Object recognition and computer vision 2015

Ivan Laptev, Jean Ponce, Cordelia Schmid and Josef Sivic

Assignment 3: Neural networks

(adapted from N. Le Roux and A. Vedaldi)

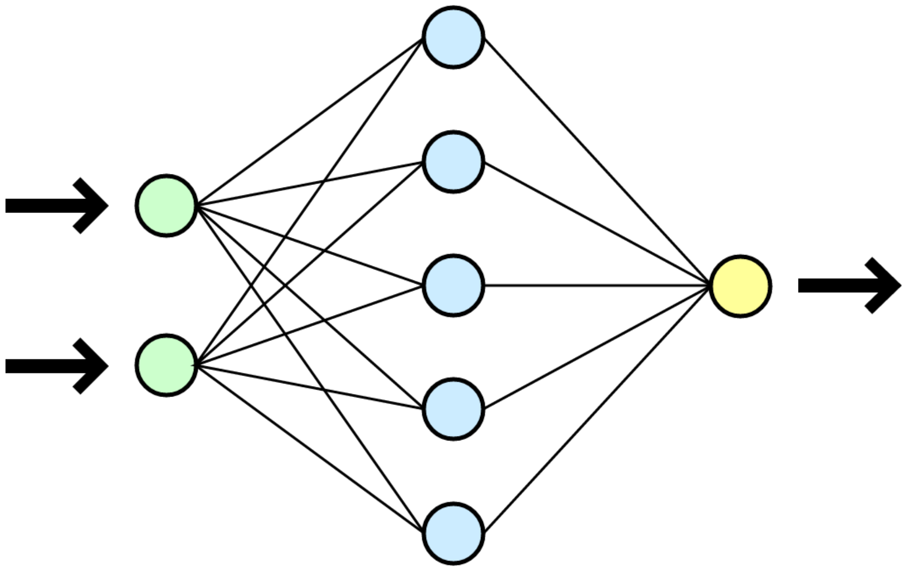


Figure 1: Two layer neural network.

Goal

The goal of this assignment is to get basic knowledge and hands-on experience with training and using neural networks. In Part 1 of the assignment you will implement and experiment with the training and testing of a simple 2 layer neural network, similar to the one depicted in Figure 1 above. In Part 2 you will use a more complex convolutional neural network with pre-trained weights for the task of learning image classification for new object classes.

Part 1: Training a neural network by back-propagation

Getting started

- Download the code and data (code and data).

- Unpack the code archive. This will create a directory called **neural_networks**.
- Start your MATLAB in the directory **neural_networks**.

As you progress in the exercises you can use MATLAB **help** command to display the help of the MATLAB functions that you need to use. For example, try typing **help nnet_forward_logloss**.

Exercise description
The image-classification directory contains Matlab script **nnet_demo.m** with a commented skeleton of the code. You will need to (1) complete this script by writing your code, and (2) produce written answers to the questions given below (**shown in red**) and include them into your report. Your answers should include results, graphs or images as specified in each question.

You will be working with a two layer neural network of the following form

$$H = ReLU(W_i X + B_i) \quad (1)$$
$$Y = W_o H + B_o \quad (2)$$

where X is the input, Y is the output, H is the hidden unit, and W_i , W_o , B_i and B_o are the network parameters that need to be trained. Here the subscripts i and o stand for the "input" and "output" layer, respectively. This network was also discussed in the class and is illustrated in the Fig.1 where the input units are shown in green, the hidden units in blue and the output in yellow. This network is implemented in the function **nnet_forward_logloss.m.** Inspect the help for that function to see the details such as the dimensions of the individual variables and parameters.

