Principles of Machine Learning: Session 2

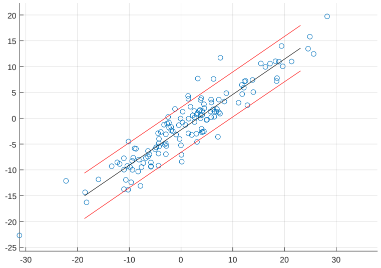
Methusalem Colloquium Mini-Course

S. Nõmm

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

08.12.2022

Linear regression: probably the oldest machine learning technique



- Find linear correlation coefficient.
- Compute coefficients of the linear equation

$$\hat{y} = ax + b$$

- Evaluate the model

- In multivariate case it is required to identify coefficients of the model

$$\hat{y} = a_1x_1 + a_2x_2 + \ldots + a_nx_n + b.$$

This leads the necessity to choose variables (perform model building).

MLE for regression least squares I

- Linear regression is the model of the form

$$p(y|x,\theta) = \mathcal{N}(y|\beta^T x, \sigma^2)$$

  where β are the coefficients of the linear model, σ is the standard deviation of x and $\theta = (\beta, \sigma^2)$
- Parameter estimation of a statistical model is usually performed by computing MLE $\hat{\theta} = \arg\max_\theta \log p(\mathcal{D}|\theta)$. remind that $\mathcal{D}$ denotes the data set

Regularization

- Overfitting may be caused by the fact that chosen model structure and data are not conform on another.
- Regularization is the technique used to overcome overfitting.
- Regularization imposes cost or penalty on the cost function and prevent larger values of the coefficients.
- Loosely speaking, regularization shrinks the coefficients towards zero and towards one another.

Logistic regression

- Linear regression may be written in the following form:

$$p(y|x,\theta) = \mathcal{N}(y|\mu(x), \sigma^2(x))$$

- This may be generalized to the binary setting as follows:

$$p(y|x,\theta) = \mathrm{Ber}(y|\mathrm{sigm}(\theta^T x))$$

  where $\mathrm{sigm}(\eta) = (1 + e^{-\eta})^{-1}$. Will be referred as *logistic regression*.
- Fitting is usually done by maximum likelihood

$$\ell(\theta) = \sum_{i=1}^{N} \log p_(g_i)(x_i|\theta) = \sum_{i=1}^{N} \left\{ y_i \beta^T x_i - \log(1 + e^{\beta^T x_i}) \right\}$$

- Solving the last one is done by means of iterative algorithm.

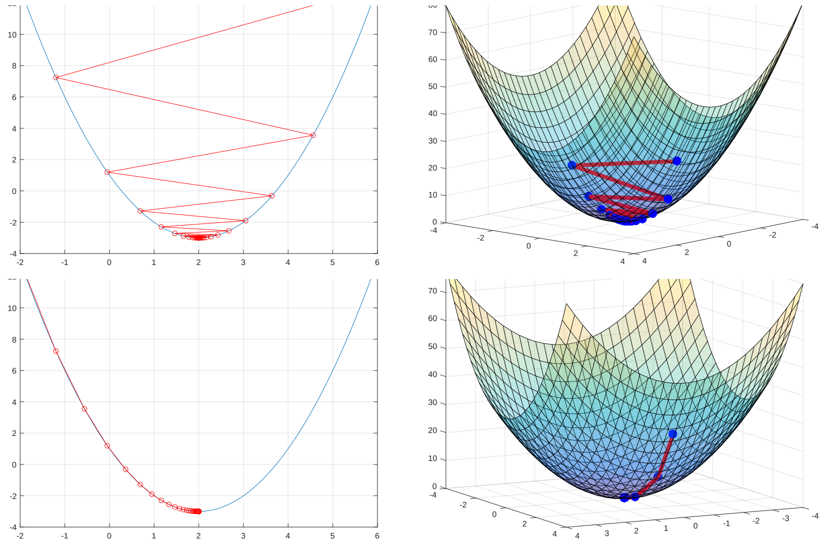
$$\begin{aligned} b^{\text{new}} &= \arg\min_b (z - Xb)^T W (z - Xb) \\ z &= Xb + W^{-1}(y - p) \end{aligned}$$

  where W is a $N \times N$ diagonal matrix with ith element $p(x_i, |b)(1 - p(x_i|b))$

Gradient descent

- Is the first order optimization algorithm.
- Requires to compute first order derivatives.
- Basic idea: If multi-variable function $f(x)$ is defined and differentiable in a neighbourhood of point x_0, then fastest decrease of $f(x)$ is in the direction of the negative gradient $\nabla f(x)$ from the point x_0.
- Formally: Let $x_{k+1} = x_k - \eta_k \nabla f(x_k)$ then $f(x_0) \geq f(x_1) \geq f(x_2) \geq \ldots$. Where η_k is the learning rate. In this form it is referred ad steepest descent.
- This method allows to find local minima of the function.
- In the case of convex function local minima are also global.
- Observe the fact that here we talk about finding minimum of a function in general. (Notation is chosen accordingly). In the case of objective function f is the objective function and its argument are the parameters θ.

Examples



Learning rate (step size)

- "Zig-zagging" effect observed in first row (previous slide).
- Choose η to be small enough. Lead slow convergence. (May not reach minimum at all.)
- Line search: Find η to minimize

  $$\phi(\eta) = f(x_k + \eta \mathrm{d}_k)$$

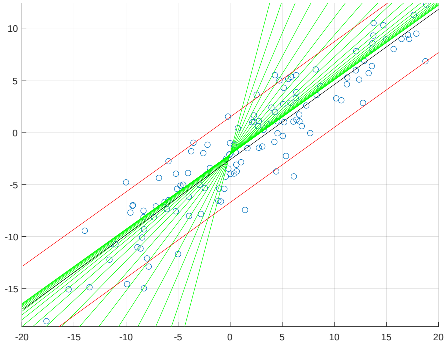
  where d is the descent direction.
- Heavy ball method. Add a *momentum term*:

  $$x_{k+1} = x_k - \eta_k \nabla f(x_k) + \mu_k (x_k - x_{k-1})$$

  where $0 \leq \mu_k \leq 1$

What actually happening with your model

Please be reminded that on each iteration you are updating the weights of your model.



As a result of each iteration one have a model (values of the parameters) (not accurate) of the modelled process.

