Learning by gradient descent

Theo Pannetier

I. Introduction

In previous assignments, we have mentioned that perceptron-like machines are limited by linear separability. Many predictive classification or regression problems involve complex interactions between many variables, so the condition of linear separability can rarely be expected to be met. A solution among others to go beyond this limitation is to integrate several perceptron in a machine as an intermediary, hidden layer between the input and output units. From the linear classifications proposed by each hidden unit, a criterion can be used (e.g. a vote in the case of soft-committee machines) to formulate a non-linear rule. The level of complexity in the classification scheme yielded by such a machine is directly dependent on the number of units in the hidden layer. Unfortunately, the complexity of real-world data can rarely be estimated in advance, and the presence of noise (e.g. a mislabelled feature in the training dataset) can render a correct classification impossible to obtain. To circumvent this issue, it is recommended to allow some level error in the classification. The initial problem of finding the rule can then be reformulated as an optimization problem, where one seeks to minimize the error in the predictions made by the (student) machine , allowing one to use the powerful tool that is differential calculus to tackle the problem. Below, I use a two-hidden-units machine to address a possibly non-linear regression problem, and study how the machine can learn using gradient descent on the error function as the core of the algorithm.

II. Problem

The feed-forward committee machine used here contains a single hidden layer of two units, linked to a single output unit. Neurons in the hidden layer use the hyperbolic tangent as their activation function. Input-to-hidden weights w_k (where $k \in \{1, 2\}$ are adapting following the procedure described in paragraphs below, while hidden-to-output weights are fixed (to 1). Input data are provided (as opposed to generated) as a pool of feature vectors ξ, associated to continuous labels τ. The actual rule is unknown. A training dataset of P input vectors ξ (of dimension $N = 50$) and labels τ is constituted from a (provided) pool of data. A second dataset of $Q = P$ points is also constituted, but to measure the generalization ability of the machine. The rule linking ξ and τ is unknown.

The output of the perceptron is then

$$\sigma(\xi) = (tanh(w_1 \cdot \xi) + tanh(w_2 \cdot \xi)) \tag{1}$$

The cost function

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

measures the error made by the machine on the training dataset and is the function we seek to minimize. We also measure the analogous generalization error

$$E_{test} = \frac{1}{Q}\frac{1}{2}\sum_{\rho=P+1}^{P+Q}(\sigma(\xi^\rho) - \tau(\xi^\rho))^2 \tag{3}$$

III. Solution

Initial weights are sampled in $\mathcal{N}(0, 1)$ and normalized such that $|w_k|^2 = 1$. At each training step t, the machine is presented P single examples ν sampled in the training dataset. The input-to-hidden weights are then adapted to minimize the contribution e^ν of point ν to the total error, such that

$$e^\nu = \frac{1}{2}(\sigma(\xi^\nu) - \tau(\xi^\nu))^2 \tag{4}$$

The derivative of e^ν is computed with respect to weight vector w_k

$$\nabla_k e^\nu = (\sigma(\xi^\nu) - \tau(\xi^\nu)) \times \xi^\nu \times (1 - tanh^2(w_k \cdot \xi^\nu)) \tag{5}$$

and w_k are updated with learning rate $\eta = 0.05$

$$w_k = w_k - \eta\nabla_k e^\nu \tag{6}$$

I perform $t_{max} = 500$ training steps and measure $E(t)$ and $E_{test}(t)$ averaged over $nD = 10$ independent runs of the algorithm on the dataset. I repeat the procedure for $P = 50, 500, 2500$. I then set $P = 500$ to study the effect of different η values.

IV. Results

In general, the machine yielded a satisfying approximation of the teacher rule after few training steps (around $t = 20$, Fig.1). However, the error did not improve significantly in later stages of the training, the error curve displaying a plateau for each value of P. The size of the training dataset was found to have a significant impact on both training error and generalization error. The final training error was smaller for the smallest dataset (top panel on Fig. 1), and was also smaller for the intermediate size compared to the largest dataset (middle and bottom panels), although the difference was smaller than with the first case. Training on the smaller dataset also resulted in a high generalization error (Fig.1, top panel). The error increased over the course of the training, indicating this dataset may have been a poor representation of the population. Increasing the size of the training dataset greatly reduced the final generalization error (Fig.1, bottom panel). For all three values of P, the generalization error also reached a plateau through the training.

