Machine Learning I – Exercise Sheet 3

Jan Zwank

1 Self organizing maps

We use a one dimensional self organizing map in order to find the shortest route connecting the beergardens in regensburg via straight lines. The topography and road system will be neglected.

For this, we adapt the algorithm from [1] for out needs. The idea of this algorithm is the following. Start by initializing a population of neurons with a ring structure in the 2-dimensional plane. In every step of the iteration, the neurons are moved closer to the beergarden locations. This is done in such a way that the perimeter is minimal.

After loading the data, we begin by normalizing it in such a way that for city coordinates x, y we have $0 \leq x, y \leq 1$ and initialize the neurons randomly in this area following a uniform distribution.

In each iteration, choose a random city and determine the winner, that is, find the neuron with the shortest distance to the chosen city. Then update the winning neuron as well as its neighborhood. The neurons are updated according to the following formula:

$$w_v(s+1) = w_v(s) + \alpha(s) \cdot \eta(u, v, s) \cdot (D(s) - w_v(s))$$

Here, s denotes the iteration, w_u is the chose neuron, $D(s)$ is the chosen beergarden, $\{w_v\}_v$ is the family of all neurons, $\alpha(s)$ is the learning rate, η is the neighborhood function.

In the chosen algorithm the parameters α and η decay exponentially and η is chosen to be constant in a neighborhood of w_u and 0 outside this neighborhood.

Good results can be achieved by using two neurons per beergarden and 5000 iterations. The learning rate is set to an initial value of 0.7 and a decay rate of 0.9999. The radius has initial value of 30 and a decay rate of 0.999.

The results can be viewed in Figures 1 to 6.

There are also options to use linear and static decay, as well as the option to set all parameters manually.

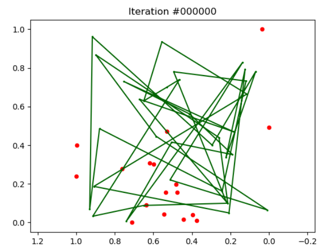

Fig. 1: Iteration 0

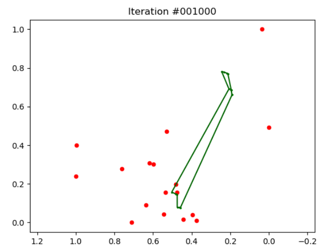

Fig. 2: Iteration 1000

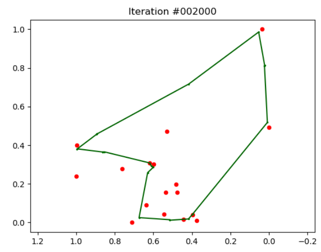

Fig. 3: Iteration 2000

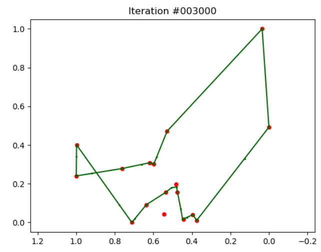

Fig. 4: Iteration 3000

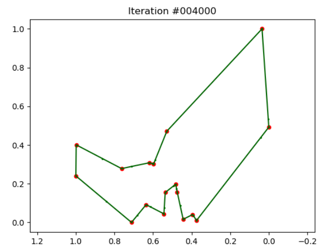

Fig. 5: Iteration 4000

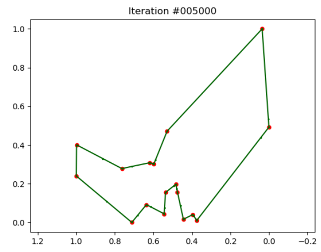

Fig. 6: Iteration 5000

2 Multi layer perceptron

2.1 Regression

We are given samples of a 1-dimensional non-linear function $f(x)$ for $x \in [0, 20]$. Using multi layer perceptrons, we approximate the function based on this input using `MLPRegressor` form `sklearn.neural_network` [2].

In order to compare the performance of various combinations of hidden layers and numbers of perceptrons, we apply the algorithm for the following `hidden_layer_sizes`: (2,3), (7,5), (7,5,3),(3), (5) and (15). The results are depicted in Figure 7

Based on the time it took to fit the respective models, we can see that additional hidden layers add more complexity than adding the same number of hidden neurons in an existing hidden layer. For example, the neural network with `hidden_layer_sizes=(2,3)` took about 4 times as long to fit as that with `hidden_layer_sizes=(5)`, even though they have the same number of neurons total. The same can be seen when comparing `hidden_layer_sizes=(7,5,3)` and `hidden_layer_sizes=(15)`. The model with only one hidden layer is faster than that with three despite having the same number of neurons. This is due to the fact that a neural network has to train all connections between neurons of different layers. It is also noteworthy that an increase in neurons, be it with additional hidden layers or not, does not guarantee a better result. It can be easily seen that the model with only three hidden neurons outperforms the models with `hidden_layer_sizes`$\in \{(2,3), (7,5), (5)\}$. Only the models with `hidden_layer_sizes`$\in \{(7,5,3), (3), (15)\}$ manage to capture the steep incline near zero of our given function.

