Introduction to Deep Learning 2014-2015

Assignment 3: CNNs, maxout (3 Points/20) Deliverable: 22/2/15

Instructor: Stavros Tsogkas, stavros.tsogkas@centralesupelec.fr

1 Theory (0.5 Point)

Prove the equality of the two objectives in Slide 60 of Lecture 7.

2 Experiments (2.5 Points)

In this exercise, we will consider training a Convolutional Neural Network (CNN) to recognize hand-written digits from the MNIST database. The goal of this assignment is to get familiar with CNNs, their fundamental computational blocks, and compare the effects of *dropout* and *maxout* in the system's performance.

We will use MatConvNet [1], a free, modular, and easy to use MATLAB/C++/CUDA library for CNNs. MatConvNet can work either in CPU mode, or take advantage of an available Nvidia GPU, building on top of MATLAB's GPU support in the Parallel Programming Toolbox. We start with a brief recap of the dropout and maxout techniques, that have already been mentioned in lecture 7. More information about the other types of CNN layers along with their implementation details can be found in the MatConvNet manual.

2.1 Dropout

Dropout is a technique that can be used as a proxy for model averaging, without having to actually train a large number of different models. The idea behind dropout is the following: consider the simple case of a neural network with one layer that has H hidden units. Every time we present the network with a training example, we randomly omit each hidden unit with a probability $p_d = 0.5$. In practice, this is equivalent to sampling from 2^H different architectures, however all these architectures share the same weights. Implementation-wise, if x, y are the input and output of the dropout layer respectively, we can use a binary mask M to randomly deactivate a subset of units: $y = M * x$, where in this case, $*$ denotes the Hadamard (element-wise) product.

At test time, we activate all hidden units but halve their responses. For a multi-layered, or nonlinear network this may not be the same as averaging all the different model architectures encountered during training, but it turns out to be a good and efficient approximation.

2.2 Maxout

Maxout is a feed-forward architecture that has recently demonstrated improved results on datasets such as MNIST and CIFAR. Given a vector $\mathbf{x} \in \mathbb{R}^d$, that may be either the input v or a hidden layer's state, the

[1] `http://www.vlfeat.org/matconvnet/`

maxout layer implements the function:

$$h_i = \max_{j \in [1,k]} z_{ij} \tag{1}$$

where $zij = x^T W_{...ij} + b_{ij}$, and $W \in \mathbb{R}^{d \times m \times k}$ and $b \in \mathbb{R}^{m \times k}$ are the weight and bias parameters to be learned. In a convolutional network, we can construct a maxout feature map by taking the maximum over feature channels: this is similar to spatial pooling, only this time pooling is performed across channels instead of neighboring nodes within the same channel. Maxout can be used in conjunction with dropout and spatial pooling.

3 Assignments

3.1 Getting started with MatConvNet

To get started with MatConvNet for the assignments of the class do the following steps:

1. Download and compile MatConvNet following the online instructions. The simplest way is to use the MATLAB function `vl_compilenn` that is already included in the library.
2. Cd in the examples/ directory; this should be your working directory. All intermediate training results and plots will be saved under examples/data/mnist-baseline/.
3. Copy `cnn_mnist_DLclass.m` in the examples/ directory. This is a modified version of `cnn_mnist.m` that you will use to do all your tests and experiments. We advise you to keep the original version so that you can readily compare with your own networks.

The functions that you will mostly use are the following:

- `cnn_mnist_DLclass.m`: Used to setup and train a CNN on the MNIST database. Defines the layers of a typical Yann Lecun CNN (LeNet) and downloads the dataset if it is not already in the MatConvNet directory. This modified version of `cnn_mnist.m` uses a subset of the MNIST database to speed up experiments.
- `cnn_train.m`: Extracts features and feeds training data to the optimization algorithm. The optimization scheme used is a mini-batch version of Stochastic Gradient Descent (SGD), combined with weight decay and momentum. This function also stores intermediate results, keeps record of the train and validation errors and plots their values in every training round (epoch).
- `vl_simplenn.m`: Implements the output functions and derivatives for every type of layer. Implementations for new layers should be added here.

Note: In MatConvNet there are no "fully-connected" layers. Those are also implemented as convolutional layers and thus are handled by `vl_nnconv()`.

3.2 Playing around with CNN layers and dropout (0.5 points)

- First train the original LeNet on the training data subset and store the results. Experiment with the network topology and see how changes affect performance on the validation set. You can try

reducing/increasing the depth of the network by removing/adding pairs of convolutional and max-pooling layers. For each topology train the network for 100 epochs and compare info.val.error(end) that is stored in the corresponding net-epoch-100.mat file.

- Add a *dropout* layer right after the *relu* layer and see how that affects performance. For all your experiments, include the respective plots of *objective-vs-epoch* and *error-vs-epoch*.

3.3 Maxout implementation (2 Points)

- Build on the template that is provided to implement a *maxout* layer, according to section 2.2. Divide the feature channels into groups of a given size g and take the maximum in each one of these groups separately. More specifically, suppose that $\mathbf{x}$ is your feature array, with dimensions $h \times w \times k \times N$, where h, w are the height and width respectively, k is the number of channels and N is the number of training samples in the batch. The output of the maxout layer should be an array $\mathbf{y}_{max}$ of dimensions $h \times w \times k/g \times N$, where $\mathbf{y}_{max}(1, 1, i, 1) = max(\mathbf{x}(1, 1, (i-1)g + 1 : i \cdot g, 1)), i \in [1, k/g]$.

  Use $g = 2$ and $g = 4$.

  Note: You will need to compute the derivative of the maxout layer that will be used in the back-propagation step.

- Add a maxout unit after the second pooling layer (fourth layer in the network) in the original LeNet. Do the same thing but this time add the maxout unit after the first pooling layer (second layer in the network). Which topology leads in better performance? Compare with the dropout version of the network from exercise 3.2. What happens if you combine dropout and maxout?

Useful MATLAB functions: `bsxfun`, `ndgrid`.