IFT3395/6390, Assignment 2

Date posted: Oct. 30, 2014,
Due: Nov. 6, 2014, at the *start* of class

---

1. (6/10) Suppose you have a training dataset $D_n = \{(\mathbf{x}_1, t_1), \ldots, (\mathbf{x}_n, t_n)\}$ with input observations $\mathbf{x}_i \in R^d$ and class-labels $t_i \in \{1, \ldots, M\}$ (there are M classes in total).

   Consider a feed-forward neural net with a single hidden layer (thus, 3 layers in total if you count the inputs and outputs as separate layers). The hidden layer contains d_h neurons fully connected to the input layer, with hyperbolic tangent non-linearity (tanh). The output layer contains M neurons (fully connected to the hidden layer) which are combined using a softmax non-linearity. The output of the jth neuron in the output layer can be interpreted as computing the probability that the input, $\mathbf{x}$, to the network belongs to class j.

   It is strongly suggested that you draw a sketch of the network in order to better understand the following stages (but we are not asking to provide your sketch in your response!).

   a) Let $\mathbf{W}^{(1)}$ be a $d_h \times d$-matrix of weights and $\mathbf{b}^{(1)}$ be a vector of biases, which together specify the synaptic weights going from the input-layer to the hidden layer. What is the dimension of $\mathbf{b}^{(1)}$?

   Write down the formula to calculate the vector of activations (i.e. before the non-linearity) of the neurons in the hidden layer, $\mathbf{h}^a$, given an input, $\mathbf{x}$, at first in matrix expression, then more detailed the element-by-element computation of the entries of $\mathbf{h}^a$. Then write down the vector of outputs of the hidden layer neurons, $\mathbf{h}^s$, in terms of the activations, $\mathbf{h}^a$.

   b) Let $\mathbf{W}^{(2)}$ be the weight matrix from the hidden to output layer and $\mathbf{b}^{(2)}$ be the vector of biases for the output layer. What are the dimensions of $\mathbf{W}^{(2)}$ et $\mathbf{b}^{(2)}$?

   Write down the formula describing the vector of activations of neurons in the output layer $\mathbf{o}^a$ given $\mathbf{h}^s$ in matrix form, then in detail element-wise form.

   c) The output of the neurons in the output layer is given by

$$\mathbf{o}^s = \text{softmax}(\mathbf{o}^a)$$

$$o_k^s = \frac{\exp\left(o_k^s\right)}{\sum_{j=1}^{M} \exp\left(o_j^s\right)}$$

   d) The negative log-likelihood loss function is given by

$$L(\mathbf{x}, t) = -\log \mathbf{o}^t(\mathbf{x})$$

Training the neural network amounts to finding the parameters which minimize the value of the loss function for the training set. Mention precisely what constitutes $\boldsymbol{\theta}$ the set of all network parameters. How many scalar parameters does $\boldsymbol{\theta}$ contain?

e) For learning we will use the method of stochastic gradient descent. The gradient of the cost, L, incurred by the i'th training example $(\mathbf{x}_i, t_i)$ wrt. the parameters is:

$$\frac{\partial L}{\partial \boldsymbol{\theta}} = \begin{pmatrix} \frac{\partial L(\mathbf{x}_i, t_i)}{\partial \theta_1} \\ \cdot \\ \cdot \\ \cdot \\ \frac{\partial L(\mathbf{x}_i, t_i)}{\partial \theta_{n_\theta}} \end{pmatrix}$$

To calculate the gradient we will use error back-propagation.

IMPORTANT: The method of error back-propagation is based on an intelligent and efficient application of the chain rule of differentiation, that avoids the unnecessary repetition of expensive calculations. It assumes that we have pre-calculated and saved off the activations and outputs of all neurons in the network during the forward propagation phase, and that one can use them without having to re-calculate them when computing the gradient. For this reason it is important to express the derivatives as a function of these pre-calculated values, without substituting their own detailed calculation. This would amount to re-computing these and would yield expressions longer, more complicated and more inefficient than necessary.

As you showed in Assignment 1, the partial derivatives of the cost, L, wrt. the activations of the neurons in the last layer are

$$\frac{\partial L(\mathbf{x}, t)}{\partial o_k^s} = \begin{cases} o_k^s - 1 & \text{si} \quad k = t \\ o_k^s & \text{si} \quad k \neq t \end{cases}$$

or, in matrix form:

$$\frac{\partial L(\mathbf{x}, t)}{\partial \mathbf{o}^s} = \mathbf{o}^s - \text{onehot}_m(t)$$

g) Show that the gradients wrt. parameters $\mathbf{W}^{(2)}$ and $\mathbf{b}^{(2)}$ are given by:

$$\frac{\partial L}{\partial \mathbf{W}^{(2)}} = \frac{\partial L}{\partial \mathbf{o}^a} \left(h^s\right)^{\mathrm{T}}$$

and

$$\frac{\partial L}{\partial \mathbf{b}^{(2)}} = \frac{\partial L}{\partial \mathbf{o}^a}$$

where $\frac{\partial L}{\partial \mathbf{o}^a}$ and $\mathbf{h}^s$ are column vectors.

Specify their dimensions.

h) Using the chain rule

$$\frac{\partial L}{h_j^s} = \sum_{k=1}^{M} \frac{\partial L}{\partial o_k^a} \frac{\partial o_k^a}{h_j^s}$$

show that the partial derivatives of the cost L wrt. the outputs of the neurons in the hidden layer are given by:

$$\frac{\partial L}{\partial \mathbf{h}^s} = \left(W^{(2)}\right)^{\mathrm{T}} \frac{\partial L}{\partial \mathbf{o}^a}$$

where $\frac{\partial L}{\partial \mathbf{o}^a}$ is a column vector.

