Homework 1

Feed forward neural network, Back propagation

Instructor: Dr. Souvik Chakraborty
APL 745: Deep Learning for Mechanics
January 23, 2024
Submission due on February 08, 2024

Instructions:
(i) You are allowed to discuss in a group; however, you must submit your own handwritten homework copy (no computer-typed submission will be accepted). Further, copying homework from your friends is forbidden. If found copied, the homework submission will be considered invalid for all the students involved and will be graded zero.
(ii) Write all the steps, including your reasoning and the formulae you referred to, if any. If sufficient reasoning is not provided and steps are unclear, step marks will not be given.
(iii) For practical submissions, the codes are accepted in *.ipynb* or *.py* format.
(iv) Unless mentioned otherwise, only *numpy*, *pytorch* and *matplotlib* libraries may be used. Direct commands for algorithms to be implemented are not allowed.
(v) The *.rar* file containing all submission related files shall be named in the format, **Name_Entrynumber.rar**

Question 1. [15 + 15 = 30 Marks] Evaluation of Gradient and Hessian:
A feed-forward network consists of a set of units, each of which computes a weighted sum of its inputs:

$$a_k = \sum_i w_{ki} z_i$$

where z_i is either the activation of another unit or an input unit that sends a connection to unit k, and w_{ki} is the weight associated with that connection. Biases can be included in this sum by introducing an extra unit or input with activation fixed at $+1$, and so we do not need to deal with biases explicitly. The sum in the above equation, known as a pre-activation, is transformed by a nonlinear activation function $h(\cdot)$ to give the activation z_k of unit k in the form

$$z_k = h(a_k)$$

Note that one or more of the variables z_i in the sum of the first equation could be an input, and similarly, the unit k in the second equation could be an output.

Consider a sum-of-squares error function, so that for data point n the error is given by

$$E_n = \frac{1}{2} \sum_{k=1}^{K} (y_k - t_k)^2$$

where y_k is the activation of output unit k, and t_k is the corresponding target value for a particular input vector x_n. Now take a two-layer neural network, and define the quantities

$$\delta_k = \frac{\partial E_n}{\partial a_k}, \quad M_{kk'} \equiv \frac{\partial^2 E_n}{\partial a_k \partial a_{k'}}.$$

a) Derive expressions for the elements of the Hessian matrix expressed in terms of δ_k and $M_{kk'}$ for elements in which (i) both weights are in the second layer, (ii) both weights are in the first layer, and (iii) one weight is in each layer.

b) Extend the above results for the exact Hessian of a two-layer network to include skip-layer connections that go directly from inputs to outputs.

Question 2. [5+10 +10 = 25 Marks] Activation functions:
a) Show that the derivative of the logistic-sigmoid activation function can be expressed in terms of the function value itself as:

$$\frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x))$$

Also derive the corresponding result for the *tanh* activation function.

b) If $\zeta(a)$ is the softplus activation function and $\sigma(a)$ is the logistic-sigmoid activation function, then show that:

$$\zeta(a) - \zeta(-a) = a$$
$$\ln \sigma(a) = -\zeta(-a)$$
$$\frac{d\zeta(a)}{da} = \sigma(a)$$
$$\zeta^{-1}(a) = \ln(\exp(a) - 1)$$

c) The swish activation function (Ramachandran, Zoph, and Le, 2017) is defined by:

$$h(x) = x\,\sigma(\beta x)$$

where $\sigma(x)$ is the logistic-sigmoid activation function. When used in a neural network, β can be treated as a learnable parameter. Plot the graphs of the swish activation function as well as its first derivative for $\beta = 0.1$, $\beta = 1.0$, and $\beta = 10$. Show that when $\beta \to \infty$, the swish function becomes the ReLU function.

