Intro to A.I. Assignment

Multilayered Perceptrons (MLP) or Artificial Neural Networks for Classification

You are required to modify the code provided which implements a MLP for XOR. As it is, there are 2 input neurons, 2 hidden neurons and 1 output neuron. This has to be generalised to m input, n hidden and o output neurons respectively. These 3 integers can be supplied to the MLP constructor. 1 hidden layer is probably sufficient.

You are then to parameterise a MLP to solve each of the 4 problems described in the problem sheet. Code and output including Cost or Loss plots should be included in a Jupyter Notebook which you will submit online.

For the first 3 problems, the training data can be manually declared in the notebook like it is for XOR. You will need to use Pandas library to import the Iris dataset or a larger one from a CSV file and copy it to a NumPy array. Requires only a few of lines of code.

For the transport and Iris problems, once the code works and network is trained, you ought to add an operation **predict(x)** which is like feedforward() except that is uses the output of feedforward(x) to for example suggest train or car in the transport problem or the name of the Iris.

If you can, find another online data set besides the Iris flowers one and train your MLP on this too. Extra marks will be awarded for initiative.

Important in some of the problems, maybe two of the larger ones, to vary the training rate, the number of hidden neurons or the number of hidden layers and to see the effect on training. Also experiment with the cross-entropy cost function.

Submission to Brightspace

- A single Jupyter Notebook with all you code and outputs for all the problems.
- Any .csv data sets that you used aside from the Iris dataset and ones provided.
- A PDF with explanations, discussion & summary of what's going on with relevant charts and code fragments from your notebook. No need to copy all the notebook stuff, just enough to illustrate your discussion/analysis.

Technical Issues

The code provided uses the mean-squared-error cost function. Once you get the code to work, use the cross-entropy cost function also and see if it speeds up the learning. See chapter 3 of Nielsen for more detailed information. Using cross-entropy only affects the **delta3** or δ^3 vector calculation for the output layer (layer 3). It makes δ^3 more responsive to output error. Back propagation code is still the same. See formulas below.

Mean Squared Error Cost Function

$$C_x = \frac{1}{2}\sum_{i=1}^{n}(y_i - a_i^3)^2$$

cost = (y – a3)**2.sum()

$\delta^3 = \frac{\partial C}{\partial a^3} * \sigma'(z^3)$ works out as

$\delta^3 = (a^3 - y) * \sigma'(z^3) = (a^3 - y) * a^3 * (1 - a^3)$

delta3 = (a3 – y) * a3 * (1 – a3)

Cross-entropy Cost Function

$$C_x = \sum_{i=1}^{n} -y_i ln(a_i^3) - (1 - y_i)\, ln(1 - a_i^3)$$

cost = ??

But when you calculate $\delta^3 = \frac{\partial C}{\partial a^3} * \sigma'(z^3)$, it simplifies to:

$\delta^3 = (a^3 - y)$

delta3 = (a3 – y)

So it does not depend on the σ'(z) but only on the error (a3 – y). More responsive than delta3 based on mean-squared-error.