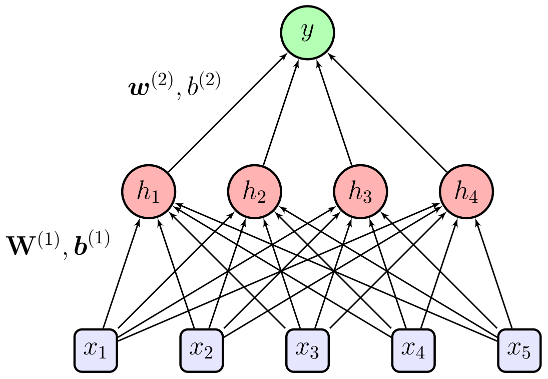
APL 745: Deep learning for mechanics
Homework – 1

Instructions

- **File name:** Submit your solutions and codes in a zip file titled `APL745_HW1_<Insert Roll Number>`. Written solutions can be produced using LATEX, Word or scan. Handwritten scans that are not legible will not be graded.
- **Submission:** : Only homeworks uploaded to MS Teams will be graded. Make sure to show all the steps of your derivations to receive full credit.
- **Integrity and Collaboration:** You are expected to work on the homeworks by yourself. You are not permitted to discuss them with any fellow students. The homework that you hand in should be entirely your own work. You may be asked to demonstrate how you got any results that you report.
- **Clarifications:** If you have any question, please look at MSTeams channel for homeworks first. Other students may have encountered the same problem and have got an answer. If not, post your question there. We will respond as soon as possible.
- **DEADLINE:** Feb 9th 2022 by 11:59pm. If you exceed the submission deadline, you will receive a penalty of 20%, and the penalty will increase by 5% with each passing day.

Problem 1 [20 pts]: In this problem you will find a set of weights and biases for a multilayer perceptron which determine if a list of four numbers in the 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 one output layer with one node.



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

The activation function of the hidden units and the output unit can be assumed to be a hard threshold function.

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

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

Problem 2: [20 pts] Consider a linear combination of M input variables $x_1, ..., x_D$ in the form

$$z_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}$$

where $j = 1, \ldots, M$, resulting in the pre-activation z_j and the superscript (1) indicates that the corresponding parameters are in the first 'layer' of the network. Also consider

$$h_j = g(z_j)$$

where $g(\cdot)$ represents some differentiable, nonlinear activation function. The outputs of the activation functions are linearly combined to obtain the output pre-activation

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} g_j + w_{k0}^{(2)}$$

where $k = 1, \ldots, K$, and k is the total number of outputs. This transformation corresponds to the second layer of the network. Finally, the output unit activations are transformed using an appropriate activation function to give a set of network outputs y_k. The choice of activation function is determined by the nature of the data and the assumed distribution of target variables. Thus, for standard regression problems, the activation function is the identity so that $y_k = a_k$. Similarly, for multiple binary classification problems, each output unit activation is transformed using a logistic sigmoid function so that

$$y_k = \sigma(a_k)$$

where,

$$\sigma(a) = \frac{1}{1 + e^{-a}}$$

Finally, for multiclass problems, a SoftMax activation function is used. We can combine these various stages to give the overall network function that, for sigmoidal (logistic) output unit activation functions, takes the form

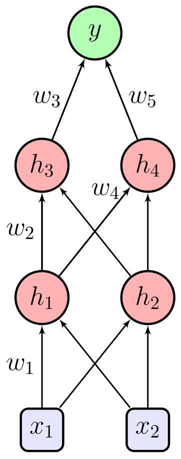
$$y_k(x; w) = \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)}\ g\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right)$$

where the set of all weight and bias parameters have been grouped together into a vector $\boldsymbol{w}$. Thus, the neural network model is simply a nonlinear function from a set of input variables x_i to a set of output variables y_k controlled by a vector $\boldsymbol{w}$ of adjustable parameters. Considering a two-layer neural network as shown above with logistic sigmoid activation function, show that there exists an equivalent network, which computes the same function, but with hidden unit activation functions given by $\texttt{tanh}(a)$ where the $\texttt{tanh}(a)$ is defined as follows:

$$\texttt{tanh}(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}$$

Note only the hidden unit activation needs to be replaced with `tanh`, the output activation will remain as sigmoid

Problem 3: [10 pts] One of the interesting features of the ReLU activation function is that it sparsifies the activations and the derivatives, i.e. sets a large fraction of the values to zero for any given input vector. Consider the following network:



Note that each w_i refers to the weight on a single connection, not the whole layer. Suppose we are trying to minimize a loss function $\mathcal{L}$ which depends only on the activation of the output unit y (For instance, $\mathcal{L}$ could be the squared error loss $\frac{1}{2}(y-t)^2$). Suppose the unit h_1 receives an input of -1 on a particular training case, so the ReLU evaluates to 0. Based only on this information, which of the weight derivatives

$$\frac{\partial \mathcal{L}}{\partial w_1}, \frac{\partial \mathcal{L}}{\partial w_2}, \frac{\partial \mathcal{L}}{\partial w_3}$$

are guaranteed to be 0 for this training case? Write YES or NO for each. Justify your answers.

Problem 4: [20 pts] Consider a MLP consisting of N input units, K hidden units and N output units. The activations are computed as follows:

$$\begin{aligned} \boldsymbol{z} &= \mathbf{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)} \\ \boldsymbol{h} &= \sigma(\boldsymbol{z}) \\ \boldsymbol{y} &= \boldsymbol{x} + \mathbf{W}^{(2)}\boldsymbol{h} + \boldsymbol{b}^{(2)} \end{aligned}$$

where $\sigma(\cdot)$ is the sigmoid function, applied element wise. The loss $\mathcal{L}$ will involve both $\boldsymbol{h}$ and $\boldsymbol{y}$:

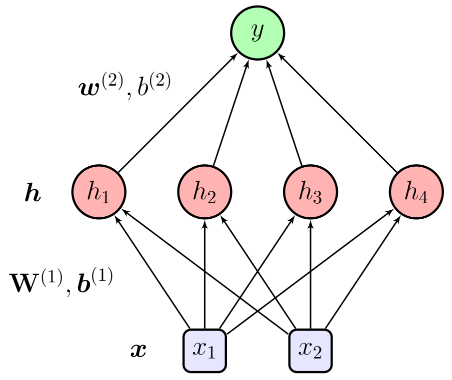
$$\begin{aligned} \mathcal{L} &= \mathcal{S} + \mathcal{R} \\ \mathcal{S} &= \frac{1}{2}\left\|\boldsymbol{y} - \boldsymbol{s}\right\|_2^2 \\ \mathcal{R} &= \boldsymbol{r}^T\boldsymbol{h} \end{aligned}$$

where $\boldsymbol{r}$ and $\boldsymbol{s}$ are given vectors.

1. Draw the computation graph relating $\boldsymbol{x}$, $\boldsymbol{z}$, $\boldsymbol{h}$, $\boldsymbol{y}$, $\mathcal{R}$, $\mathcal{S}$ and $\mathcal{L}$.

2. Derive the backpropagation equations for computing $\bar{\boldsymbol{x}} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{x}}$. You may use σ' to denote the derivation of the sigmoid function, so you do not have to write it out explicitly.

Problem 5: [30 pts] Coding Assignment Create a neural network that can learn the XOR operator. The network you implement here should have two input units (the two Boolean values that are being processed by the XOR operator), 1 hidden layer with four neurons, and one neuron in the output layer. Use a sigmoid function as the non-linear activation function in the hidden units. There need not be an


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

output activation, so your output prediction can be a linear function of the outputs from hidden units. The output of your code should be the answer for XOR for any *binary* input. You are given 15 training data points (inputs x_1 and x_2 and their corresponding output labels) for the XOR operator:

Data	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
x_1	0	0	1	0	1	1	1	1	0	0	0	1	0	0	1	1	1	1	0	0
x_2	0	1	0	1	0	1	0	1	1	0	0	1	1	1	1	1	0	1	0	1
t	0	1	1	1	1	0	1	0	1	0	0	0	1	1	0	0	1	0	0	1

For this activity, you are encouraged to use a Google colab notebook in Python. Please put down comments in text blocks of the notebook explaining important parts of the code. Consider the following:

- Set the last five examples of the 20-points dataset for testing and use the first 15 points for training. Use a mean squared loss function. Manually derive the gradient for the individual loss with respect to the parameters (weights and biases) of the single-hidden-layer neural net.

- Implement a manually coded SGD using the derived gradients for updating the parameters. Choose a learning rate that works best for you. Train your network for *atleast* 100 epochs. Your training code should incrementally print the loss value as the training proceeds. Also print the predictions of your learned model on the test input dataset and show the comparison in a tabular format.

Note you are **not allowed** to use PyTorch (or Tensorflow) to build your model, to compute gradients or to use built-in optimizers for this problem