You will train the parameters of the network from labelled training data $\{X^n, Y^n\}$ where X^n are points in R^2 and $Y^n \in \{-1,1\}$ are labels for each point. You will use the stochastic gradient descent algorithm discussed in the class to minimize the loss of the network on the training data given by

$$L = \sum_n s(Y^n, \overline{Y}(X^n)) \; (3)$$

where Y^n is the target label for the n-th example and $\overline{Y}(X^n)$ is the network's output for the n-th example X^n. The skeleton of the training procedure is provided in the Matlab script **nnet_demo.m.**

We will use the logistic loss, which has the following form

$$s(Y, \overline{Y}(X)) = log(1 + exp(-Y.\overline{Y}(X)), \; (4)$$

where Y is the target label and $\overline{Y}(X)$ is the output of the network for input example X. With the logistic loss, the output of the network can be interpreted as a probability $P(class = 1|X) = \sigma(X)$, where $\sigma(X) = 1/(1 + exp(-X))$ is the sigmoid function. Note also that $P(class = -1|X) = 1 - P(class = 1|X)$.

Stage A: Computing gradients of the loss with respect to network parameters

Derive the form of the gradient of the logistic loss (4) with respect to the parameters of the network W_i , W_o , B_i and B_o. Hint: Use the chain rule as discussed in the class. First, write down the derivative of the loss (4) with respect to the output of the network $\bar{Y}(X)$. Then write down the derivatives of the output $\bar{Y}$ with respect to the parameters W_o , B_o of the output layer, and so on.

QA: In your report, derive using the chain rule the form of the gradient of the logistic loss (4) with respect to the parameters of the network W_i , W_o , B_i and B_o.

Following your derivation from QA, implement the gradient computation in the function:

```
[grad_s_Wi, grad_s_Wo, grad_s_bi, grad_s_bo] =
gradient_nn(X,Y,Wi,bi,Wo,bo)
```

See the code for the description of the required inputs / outputs of this function.

Stage B: Numerically verify the gradients

Here you will numerically verify that your analytically computed gradients in function **gradient_nn.m** are correct.

QB1: In your report, write down the general formula for numerically computing the approximate derivative of the loss $s(\Theta)$, with respect to the parameter Θ_i using finite differencing. Hint: use the first order Taylor expansion of loss $s(\Theta+\Delta\Theta)$ around point Θ.

For this part of the assignment set the number of hidden units h=3. Run the learning algorithm for few iterations. Now, compute numerically the derivative of the loss with respect to the scalar parameter B_o and store it into variable `grad_s_bo_approx`. You can use the function **nnet_forward_logloss.m** to compute the loss of the network for a particular example {X,Y} and specific values of parameters W_i , W_o , B_i and B_o. Check that the analytically computed derivative `grad_s_bo` at the same training example {X,Y} using function **gradient_nn.m** is the same (up to small errors) as your numerically computed value of the derivative `grad_s_bo_approx`.

Use this idea to implement a function that will numerically compute the derivatives of the loss function with respect to all the parameters of the network W_i , W_o , B_i , B_o:

```
[grad_s_Wi_approx, grad_s_Wo_approx, grad_s_bi_approx,
grad_s_bo_approx] = gradient_nn_approx(X,Y,Wi,bi,Wo,bo);
```

