Neuronale Netze (SS 2002), 3.6., 5.6., 10.6.

Efficiency of training

- For gradient descent methods: one cycle needs $\sim WP$ time steps
- For local optima with first derivatives 0 and a positive definite matrix of second derivatives, convergence to the local optimum is exponentially fast if the learning rate is small enough and one starts sufficiently close at the optimum.
- In general, it is NP-complete to decide whether even for a single sigmoidal neuron, a given training set and a number k, weights can be found such that the quadratic error is smaller than k.
- There exists a bunch of NP-results mostly with proofs which are not really nice ;)

Tricks to avoid the complexity and overfitting

- start with large networks and appropriately preprocessed data to avoid local optima.
- Early stopping: stop when error on a validation set is minimum.
- Pruning: iteratively drop least important weight/neuron until the error severlely insreases
  - magpruning: drop smallest weight
  - sceletonization: drop neuron with minimum influence on the error
  - optimum brain damage: drop weight with minimum influence on the error

Interpretation of training:

- rescaling of outputs/interpretation as a classification
- compare to simple models, known results, different methods

- in principle: error divides into the systematic error which should be small and the unsystematic error due to noise in the data which constitutes the lower bound for every training algorithm

The training pipeline revisited:

- Preprocessing – tedious and crucial
- architecture/model selection, important: approximation capability
- training as error minimization, important: efficient training algorithms
- interpretation of learning results, important: generalization ability

Alternative: Support Vector Machine (SVM)

- General idea:
  - fixed nonlinear preprocessing of data followed by a trainable linear separation
  - nonlinear preprocessing given by kernels for efficiency
  - linear separation with maximum margin for generalization ability
- data $(\vec{x}_i, \vec{y}_i) \in \mathbb{R}^n \times \{-1, 1\}$ are to be learned

  fix a nonlinear preprocessing via $\Phi(\vec{x})$ with $\Phi(\vec{x})^t\Phi(\vec{y}) = k(\vec{x}, \vec{y})$ for a kernel k

  typical kernels:
  polynomial kernel $(\vec{x}^t\vec{y} + 1)^d$
  Gaussian kernel $\exp(-|\vec{x} - \vec{y}|^2/\sigma^2)$
- primary problem:

  minimize $0.5 \cdot |\vec{w}|^2 s.t.\ y_i\vec{w}^t\Phi(\vec{x}_i) - 1 \geq 0$
- dual equivalent problem:

  maximize $\sum \alpha_i - 0.5 \cdot \sum \alpha_i\alpha_j y_i y_j k(\vec{x}_i, \vec{x}_j)$ s.t. $\alpha_i \geq 0$

  in this case: $\vec{w} = \sum \alpha_i y_i \Phi(\vec{x}_i)$,
  the classification is given by $H(\vec{x} \mapsto \sum \alpha_i y_i k(\vec{x}_i, \vec{x}))$

α_i correspond to measures for the importance of the points; only points with minimum margin to the separating hyperplane have nonzero dual variables α_i; these are called **support vectors**; they determine the classification!

- generalizations such that errors are tolerated (soft margin) or functions can be approximated (ϵ-tube) exist

Alternative: Prototype based classifiers

- **Learning Vector Quantization (LVQ)**:

  Classification of data in $\mathbb{R}^n$ is given by a set of labeled prototypes $\vec{w}_i$, a given data point $\vec{x}$ is mapped to the label of the respective closest prototype

  Learning:

  initialize the prototypes
  Repeat
  choose a data point $\vec{x}_i$
  compute the closest prototype $\vec{w}_j$
  adapt $\vec{w}_j += \epsilon(\vec{x}_i - \vec{w}_j)$ if the classes coincide or
  adapt $\vec{w}_j -= \epsilon(\vec{x}_i - \vec{w}_j)$ if the classes do not coincide

- **Dynamic LVQ:**
  Start with one prototype per class and successively add new prototypes if incorrect classifcations exist

- **Relevance LVQ:**
  Substitute the Euclidian metric by a diagonal metric $\|\vec{x}-\vec{y}\|_{\vec{\lambda}} = (\sum \lambda_i (x_i - y_i)^2)^{0.5}$ with nonnegative λ_i with $\sum \lambda_i = 1$

  λ_l are simultaneously adapted via:
  $\lambda_l -= \epsilon'((\vec{x}_i)_l - (\vec{w}_i)_l)^2$ if the classes coincide or
  $\lambda_l += \epsilon'((\vec{x}_i)_l - (\vec{w}_i)_l)^2$ if the classes do not coincide
  normalize $\vec{\lambda}$

- **Generalized RLVQ:**
  gradient descent on

$$\sum \text{sgd}\left(\frac{d_1(\vec{x}_i) - d_2(\vec{x}_i)}{d_1(\vec{x}_i) + d_2(\vec{x}_i)}\right)$$

where $d_1(\vec{x}_i)$ is the weighted Euclidian distance from the closest correct prototype and $d_2(\vec{x}_i)$ is the weighted Euclidian distance from the closest incorrect prototype.

Yields Hebb updates for the weights and relevance factors with appropriate scaling factors.

Rule extraction (decision trees) based on GRLVQ-networks is approximately possible.