Specify their dimensions.

i) Calculate the partial derivatives wrt. the activations of the neurons of the hidden layer. Since L depends on the activation, h_j^a, of a neuron in the hidden layer only through its output h_j^s, the chain rule yields:

$$\frac{\partial L}{\partial h_j^a} = \frac{\partial L}{\partial h_j^s} \frac{\partial h_j^s}{\partial h_j^a}$$

Note that $\mathbf{h}^s = \tanh(\mathbf{h}^a)$, where the hyperbolic tanget activation is applied element-wise. The hyperbolic tangent is given by $\tanh(z) = \frac{\sinh z}{\cosh z} = \frac{e^z - e^{-z}}{e^z + e^{-z}} = \frac{e^{2z}-1}{e^{2z}+1}$ First show that $\frac{\partial \tanh z}{\partial z} = 1 - \tanh^2(z)$. Express the result in matrix form, and define the dimension of each matrix or vector involved.

j) Calculate the gradients wrt. the parameters $\mathbf{W}^{(1)}$ and $\mathbf{b}^{(1)}$ of the hidden layer. Express the gradients in matrix form.

k) Consider quadratic "weight decay " regularization, which penalizes the squared (L2)-norm of of the weights (but not the biases). How does this change the gradient of the cost wrt. to the parameters?

l) Describe in detail how any derivatives will change if we use the rectifier non-linearity "RELU":

$$\mathrm{RELU}(h_j^a) = \begin{cases} h_j^a & \text{if} \quad h_j^s \geq 0 \\ 0 & \text{if} \quad h_j^s < 0 \end{cases}$$

instead of the tanh.

2. (4/10) Implement the neural network, and apply it to the classification data used in Assignment 1 (see Assignment 1 for more information about the data if you forgot the details):

`www.iro.umontreal.ca/~memisevr/teaching/ift3395_2014/devoirs/train_images.txt`

`www.iro.umontreal.ca/~memisevr/teaching/ift3395_2014/devoirs/test_images.txt`

`www.iro.umontreal.ca/~memisevr/teaching/ift3395_2014/devoirs/train_labels.txt`

`www.iro.umontreal.ca/~memisevr/teaching/ift3395_2014/devoirs/test_labels.txt`

You can either (i) use the step-by-step calculation of gradients derived in the previous question (including for weight decay), or (ii) use an existing implementation (such as the library theano).

Hints:

- Parameter initialisation: It is necessary to initialize the parameters randomly (making sure to avoid symmetries and saturation of neurons). Initialize the weights of each layer by drawing from the uniform distribution defined by $\left[\frac{-1}{\sqrt{n_c}}, \frac{1}{\sqrt{n_c}}\right]$, where n_c is the number of inputs to this layer (the number of neurons in the input-layer to which each neuron of this layer is connected, so this number typically changes from layer to layer). The biases can be initialized to 0. Clearly justify any deviation from this choice of initialisation.
- If you use the step-by-step calculation of the gradient we suggest writing the methods *fprop*, *bprop* and *grad* as discussed in class.
- Verification of gradients using finite differences: On can estimate the gardient numerically using finite differences. You need to implement this estimation in order to verify correctness of your derivatives (or of those computed by the librairy that you use). To do so, first calculate the value of the loss for the current parameter value (for one training exemple). After that, modify each (scalar) parameter θ_k by a small value ($10^-6 < \epsilon < 10^-4$) and re-calculate the loss (for the same training exemple), then set the parameter back to its initial value. The partial derivative wrt. each parameter can be estimated by dividing the change in loss by ϵ. The ratio of your gradient calculated using back-prop to the gradient estimated by finite differences should be situer somewhere between 0.99 and 1.01.
- Batchsize:. Perform gradient descent using mini-batches of size 100. In the case of mini-batches, you do not work with individual data-points, but rather a batch of examples at a time, grouped in a matrix (which will also yield a matrix representing values at the hidden layer and at the outputs). (The case of a minibatch size of 1 would be the exact equivalent of stochastic gradient descent.)

What to hand in:

- Gradient verification: produce a plot showing the gradient for your network computed with finite differences using the *first training example*. In the same plot show the gradient computed using back-propagation. Make these plots, even if you use a software like theano for computing derivatives automatically.
- Train your network on the training data. Show training and test curves (curves showing the classification error and the cost function as a function of training epochs). Include in your report the curves obtained with the best value of hyper-parameters, ie. for which your model attains the lowest classification error on the test data. Make two graphs: One for the classification error rates (train and test, clearly identified in the legend) and another for the mean loss function values (train and test). Report the values for the best hyper-parameter settings that you found.
- Rectifier non-linearity: Repeat the preceding experiment using the rectifier non-linearity. You may want to verify the derivatives like before before using this non-linearity (but it is not necessary to hand in the result of this verification).
- (Bonus, 1 point) Repeat the preceding experiment by using Dropout (proposed recently in http://arxiv.org/pdf/1207.0580.pdf) instead of weight-decay for regularization during training: Randomly multiply the hidden unit outputs h_j^s in the

current mini-batch by 0 with probability $\frac{1}{2}$. (This should be done independently for each training example and each hidden dimension.) At test-time, do not perform this random corruption of hiddens, and instead multiply the weights in the hidden-to-output layer by 0.5.