Assignments

Assignment 3 - Multi-module gradient-based learning

Written by Durk Kingma (dpkingma [at] gmail.com) and Xiang Zhang (xiang.zhang [at] nyu.edu)

In this assignment you'll (1) create a Torch-compatible dataset, (2) implement whitening transform, (3) create your own custom neural networks and (4) create modules for a RBF network.

Practicalities

- The deadline is: November 26, before class
- Deliver your solutions (including derivations and graphs) in one non-large PDF file. You may create your solutions in LaTeX or a word processor with proper math functionality, as long as you convert it to small PDF. If your handwriting is readable, you may also submit scanned hand-written solutions.
- Your code and solutions should be send to dpk2 [at] nyu.edu or xiang.zhang [at] nyu.edu. As usual, do not include the data files.
- Put everything in one zip or tar.gz. Please EXCLUDE datasets (.t7, data), or your email may not be received.
- You may discuss your results with other students, but you must submit your own implementations.

Questions

Part 1 - Isolet Dataset and Pre-processing

We're going to experiment with Isolet, a spoken letter recognition dataset. This is a relatively large set with over 600 input features per sample, 26 categories, and over 6000 training samples. The dataset contains speech data from 150 subjects who spoke the name of each letter of the alphabet twice. Hence, there are 52 training examples from each speaker. The features include spectral coefficients (resulting from Fourier transform) as well as other features (contour features, sonorant features, pre-sonorant features, and post-sonorant features). There are 5 parts: 1 to 4 with training data, part 5 with testing data. Information and dataset can be found here: http://archive.ics.uci.edu/ml/datasets/ISOLET [http://archive.ics.uci.edu/ml/datasets/ISOLET]

You're going to write a script 'isolet.lua' containing 'getIsoletDatasets()' that returns the 'train' and 'test' datasets.

The train and test datasets should follow the Torch dataset convention: “A dataset is an object which implements the operator dataset[index] and implements the method dataset:size(). The size() methods returns the number of examples and dataset[i] has to return the i-th example. An example has to be an object which implements the operator example[field], where field often takes the value 1 (for input features) or 2 (for corresponding labels), i.e an example is a pair of input and output objects." Note that spambase.lua of the previous assignment also followed this convention.

Like in 'spambase.lua', the features should be invididually normalized (zero mean, unit variance) and the training set should be randomly shuffled.

Q1.1: Download dataset and implement 'isolet.lua' as described above.

Q1.2: Write a script 'whitening.lua' containing 'whitenDatasets(train, test, k)' that performs a whitening transform on your dataset. The function should transforms the 'N' original (normalized) features to the top 'k' principal components. Tip: use the 'torch.svd(…)' function. The resulting features should be uncorrelated (in the trainingset) and have zero mean and unit variance. The function 'whitenDatasets(…)' should return the whitened train- and testsets.

Part 2 - Torch neural-network package

Take a look at the powerful Torch7 neural-network package as described in the manual: http://www.torch.ch/manual/nn/index [http://www.torch.ch/manual/nn/index]

Q2.1: What basic algorithm is used to differentiate a neural network (consisting of a sequence of 'modules') w.r.t. its parameters? Explain concisely, in max 10 sentences, the algorithm.

Q2.2: In 'models.lua', write functions that return the following Torch models. The last module of the model should be LogSoftMax, such that the models outputs log-probabilities of the classes.

- Logistic regression
- A two-layer neural network (linear→tanh→linear→logsoftmax), where the number of hidden units is a construction parameter. Note that the Linear module contains a 'bias' term.

Part 3 - Radial Basis Function Network

You're now going to implement some new modules to create a RBF network. Make sure your modules are compatible with the 'nn' package. Put every module in their own file.

Q3.1: Implement a 'RBF' (Radial Basis Function) module:

- The function is: z_i = sum_j((x_j - W_ij)^2) where 'x_j' is j-th element of the input vector, 'W_ij' is the element of the i-th row and j-th column of a matrix of prototypes W and 'z_i' is the i-th element of the output vector 'z'.
- Justify your differentiation function ('updateGradInput(…)') with a full derivation.

Q3.2: Implement a 'MulPos' (multiplication with positive number) module:

- The module is similar to the existing 'Mul' module, but with strictly positive multiplicative factor. How could you enforce such constraint? Hint: you can let the module compute z_i = e^w * x_i, where 'w' is your weight. The weight can be any real number but e^w will always be strictly positive.
- Justify the differentiation function with a derivation

Q3.3: Implement a 'NegExp' (negative exponent) module:

- The function is: z_i = e^(-x_i)
- Justify the differentiation function with a derivation

You have now written all necessary modules to create a standard RBF network.