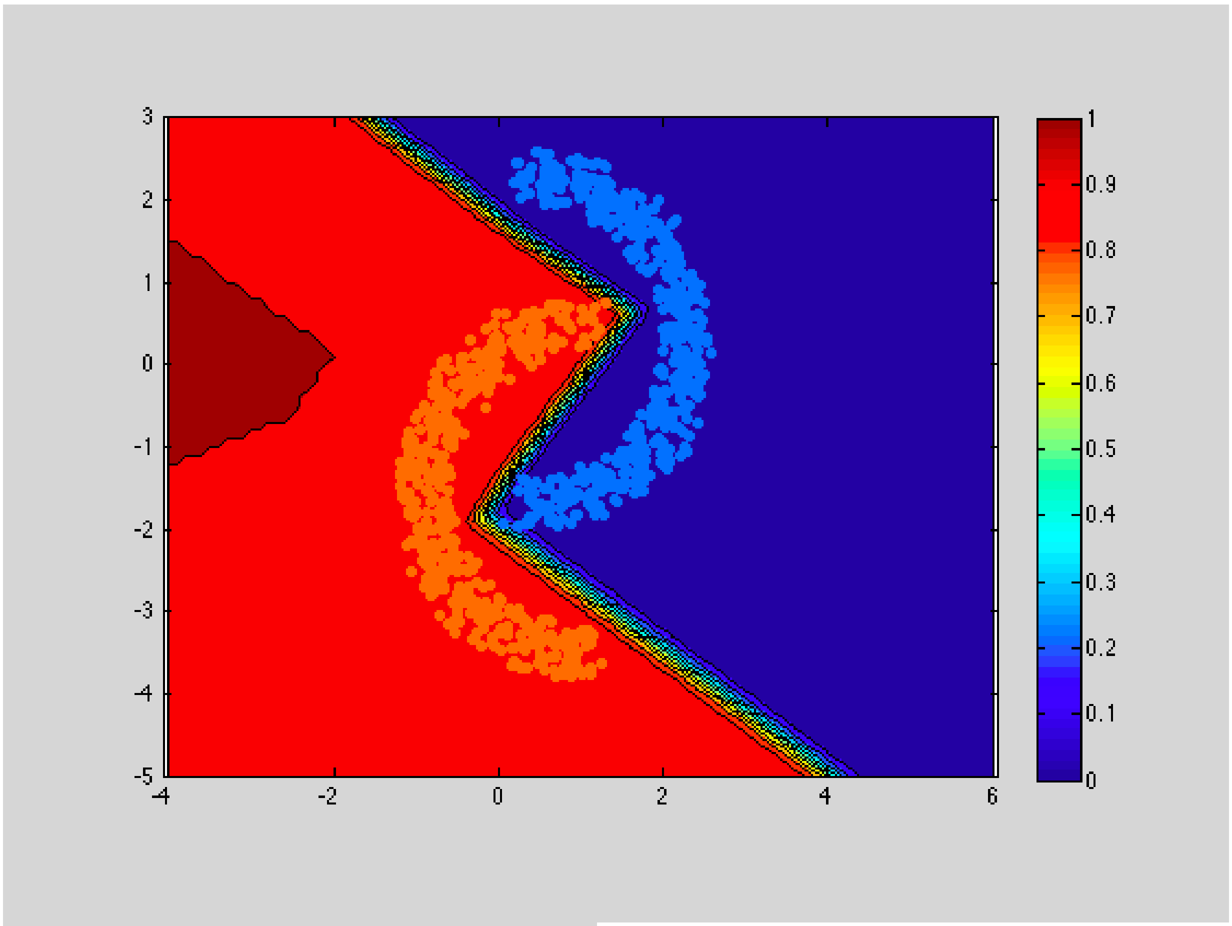
QB2: In your report, choose a training example {X,Y} and report the values of the analytically computed gradients : `grad_s_Wi`, `grad_s_Wo`, `grad_s_bi`, `grad_s_bo`

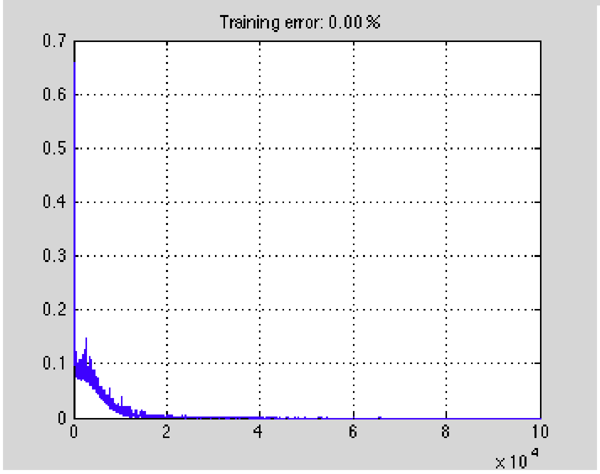
as well as their numerically computed counterparts:
`grad_s_Wi_approx, grad_s_Wo_approx, grad_s_bi_approx, grad_s_bo_approx`
Are their values similar?