Using large η values was found to increase the error made by the student machine in the training phase (Fig.2, middle

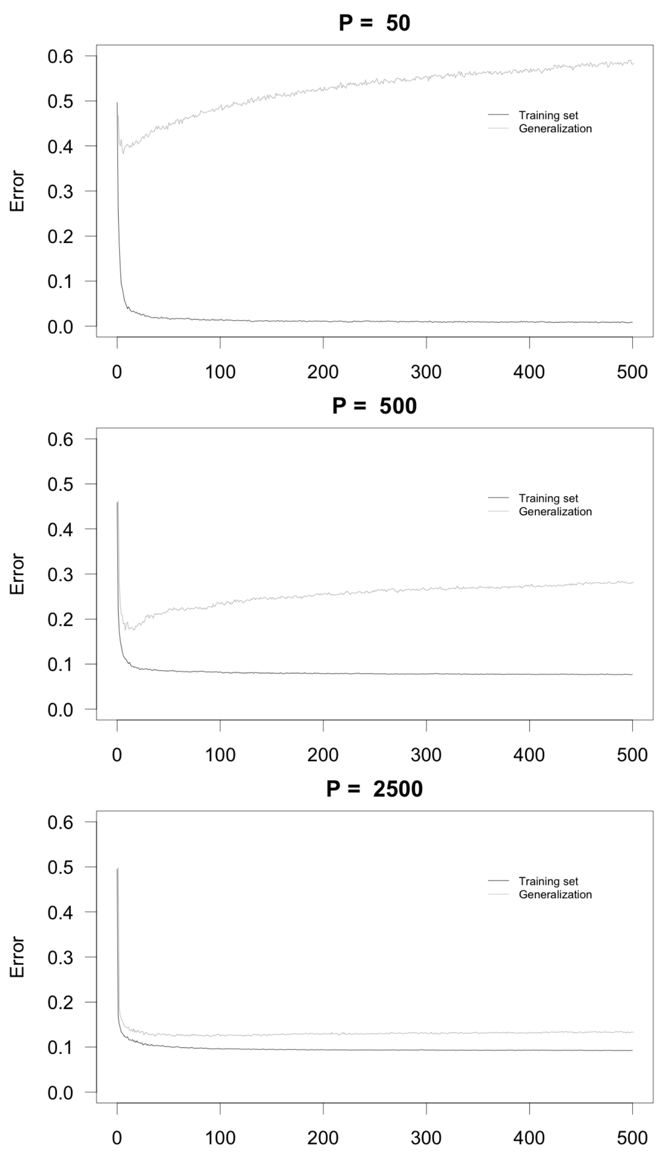

Fig. 1. Mean training (E, in black) and generalization (E_{test}, in grey) error through time for 10 independent runs of the gradient descent algorithm, for different sizes of the training data set.