Q3.4: Write a RBF model constuctor function: (rbf → mulpos → negexp → linear → logsoftmax).

Q3.5: Bonus points: Write an advanced module of your choice, for instance:

- An extended 'RBF' module, with the following function: z_i = sum_j(A_ij * (x_j - W_ij)^2), where 'A' is some learnable scaling matrix, where each row is normalized to unit L2 norm
- A multi-layer (hierarchical) RBF network

Q3.6: Bonus points: Module differentiation testing:

- Devise a method that can test correctness of 'updateGradInput(…)' of any (differentiable) module.
- Implement the method

Part 4 - Experiments

You're going to train some models on the Isolet dataset.

Q4.1: Plot a graph of “k” vs ”% error on the test set” where 'k' is the number of component we keep in the whitening step, with 'k' at least in [1,2,4,8,16,32,64,128,256,512,617]. Include performance of all models from Q2.2, and a RBF network. For the 2-layer neural net and RBF, include networks containing 10, 20, 40 and 80 hidden units.

Q4.2: Create a model that beats all models from (Q4.1) in terms of accuracy on the test-set. Any model is allowed (e.g. regularization, other modules, etc.). Explain what you did, and give a function that exaclty reproduces your results.

Q4.3: Bonus points if:

- your model has lower than 3.27% error on the entire test set (which was the state of the art on September 12, 1994, when the dataset was released).
- performance is in the top 3 of submissions.

Additional tips

Here's some additional tips.

- Some people asked if it's necessary to apply cross-validation. No, if you read the assignment, you see that we don't ask for crossvalidation this time, just for the performance on the test set.
- For part 3, you have to write some modules. It's helpful to look at examples of existing modules, you can take a look at the sourcecode of existing Torch modules:

https://github.com/andresy/torch/tree/master/extra/nn
[https://github.com/andresy/torch/tree/master/extra/nn]

- Some people asked if the “model” class should be like the previous assignment. No. We ask for “torch” models. In other words, you can just take nn.Sequential(), add the correct modules and criterion, and the model is fully functional.

You can train the models using any method you like, e.g. you could use the Trainer as in the nn manual page. But, as far as I know Torch has no function like that for test performance on a test set, so you still need to write that function.

- Some students have trouble deriving the gradient functions for the “updateGradInput()” and “accGradParameters()” functions (Q3.1-Q3.3). The general idea for deriving a module's derivatives is as follows:

Write down the module's (mathematical) function in the form “f(x, w) = …” where “x” are the inputs and “w” are the weights. This is what I already described in the assignment. Derive the derivatives of the function's value (= output) with respect to its variables (= input and weights), i.e. df/dx and df/dw. If your full-blown vector calculus skills are rusty, you could start with derivation of simple partial derivatives of the individual x's and w's. Most often, this comes down to repeated application of the chain rule. (What else is backpropagation then repeated application of chain rule?) This results in expressions of the derivative of the module's output resulting in the gradients w.r.t. the inputs/weights, i.e. individual df/dw = … and df/dx = …. You could translate these functions directly into Lua in the updateGradInput() and accGradParameters methods, but that would be pretty inefficient. It's computationally much more efficient to “vectorize” these gradient computations using Tensor operations, so that gradient vectors are computated as a whole. Late submissions are accepted, but you get points deducted for the amount of days late (that's something the professor decided).

Assignment 2 - Support Vector Machines

Practicalities

Deadline: October 29th before Midnight

The datasets for this assignment are here: http://cs.nyu.edu/~xz558/ml12/assign2-dataset.tar.gz [http://cs.nyu.edu/~xz558/ml12/assign2-dataset.tar.gz]

The skeleton source code is here: http://cs.nyu.edu/~xz558/ml12/assign2-source.tar.gz [http://cs.nyu.edu/~xz558/ml12/assign2-source.tar.gz]

Note: ***Please start this assignment as early as possible***, since part 2 requires some quite long computations. We recommend to test your code first on a scaled-down problem (polynomial degree, smaller range of C and/or less cross-validation folds).

Your assignment should be sent by email to dpk2@nyu.edu OR xz558@nyu.edu

- Send all the source code files (.lua) plus a description in one zip or tar.gz. Please EXCLUDE datasets (.t7, data), or your email may not be received.
- The description can be a simple text file, with any mathematical notion (if there is any) written in LaTeX convention. Please do NOT include any msword, rtf or html files. If you do not like writing formulas in LaTeX convention, we can accept pdf if it is small. Write all your answers to the QUESTIONS section (below) in this description file.
- If you choose to write a pdf file, please embed all the plots into it. Or you can save your plot as eps, ps, or pdf files and send them altogether with your description file,