Question 3. [10+10+20 = 40 Marks] Loss function for a multi-class neural network:
a. Show that maximizing the likelihood for a multi-class neural network model in which the network outputs have the interpretation $y_k(\mathbf{x}, \mathbf{w}) = p(t_k = 1|\mathbf{x})$ is equivalent to minimizing the cross-entropy error function given below.

$$E(\mathbf{w}) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{kn} \ln y_k\left(\mathbf{x}_n, \mathbf{w}\right)$$

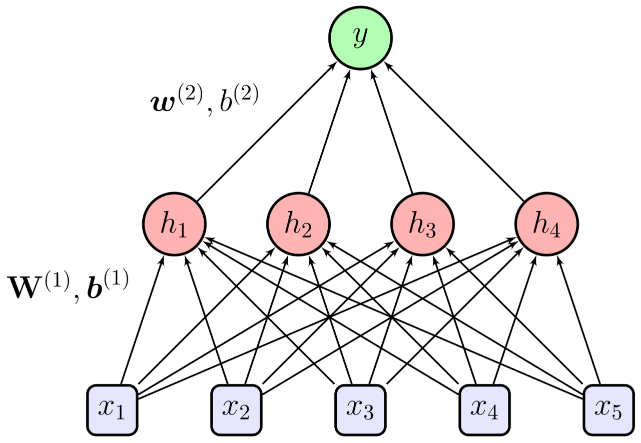
b. Show that the derivative of the error function shown above with respect to the pre-activation a_k for output units having a softmax activation function satisfies $\dfrac{\partial E}{\partial a_k} = y_k - t_k$.

c. Use cross-entropy error as the loss function and optimize the neural network having two hidden layers with 20 neurons each and sigmoid activation function. Take suitable number of nodes in input and output layers. The output layer will have a softamx activation function. The dataset to be used is given in this link. Predict the output class labels for the for the test dataset (use 80:20 as the train: test ratio for the given dataset). Also, compute the misclassification ratio. Please note that you should write your own code for the optimizer algorithm for the training the network parameters.

Question 4. [5+5+5+5+5 = 25 Marks] Feed Forward Neural Network:
In this problem, you will find a set of weights and biases for a multilayer perceptron which determines if a list of five numbers is in descending order. More specifically, you receive five inputs x_1, x_2, x_3, x_4 and x_5, where $x_i \in \mathbb{R}$, and the network must output 1 if $x_1 > x_2 > x_3 > x_4 > x_5$ and 0 otherwise. You will use the following Multilayer Perceptron (MLP) architecture consisting of one input layer with five nodes, one hidden layer with four nodes, and one output layer with one node. The activation function of the hidden units and the output unit can be assumed to be a hard threshold function:

$$\sigma(z) = \begin{cases} 1, & \text{if } z \geq 0 \\ 0, & \text{if } z < 0 \end{cases}$$

Find a set of weights and biases for the network which correctly implements this function (**including cases where some of the inputs are equal**). Note, in some cases, if some of the inputs are equal to each other (say $x_2 = x_3$), your output should be 0. Your answer should include:
1. A 4×5 weight matrix $W^{(1)}$ for the hidden layer.



y
$\boldsymbol{w}^{(2)}, b^{(2)}$
h_1
h_2
h_3
h_4
$\mathbf{W}^{(1)}, \boldsymbol{b}^{(1)}$
x_1
x_2
x_3
x_4
x_5

FIGURE 1. Architecture

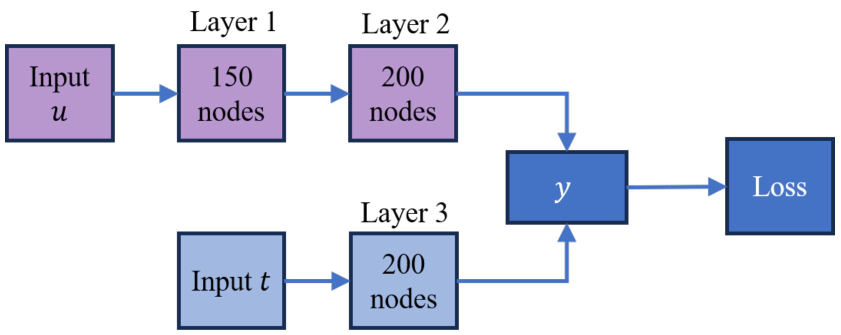
2. A 4-dimensional vector of biases $b^{(1)}$ for the hidden layer.
3. A 4-dimensional vector of weights $w^{(2)}$ for the output layer.
4. A scalar bias $b^{(2)}$ for the output layer.

