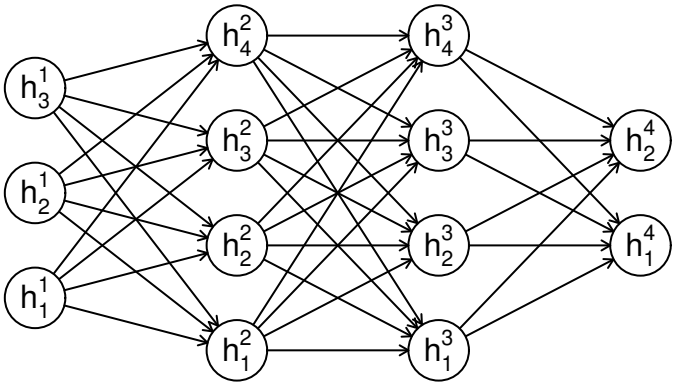
Practical 4 (group-work): Stochastic gradient descent

Programming Task: In your work group of 3, to write functions to set up a simple neural network for classification, and to train it using stochastic gradient descent. Only functions in base R and its packages (including `recommended` packages) can be used and your code must not install any packages.



The figure is a schematic diagram of a simple fully connected 4 layer neural net with layer dimensions $(3, 4, 4, 2)$. Note that superscripts denote layer index, not powers. The nodes of the network each contain values (h_j^l). The values for the first layer nodes (h_j^1) are set to the values of input data. The network then combines and transforms the values in each layer to produce the values at the next layer, until we reach the output layer (h_j^4 in the picture). The output layer nodes are used to predict output data (aka response data) associated with the input data. The network has parameters controlling the combinations and transformations linking each layer to the next. These parameters are adjusted in order to make the input data best predict the output data according to some loss function. This is known as training the network.

Mathematically each node value is determined by linearly combining the values from the nodes in the previous layer and non-linearly transforming the result. The simplest non-linear transformation is the ReLU transform, $\phi(z) = \max(0, z)$. Using this we have

$$h_j^{l+1} = \max(0, \mathbf{W}_j^l \mathbf{h}^l + \mathbf{b}^l) \tag{1}$$

where $\mathbf{W}_j^l$ is the jth row of the weight parameter matrix $\mathbf{W}^l$ linking layer l to layer $l + 1$, $\mathbf{h}^l$ is the vector of node values for layer l and $\mathbf{b}^l$ is a vector of offset parameters linking layer l to layer $l + 1$.

For a classification task in which numeric variables are used to predict what class an observation belongs to, the input layer would have a node for each numeric variable, and the output layer (layer, L, say) a node for each possible class. Then we define

$$p_k = \exp(h_k^L) / \sum_j \exp(h_j^L)$$

to be the probability that the output variable is in class k. We can then use the negative log-likelihood of a multinomial distribution as our loss function. That is, if we have n training data pairs with input, $\mathbf{x}_i$, and output class, k_i, then the loss is

$$\mathcal{L} = -\sum_{i=1}^{n} \log(p_{k_i})/n$$

The idea is to adjust the parameters of the network to minimize $\mathcal{L}$. The parameters are all the elements of all the $\mathbf{W}^l$ and $\mathbf{b}^l$. A key feature of 'deep learning' is that we may have more parameters than data, and therefore no unique set of parameters minimizing the loss, but that this will not matter provided that we can find some set of parameters giving an acceptably low loss, and that we train the network in a way that ensures it has good generalizing power, without getting too fixated on fitting any particular set of data. Stochastic Gradient Descent is an optimization approach that helps to achieve both these aims.

The idea is to repeatedly find the gradient of the loss function w.r.t. the parameters *for small randomly chosen subsets of the training data*, and to adjust the parameters by taking a step in the direction of the negative gradient.

Sometimes the small subset is just one randomly chosen $\mathbf{x}_i, k_i$ pair. How do we find the gradient? First we note that the gradient of $\mathcal{L}$ is just the average of the gradients of $\mathcal{L}_i = -\log p_{k_i}$ for each element of the training data. So we can compute the gradient one data point at a time. Here's how (for datum i):

1. Set $\mathbf{h}^1 = \mathbf{x}_i$ and compute the remaining node values h_j^l corresponding to this using (1).

2. Compute the derivative of the loss for k_i w.r.t. h_j^L:

$$\frac{\partial \mathcal{L}_i}{\partial h_j^L} = \begin{cases} \exp(h_j^L)/\sum_q \exp(h_q^L) & j \neq k_i \\ \exp(h_j^L)/\sum_q \exp(h_q^L) - 1 & j = k_i \end{cases}$$

3. Compute the derivatives of $\mathcal{L}_i$ w.r.t. all the other h_j^l by working backwards through the layers applying the chain rule (*back-propagation*):

$$\frac{\partial \mathcal{L}_i}{\partial \mathbf{h}^l} = \mathbf{W}^{l\mathsf{T}}\mathbf{d}^{l+1} \text{ where } d_j^{l+1} = \begin{cases} \partial \mathcal{L}_i/\partial h_j^{l+1} & h_j^{l+1} > 0 \\ 0 & h_j^{l+1} \leq 0 \end{cases}$$

It then follows that

$$\frac{\partial \mathcal{L}_i}{\partial \mathbf{b}^l} = \mathbf{d}^{l+1} \text{ and } \frac{\partial \mathcal{L}_i}{\partial \mathbf{W}^l} = \mathbf{d}^{l+1}\mathbf{h}^{l\mathsf{T}}.$$

If we want the derivative of the loss based on some randomly chosen set of i values, we just average the derivatives for each i in the set.