Online learning

- Suppose that at each step a sample z_k is presented and "learner" is expected to respond by giving parameter estimate θ_k.
- In the case of on-line learning objective is the *regret*.

$$\mathcal{P}_k = \frac{1}{k}\sum_{t=1}^{k} f(\theta_t, z_t) - \min_{\theta^*\in\Theta} \frac{1}{k}\sum_{t=1}^{k} f(\theta^*, z_t)$$

- Loss function in this case $f(\theta, z) = -\theta^T z$ and *regret* is how much batter/worst one did by adopting different strategy.
- Online gradient descent:

$$\begin{aligned} \theta_{k+1} &= \mathrm{proj}_{\Theta}(\theta_k - \eta_k g_k) \\ \mathrm{proj}_{\mathcal{V}}(v) &= \arg\min_{\mathrm{w}\in\mathrm{V}} ||\mathrm{w} - \mathrm{v}||_2 \\ g_k &= \nabla f(\theta_k, z_k) \end{aligned}$$

Stochastic optimization and risk minimization

- Contrary to minimizing regret one may minimize expected loss. $f(\theta) = \mathbb{E}[f(\theta, z)]$.
- One have to optimize functions where some variables in the objective are random.
- Stochastic gradient descent:

$$\bar{\theta} = \frac{1}{k}\sum_{t=1}^{k}\theta_t$$

whereas recursive implementation is as follows:

$$\bar{\theta}_k = \bar{\theta}_{k-1} - \frac{1}{k}(\bar{\theta}_{k-1} - \theta_k)$$

The perceptron

- Fitting logistic regression model in an online manner.
- The weight update rule is given by:

$$\theta_k = \theta_{k-1} - \eta_k g_i = \theta_{k-1} - \eta_k(\mu_i - y_i)x_i$$

where $\mu_i = p(y_i = 1|x_i, \theta_k) = \mathbb{E}[y_i|x_i, \theta_k]$

- Consider an approximation of this algorithm, denote most probable class label as

$$\hat{y} = \arg\max p(t|x_i, \theta)$$

- The update rule may be simplified as $\theta_k = \theta_{k-1} + \eta_k y_i x_i$.
- The gradient expression $g_i = x_i(\theta_k^T x_i - y_i)$ becomes approximate:

$$g_i \approx (\hat{y}_i - y_i)x_i$$

- Assume that $y \in \{-1, 1\}$ not just 0 or 1 then prediction becomes

$$\hat{y}_i = \mathrm{sign}(\theta^T, x_i)$$

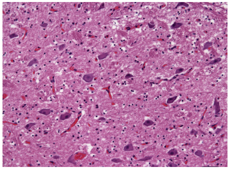
- Final update rule is given by

$$\theta_k = \theta_{k-1} + \eta_k y_i x_i$$

What is *Artificial Neural Network* ?

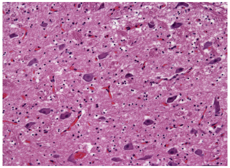
Biology inspired mathematical abstraction

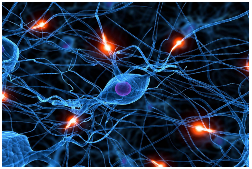
What is *Artificial Neural Network* ?



Biology inspired mathematical abstraction

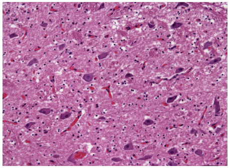
What is *Artificial Neural Network* ?

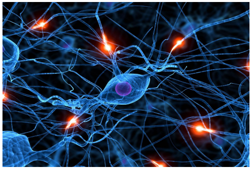




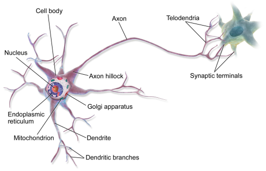
Biology inspired mathematical abstraction

What is *Artificial Neural Network* ?

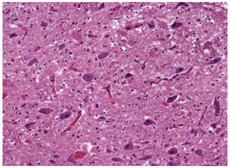


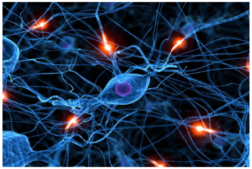


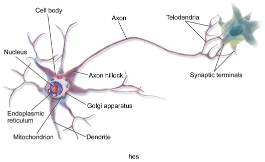
Biology inspired mathematical abstraction



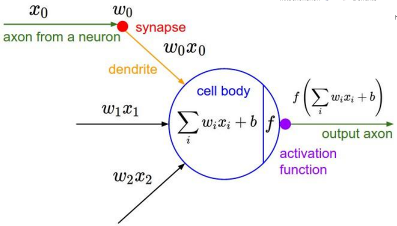
What is *Artificial Neural Network* ?



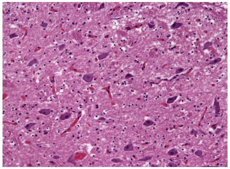


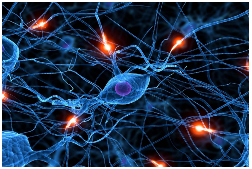


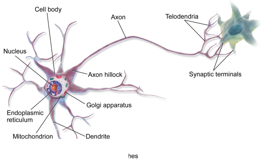
Biology inspired mathematical abstraction



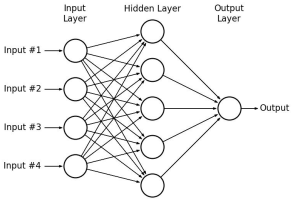
What is *Artificial Neural Network* ?

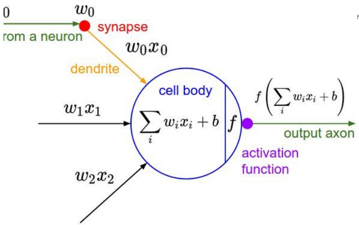






Biology inspired mathematical abstraction





Artificial Neural Network

- Brain consists of neurons.
- Basic idea of the artificial neural network is to combine models of the single neurons.
- Mathematical model of the single neuron is called *perceptron*.
- Perceptron has a number of drawbacks motivating the idea of artificial neural networks.

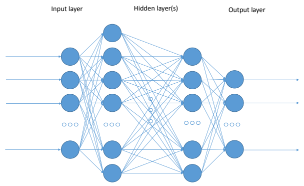
The model of a single neuron

$$y = f\Big(\sum_{j=1}^{d} w_j x_j\Big) = f\big(w^T x\big)$$

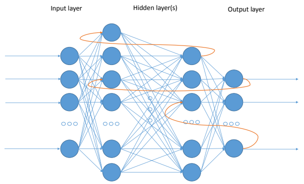
- d is the dimension of the input vector and number of *weights* w_j.
- $x \in \mathbb{R}^d$ - *input vector*.
- $f(\cdot)$ is a smooth nonlinear function referred as *activation function*. Most popular activation functions are:
  - Log-Sigmoid function $logsig(x) = 1/(1 + e^{-x})$.
  - Tan-Sigmoid function $tansig(x) = 2/(1 + e^{(-2*n)}) - 1$
  - Linear transfer function
- In case of Log -Sigmoid activation function the model of the neuron takes form:

$$y = \frac{1}{1 + e^{-\sum_{j=1}^{d} w_j x_j}}$$

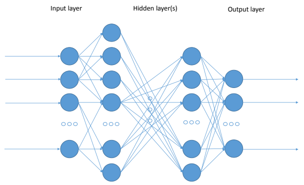
Different topologies of the ANN



Fully connected feed forward neural network



Recurrent neural network



Restricted connectivity feed forward neural network

ANN

- How to choose optimal topology of the net? Nr of layers? Nr of neurons on hidden layer etc?
- Universal approximation theorem:
  - G.Cybenko (1989) Single hidden layer with finite number of neurons (multilayer perceptron) can approximate continuous functions on a compact subsets of $\mathbb{R}^n$. Some weak assumptions about activation function were made. Algorithmic aspects were not touched.
  - K.Hornik (1991) Demonstrated importance of the choice of architecture over the choice of activation function.

