Worksheet 1

Implementation exercises

These assignments are intended to build some practical foundations. They make up a bigger part of your final grade. You are supposed to solve these problems on your own; however, active discussion of approaches, problems, obstacles, etc. on Piazza is very much encouraged.

How to hand in?

Place each assignment in a subdirectory of your repository and also include the output plots. This directory should not contain any large data files. Provide a `README.md`, which describes very briefly what's going on. Here you should also mention if we need to copy some files into the directory. If these are non-standard files, please describe what we should do to get them.

Your solution program should be runnable from the command line without requiring additional arguments or libraries. Your programs will be tested on Microsoft Windows 7 x64 using the software stack described in section 1.

Also, mention in the brief description what you did *beyond* the original assignment; all the extra work you do will be favourable with respect to your grade.

GitHub Repositories

Create a free account on `github.com`. We have private space on our account (`https://github.com/BRML`); contact us to get access to our space, we will create a project for you and need your GitHub username for that. Put all the assignment directories in this repository. Use `.gitignore` to avoid checking in large files. The repository should be private, for the time being. To hand in your solutions, simply share your repository with us. Do that before the deadline.

1 Software stack and datasets

Problem 1: Install Anaconda Python 2.7 from `http://continuum.io/downloads`.

Problem 2: (optional) We recommend the excellent JetBrains PyCharm IDE available from `https://www.jetbrains.com/pycharm/download/` for development. The Community Edition is perfectly sufficient for our needs.

Problem 3: Install bleeding-edge Theano from `http://deeplearning.net/software/theano/` with the command `python setup.py develop`. (In some cases we've encountered some issues with Theano in Windows. You may want to use the version `https://github.com/surban/theano` instead.)

Problem 4: (optional) If you have an nVidia GPU with CUDA support in your computer, you will greatly accelerate your programs by letting Theano do its computations on the GPU. See http://deeplearning.net/software/theano/tutorial/using_gpu.html and http://deeplearning.net/software/theano/install_windows.html#configure-theano-for-gpu-use and how to configure Theano accordingly.

Windows OS: To use the GPU, you will need to install the nVidia CUDA Toolkit 5.5 (https://developer.nvidia.com/cuda-toolkit-55-archive) and Microsoft Visual C++ 2008 compilers (https://www.microsoft.com/en-us/download/details.aspx?id=3138). Launch your programs from a `Microsoft Windows SDK v7.0 -> CMD Shell` to make sure that the correct compiler is used. The reason for using such an old compiler and CUDA toolkit is that the compiler version must match the version with which the Python interpreter was compiled. [Note that you can also use Visual C++ 2013 with CUDA 7.5 (CUDA does not support Visual C++ 2015 yet) internally. However this means you will need a full Visual Studio installation (downloadable for free from https://www.visualstudio.com/), as the newer SDKs do not ship with a compiler anymore.]

In file `theano/sandbox/cuda/cuda_ndarray.cuh` around line 37 you might need to comment out the inclusion of `stdint.h`, i.e. change the line `#include <stdint.h>` to `// #include <stdint.h>`, to fix a compile error. To check that the correct Python and compiler versions are used, execute the following commands and compare the output:

```
C:\>cl
Microsoft (R) C/C++ Optimizing Compiler Version 15.00.30729.01 for x64
Copyright (C) Microsoft Corporation.  All rights reserved.

usage: cl [ option... ] filename... [ /link linkoption... ]

C:\>nvcc --version
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2013 NVIDIA Corporation
Built on Wed_Jul_10_13:36:45_PDT_2013
Cuda compilation tools, release 5.5, V5.5.0

C:\>python --version
Python 2.7.10 :: Anaconda 2.4.0 (64-bit)
```

Problem 5: Read and perform the following sections from the Theano tutorial at http://deeplearning.net/software/theano/tutorial/index.html: Baby Steps – Algebra, More Examples, Graph Structures, Derivatives in Theano.

Problem 6: Download and install the Climin optimization library from http://climin.readthedocs.org/en/latest/.

Problem 7: Acquire and acquaint yourself with the following datasets that you will be using through this worksheet. To simplify things convert CIFAR-10 into grayscale images.