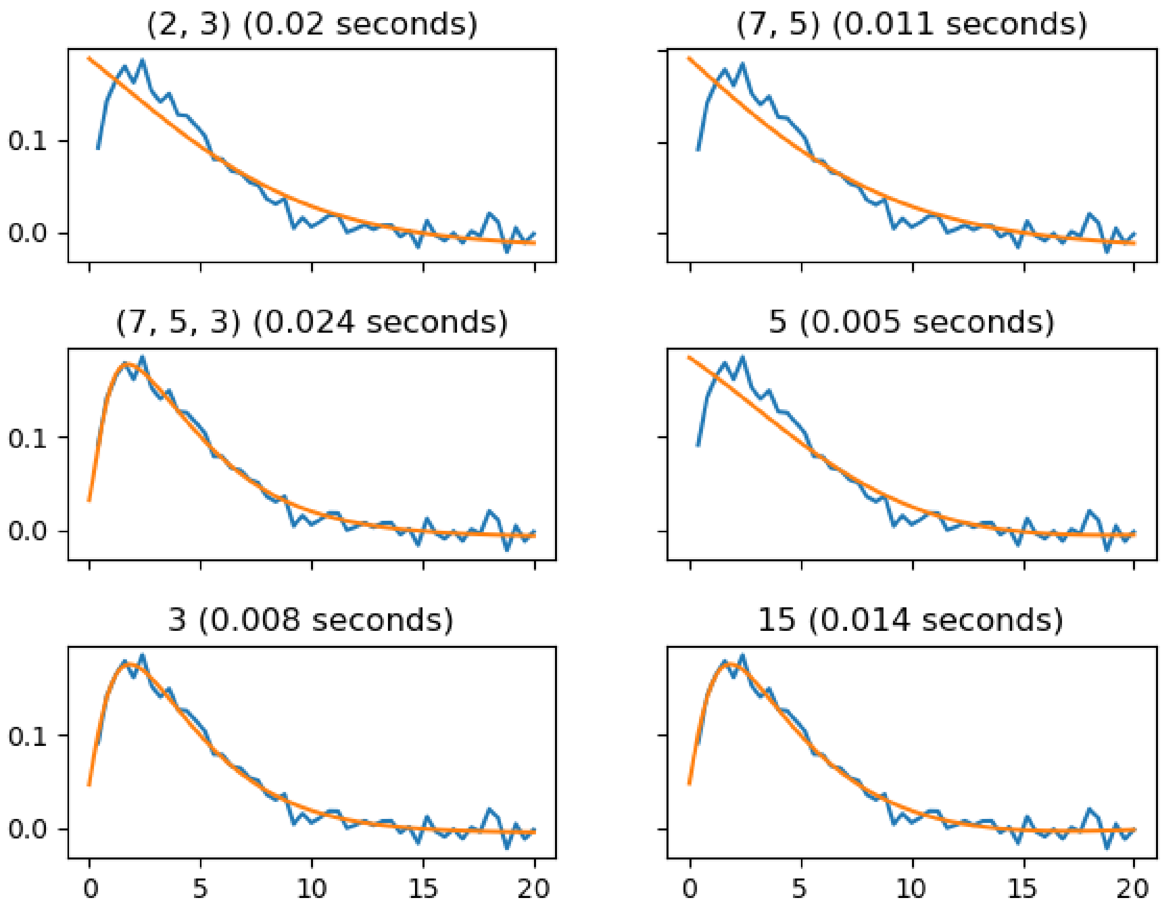

Fig. 7: Comparision of various designs for neural network.

2.2 Classification

For this exercise we choose the p53 mutant dataset from the previous sheet (`http://archive.ics.uci.edu/ml/datasets/p53+Mutants`).