Steps specific to the modelling of NN

- Initialization of the network.
- Training algorithms.
  - Levenberg-Marquardt backpropagation.
  - Bayesian regularization backpropagation.
  - Scaled conjugate gradient backpropagation.
  - Resilent backpropagation.
- Stopping criteria.
- Epoch - is the measure of the number of times all the training vectors are used to update the weights.

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

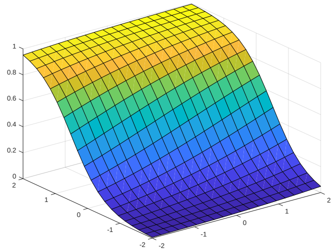
Neuron

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:

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

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

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

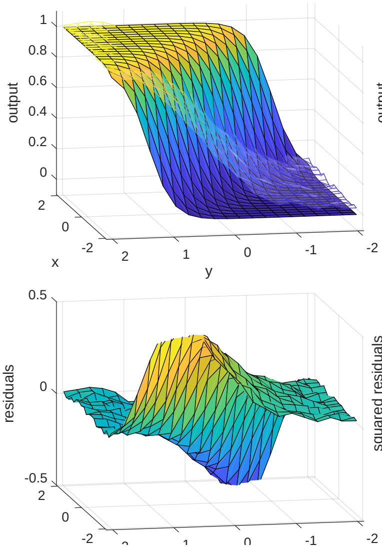
  Elements of the second column:

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

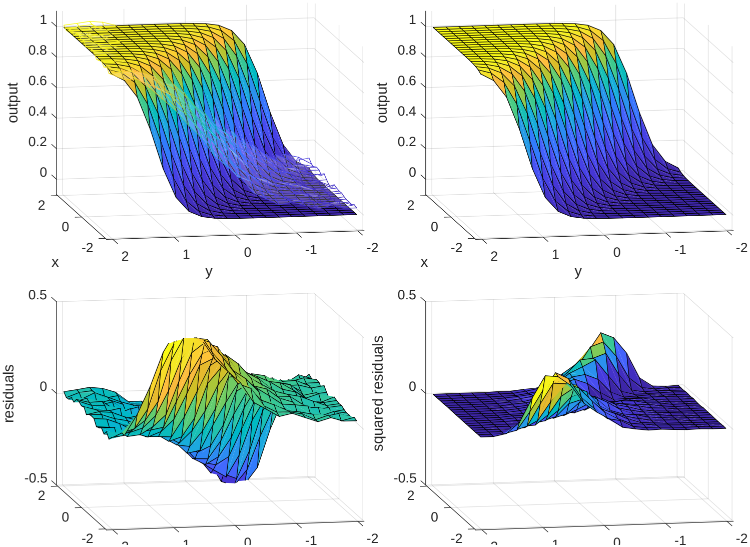
Neuron

- 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

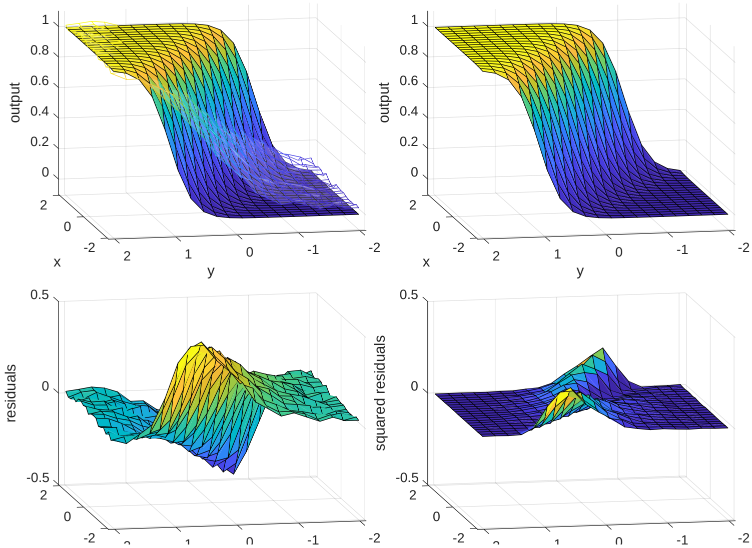
Initial guess, SSR $= 17.51$



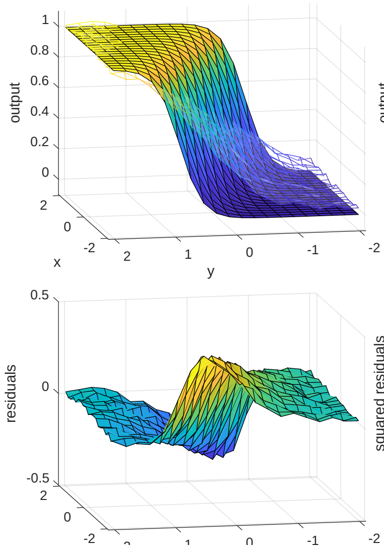
Epoch 2, SSR $= 15.09$



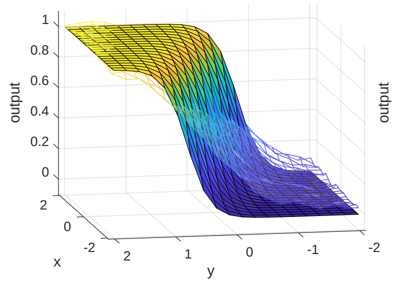
Epoch 4, SSR $= 11.05$

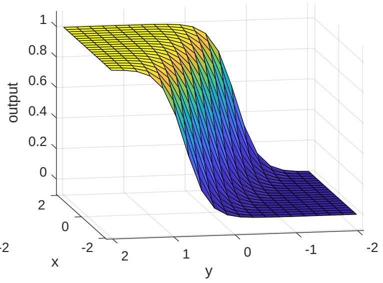


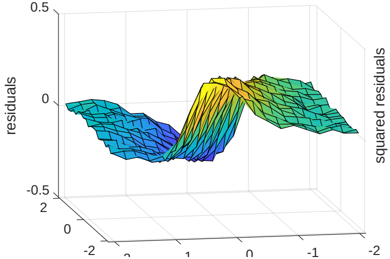
Epoch 6, SSR $= 8.45$

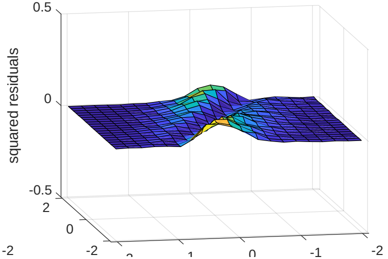


Epoch 8, SSR $= 7.19$

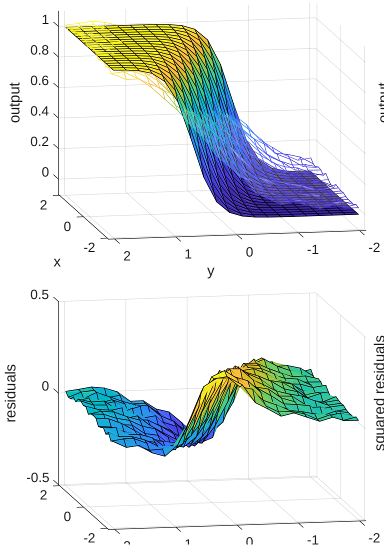




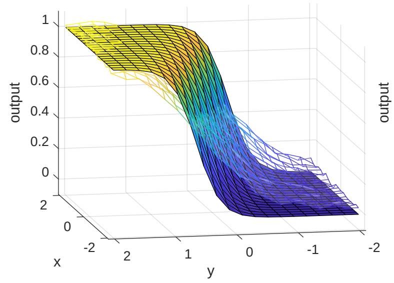


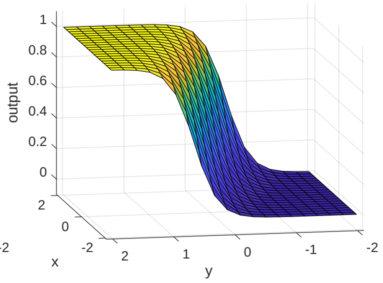


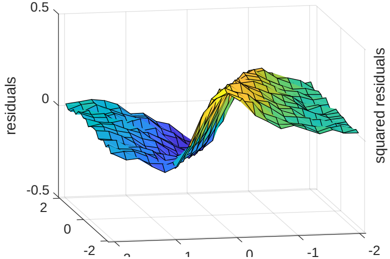
Epoch 10, SSR $= 6.55$