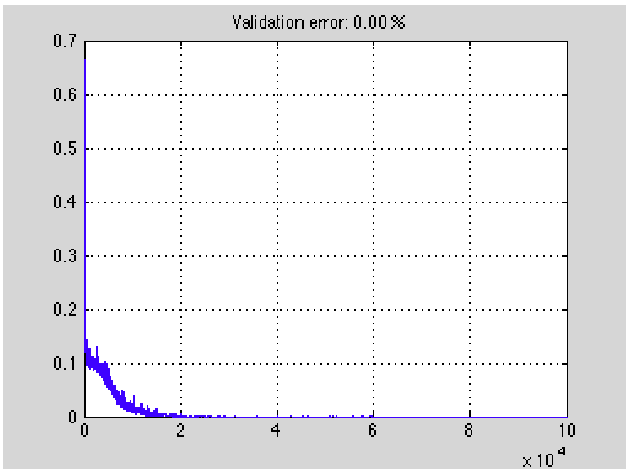
Stage C: Training the network using backpropagation and experimenting with different parameters.

Use the provided code in **nnet_demo.m** to load the training data and validation data from files **double_moon_train1000.mat** and **double_moon_val1000.mat**, respectively. This is the same data as shown in the demo in the class.

Set the number of hidden units to 7 by setting h=7 in the code and set the learning rate to 0.02 by setting lrate = 0.02. Run the training code. Visualize the trained hyperplane using the provided function **plot_decision_boundary(Xtr,Ytr,Wi,bi,Wo,bo).** Show also the evolution of the training and validation errors. You may get plots like this:







Note that you may not get identical plots as the network is initialized randomly and may converge to a different solution.

QC1: Include the decision hyper-plane visualization and the training and validation error plots in your report. What are the final training and validation errors? After how many iterations did the network converge?

QC2: **Random initializations.** Repeat this procedure 5 times from 5 different random initializations. Record for each run the final training and validation errors. Did the network always converge to zero training error? Note: to speed-up the training you can set the variable `visualization_step = 10000`. This will plot the visualization figures less often and hence will speed-up the training.

QC3: **Learning rate.** Keep h=7 and change the learning rate to values lrate = {2, 0.2, 0.02, 0.002}. For each of these values run the training procedure 3 times and observe the training behaviour. For each run and the value of the learning rate report: the final (i) training and (ii) validation errors, and (iii) after how many iterations the network converged (if at all). Visualize the decision hyperplane and the evolution of the training error for each value of the learning (here only one of the three runs is sufficient). Briefly discuss the different behaviour of the training for different learning rates.

QC4: **The number of hidden units.** Set the learning rate to 0.02 and change the number of hidden units h = {1, 2, 5, 7, 10, 100}. For each of these values run the training procedure 3 times and observe the training behaviour. For each run and the value of the number of hidden units record and report: the value of the final (i) training and (ii) validation error, and (iii) after how many iterations the network converged (if at all). Show the plot of the final decision hyperplane for each value of h (only one of the three plots for each h is sufficient). Discuss the different behaviours for the different numbers of hidden units.

Part 2: Image classification with CNN features

In this part of the exercise you will experiment with a convolutional neural network (CNN) that has been pre-trained on a large-scale ImageNet classification task. While training such a network can take days on powerful GPU hardware, once trained, CNNs can be used for new classification tasks. Here you will use the ImageNet pre-trained CNNs to classify images of four classes from your previous Assignment 2.

