CS-E4850 Computer Vision
Exercise Round 9

The problems should be solved before the exercise session and solutions returned via the MyCourses page. Upload two files: (1) a PDF file containing your written answers to all problems, (2) a Matlab m-file containing the source code for the problem 2. Scanned handwritten solutions are ok for problem 1.

Notice also that the last two problems can be done without solving problem 1 since the solutions are already written out in the subtasks of problem 1 (i.e. only the derivations are missing and asked in problem 1).

If you have not studied basics of neural networks in previous courses and the problem context of these exercises is not clear, it may be helpful to check the slides of the first four lectures of prof. Hinton's course "Introduction to neural networks and machine learning":
`http://www.cs.toronto.edu/~hinton/coursera_slides.html`
`http://www.cs.toronto.edu/~hinton/coursera_lectures.html` (lecture videos).

Exercise 1. Neural networks and backpropagation. (Pen & paper problem)

This exercise presents the backpropagation algorithm for a multi-layer neural network with L layers. To keep things as simple as possible, we do not include bias terms in the network. Thus, the only parameters of the network are the weights for the connections between the neurons of consecutive layers. The consecutive layers are fully connected, i.e., each neuron in a certain layer is directly connected to all neurons of the previous layer. In the hidden layers, the nonlinear function of each neuron is the logistic function σ, $\sigma(z) = 1/(1 + e^{-z})$. In the final output layer the nonlinear function is a softmax function s with n_L outputs. The number of neurons in layer l is denoted by n_l.

Given input vector $\mathbf{x}$ with dimension n_0, the output $\mathbf{y} = f(\mathbf{x})$ produced by our network f is mathematically defined by the following recursion

$$\begin{aligned}
\mathbf{z}^{(1)} &= \mathbf{W}^{(1)}\mathbf{x} & \mathbf{y}^{(1)} &= \sigma(\mathbf{z}^{(1)}), \\
\mathbf{z}^{(l)} &= \mathbf{W}^{(l)}\mathbf{y}^{(l-1)} & \mathbf{y}^{(l)} &= \sigma(\mathbf{z}^{(l)}), \quad l = 2, \ldots, L-1 \\
\mathbf{z}^{(L)} &= \mathbf{W}^{(L)}\mathbf{y}^{L-1} & \mathbf{y}^{(L)} &= s(\mathbf{z}^{(L)}) \\
\mathbf{y} &= \mathbf{y}^{(L)}, &&
\end{aligned}$$

where the weight matrix $\mathbf{W}^{(l)}$ for layer l has size $n_l \times n_{l-1}$, the logistic function is applied to a vector element-wise, and $\mathbf{y}^{(L)} = s(\mathbf{z}^{(L)})$ denotes the output vector of the softmax layer with elements

$$s(z_i^{(L)}) = \frac{e^{z_i^{(L)}}}{\Sigma_{k=1}^{n_L} e^{z_k^{(L)}}}, \quad i = 1, \ldots, n_L.$$

The softmax layer is typically used in classification problems so that the number of outputs n_L equals the number of classes. We may observe that the sum of the elements of the output vector $\mathbf{y}^{(L)}$ equals 1, and hence, they can be interpreted as the predicted probabilities of the classes.

Given training samples, $\mathbf{x}_1, \ldots, \mathbf{x}_m$ and their target class probabilities $\mathbf{t}_1, \ldots, \mathbf{t}_m$, the weights of the neural network can be trained by minimising the cross-entropy loss which is defined as follows

$$E = \frac{1}{m} \sum_{j=1}^{m} -\mathbf{t}_j \cdot \log(\mathbf{y}_j), \tag{1}$$

where $\cdot$ denotes vector dot product and log is the natural logarithm applied to the vector element-wise. The target vectors $\mathbf{t}_j$ are typically "one-hot vectors", i.e. vectors where the element corresponding to the correct class is 1 and other elements are 0. Sometimes the cross-entropy loss is supplemented with so called weight-decay term, which regularises the solution and penalises large weigths by adding their sum of squares to the loss, as follows:

$$E = \left(\frac{1}{m} \sum_{j=1}^{m} -\mathbf{t}_j \cdot \log(\mathbf{y}_j) \right) + \frac{\lambda}{2} ||\mathbf{w}||^2 \tag{2}$$

where λ is a positive scalar and the vector $\mathbf{w}$ contains all the weights, i.e. the elements of all matrices $\mathbf{W}^{(l)}$.

