Machine Learning - III

Methods of Knowledge Based Software Development

S. Nõmm

[1]Department of Software Science, Tallinn University of Technology

15.12.2017

- Part I Neural Networks training

Training

- Training is iterative process which starts with initialization, when initial weights are generated randomly or defined using some other technique.
- During each iteration (epoch) measure of imprecision (loss function (usually nonlinear)) is calculated then values of the weights are updated. The idea is to solve optimization problem with respect to loss function.
- Iterations are repeated until stopping criteria is met.
- The choice of loss function, number of weights to identify and computational restrictions define the choice of the training algorithm.
- Gradient descent, Newton's method, Conjugate gradient, Quasi Newton method, Levenberg - Marquardt algorithm,

Training Algorithms

- Gradient descent (steepest descent). Slow but does not require a lot of memory. Denote $\nabla f(w_i) = g_i$ then weights update rule is

$$w_{i+1} = w_i - g_i \cdot \eta_i, \quad i = 0, 1, \dots$$

  where η_i is the learning rate.
- Levenberg - Marquardt algorithm. Fast but requires a lot of memory. Loss function:

$$f = \sum e_i^2$$

  where e_i are the residuals for each point of the data set. Jacobian matrix of the loss function:

$$J_{i,j} = \frac{\partial e_i}{\partial w_j}$$

  where i indexes data points and j - weights of the network. Weights update rule (here i is the epoch identifier):

$$w_{i+1} = w_i - (J_i^T J_i + \lambda_i I)^{-1}(2 J_i e_i), \quad i = 0, 1, \dots$$

  where λ plays the role of a learning rate.

Example

- Data set is represented by the surface in 3D set.
- Let us adopt LM algorithm to just single neuron and observe convergence process step by step.
- Initialize the process by randomly generating the weights.
- Denote the sum of squared residuals as SSR

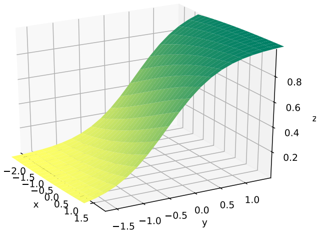
Neuron

Consider one neuron with two inputs and logsig activation function

- Consider one neuron with two inputs (denoted x and y) and logsig activation function. (Notation is chosen to simplify geometric interpretation).
- Mathematical model of such neuron is given by

$$z = \frac{1}{1 + e^{w_1 x + w_2 y}}$$

- For the values $w_1 = 0.5$ and $w_2 = 2$ this function graph in 3D space is



Implementation

- Loss function:

$$f = \sum \left(z_i - \frac{e^{(w_1 x_i + w_2 y_i)}}{e^{(w_1 x_i + w_2 y_i)} + 1} \right)^2$$

- Jacobian matrix of the loss function:
  Elements of the first column:

$$j_{i,1} = \frac{\partial e_i}{\partial w_1} = 2\left(\frac{-z_i x_i e^{(w_1 x_i + w_2 y_i)}}{(e^{(w_1 x_i + w_2 y_i)} + 1)^2} + \frac{x_i e^{2(w_1 x_i + w_2 y_i)}}{(e^{(w_1 x_i + w_2 y_i)} + 1)^3} \right)$$

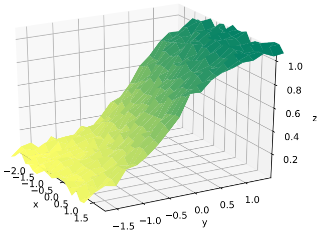
  Elements of the second column:

$$j_{i,2} = \frac{\partial e_i}{\partial w_2} = 2\left(\frac{-z_i y_i e^{(w_1 x_i + w_2 y_i)}}{(e^{(w_1 x_i + w_2 y_i)} + 1)^2} + \frac{y_i e^{2(w_1 x_i + w_2 y_i)}}{(e^{(w_1 x_i + w_2 y_i)} + 1)^3} \right)$$

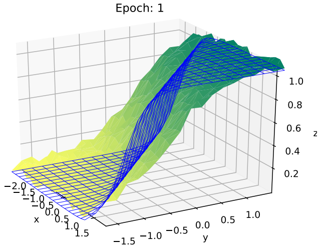
Neuron

- Generate the surface to approximate adding some noise.
- Levenberg - Marquardt algorithm uses sum of squared errors as the loss function.
- Initialize the process by randomly generating the weights.
- Denote sum of squared residuals as SSR

