Neural Networks and Computational Intelligence
Practical Assignment III: Learning by gradient descent

Stochastic gradient descent
The aim of this problem is to get acquainted with gradient descent based training in practice and to do some *hands on* experiments. Take the actual assignment as a starting point for further exploration and self-study.
We consider a simple feedforward neural network with real-valued output as our *student network:*

$$\sigma(\boldsymbol{\xi}) = (\tanh[\boldsymbol{w}_1 \cdot \boldsymbol{\xi}] + \tanh[\boldsymbol{w}_2 \cdot \boldsymbol{\xi}])$$

where $\boldsymbol{\xi} \in I\!R^N$ represents an input vector and $\boldsymbol{w}_1$ and $\boldsymbol{w}_2$ are the N-dim. vectors of adaptive input–to–hidden weights. The fixed hidden-to-output relation is given as the sum of the hidden states *(soft committee machine).*

a) Stochastic gradient descent
Formulate and implement a stochastic gradient descent procedure w.r.t. the weight vectors $\boldsymbol{w}_1$ and $\boldsymbol{w}_2$, which aims at the minimization of the cost function

$$E = \frac{1}{P}\frac{1}{2}\sum_{\mu=1}^{P}\left(\sigma(\boldsymbol{\xi}^\mu) - \tau(\boldsymbol{\xi}^\mu)\right)^2 \tag{1}$$

for a given data set $I\!D = \{\boldsymbol{\xi}^\mu, \tau(\boldsymbol{\xi}^\mu)\}_{\mu=1}^{P}$ with continuous training labels $\tau(\boldsymbol{\xi}^\mu) \in I\!R$.
In each learning step, select one of the P examples, say $\boldsymbol{\xi}^\nu$, randomly with equal probability and use the gradient with respect to its contribution $e^\nu = (\sigma(\boldsymbol{\xi}^\nu) - \tau(\boldsymbol{\xi}^\nu))^2/2$, only (stochastic gradient descent):

$$\begin{aligned} \boldsymbol{w}_1 &\leftarrow \boldsymbol{w}_1 - \eta\,\nabla_1\, e^\nu \\ \boldsymbol{w}_2 &\leftarrow \boldsymbol{w}_2 - \eta\,\nabla_2\, e^\nu \end{aligned}$$

where ∇_j denotes the gradient with respect to $\boldsymbol{w}_j$.
Perform $t_{max} \cdot P$ single training steps, where $t \leq t_{max}$ measures the *training time* in units of P single example updates. Initialize the weights as independent random vectors with $|\boldsymbol{w}_1|^2 = 1$ and $|\boldsymbol{w}_2|^2 = 1$. For comparability of results between different reports, use a constant learning rate $\eta = 0.05$. In addition, you may want to explore different values or time dependent η.

b) A regression problem
In Nestor you will find the file `data3.mat` which you should import into Matlab (`load data3.mat`). It provides a 50×5000-dim. array `xi` corresponding to 5000 input vectors (dimension $N = 50$) and a 5000-dim. vector `tau` corresponding to the target values. Note that the data set (or subsets thereof) is not necessarily learnable for the student network.
Consider (at least) the first $P = 100$ examples as the training set.
In the course of stochastic gradient descent training, measure the cost function E, Eq. (1), and plot it vs. the time t as defined above.
In addition, evaluate the quantity

$$E_{test} = \frac{1}{Q}\frac{1}{2}\sum_{\rho=P+1}^{P+Q}\left(\sigma(\boldsymbol{\xi}^\rho) - \tau(\boldsymbol{\xi}^\rho)\right)^2 \tag{2}$$

which corresponds to the *test* or *generalization error* in terms of quadratic deviation from the target function for Q test examples, set $Q = 100$ or larger.
Plot and compare the evolution of E and E_{test} with the training time t. You should consider large enough training times t_{max} after which the errors seem to decrease no further (apart from random

fluctuations). At the end of the training process, display the obtained, final weight vectors, for instance as bar graphs.

Hand in **at least** the following:

- A brief description of the problem, including the full update equations according to the stochastic gradient descent
- The curves $E(t)$ and $E_{test}(t)$ corresponding to the above specified parameters
- Bar graphs displaying the two final weight vectors after t_{max}

Remarks:

- You can follow the supplementary material (grad-example.pdf) in Nestor for the calculation of derivatives. Note, however, that here hidden-to-output weights and the gain parameters are fixed!
- If you want to solve the problem using some other programming language you can export the arrays in text or csv format from within matlab using the commands `save` or `csvwrite`, for instance. See the matlab documentation for syntax and options.
- You could also consider the concatenated vector $\underline{W} = [\boldsymbol{w}_1, \boldsymbol{w}_2]$ and use the gradient with respect to $\underline{W}$, but since $\nabla_{\underline{W}} = [\nabla_1, \nabla_2]$ this is completely equivalent to the above.
- Compute E and E_{test} after P single randomized steps, not after each individual update. It is recommended to define a (matlab) function which calculates E and E_{test} with $\boldsymbol{w}_1, \boldsymbol{w}_2$ and the corresponding data set (inputs and labels) as arguments.
- Of course, your results will be more reliable if you repeat the training process over several runs from random initializations and take an average of $E(t)$ and $E_{test}(t)$ over these runs. However, this is not obligatory and depends on your patience and and available CPU time.

Possible bonus problems

- consider smaller and larger values of P, e.g. a selection from $P = 20, 50, 200, 500, 1000, 2000$ for the training process. How do the final training and test errors depend on P? Make sure that the Q test examples are not used in the training, of course.
- study systematically the influence of the learning rate η. Potentially consider a time dependent rate as discussed in class.
- Can you observe plateau states? If so, display the corresponding weight vectors and compare with the final ones after leaving the plateau.
- consider a student network with adjustable hidden-to-output weights for the same data
- ...