The parameter updates are then of the form

$$\mathbf{W}^l \leftarrow \mathbf{W}^l - \eta\frac{\partial \mathcal{L}_i}{\partial \mathbf{W}^l} \text{ and } \mathbf{b}^l \leftarrow \mathbf{b}^l - \eta\frac{\partial \mathcal{L}_i}{\partial \mathbf{b}^l}$$

(or equivalent when we have chosen a set of i values). η is a step length.

1. Write a function `netup` with single argument `d`, a vector giving the number of nodes in each layer of a network. `netup` should return a list representing the network. This should contain at least the following elements:

   - `h` a list of nodes for each layer. `h[[l]]` should be a vector of length `d[l]` which will contain the node values for layer `l`.
   - `W` a list of weight matrices. `W[[l]]` is the weight matrix linking layer `l` to layer `l+1`. Initialize the elements with $U(0, 0.2)$ random deviates.
   - `b` a list of offset vectors. `b[[l]]` is the offset vector linking layer `l` to layer `l+1`. Initialize the elements with $U(0, 0.2)$ random deviates.

2. Write a function `forward(nn,inp)` where `nn` is a network list as returned by `netup` and `inp` a vector of input values for the first layer. `forward` should compute the remaining node values implied by `inp`, and return the updated network list (as the only return object).

3. Write a function `backward(nn,k)` for computing the derivatives of the loss corresponding to output class `k` for network `nn` (returned from `forward`). Derivatives w.r.t. the nodes, weights and offsets should be computed and added to the network list as lists `dh`, `dW` and `db`. The updated list should be the return object.

4. Write a function `train(nn,inp,k,eta=.01,mb=10,nstep=10000)` to train the network, `nn`, given input data in the rows of matrix `inp` and corresponding labels (1, 2, 3 ...) in vector `k`. `eta` is the step size η and `mb` the number of data to randomly sample to compute the gradient. `nstep` is the number of optimization steps to take.

5. Train a 4-8-7-3 network to classify irises to species based on the 4 characteristics given in the `iris` dataset in R. Divide the `iris` data into training data and test data, where the test data consists of every 5th row of the `iris` dataset, starting from row 5. Occasionally the network will fail to train well, so in your submitted work set the seed to provide an example in which training has worked and the loss has been substantially reduced from pre- to post-training.

6. After training write code to classify the test data to species according to the class predicted as most probable for each iris in the test set, and compute the misclassification rate (i.e. the proportion misclassified) for the test set.

As a group of 3 you should aim to produce well commented[1], clear and efficient code for the task. The code should be written in a plain text file (no special characters) called `Gxx.r` where `xx` is your group number. This file is what you submit. Your solutions should use only the functions available in base R (or packages supplied with base R - no other packages). The work must be completed in your work group of 3, which you must have arranged and registered on Learn, and collaborative code development must be done using a githup repo set up for this purpose.

The first comment in your code should list the names and university user names of each member of the group. The second comment should give the address of your github repo, which should be made public *after* the hand in deadline.The third comment **must** give a brief description of what each team member contributed to the project, and roughly what proportion of the work was undertaken by each team member. Contributions never end up completely equal, but you should aim for rough equality, with team members each making sure to 'pull their weight', as well as not unfairly dominating[2]. Any team member who did not take an active part in actually writing code for earlier group projects, should ensure they do so this time.

One piece of work - the text file (no tab characters) containing your commented R code - is to be submitted for each group of 3 on Learn by 12:00 Friday 17th November 2023. Format the code and comments tidily, so that they display nicely on an 80 character width display, without line wraps (or only occasional ones). No extensions are available on this course, because of the frequency with which work has to be submitted. So late work will automatically attract a hefty penalty (of 100% after work has been marked and returned). Technology failures will not be accepted as a reason for lateness (unless it is provably the case that Learn was unavailable for an extended period), so aim to submit ahead of time.

Marking Scheme: Full marks will be obtained for code that:

1. does what it is supposed to do, in a reasonably efficient manner (e.g. avoids loops for things easily done with matrix operations), so that it operates correctly on both the iris data, and other examples.
2. exactly follows the specification with regard to inputs and outputs of functions.
3. is carefully commented, so that someone reviewing the code can easily tell exactly what it is for, what it is doing and how it is doing it without having read this sheet, or knowing anything else about the code. Note that *easily tell* implies that the comments must also be as clear and *concise* as possible. You should assume that the reader knows basic R, but not that they know exactly what every function in R does.
4. is well structured and laid out, so that the code itself, and its underlying logic, are easy to follow.
5. was prepared collaboratively using git and github in a group of 3.
6. contains no evidence of having been copied, in whole or in part, from other students on this course, students at other universities (there are now tools to help detect this, including internationally), previous students, online sources etc.
7. includes the comment stating team member contributions.

Highest marks will be awarded for concise, efficient, well structured, well commented code behaving according to the specification. Individual marks may be adjusted within groups if contributions are widely different.

[1] Good comments give an overview of what the code does, as well as line-by-line information to help the reader understand the code. Generally the code file should start with an overview of what the code in that file is about, and a high level outline of what it is doing. Similarly each function should start with a description of its inputs outputs and purpose plus a brief outline of how it works. Line-by-line comments aim to make the code easier to understand in detail.

[2] all team members must have git installed and use it