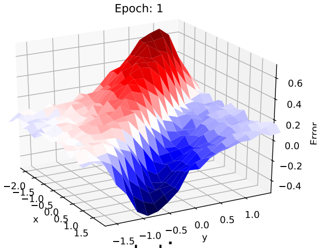
Initial guess, SSR $= 18.94$



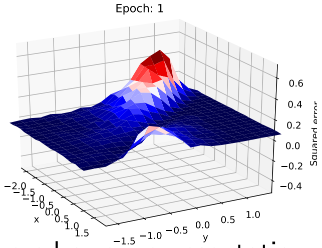
Surface to approximate



Initial approximation

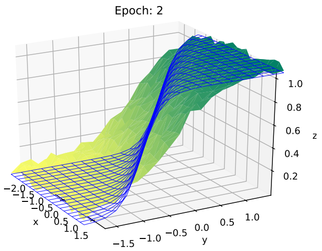


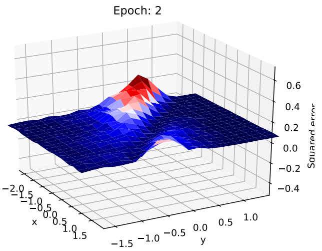
Error representation



Squared error representation

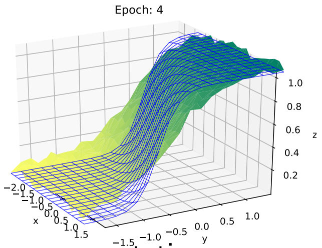
Epoch 2 & 4, SSR $= 12.3/7.8$

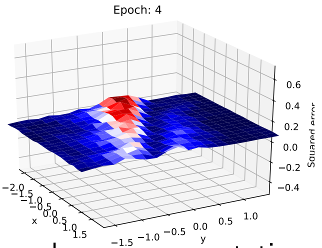




Surface to approximate

Initial approximation

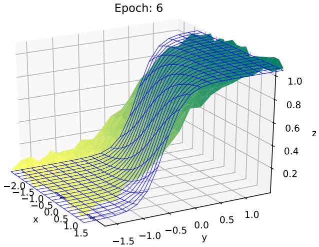


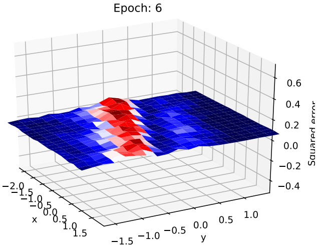


Error representation

Squared error representation

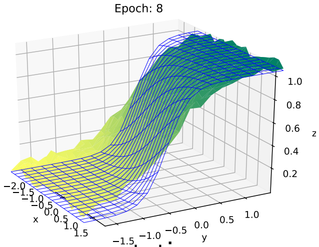
Epoch 6 & 8 , SSR $= 7.12/6.7$

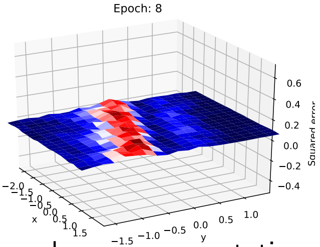




Surface to approximate

Initial approximation

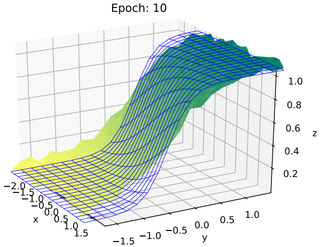




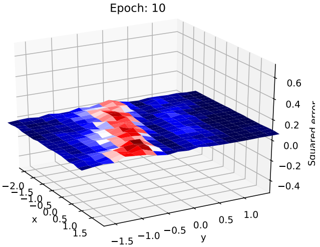
Error representation

Squared error representation

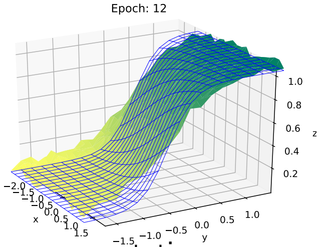
Epoch 10 & 12, SSR $= 6.08/5.39$



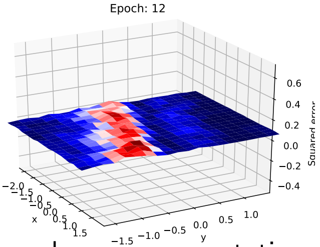
Surface to approximate



Initial approximation

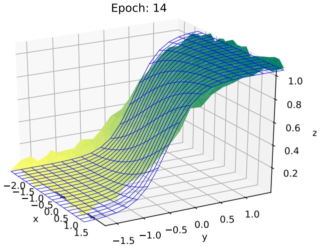


Error representation

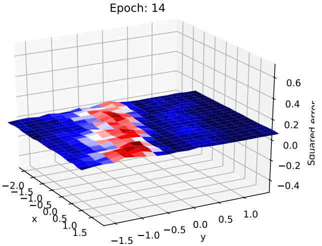


Squared error representation

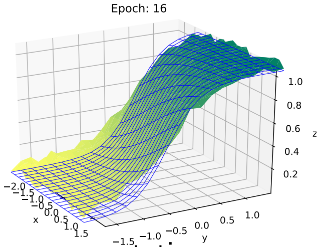
Epoch 14 & 16, SSR $= 4.59/3.71$



