Machine Learning for Computer Vision, MVA 2015-2016

Instructor: Iasonas Kokkinos, iasonas.kokkinos[at]ecp.fr

Programming Assignment 3 - 3.0 Points/20 Deliverable: 3/4/16

1 Building classifiers as neural network layers (2.0/3.0)

In neural networks the relationship between input and output is modelled as a sequence of layers, where each layer performs a specific operation on its input. For a network with N layers, the network function can be written as

$$y(\mathbf{x}, \mathbf{w}) = f_N\left(...f_2\left(f_1\left(\mathbf{w_1}, \mathbf{x}\right), \mathbf{w_2}\right), \mathbf{w_N}\right) \tag{1}$$

where $\mathbf{w} = [\mathbf{w_1}, \mathbf{w_2}, \dots \mathbf{w_N}]$ and f_i is the function corresponding to layer i.

Optimization amounts to minimizing a generic loss function $L(\mathbf{w})$, which is commonly implemented as the last layer of the NN architecture. The update rule for weight w^i is

$$w^i_{t+1} = w^i_t + \Delta w^i_{t+1} = w^i_t - \alpha \frac{\partial L(\mathbf{w})}{\partial w^i} - \alpha \lambda w^i_t. \tag{2}$$

Since each layer in the network only has access to its own input (the output of the previous layer) the partial derivative is computed by applying the chain rule of differentiation[1]

$$\frac{\partial L(\mathbf{w})}{\partial w^i} = \frac{\partial L(\mathbf{w})}{\partial \mathbf{x_N}} \frac{\partial \mathbf{x_N}}{\partial \mathbf{x_{N-1}}} \cdots \frac{\partial \mathbf{x_2}}{\partial \mathbf{x_1}} \frac{\partial \mathbf{x_1}}{\partial w^i}. \tag{3}$$

For a more detailed analysis on the implementation of backpropagation, we refer to the MatConvNet manual.

Your task in this assignment is to re-implement the linear regression, logistic regression and linear SVM classifiers as layers of a neural network, using MatConvnet. First download and install MatConvNet following the instructions on the website. Make sure you add the MatConvNet directory in your working path. You can do that with the command: `>>run('MATCONVNETROOT/matlab/vl_setupnn')`.

You are provided with three new m-files: `cnn_exp.m` contains all the necessary functions to train a (convolutional) neural network and `myloss.m` responsible for computing different types of loss functions and their derivatives. Finally, `week_3_a.m` is a script used to facilitate your experiments with different parameter values.

To integrate `myloss.m` in the MatConvNet code, open the `vl_simplenn.m` function and add a new entry in the switch command. Feel free to create any additional layers you consider necessary. **Do not forget to add both forward and backward steps for all new layers you create.**

Deliverables

- Fill in the missing code in `cnn_init` function in `cnn_exp.m`, to create networks that use different types of losses (log-loss and hinge loss). Check `MATCONVNETROOT/matlab/examples/mnist/cnn_minst_init.m` to get an idea of how to setup CNN architectures.

[1] In deep learning this procedure is also known as *backpropagation*.

- Fill in the missing code in `myloss.m` to compute the loss over the training set and its derivative. Note that the derivative should be computed with respect to the *input* of that layer - *not* the weights $\mathbf{w}$.
- Train networks using different loss functions (square, log, hinge) and print the resulting curves of the loss function across iterations (epochs). Vary the learning rate α and λ (parameters `opts.train.learningRate` and `opts.train.weightDecay` respectively) and describe what you observe.
- Is parameter λ in Eq. 2 related to a quantity encountered in lectures 1-2? If yes, in what way?
- Consider the new weight update rule:

$$w_{t+1}^i = w_t^i + \mu \Delta w_t^i - \alpha \frac{\partial L(\mathbf{w})}{\partial w^i} - \alpha \lambda w_t^i. \tag{4}$$

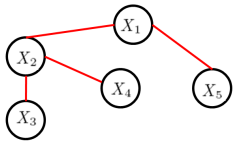
  The additional term $\mu \Delta w_t^i$ is called *momentum* and μ can be set using `opts.train.momentum`. Training networks using $\mu = 0.5$ and $\mu = 0.9$, and plot L. What do you observe? How does momentum affect training?

Hints

- When completing `myloss.m`, keep in mind that the goal is to *minimize* an *error/loss* function.
- The inner product of two vectors $\langle \mathbf{w}, \mathbf{x} \rangle$ can be implemented as a fully connected CNN layer (convolutional layer with a 1×1 kernel size) and a single node (neuron).
- If training diverges, try using a smaller learning rate.

2 Sum/Max-Product algorithms (1.0)

Consider the following MRF:



It has discrete variables, each of which can take a distinct number of values. Specifically, $|X_1| = 3, |X_2| = 2, |X_3| = 4, |X_4| = 2, |X_5| = 2$.

The potential functions of the network are as follows:

$$\Phi(X_1, X_2) = \begin{bmatrix} .1 & .9 \\ .5 & .2 \\ \underbrace{.1}_{\Phi(X_1=3, X_2=1)} & .1 \end{bmatrix} \Phi(X_2, X_3) = \begin{bmatrix} .1 & .9 & .2 & .3 \\ .8 & .2 & .3 & .6 \end{bmatrix} \Phi(X_2, X_4) = \begin{bmatrix} .1 & .9 \\ .8 & .2 \end{bmatrix} \Phi(X_1, X_5) = \begin{bmatrix} .1 & .2 \\ .8 & .1 \\ .3 & .7 \end{bmatrix}$$

$$\Phi(X_j, X_i) = \Phi(X_i, X_j)^T$$

while the joint distribution of the state is given by

$$\begin{aligned} P(X) &= \frac{1}{Z} \Phi(X_1, X_2) \Phi(X_2, X_3) \Phi(X_2, X_4) \Phi(X_1, X_5) \\ Z &= \sum_X \Phi(X_1, X_2) \Phi(X_2, X_3) \Phi(X_2, X_4) \Phi(X_1, X_5) \end{aligned} \tag{5}$$

As we do not know the value Z in advance, we can consider an unnormalized distribution,

$$\tilde{P}(X) = \Phi(X_1, X_2)\Phi(X_2, X_3)\Phi(X_2, X_4)\Phi(X_1, X_5) = P(X)Z, \tag{6}$$

where, by definition, $Z = \sum_X \tilde{P}(X)$.

- Implement the sum product algorithm; execute it for $\tilde{P}(X)$ in order to compute

$$\tilde{P}(X_1) = \sum_{X_2, X_3, X_4, X_5} \tilde{P}(X) \tag{7}$$

$$\tilde{P}(X_2) = \sum_{X_1, X_3, X_4, X_5} \tilde{P}(X) \tag{8}$$

- Use these to compute the Partition function, Z
- Use Z to compute the normalized marginal distributions for $P(X_1), P(X_2)$ and report your results.

- Construct a function that returns P(X). Evaluate it for

$$\begin{aligned} X &= \{1, 2, 4, 2, 1\} \\ X &= \{3, 1, 2, 1, 2\} \\ X &= \{2, 2, 1, 2, 1\} \end{aligned}$$

- Implement the max-product algorithm for this network.

Bonus, 0.5 Consider we know $X_5 = 1, X_4 = 2, X_1 = 3$. Compute $P(X_2), P(X_3)$. You will need to modify the potentials and therefore recompute the partition function.