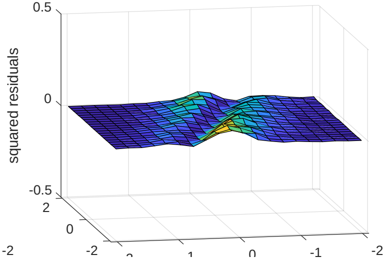


Epoch 12, SSR $= 6.14$

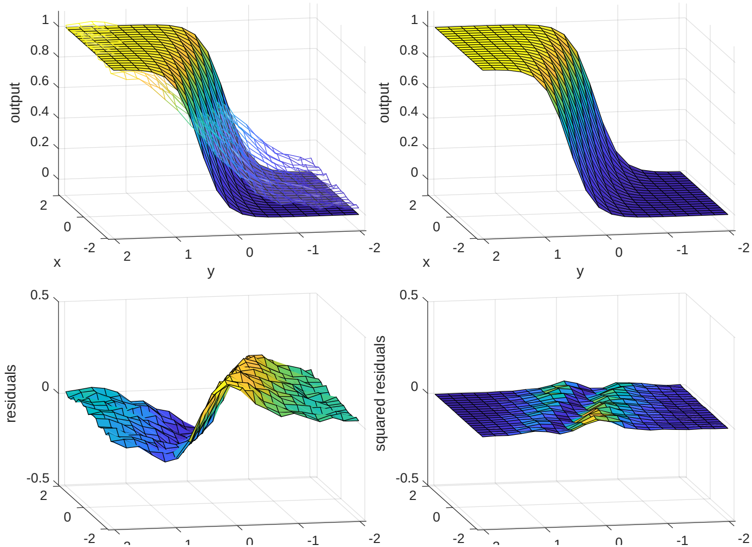




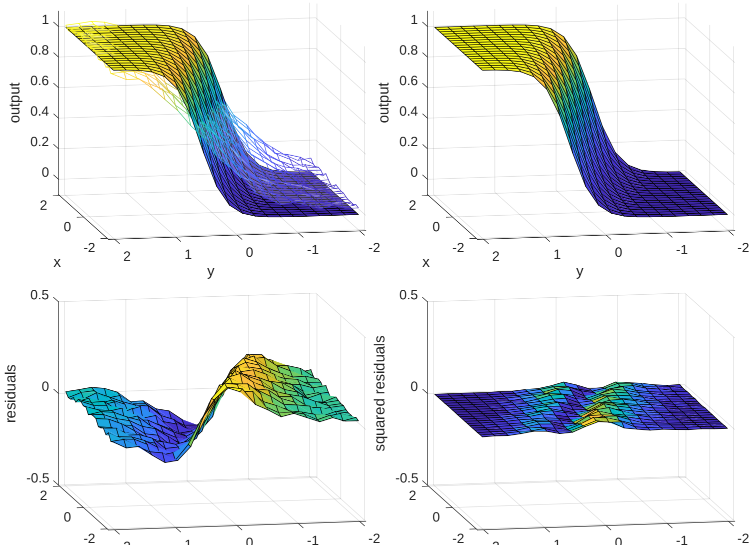




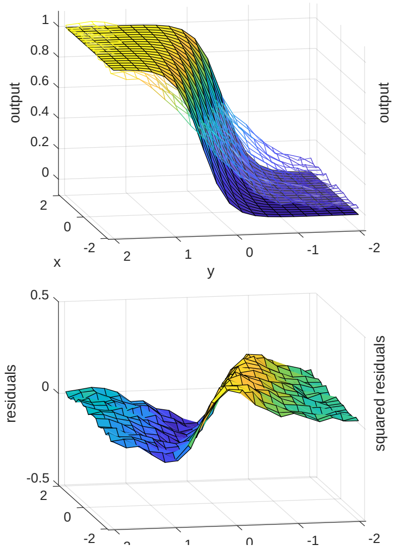
Epoch 14, SSR $= 5.82$

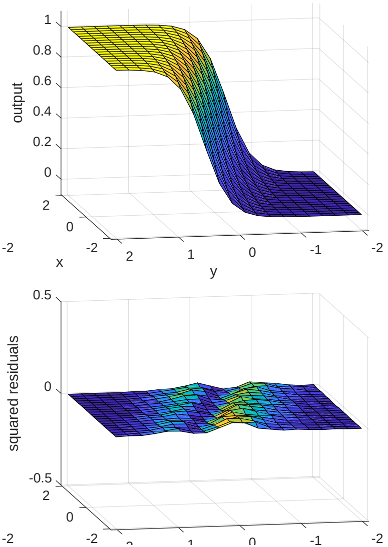


Epoch 16, SSR $= 5.53$

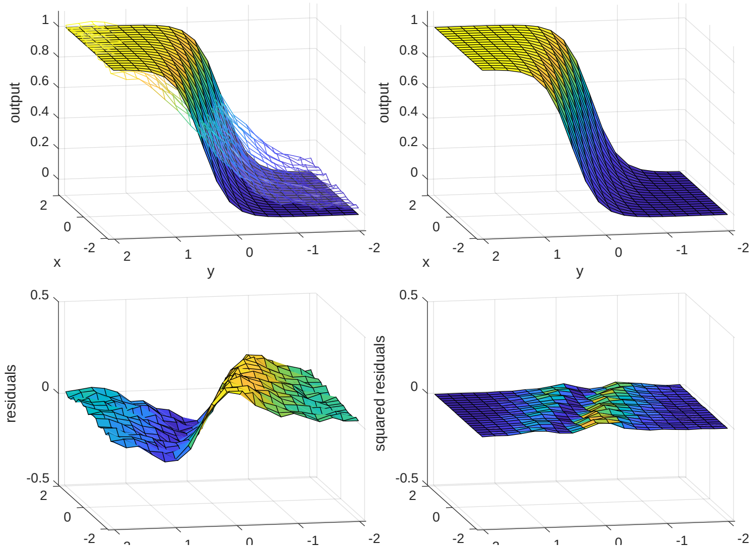


Epoch 18, SSR $= 5.26$

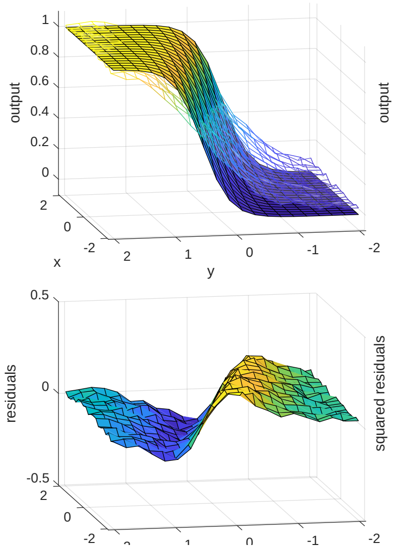


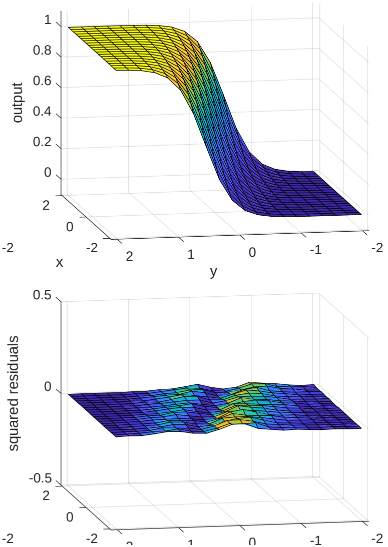


Epoch 20, SSR $= 4.98$

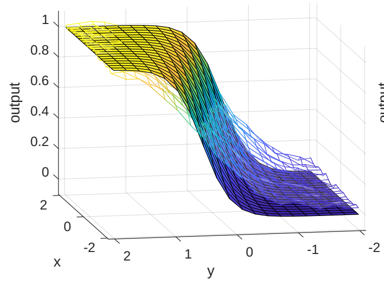


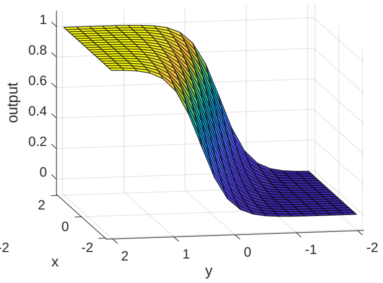
Epoch 22, SSR $= 4.69$

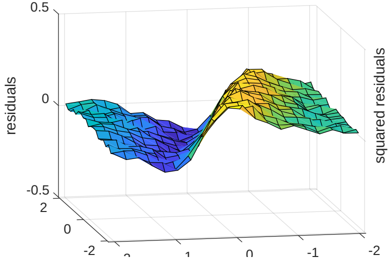


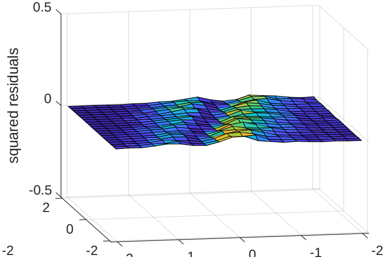


Epoch 24, SSR $= 4.4$

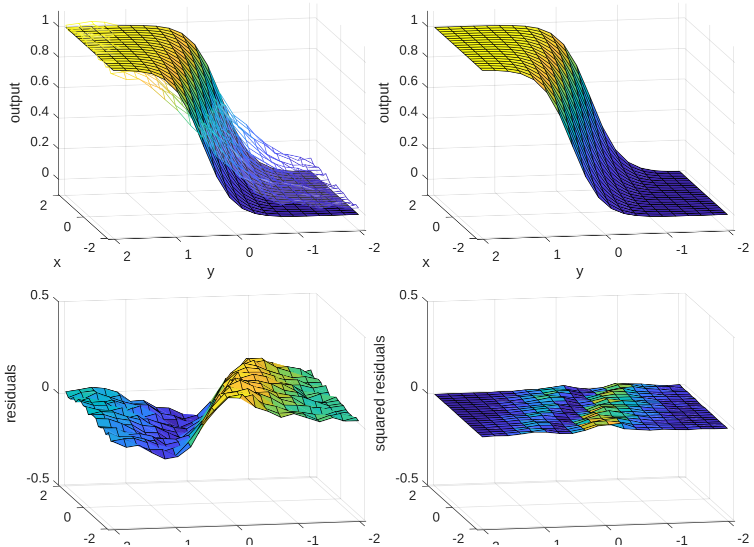




