

STRASBOURG UNIVERSITY / FRENCH-AZERBAIJANI UNIVERSITY
ARTIFICIAL INTELLIGENCE
Computer Science track - Year 3
Lab : Artificial Neural Network training, from scratch

Your objective during this lab is to build an artificial neural network from scratch and train it on the Iris dataset. A Java template is available to you on Moodle.

Specific objectives :

- Observe the data you want to learn from, understand their nature and how to adapt them (if needed) to use them with a neural network model.
- Understand how a neural network model works so as to fully implement one in a program.

1 Network architecture and data

As discussed during the lectures, a multi-layer perceptron is nothing more than a set of matrices holding the parameters of the network for each layer. As such, we will only work on matrices that hold 1) the parameters of the model and 2) the data we are working on.

Loading the data into a matrix : `DataLib.java` The data file is read using a method in the class `DataLib` and used in the `main`. The data matrix has type float, 2 dimensions and it is used to instantiate the neural network. You may also have noticed that **the data are shuffled** before they are returned as a 2D array.

1.1 Dealing with matrices : `NNLib.java`

This class contains two sets of methods. On the one hand, methods are dedicated to setting up a neural network (weights initialization, activation functions and their derivative and a cost function). On the other hand, you will find methods to work with matrices (multiplication, Hadamard product, sum, etc.).

Questions

1. How are the weights initialized?
2. Which activation functions are available to you in `NNLib.java`?
3. Since you are dealing with a classification problem, what activation function will you use for your output layer?
4. What cost function is available to you?

1.2 Neural Network architecture : `NeuralNet.java`

Questions

1. What proportion of the data are held for training? for testing?
2. What is contained in `X_train` (resp. `X_test`) and `Y_train` (resp. `Y_test`)?
3. How many hidden layers are used in this network? How many units are there per layer?

2 Preparing the data

$$\text{Data}_{\text{raw}} = \begin{pmatrix} x_1 & \dots & x_n & y \\ x_1 & \dots & x_n & y \\ x_1 & \dots & x_n & y \end{pmatrix}$$

FIGURE 1 – Raw data, in matrix form, as imported from `DataLib.java`. Each line represents an instance of the dataset.

$$\mathbf{X}_{\text{train}} = \begin{pmatrix} x_1 & x_1 & x_1 \\ x_2 & x_2 & x_2 \\ x_3 & x_3 & x_3 \\ x_4 & x_4 & x_4 \end{pmatrix}$$

FIGURE 2 – Input data in matrix form. Each column represent an instance of the dataset. For a given training pass, several instances can be loaded (i.e. a batch). In the present case, the batch contains 3 instances.

$$\mathbf{Y}_{\text{train}} = \begin{pmatrix} y_1 & y_1 & y_1 \\ y_2 & y_2 & y_2 \\ y_3 & y_3 & y_3 \end{pmatrix}$$

FIGURE 3 – Labels of the data in matrix form, using one-hot encoding. Each line is a possible class (here, we know that $K = 3$). Each column represent an instance of the dataset. For a given training pass, several instances can be loaded (i.e. a batch). In the present case, the batch contains 3 instances.

All the attributes are length (in cm) and their scale is similar : we won't have to normalize these attribute.

We must adapt the representation of the labels ($\mathbf{Y}_{\text{train}}$ et \mathbf{Y}_{test}) : using a softmax on the output layer, we expect probabilities. For now, labels are encoded using integers (representing the 3 possible categories) : $0 \rightarrow \text{Iris-setosa}$, $1 \rightarrow \text{Iris-versicolor}$ and $2 \rightarrow \text{Iris-virginica}$.

We will transform those labels into one-hots. The actual class will be encoded by 1 while all the other will be encoded by 0 :

$$\mathbf{Y}_{\text{Iris-setosa}} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \quad \mathbf{Y}_{\text{Iris-versicolor}} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \quad \mathbf{Y}_{\text{Iris-virginica}} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$


Implementation

1. Write the method `createOneHot()` : for a given instance `indexInBatch` (a column index), set line k of the vector to 1 and all other lines to 0.
2. Write the method `loadAttributesAndLabels()`. For each instance, you have to report each attribute values (except for the label) in the instance corresponding line of the matrix.