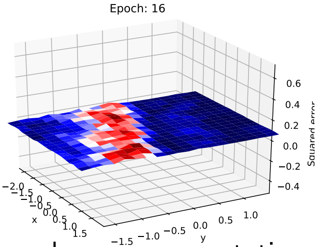
Surface to approximate



Initial approximation

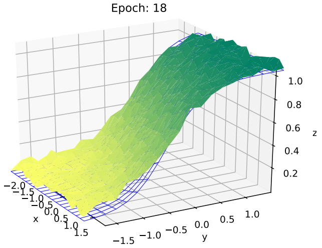


Error representation

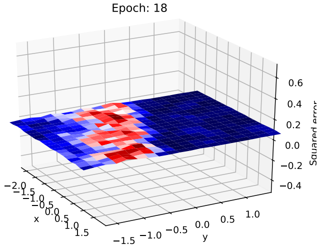


Squared error representation

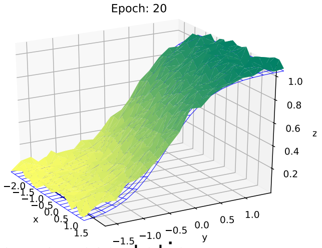
Epoch 18 & 20, SSR $= 2.83/2.08$



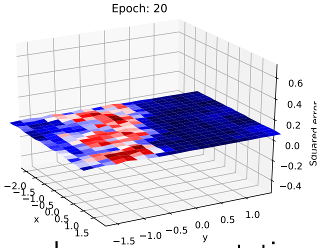
Surface to approximate



Initial approximation

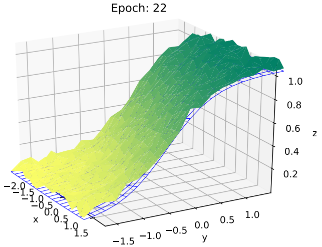


Error representation

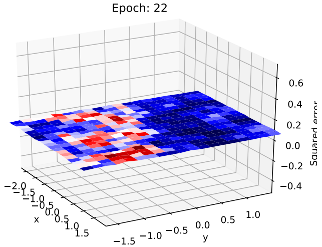


Squared error representation

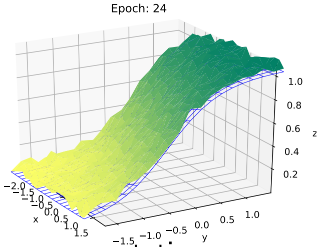
Epoch 22 & 24, SSR $= 1.61/1.45$



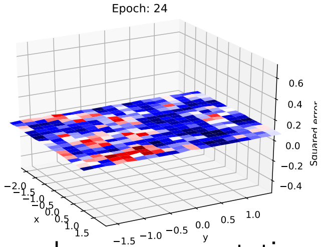
Surface to approximate



Initial approximation

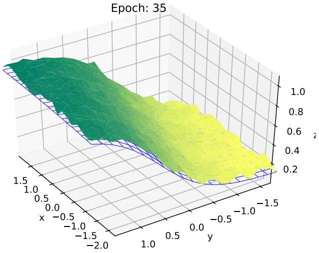


Error representation

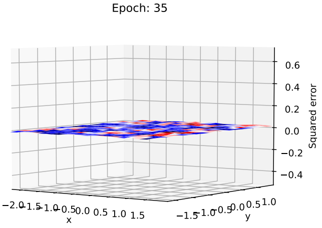


Squared error representation

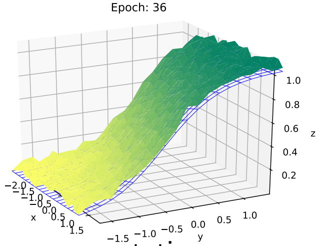
Epoch 35 & 36, SSR $= 1.41592182/1.41592116$



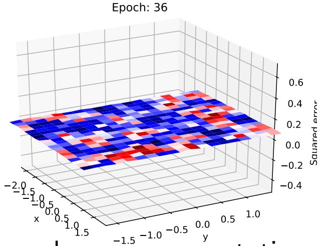
Surface to approximate



Initial approximation



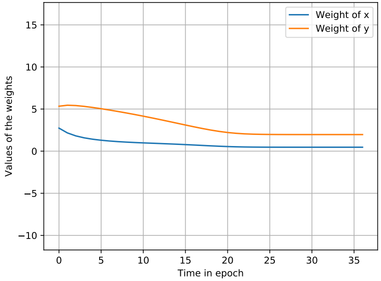
Error representation



Squared error representation

Final result

- Sum of squares of the residuals 1.41592116 at last iteration.
- Weights at last iteration $w_1 = 0.4787443$ and $w_2 = 1.99086222$.
- Could we build a better approximation?



