752 0.016213 -0.024596 0.013910 -0.086675
-0.000299 -0.002600 0.025188 -0.016305 -0.093204
-0.066342 -0.015584 -0.040319 0.037159 0.002399
-0.000325 -0.027802 -0.001942 0.014363 0.051891
-0.019013 -0.085118 0.004838 0.021656 -0.026425
-0.010305 0.007372 -0.003590 0.009972 0.002364
0.000439 -0.102526 0.014582 -0.039927 -0.066515
0.034031 0.008517 0.021302 -0.038425 0.002871
-0.008871 0.013301 -0.025234 0.009841 -0.008295
-0.040072 0.202526 -0.106859 0.028711 0.004015

Its L2 norm is 0.588675 L_inf norm: 0.231517 L1_narm: 3.727896.

(5) **(extra credit option 3 points)** As in (4), could you make a small change to the given sample so that it will be classified as from the second class but with the maximal changed value to be minimized? In

other words, we like the vector to be as small as possible according to the L_∞ norm. Explain how you have obtained your solution and confirm that the resulting sample is indeed classified as from the second class.

In this case, we need to introduce an auxiliary variable t; which will be non-negative. We will require all the variables to be bounded by t. Then just need to minimize t subject to the bounds given by t and also that the second one needs to the largest to be classified correctly. I solved it using Octave's linear programming function. Here is the solution I computed:

0.098971 -0.098971 0.098971 -0.098971 -0.098971
0.098971 -0.098971 0.098971 -0.098971 0.098971
0.098971 0.000047 -0.098971 -0.098971 0.098971
-0.098971 -0.098971 -0.098971 -0.098971 -0.098971
-0.098971 0.098971 0.098971 -0.098971 -0.098971
0.098971 0.098971 -0.006501 0.098971 0.098971
-0.098971 -0.098971 -0.098971 0.098971 -0.098971
-0.098971 0.098971 -0.098971 0.098971 -0.055109
-0.098971 0.098971 -0.098971 -0.098971 0.098971
0.098971 0.098971 -0.098971 -0.098971 -0.098971
0.098971 0.098971 -0.038806 0.098971 -0.098971
-0.098971 0.098971 0.098971 -0.098971 -0.098971
-0.098971 -0.098971 0.098971 0.098971 0.098971
0.098971 -0.098971 0.098971 0.098971 0.098971
-0.098971 -0.098971 0.098971 0.098971 -0.098971
-0.098971 0.098971 -0.098971 0.098971 0.098971
0.098971 -0.098971 0.098971 -0.098971 -0.098971
0.098971 0.098971 0.098971 -0.098971 0.098971
-0.098971 0.098971 -0.098971 0.098971 -0.098971
-0.098971 0.098971 -0.098971 0.098971 0.098971

L2_norm: 0.972074 L_inf norm: 0.098971 L1_narm: 9.601673.

Note that its infinity norm is much smaller even though its L2 norm and L1 norm have increased.

Extra Credit Problem

Problem 5 (8 points) In the paper "Distributed Representations of Words and Phrases and their Compositionality" (which can be downloaded from https://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf), an objective function based on negative sampling is defined (see Equation (4) in the paper).

(1) (**2 points**) Suppose that we use k negative samples to approximate the expectation, write down the objective function for one sample. Make sure that you define/explain all the variables in your function.

(2) (**4 points**) Derive the learning rule using gradient descent optimization for the objective function given in (1).

(3) (**2 points**) Explain the key differences between the learning rule for part (2) and the one we derived in class for the skip gram model; if you think that the two learning rules are equivalent, justify your answer.