MNIST:	`http://deeplearning.net/data/mnist/mnist.pkl.gz`
	`http://deeplearning.net/tutorial/gettingstarted.html`
CIFAR-10:	`http://www.cs.toronto.edu/~kriz/cifar.html`

2 Multiclass logistic regression

Subfolder:	`logreg`
Theoretic prerequisites:	logistic regression, optimisation (gradient descent, conjugate gradient), receptive field
Literature:	Bishop: section 4.3.2 – 4.3.4
Tutorial:	`http://deeplearning.net/tutorial/logreg.html`

Problem 8: Implement multiclass logistic regression in Python using Theano. Use standard gradient descent with minibatches as the initial optimisation method. You may follow the tutorial or, better, write your own implementation from ground up.

Problem 9: Evaluate your implementation to classify handwritten digits from the MNIST dataset.

Problem 10: Try improving your results by using better optimisation methods. You can use readily available optimiser from the Climin library.

Problem 11: Visualize the receptive fields of each class (i.e., digit) and write them to file `repflds.png`. In logistic regression the receptive fields are the weight matrices for each class.

Problem 12: Plot the error curves (i.e., error over iteration) for the training, evaluation and test set into file `error.png`. When do you stop training?

Problem 13: Fine tune your implementation until you achieve an error rate of about 7% on the test set. One approach might be to augment the training set by including transformed (slightly rotated, elastically deformed) versions of the digits from the vanilla training set. Do not spend too much time on this problem, since we have not verified that this error rate is achievable using logistic regression.

Bonus question: Why is the last problem statement actually very bad scientific practice?

3 Two-layer neural network

Subfolder:	`nn`
Theoretic prerequisites:	artificial neural networks, backpropagation, optimization (gradient descent, rmsprop), receptive field
Literature:	Bishop: sections 5.1–5.2.1, 5.3.1 Murphy: sections 16.5, 16.5.4, 16.5.6 COURSERA Neural Networks for Machine Learning: lectures 6a, 6b, 6e
Tutorial:	`http://deeplearning.net/tutorial/mlp.html`

Problem 14: Implement a neural network with one hidden layer. We suggest that you implement it in a way that it works with different kinds of optimisation algorithms. Use stochastic gradient descent with mini-batches and rmsprop as the initial optimisation method. Implement early stopping. You may follow the tutorial or, better, write your own implementation from ground up.

Bonus question: Implement different regularisation approaches, like momentum, weight decay, ℓ_1 regularisation, dropout.

Problem 15: Evaluate your implementation on MNIST. Initially use 300 hidden units with tanh activation functions.

Problem 16: Try different nonlinear activation functions for the hidden units. Evaluate logistic sigmoid, tanh and rectified linear neurons in the hidden layer. Think about how the different activation functions look like and how they behave. Does—and if it does, how does—this influence, e.g., weight initialization or data preprocessing? Implement and test your reasoning in your code to see if the results support your conclusions.

Problem 17: Plot the error curves for the training, evaluation and test set for each of the activation functions evaluated in the previous problem into file `error.png`. That is, either provide one file with three subplots (one per activation function) and three error curves each, or provide three different files (`error_tanh.png`, `error_sigmoid.png`, and `error_relu.png`).

Problem 18: Visualize the receptive fields *of the hidden layer* and write them to file `repflds.png`. As in the previous problem, either provide one file with three subplots, or three distinct files.

Problem 19: Fine tune your implementation until you achieve an error rate of about 2%. Optionally try augmenting the training set as described in section 2. Do not spend too much time on this problem.

4 PCA and sparse autoencoder

Subfolder:	`latent`
Theoretic prerequisites:	principal component analysis (PCA), autoencoder, sparsity
Literature:	Bishop: section 12.1, Murphy: sections 28.3.2, 28.3.3, 28.4.2, 28.4.3 http://ufldl.stanford.edu/wiki/index.php/Autoencoders_and_Sparsity http://stats.stackexchange.com/questions/45643/why-l1-norm-for-sparse-models
Tutorial:	http://deeplearning.net/tutorial/dA.html#autoencoders

Problem 20: Implement PCA in Python using Theano. Write your own implementation from ground up.