Part II

- Committee learning.

Committee learning

- Some times referred as ensemble learning.
- The idea is to combine a number of weak (accuracy is slightly larger than of random guessing) classifiers into a powerful committee.
- Motivation is to improve estimate by reducing variance and sometimes bias.
- Bootstrap is the technique used to assess the model quality. It requires one to draw data sets with replacement from the training data. The samples are independent.

Bagging

- Induced from the bootstrap technique (which is used to assess accuracy of estimate).
- Draw B samples with replacements and train the model on each sample.
- The bagging estimate then is defined by:

$$\hat{f}_{\mathrm{bag}}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}^{*b}(x).$$

Boosting

- The final prediction is given by:

$$G(x) = \operatorname{sign}\Big(\sum_{m=1}^{M} \alpha_m G_m(x)\Big).$$

  which is weighted majority vote of classifiers $G_m(x)$. Here α_m are weights describing contribution of each classifier.
- While on the first view result is very similar to the bagging, there are some major differences.
- Two class problem where output variable coded as $Y \in \{-1, 1\}$.
- For the classifier $G(X)$ error rate is given by:

$$\overline{\mathrm{err}} = \frac{1}{N} \sum_{i=1}^{N} I(y_i \neq G(x_i)),$$

  where N is the power of training data set.

Ada Boost

AdaBoost.M1. by Freund and Shcapire (1997).

- Initialize observation weights $w_i = 1/N$, $i = 1, \dots, N$.
- For $m = 1$ to M:
  - Fit weak classifier G_m that minimizes the weighted sum error for misclassified points.

    $$\epsilon_m = \sum_{i=1, G_m(x_i) \neq y_i}^{N} w_i$$

  - Compute $\alpha_m = \log((1 - \mathrm{err}_m)/\mathrm{err}_m)$.
  - Update weights w_i as

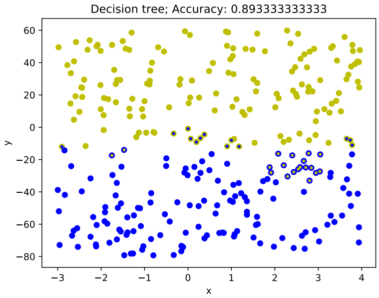
    $$w_i = w_i * \exp(\alpha_m * I(y_i \neq G_m(x_i))), \quad i = 1, \dots, N.$$

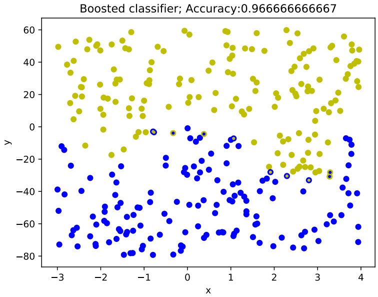
- Output classifier:

$$G(x) = \mathrm{sign}\Big(\sum_{m=1}^{M} \alpha_m G_m(x)\Big).$$

Single tree *vs* Boosted trees

The case when classes are linearly separated is rather rare.





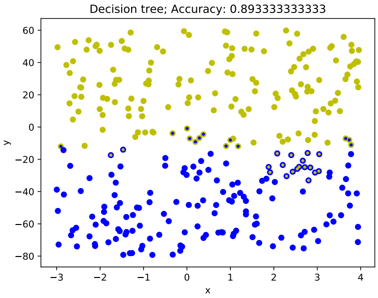
Random Forests

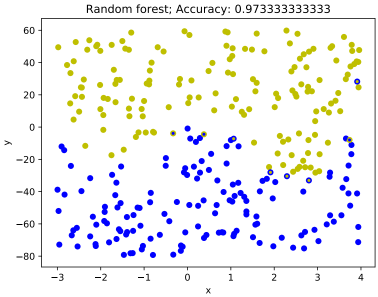
The idea is to build large collection of de-correlated trees, and then average them.

- For $b = 1$ to B:
  - Draw a bootstrap sample Z^* of size N from the available training data.
  - Grow tree T_b. Repeat recursively for each terminal node until minimum node size is reached.
    - Select m variables from p.
    - Pick the best variable among m.
    - Split the node.
- Output the ensemble of trees $\{T_b\}_1^B$.
- Prediction:
  - Regression: $\hat{f}_{\mathrm{rf}}^B(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$.
  - Classification: $\hat{C}_{\mathrm{rf}}^B(x) = \mathrm{mode}\{\hat{C}_b(x)\}_1^B$.

Single tree *vs* Random forest

The case when classes are linearly separated is rather rare.


Decision tree; Accuracy: 0.893333333333
y
x


Random forest; Accuracy: 0.973333333333
y
x