Give brief explanation supporting your solutions and compute output for the following sets of inputs using the various weights and bias matrices.
Test inputs (x_1, x_2, x_3, x_4, x_5):
1. 10, 20, 30, 40, 50
2. -20, -10, 0, 10, 20
3. 10, 20, 20, 20, 30
4. 10, 5, 20, 15, 30
5. 50, 40, 30, 20, 10

Question 5. [5+10+10 = 25 Marks] Backpropagation:
1. For the network shown in Fig. 2,

- Compute the total number of trainable parameters.
- Compute the expressions for the gradients, $\dfrac{dL}{d\mathbf{W}_i}$ and $\dfrac{dL}{d\boldsymbol{b}_i}$, i = 1,2,3, using backpropagation. Use matrix notation and write the dimensions for various gradient tensors (obtained throughout the computational graph). Take $\sigma'(\boldsymbol{z})$ as the gradient for activation function w.r.t. $\boldsymbol{z}$.

Note: Layer 1, 2 and 3 in the network are densely connected layers.
2. If *logistic map* is a function defined by the iterative relation $L_{n+1}(x) = 4L_n(x)(1 - L_n(x))$ with $L_1(x) = x$. Write down the evaluation trace equations for $L_2(x)$, $L_3(x)$, and $L_4(x)$ and then write down expressions for the corresponding derivatives $L_0^1(x)$, $L_0^2(x)$, $L_0^3(x)$, and $L_0^4(x)$. Do not simplify the expressions but instead simply note how the complexity of the formulae for the derivatives grows much more rapidly than the expressions for the functions themselves.



Loss function:	mean square error
Activation for layers 1-3:	$\sigma(\boldsymbol{z})$
True output:	s, scalar quantity
Inputs:	u, vector of size 100×1
	t, scalar quantity

Layer	Weights	Bias	Output
Layer 1	$\mathbf{W}_1$	$\boldsymbol{b}_1$	$\boldsymbol{O}_1 = \sigma(\mathbf{W}_1^{\mathrm{T}}\boldsymbol{I}_1 + \boldsymbol{b}_1)$
Layer 2	$\mathbf{W}_2$	$\boldsymbol{b}_2$	$\boldsymbol{O}_2 = \sigma(\mathbf{W}_2^{\mathrm{T}}\boldsymbol{I}_2 + \boldsymbol{b}_2)$
Layer 3	$\mathbf{W}_3$	$\boldsymbol{b}_3$	$\boldsymbol{O}_3 = \sigma(\mathbf{W}_3^{\mathrm{T}}\boldsymbol{I}_3 + \boldsymbol{b}_3)$

* $\boldsymbol{I}_i$ here refers to input to i^{th} layer.

$$y = \boldsymbol{O}_2 . \boldsymbol{O}_3$$

FIGURE 2. Network details

Question 6. [25 marks] Coding exercise:
For the computation shown below:

$$x = \sigma_1(wu + b)$$
$$y = \sigma_2(mv + c)$$
$$z_1 = x + y^2$$
$$z_2 = x - y$$
$$z_3 = \ln(xy)$$
$$z = \left(\frac{z_1}{z_2}\right)^{z_3}$$
$$\sigma_1(\alpha) = ReLU(\alpha)$$
$$\sigma_2(\alpha) = Sigmoid(\alpha)$$

Write a code, which utilizes concepts of computational graph, backpropagation and/or chain rule of derivatives to compute the gradients $\frac{dz}{dw}, \frac{dz}{db}, \frac{dz}{dm}$ and $\frac{dz}{dc}$. The following library of derivatives may assumed to be known apriori:

$$\frac{d}{dt}t^n = n\,t^{n-1}, \qquad \frac{d}{dt}e^t = e^t,$$
$$\frac{d}{dt}t = 1, \qquad \frac{d}{dt}c = 0, \text{c} = \text{constant},$$
$$\frac{d}{dt}\ln t = \frac{1}{t}, \qquad \frac{d}{dt}c^x = c^x \ln c, \text{c} = \text{constant},$$
$$\frac{d}{dt}ct = c, \text{c} = \text{constant}$$

Sum, product and quotient rule may also assumed to be known. Backpropagation functions of existing libraries are NOT to be used directly. The same may ONLY be used for comparison, which is to be included in the final results. For generating data, assume u is sampled from uniform distribution $\mathcal{U}(5, 10)$ and v is sampled from a normal distribution $\mathcal{N}(-5, 2)$.