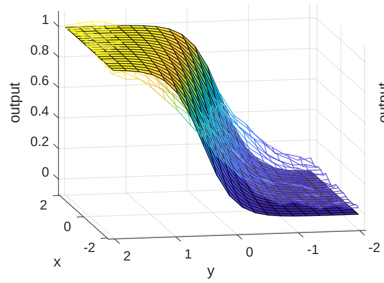


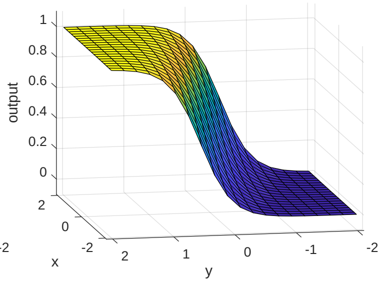


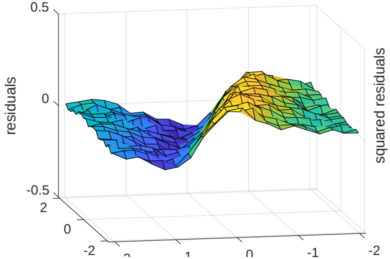
Epoch 26, SSR $= 4.08$

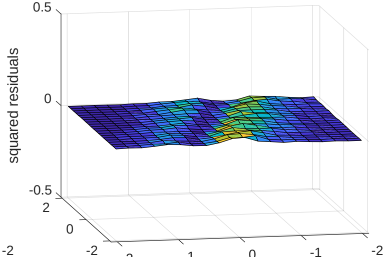


Epoch 28, SSR $= 3.76$

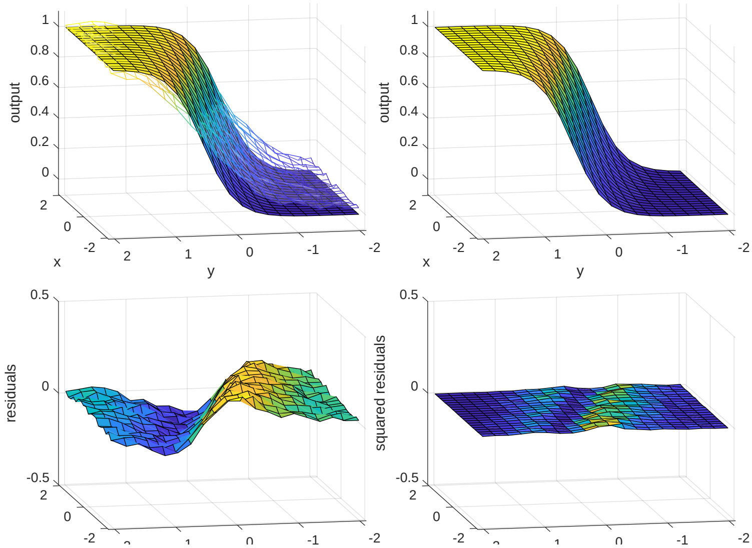




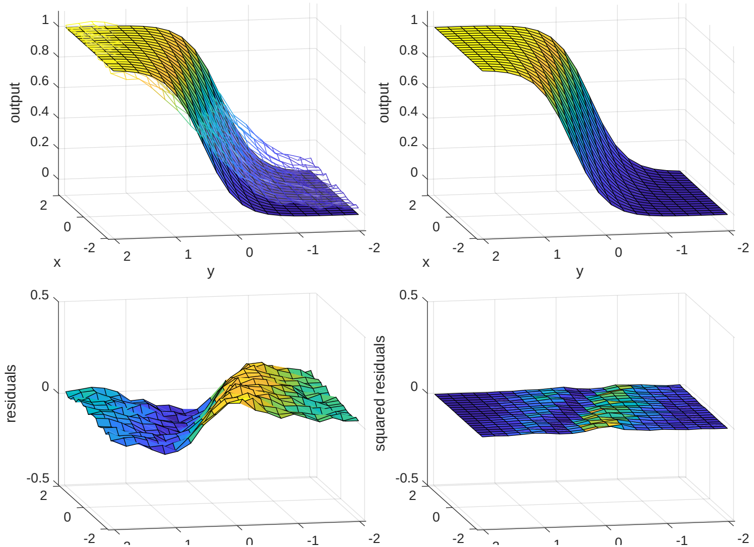




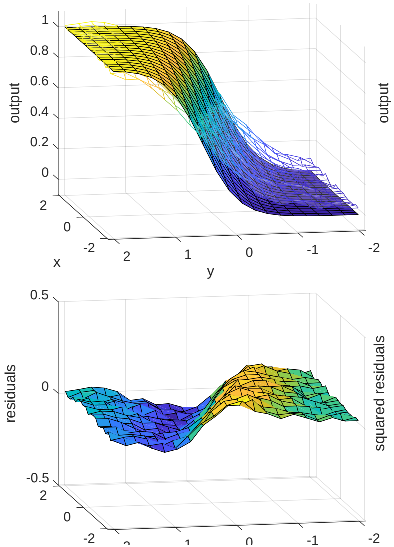
Epoch 30, SSR $= 3.42$

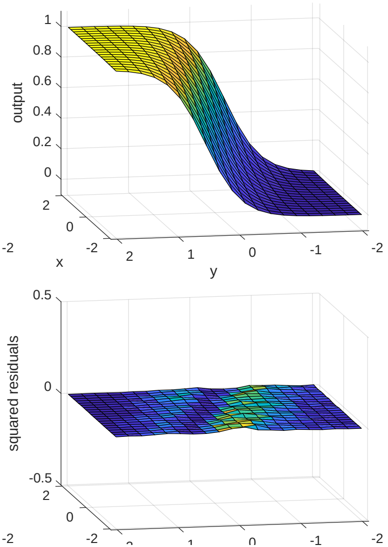


Epoch 32, SSR $= 3.07$

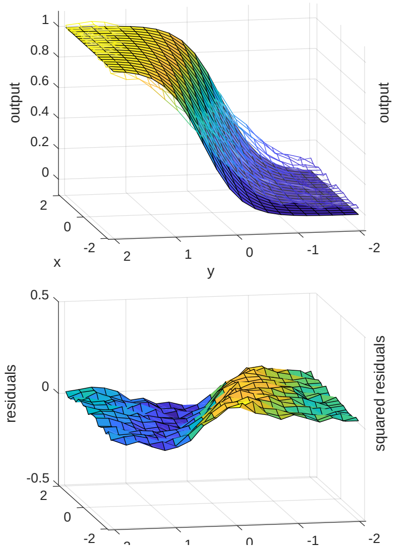


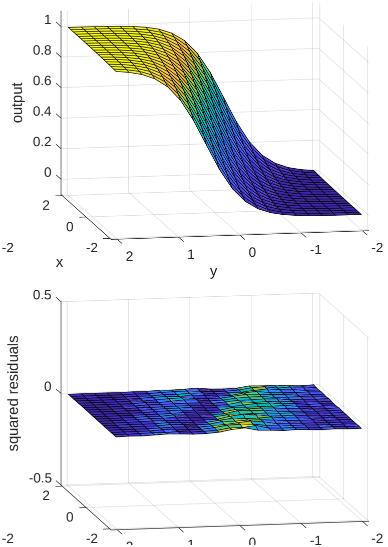
Epoch 34, SSR $= 2.7$



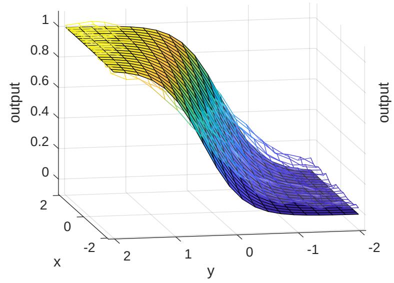


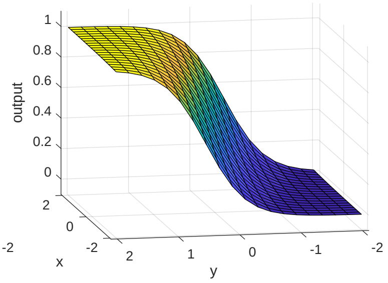
Epoch 36, SSR $= 2.33$

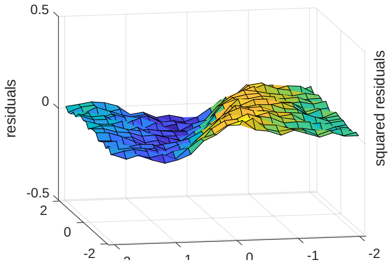


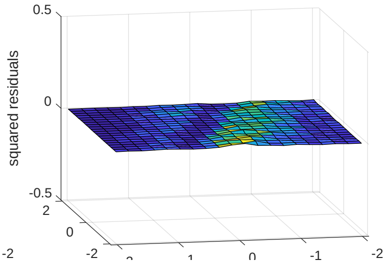