Getting started

- Download the MatConvNet package http://www.vlfeat.org/matconvnet from http://www.vlfeat.org/matconvnet/download/matconvnet-1.0-beta7.tar.gz
- Unpack the code (`tar zxf matconvnet-1.0-beta7.tar.gz`) and compile mex functions by typing `make` in `matconvnet-1.0-beta7` directory. Note: you will need to edit `Makefile` to direct `MATLABROOT` (line 18) to the Matlab directory on your system. Compilation may also require some other adjustments depending on your system and compiler, see more instructions here http://www.vlfeat.org/matconvnet/#installing
- Start Matlab in `matconvnet-1.0-beta7/matlab` directory and initialize MatConvNet package by running `vl_setupnn`
- Download pre-trained CNN model (244Mb) from http://www.vlfeat.org/sandbox-matconvnet/models/imagenet-vgg-f.mat and load the model into Matlab `net = load('imagenet-vgg-f.mat')`
- You will also need the code and the data from your previous class Assignment 2 http://www.di.ens.fr/willow/teaching/recvis15/assignment2/

Stage A: Computing CNN features

Compute forward pass of the network and save network output for all images in the `data/images` directory of Assignment 2. For this purpose you should first normalize each image before passing it through the network by adapting the following example code:

```
im  = imread([A2path '/data/images/000005.jpg']);
im_ = single(im);
im_ = imresize(im_, net.normalization.imageSize(1:2));
im_ = im_ - net.normalization.averageImage;
```

Q2A1: What is done by the above normalization code and why is it needed?

The output of the CNN forward pass given `im_` as input can now be computed as
`res = vl_simplenn(net, im_);`

Q2A2: Look at the content of variables `net` and `res`. What is their structure? What is encoded by `net.layers{k}` for different values of `k`? What do values in `res(k).x` represent for different values of `k`? For hints refer to http://www.vlfeat.org/matconvnet/matconvnet-manual.pdf

For each image save outputs of the softmax (last) layer of the network as well as the outputs of fully-connected layers fc7 and fc8.

Stage B: Image classification using CNN features and linear SVM

Follow your code in Assignment 2, Stages B,C,D and train a linear

SVM classifier for aeroplanes, motorbikes and persons using network output as feature vectors. Cross-validate the C parameter of SVM and evaluate precision-recall and average precision (AP) for each class. Try using different layers of the network output as input features for SVM. Experiment with different feature normalizations (no normalization, l1, l2).

Q2B1: Report your classification results for the three object classes by including precision-recall plots and AP values into the report. What CNN layers work best as features? Are feature normalization and SVM cross-validation important steps to obtain best results? Motivate your answer by performance numbers. ImageNet dataset contains no person class. Can you connect this fact to the difference in the performance of your person classifiers trained/tested using different layers of CNN output?

Q2B2: Compare your best results using CNN features to your results obtained in Assignment 2 for corresponding object classes and training/testing image subsets. What conclusions can you draw? Illustrate a few negative test samples with the highest classification scores (false positives) and a few positive test samples with the lowest classification scores (false negatives) for each class. Can you explain the errors of the classifier?

Note: MatConvNet package should compile on different platforms. In case you try different options and still fail to compile it, please contact Ivan Laptev or Josef Sivic.

What to hand-in:

Hand-in a written report containing brief answers to the above questions (**shown in red**). Include the label of the question, e.g. "QA1" at the beginning of each answer. Your answers should include results, graphs or images as requested in each question.

Instructions for formatting and handing-in assignments:

- At the top of the first page of your report include (i) your name, (ii) date, (iii) the assignment number and (iv) the assignment title.
- The report should be a single pdf file and should be named using the following format: A#_lastname_firstname.pdf, where you replace # with the assignment number and "firstname" and "lastname" with your name, e.g. **A3_SIVIC_Josef.pdf**.
- Zip your code and any additional files (e.g. additional images) into a single zip file using the same naming convention as above, e.g. **A3_SIVIC_Josef.zip**. We do not intend to run your code, but we may look at it and try to run it if your results look wrong.

Send your report (a single pdf file) and code (excluding the data) in a single zip file to **Josef Sivic <Josef.Sivic@ens.fr>**.