By starting from some random initial guess $\mathbf{w}_0$, the weights can be improved iteratively via gradient-descent minimisation of the loss, i.e.

$$\mathbf{w}_\tau = \mathbf{w}_{\tau-1} - \alpha \frac{\partial E}{\partial \mathbf{w}},$$

where τ indicates the iteration count and parameter α is called *learning rate*.

The gradient $\partial E / \partial \mathbf{w}$ of the cross-entropy loss (1) for a single training sample (i.e. for case $m = 1$) can be calculated with the backpropagation algorithm, which utilises the chain rule of differential calculus recursively, as follows:

$$\frac{\partial E}{\partial \mathbf{z}^{(L)}} = \frac{\partial E}{\partial \mathbf{y}^{(L)}} \frac{\partial \mathbf{y}^{(L)}}{\partial \mathbf{z}^{(L)}}$$

$$\frac{\partial E}{\partial \mathbf{y}^{(L-1)}} = \frac{\partial E}{\partial \mathbf{z}^{(L)}} \frac{\partial \mathbf{z}^{(L)}}{\partial \mathbf{y}^{(L-1)}} = \frac{\partial E}{\partial \mathbf{z}^{(L)}} \mathbf{W}^{(L)} \quad \text{and} \quad \frac{\partial E}{\partial w_{uv}^{(L)}} = \frac{\partial E}{\partial \mathbf{z}^{(L)}} \frac{\partial \mathbf{z}^{(L)}}{\partial w_{uv}^{(L)}} = \frac{\partial E}{\partial z_u^{(L)}} y_v^{(L-1)},$$

whereafter we can continue for all $l = L-1, \ldots, 1$ by computing

$$\frac{\partial E}{\partial \mathbf{z}^{(l)}} = \frac{\partial E}{\partial \mathbf{y}^{(l)}} \frac{\partial \mathbf{y}^{(l)}}{\partial \mathbf{z}^{(l)}} = \frac{\partial E}{\partial \mathbf{y}^{(l)}} \frac{\partial \sigma(\mathbf{z}^{(l)})}{\partial \mathbf{z}^{(l)}}$$

$$\frac{\partial E}{\partial \mathbf{y}^{(l-1)}} = \frac{\partial E}{\partial \mathbf{z}^{(l)}} \frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{y}^{(l-1)}} = \frac{\partial E}{\partial \mathbf{z}^{(l)}} \mathbf{W}^{(l)} \qquad \frac{\partial E}{\partial w_{uv}^{(l)}} = \frac{\partial E}{\partial \mathbf{z}^{(l)}} \frac{\partial \mathbf{z}^{(l)}}{\partial w_{uv}^{(l)}} = \frac{\partial E}{\partial z_u^{(l)}} y_v^{(l-1)},$$

where $\partial E / \partial w_{uv}^{(l)}$ is the partial derivative with respect to element (u, v) of matrix $\mathbf{W}^{(l)}$.

Finally, in the case of multiple training samples, i.e. $m > 1$, we can compute the partial derivates of the weights for each sample as above and then average them to get the partial derivate with respect to the entire batch of training samples.

Task: Your task is to compute the expressions for the partial derivates $\partial E/\partial w_{uv}^{(l)}$ in a special case, where we have two layers in the network, the hidden layer has 7 neurons, and the softmax layer has 10 neurons, i.e. $L = 2$, $n_1 = 7$ and $n_2 = 10$. Draw a schematic picture of the network and proceed with the following stages:

1. Use the cross-entropy loss of Eq. (1) and assume $m = 1$. That is, first compute the partial derivatives for single training sample $\mathbf{x}, \mathbf{t}$.
   (Hint: You do not need to do anything else than just write the loss (1) in case $m = 1$. This is just an introduction to stages 2-7 where the mentioned partial derivates will be calculated in a step-wise manner for the case $m = 1$. The main purpose of this sub-task is to note that one can first focus on the case $m = 1$.)