Epoch 38, SSR $= 1.97$

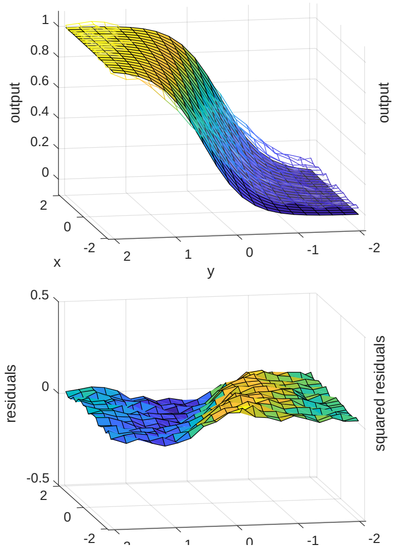




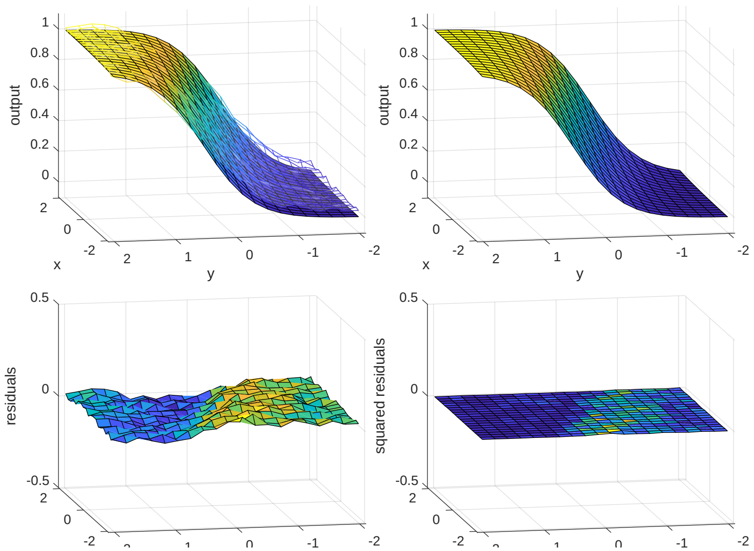




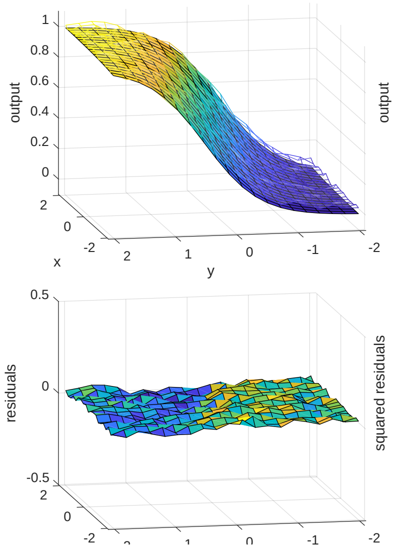
Epoch 40, SSR $= 1.61$

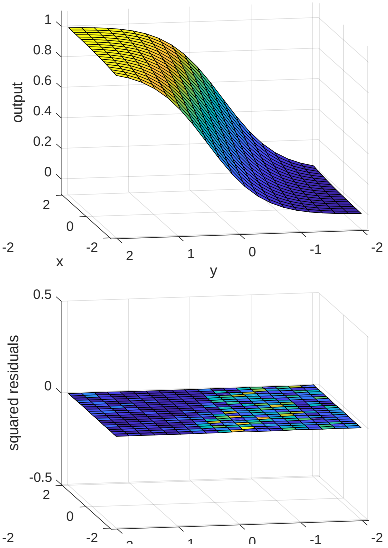


Epoch 45, SSR $= 0.87$

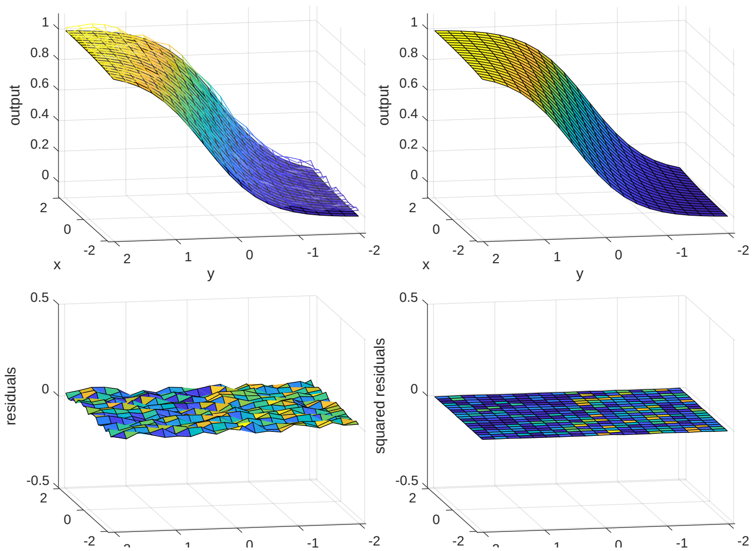


Epoch 50, SSR $= 0.46$

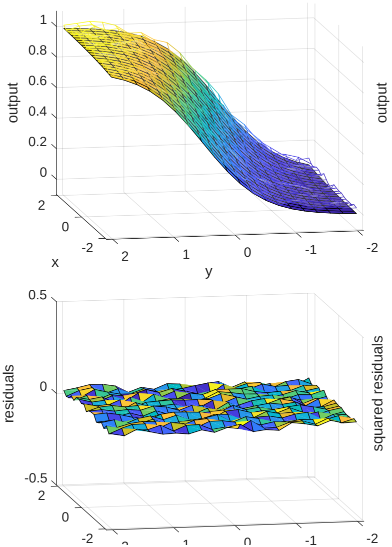


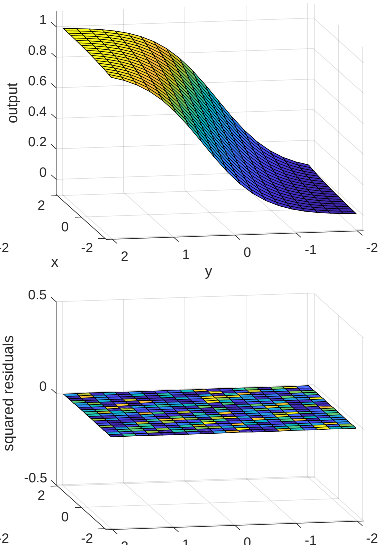


Epoch 55, SSR $= 0.33$

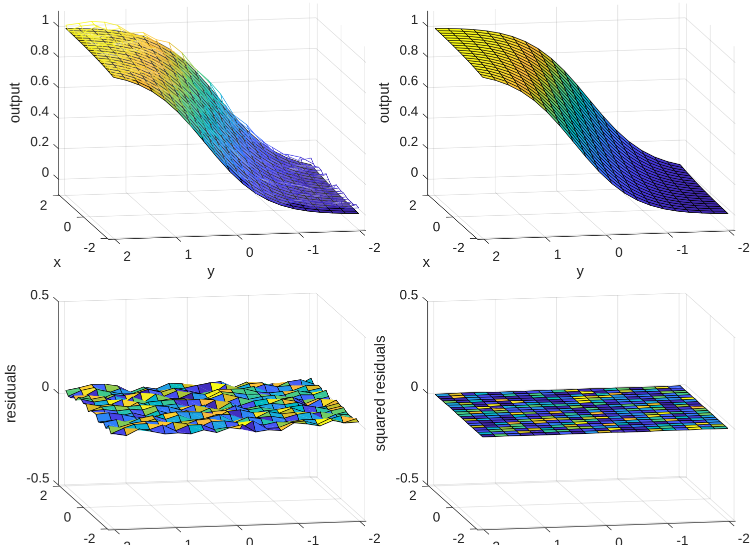


Epoch 60, SSR $= 0.32$



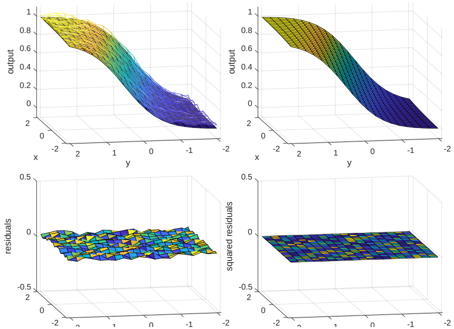


Epoch 65, SSR $= 0.32$



Final result

- Sum of squares of the residuals 1.41592116 at last iteration.
- Weights at last iteration $w_1 = 0.4984$ and $w_2 = 2.00102$.
- Could wone build a better approximation?