For this exercise we apply the `MLPClassifier` form `sklearn.neural_network` [2] with various architectures. The results are included in Table 1

Compared with the classification using SVM and RF from the previous sheet, the `MLPClassifier` is less accurate than both SVM and RF with top accuracy of 99.42% compared to 99.64% for SVC and 99.84% for random forest with at least 500 trees. However, the classification with neural networks is with training times between 10 and 90 seconds significantly faster than classification with SVC, which took several minutes to finish. The range of training times for random forests and neural networks overlap. A random forest with 100 trees takes 24 seconds, one with 500 trees takes 93 seconds and training time increases further with more trees.

References

[1] Maximilian Leonard Kleinhans and Diego Vicente Martín. *Using Self-Organizing Maps to solve Travelling Salesman Problem.* 2017. URL: `https://github.com/DiegoVicen/ntnu-som`.

[2] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python." In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

Design	(3,)	(5,)	(7,)	(10,)	(3,3)	(3,5)	(3,7)	(3,10)
Accuracy	0.9924	0.9928	0.9934	0.993	0.9785	0.9918	0.9916	0.9942
Time	19.55	36.3	37.18	22.56	37.31	75.72	34.87	33.32
Design	(5,3)	(5,5)	(5,7)	(5,10)	(7,3)	(7,5)	(7,7)	(7,10)
Accuracy	0.9934	0.9914	0.9928	0.993	0.9922	0.9916	0.9922	0.9924
Time	40.31	66.22	38.28	47.4	50.49	42.17	59.69	49.69
Design	(10,3)	(10,5)	(10,7)	(10,10)	(3,3,3)	(3,3,5)	(3,3,7)	(3,3,10)
Accuracy	0.994	0.992	0.9883	0.9926	0.992	0.992	0.9924	0.9926
Time	26.82	90.51	49.3	23.79	27.73	28.73	62.36	40.77
Design	(3,5,3)	(3,5,5)	(3,5,7)	(3,5,10)	(3,7,3)	(3,7,5)	(3,7,7)	(3,7,10)
Accuracy	0.9924	0.9924	0.992	0.993	0.9918	0.993	0.992	0.992
Time	58.68	32.95	70.24	23.43	35.89	47.5	14.09	10.18
Design	(3,10,3)	(3,10,5)	(3,10,7)	(3,10,10)	(5,3,3)	(5,3,5)	(5,3,7)	(5,3,10)
Accuracy	0.9924	0.9938	0.993	0.9928	0.992	0.9926	0.9934	0.9928
Time	32.3	39.42	19.96	43.48	94.46	38.48	63.7	50.91
Design	(5,5,3)	(5,5,5)	(5,5,7)	(5,5,10)	(5,7,3)	(5,7,5)	(5,7,7)	(5,7,10)
Accuracy	0.992	0.9932	0.9916	0.9934	0.9918	0.993	0.9936	0.992
Time	57.19	21.83	50.01	49.4	40.7	29.18	28.14	14.44
Design	(5,10,3)	(5,10,5)	(5,10,7)	(5,10,10)	(7,3,3)	(7,3,5)	(7,3,7)	(7,3,10)
Accuracy	0.9918	0.9924	0.9924	0.992	0.9924	0.992	0.9934	0.992
Time	37.63	60.58	46.47	40.74	41.71	20.58	34.96	21.94
Design	(7,5,3)	(7,5,5)	(7,5,7)	(7,5,10)	(7,7,3)	(7,7,5)	(7,7,7)	(7,7,10)
Accuracy	0.9932	0.9922	0.9924	0.992	0.992	0.9926	0.9888	0.9932
Time	30.58	49.26	31.27	31.3	60.56	51.44	42.96	62.71
Design	(7,10,3)	(7,10,5)	(7,10,7)	(7,10,10)	(10,3,3)	(10,3,5)	(10,3,7)	(10,3,10)
Accuracy	0.9936	0.9898	0.9934	0.9914	0.9926	0.992	0.9924	0.9922
Time	41.12	35.52	37.9	41.08	63.57	36.59	49.47	45.09
Design	(10,5,3)	(10,5,5)	(10,5,7)	(10,5,10)	(10,7,3)	(10,7,5)	(10,7,7)	(10,7,10)
Accuracy	0.992	0.9926	0.992	0.9932	0.9928	0.9938	0.9932	0.992
Time	94.1	38.8	90.27	45.97	65.44	61.05	48.84	44.54
Design	(10,10,3)	(10,10,5)	(10,10,7)	(10,10,10)				
Accuracy	0.9922	0.9934	0.9926	0.9926				
Time	48.74	36.44	47.82	58.55				

Table 1: Accuaracy and training time (in sec.) for MLPClassifier