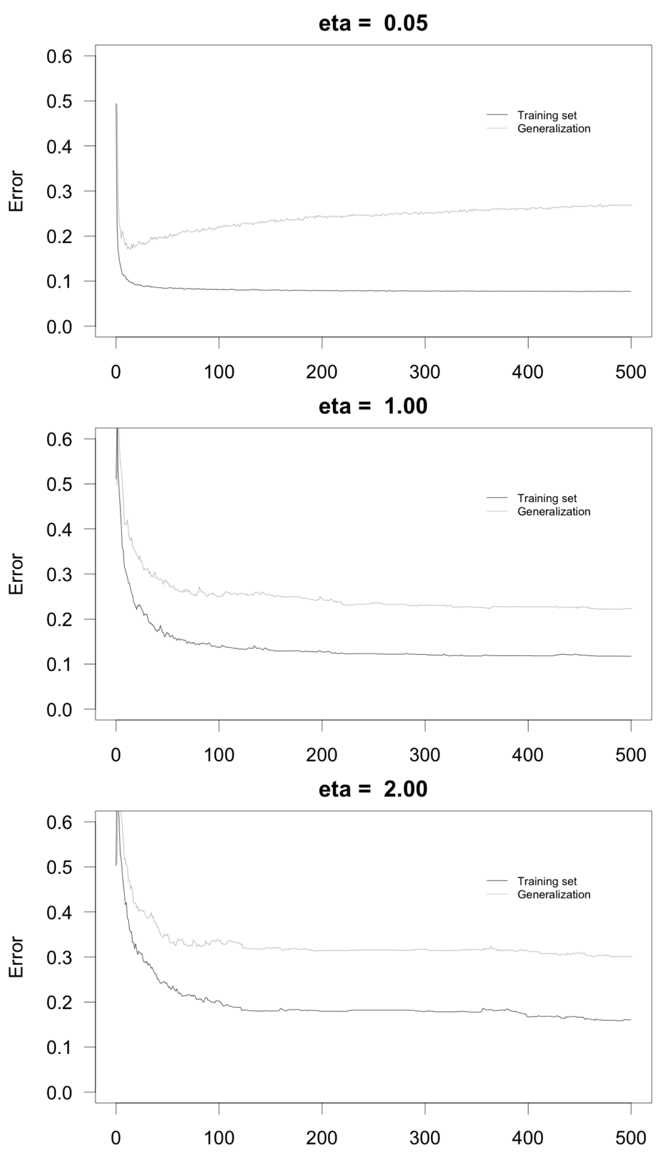

Fig. 2. Mean training (E, in black) and generalization (E_{test}, in grey) error through time for 10 independent runs of the gradient descent algorithm, for different values of the learning rate .

and bottom panels). Generalization error was improved in the case of $\eta = 1$, but not for $\eta = 1$.

Fig 3. displays the elements of w_1 and w_2. at t_{max}. Here, I only present the vectors for the first replicate of each training, but the patterns are similar across all 10 replicates. For $P = 50$, visual inspection does not reveal any pattern, though it seems that weights in one vector are more often associated with weights of opposite sign than same sign. This more striking at $P = 500$, and evident for $P = 2500$. In the latter case, the weights in the first vector even approximate the negative value of the first.

V. Discussion

The results are consistent with results of the previous assignments and theoretical expectations. Both the training error and the generalization error were heavily dependent on the number examples provided for the training. First, the rule dictated by the teacher was best approximated for a small dataset (Fig.1, top panel), i.e. better storage of the data

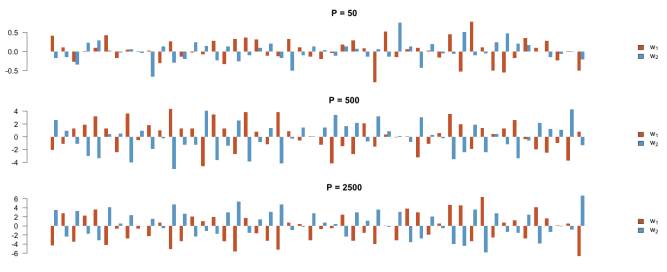

Fig. 3. Final state of the weight vectors at t_{max} for different values of P. N = 50, $\eta = 0.05$

was achieved when the number of examples was limited. Second, the generalization error reduced when a high number of examples was presented. That is, the machine was able to learn most from a large training dataset. This echoes what had been discussed previously in the case of the percpetron; the learning ability of the machine seemed to improve beyond the storage capacity (albeit it was not esimated).

Although the approximation of the rule was generally good, it stopped to improve early in the training in all cases. This is probably indicative of the gradient descent falling in local minima. To circumvent this, I tried to increase the learning rate, in an attempt to allow the algorithm to explore larger sections of the error landscape.

In the example presented here, it proved to actually worsen approximation of the teacher rule, but also (slightly) improved generalization in one case (Fig.2, middle panel). Higher learning rates may have caused large oscillations around a minimal value, preventing the algorithm to converge. Oscillations cannot be observed in the output data due to the results being averaged (see Solution section), but the rougher look of the curves support this interpretation (Fig.2, middle and bottom panel). A perhaps more efficient procedure would be to adapt the learning rate over the course of training (time-dependent learning rate). One could example adapt the pocket algorithm (Gallant, 1990) to store a weight vector and momentarily increase the learning rate to search for better minima. Local optima are a common problem, and numerous methods to address it are described in the optimization literature, for example simulated annealing (Kirkpatrick *et al.*, 1983) or Monte Carlo methods (Metropolis & Ulam, 1949).

VI. Conclusion

The results presented here show how a regression problem can be tackled efficiently with gradient descent. A great advantage, and perspective to this approach is that it allows to take advantage of powerful optimization techniques to improve the learning ability of the machine.

VII. Code availability

The C++ code used to generate the data and the R code used for the plots can both be found at https://github.com/TheoPannetier/NNCI_winter2018

References

Gallant, S.I. (1990). Perceptron-based learning algorithms. *IEEE Transactions on Neural Networks*, 1, 179–191.

Kirkpatrick, S., Gelatt, C.D. & Vecchi, M.P. (1983). Optimization by Simulated Annealing. *Science*, 220, 671–680.

Metropolis, N. & Ulam, S. (1949). The Monte Carlo Method. *Journal of the American Statistical Association*, 44, 335–341.