2. Show that $\partial E/\partial \mathbf{z}^{(2)} = (\mathbf{y}^{(2)} - \mathbf{t})^\top$. Hint: Differentiate the softmax function and assume that the elements of each $\mathbf{t}_j$ sum to one.

3. Show that $\partial E/\partial \mathbf{y}^{(1)} = (\mathbf{y}^{(2)} - \mathbf{t})^\top \mathbf{W}^{(2)}$.

4. Show that $\partial E/\partial w_{uv}^{(2)} = (y_u^{(2)} - t_u) y_v^{(1)}$ and hence $\partial E/\partial \mathbf{W}^{(2)} = (\mathbf{y}^{(2)} - \mathbf{t})\mathbf{y}^{(1)\top}$.

5. Show that $\partial \mathbf{y}^{(1)}/\partial \mathbf{z}^{(1)} = \mathrm{diag}(\mathbf{y}^{(1)} .* (\mathbf{1} - \mathbf{y}^{(1)}))$, where $.*$ denotes element-wise multiplication, $\mathbf{1}$ is a vector of ones and $\mathrm{diag}(\cdot)$ converts the input vector to a diagonal matrix (similar to Matlab function `diag`). Hint: Start by differentiating the logistic function. You should get $\partial \sigma(z)/\partial z = \sigma(z)(1 - \sigma(z))$.

6. Show that $\partial E/\partial \mathbf{z}^{(1)} = (\mathbf{y}^{(2)} - \mathbf{t})^\top \mathbf{W}^{(2)} \mathrm{diag}(\mathbf{y}^{(1)} .* (\mathbf{1} - \mathbf{y}^{(1)}))$.

7. Show that $\partial E/\partial w_{uv}^{(1)} = \frac{\partial E}{\partial z_u^{(1)}} x_v$ and hence $\partial E/\partial \mathbf{W}^{(1)} = \left(\frac{\partial E}{\partial \mathbf{z}^{(1)}}\right)^\top \mathbf{x}^\top$, where vector $\frac{\partial E}{\partial \mathbf{z}^{(1)}}$ is computed above.

8. Finally, if we have multiple training samples (i.e. $m > 1$) we can average the elements of the corresponding matrices $\partial E/\partial \mathbf{W}^{(2)}$ and $\partial E/\partial \mathbf{W}^{(1)}$ that contain the partial derivatives of the network weights for each sample.

9. Further, if we use weight decay as in Eq. (2), its contribution to each partial derivative is simply λ times the value of the corresponding weight (i.e. $\lambda w_{uv}^{(l)}$) and it can be added in the end.

Exercise 2. Image classification using a neural network. (Matlab exercise)

The first exercise problem above gives the solution to Part 2 of the second programming assignment of professor Hinton's course "Introduction to neural networks and machine learning". The assignment and related material are available at
`https://www.cs.toronto.edu/~tijmen/csc321/assignments/a2/`.

Download the code (`a2.m`) and data (`data.mat`) from the above web page and complete the programming task of Part 2 according to the instructions. The solution for the pen and paper part of the task is already given above. Hence, the programming part is a relatively straightforward implementation and can be done without carrying out the derivations since the results of the derivations are already given in **Exercise 1** above.

When you have completed the programming part, run `a2(0,10,30,0.01,0,false,10)` and report the resulting training data classification loss in the returned PDF file.

Exercise 3. Optimisation using backpropagation. (Matlab exercise)

Do Part 3 of the assignment which is available at
`http://www.cs.toronto.edu/~tijmen/csc321/assignments/a2/`

The task is to experiment with the given example code and report your findings. There is no need to program anything in this part but completing it requires that Part 2 is successfully solved.