3 Training and testing the model

Questions

1. What is an epoch of training?
2. Say we want to train on batches of size n . How will you implement an epoch of training?
3. We know how to measure the error between our predictions and the expected labels on the output label. Do we have target values for the hidden units? How can we update the parameters on those hidden units?

 **Implementation** The method `train()` is used to train the model (the number of epochs is set as a parameter). Before we can use it, we must first implement the procedure for an epoch of training.

1. Write the method `shuffleTrainingData()`. You can use the Fisher-Yates¹ algorithm for this.
2. With the help of the algorithm 1 page 4, write the method `trainingEpoch()`.
3. Write the method `testPrediction()` (it is used by `trainingEpoch()`) : you must first load the data in `X_test` and `Y_test` and perform a forward pass. Then, measure the error and the model precision using `NNLib.checkAccuracy()`.

4 So what ?

The method `train()` in `NeuralNet.java` exports the cost (or the precision, given what you return in method `testPrediction()`). Open this file (using a spreadsheet, or Gnuplot or whatever) and display the error (or the cost) as a function of the training epoch.

The following figure 4 is an example of such plot using my version of the program. We can observe saddle-points, that may denote reaching local minimum in the error function.

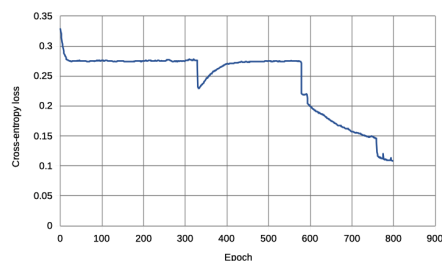


FIGURE 4 – Model error as a function of training epoch

1. https://en.wikipedia.org/wiki/Fisher-Yates_shuffle#The_modern_algorithm

Algorithm 1: Performing an epoch of training

```
shuffleTrainingData()
while epoch not finished do
  foreach instance in current batch do
    | loadAttributesAndLabels(trainingData, X_train, Y_train, currentDataIndex, batchsize)
  end

  /* Forward pass */
   $Z^{[1]} \leftarrow W^{[1]} X_{\text{train}} + b^{[1]}$ 
   $A^{[1]} \leftarrow g_{\text{hidden}}(Z^{[1]})$ 
   $Z^{[2]} \leftarrow W^{[2]} A^{[1]} + b^{[2]}$ 
   $A^{[2]} \leftarrow g_{\text{out}}(Z^{[2]})$ 

  /* Measuring the error */
  training_error  $\leftarrow \text{error}(A^{[2]}, Y_{\text{train}})$ 

  /* Error backpropagation */
   $\text{delta}^{[2]} \leftarrow \dots$  /* use the adequate expression here */
   $dW^{[2]} \leftarrow \text{delta}^{[2]} A^{[1]T}$ 
   $db^{[2]} \leftarrow \text{delta}^{[2]}$ 

   $\text{delta}^{[1]} \leftarrow W^{[2]} \text{delta}^{[2]} \circ g'_{\text{hidden}}(Z^{[1]})$ 
   $dW^{[1]} \leftarrow \text{delta}^{[1]} X_{\text{train}}^T$ 
   $db^{[1]} \leftarrow \text{delta}^{[1]}$ 

  /* Parameters update */
   $W^{[2]} \leftarrow W^{[2]} - \eta dW^{[2]}$ 
   $b^{[2]} \leftarrow b^{[2]} - \eta db^{[2]}$ 

   $W^{[1]} \leftarrow W^{[1]} - \eta dW^{[1]}$ 
   $b^{[1]} \leftarrow b^{[1]} - \eta db^{[1]}$ 
end
return testPrediction()
```