Problem 21: Produce a PCA scatterplot (see http://peekaboo-vision.blogspot.de/2012/12/another-look-at-mnist.html) on MNIST, also do this on CIFAR-10. Write them to file `scatterplotMNIST.png` and `scatterplotCIFAR.png` respectively.

Problem 22: Implement an autoencoder in Python using Theano. Train the network using the squared error loss $L(\boldsymbol{x}) = ||\boldsymbol{f}(\boldsymbol{x}) - \boldsymbol{x}||_2^2$ where $\boldsymbol{x}$ is a data sample and $\boldsymbol{f}(\boldsymbol{x})$ is the output of the autoencoder. You may follow a part of the tutorial or, better, write your own implementation from ground up. If training is difficult using gradient descent, try using the RMSprop optimizer.

Problem 23: Increase the number of hidden units, but add a sparsity constraint on the hidden units. This means that the network should be encouraged to have most hidden units close to zero for a sample from the data set. This can be done by adding an L_1 penalty (see literature section for reasons behind this) to the loss function, for example $L_{\text{sparse}}(\boldsymbol{x}) = L(\boldsymbol{x}) + \lambda|\boldsymbol{h}(\boldsymbol{x})|_1$ where $\boldsymbol{h}(\boldsymbol{x})$ denotes the hidden layer values for sample $\boldsymbol{x}$ and $|\boldsymbol{z}|_1 = \sum_i |z_i|$ is the L_1-norm of $\boldsymbol{z}$. $\lambda > 0$ is a new hyperparameter that determines the trade-off between sparsity and reconstruction error.

Problem 24: Train the sparse autoencoder on MNIST. Write the reconstructions (i.e. outputs of the autoencoder) of the first 100 samples from the test set of MNIST into file `autoencoderrec.png`. Adjust λ and see how it affects the reconstructions.

Problem 25: Visualise the learnt receptive fields (weights of the first layer). Write them to file `autoencoderfilter.png`. Adjust λ and see how it affects the receptive fields.

Problem 26: Explain the meaning of a sparse encoding of MNIST.

Bonus problem: Replace the sparsity-inducing L_1 penalty by a KL-divergence penalty between the data-induced distribution of the hidden units and Bernoulli random variables with a low ($p < 0.05$) success probability. This is described in http://ufldl.stanford.edu/wiki/index.php/Autoencoders_and_Sparsity.

5 t-SNE

Subfolder: tsne
Literature: L.J.P. van der Maaten. Accelerating t-SNE using Tree-Based Algorithms.
http://lvdmaaten.github.io/publications/papers/JMLR_2014.pdf
L.J.P. van der Maaten. Barnes-Hut-SNE.
http://arxiv.org/pdf/1301.3342v2.pdf

Problem 27: Read the recommended literature to get a feeling of what t-SNE is about.

Problem 28: Download the Python implementation from http://lvdmaaten.github.io/tsne and experiment with it to produce 2D embeddings for a large number of data samples.

Problem 29: Reproduce Figure 5 of the Barnes-Hut-SNE paper listed above.

6 k-Means

Subfolder: kmeans
Literature: Bishop: section 9.1
http://ai.stanford.edu/~acoates/papers/coatesng_nntot2012.pdf

Problem 30: Implement k-Means in Python using Theano. Follow the above paper by Adam Coates and implement the steps on page 5 (skip everything from section 4 on).

Problem 31: Train your model on the CIFAR-10 dataset and visualise your receptive fields. Save them to file `repflds.png`. Make sure to rescale the images in the dataset from 32×32 to 12×12 pixels and choose no more than 500 centres.

Bonus problem: Implement minibatch k-Means.

Bonus problem: Implement the k-Means version from the paper B. Kulis, M. Jordan: Revisiting k-means: New Algorithms via Bayesian Nonparametrics (http://arxiv.org/pdf/1111.0352.pdf). Apply it on a (rather large) set of of colour images and try to determine a good colour palette.