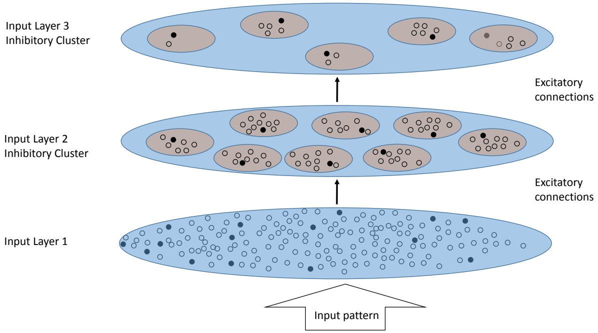
What is *Competitive Learning?*

- Usually considered and implemented with the NN framework.
- Unsupervised learning
- Corrections to the network weights are **not** performed by an external agent.
- The network itself decides what output is best for a given input.
- During the training process, output of the unit or neuron which provides the highest activation to a given input pattern is declared a winner.
- Only winning neuron (some times closest associates) learn.
  - Hard learning (The winner takes it all) - weight of only the winning neuron is updated.
  - Soft learning - weight of the winning neuron and its close associated is updated.
- There is a number of different approaches to the problem: von der Malsburg(1973), Fukushima(1975), Grossberg(1976).

Grossberg(1976) version has the following propoerties:

- Neurons in each layer are split into several non-overlapping clusters.
- Each neuron within the cluster inhibits other neurons of the cluster.
- Within the cluster, the neuron receiving the largest input achieves its maximum value (1) and all the others set to minimum (0)
- Every neuron in each cluster connected (receives signals) from all the input neurons (from all the inputs).
- Neuron learns if an only if it wins the competition within its cluster.
- A stimulus pattern is binary.
- Each neuron has fixed amount of weigh which is distributed among the inputs $\sum \omega_{i,j} = 1$.
- A neuron learns by shifting weight from inactive to active inputs.

The learning rule



The learning rule in such case is as follows:

$$\Delta\omega_{i,j} = \begin{cases} 0 & \text{if } i \text{ loses on stimulus } k \\ \epsilon\dfrac{\delta_{j,k}}{\sigma_k} - \epsilon\omega_{i,j} & \end{cases}$$

where $\delta_{j,k}$ takes the value 1 if stimulus S_k neuron j is active and 0 otherwise, σ_k number of active neurons in stimulus S_k.

Hamming Net + MaxNet

Another approach is to consider competitive network to be composed of two neural networks:

- Hamming net: measures how much the input vector resembles the weight vector of each perceptron.
- Max net: Finds the perception with the maximum value.
- Competitive learning rule:

$$\omega_i(n+1) = \omega_i(n) + \epsilon(x(n) - \omega_i(n))$$

  where i - neuron index, n-instance index, $\omega - i$ x - input, ϵ -learning rate.

Self - organizing maps (Kohonen - maps)

Self organizing map (SOM)

- Proposed by Kohonen in 1982. Key feature is topology preservation.
- Competitive learning model with a neighbourhood constrain on the output neurons.
- Usually 1D or 2D grid to restrict dimensionality and provide better visualization.
- Have a strong neurobiological basis.
- Kohonen SOM is the result from three processes Competition, cooperation, adaptation.

Competition

- Each neuron in a SOM is assigned a weight vector with the same dimensionality N as the input space.
- Any given input pattern is compared to the weight vector of each neuron and the closest neuron is declared the winner.
- The Euclidean norm is commonly used as the distance measure.

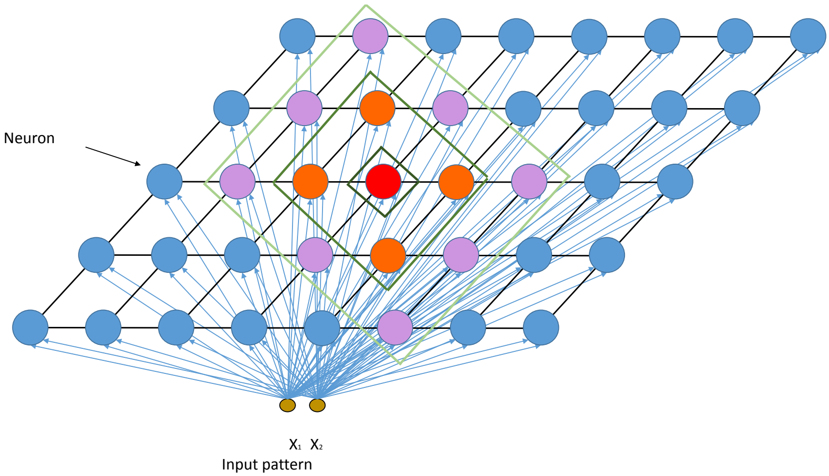
Cooperation

- In order to allow topologically close neurons to become sensitive to similar patterns the activation of the winning neuron is spread to neurons in its immediate neighborhood.
- Neighbourhood of the wining neuron is determined on the basis of topology with city-block (Manhattan or taxi driver) distance.
- The size of the neighbourhood is changes from large to small. Large neighbourhood in the beginning allows to preserve the topology. Smaller neighbourhood on the later stages allows to specialization of the neurons.

ANN

- During the training, winning neuron and its topological neighbors are adapted to make their weight vectors more similar to the input pattern that caused the activation.
- Neurons that are closer to the winner will adapt more than neurons that are further away.
- The magnitude of the adaptation is controlled with a learning rate, which decays over time to ensure convergence of the SOM.

Kohonen SOM



Kohonen SOM algorithm

- A learning rate decay rule

$$\epsilon(t) = \epsilon - \epsilon_0 e^{-t/\tau_1}$$

- A neighborhood kernel function

$$h_{ik}(t) = e^{-\frac{d_{ik}^2}{2\sigma^2(t)}}$$

  where d_{ik} distance in the grid between ω_i and ω_k.

- A neighborhood size decay rule

$$\sigma(t) = \sigma_0 e^{\frac{-t}{\tau_2}}$$

Kohonen SOM algorithm

- Initialize weights
- Repeat until converge
  - Select following pattern from the input sequence
  - Find ω_i that best matches the input pattern ξ^n

$$i = \mathrm{argmin}_j \|\xi^n - \omega_j\|$$

  - Update the weights of the winner ω_k

$$\omega_k = \omega_k + \epsilon(t) h_{ik}(t)(\xi^n - \omega k)$$

  - decrease learning rate $\epsilon(t)$
  - Decrease neighbourhood size $\sigma(t)$