.

Machine Learning

Generalisation in Multilayer perceptrons

Prof. Dr. Martin Riedmiller
AG Maschinelles Lernen und Natürlichsprachliche Systeme
Institut für Informatik
Technische Fakultät
Albert-Ludwigs-Universität Freiburg

Martin.Riedmiller@uos.de

Overview of Today's Lecture: Generalisation in MLPs

1. Motivation

2. Training and Validation

3. Regularisation Techniques

Motivation

Neural networks are very powerful function approximators:
Any boolean function can be realized by a MLP with one hidden layer. Any bounded continuous function can be approximated with arbitrary precision by a MLP with one hidden layer. Cybenko (1989)
Therefore, they incorporate the danger of overfitting.

- At first glance, the size of the network matters: the larger the number of hidden neurons, the more powerful the representation capacity.
- Tradeoff: being able to reasonably learn (avoid underfitting), but not just memorize training data (overfitting)
- Unfortuntately, there is no analytical way to determine the right network size out of the training data.
- couple of techniques are applied, most of which can also be applied to other learning methods. These are discussed in the following.

Examples of Overfitting/ Underfitting

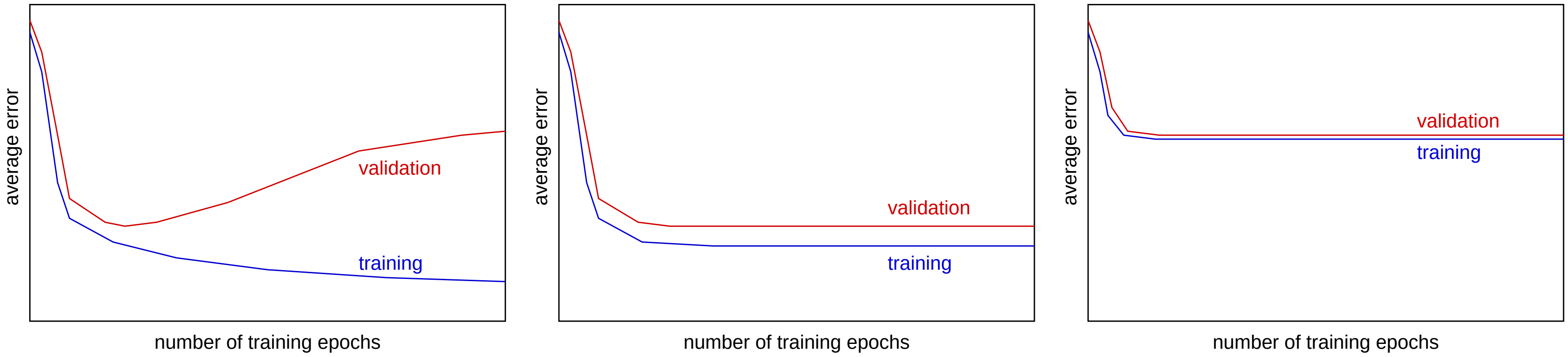
- overfitting/ underfitting in regression (blackboard)
- overfitting/ underfitting in classification (blackboard)

Overfitting/ Underfitting: Training/ Test error behaviour

- example of overfitting:
  validation error increases while training error decreases

- example of successful learning:
  validation error and training error monotonically decrease

- example of underfitting:
  validation and training error remain large



Training and validation

- learning algorithms try to find a hypothesis that fits the training data in the best way
- we would like to find a hypothesis that works well for all data sets that can be derived from the true function f^*
- assume that the input patterns $\vec{x}$ are taken from some probability distribution $\mathcal{P}$
- the best hypothesis f should minimize the expected error (generalization error):

  $$E\Big[\frac{1}{2}\big(f(\vec{x}) - f^*(\vec{x})\big)^2\Big]$$

  (here: E means the expectation value over all possible $\vec{x}$)
- Note: we cannot calculate the expected error since we don't know $\mathcal{P}$ and f^*

Training and validation

- if we are given a set of examples $\{(\vec{x}^{(1)}, d^{(1)}), \dots, (\vec{x}^{(p)}, d^{(p)})\}$ with $\vec{x}^{(i)} \sim \mathcal{P}$ and $d^{(i)} = f^*(\vec{x}^{(i)})$, we can approximate the expected error by the mean error (training error):

$$\frac{1}{p}\sum_{i=1}^{p}\Big(\frac{1}{2}\big(f(\vec{x}^{(i)}) - d^{(i)}\big)^2\Big)$$

- whether or not this approximation is good is discussed in the $Computational\ Learning\ Theory$
- if the approximation is good, a hypothesis learned on the training set will also perform well on other data sets
- if the approximation is bad, a hypothesis learned on the training set will perform poorly on other data sets

Training and validation (cont.)

- Validation is the process to check the performance of a learned function on independent validation/test data.
- simple approach for validation:
  - before training, split the available data set into two disjoint subsets: training set and validation set
  - apply training only on the training set
  - apply testing the learned functions on the validation set
- disadvantage: only a subset of available patterns is used for training, only a subset of available patterns is used for testing

Cross-validation

- k-fold cross validation:

```
Require: $\mathcal{D}$, $2 \le k \le p$
split $\mathcal{D}$ into $k$ disjoint subsets of equal size: $\mathcal{D}_1, \dots, \mathcal{D}_k$
for $i = 1$ to $k$ do
    (re-) initialize neural network
    train neural network on set $\mathcal{D}_1 \cup \dots \cup \mathcal{D}_{i-1} \cup \mathcal{D}_{i+1} \cup \dots \cup \mathcal{D}_k$
    calculate average test error $e_i$ on $\mathcal{D}_i$
end for
return $\frac{1}{k} \sum_{i=1}^{k} e_i$
```

- advantage: model is learned on $\frac{k-1}{k}p$ data points and evaluated on all data points
- disadvantage: model has to be learned k times
- k-fold cross validation with $k = p$ yields the leave-one-out error

Cross-validation

- example: $\mathcal{D} = \{z_1, \ldots, z_{17}\}$, 3-fold crossvalidation
- split training set randomly:

$$\mathcal{D}_1 = \{z_1, z_2, z_8, z_{13}, z_{14}, z_{15}\}$$
$$\mathcal{D}_2 = \{z_4, z_5, z_{10}, z_{11}, z_{12}, z_{17}\}$$
$$\mathcal{D}_3 = \{z_3, z_6, z_7, z_9, z_{16}\}$$

- train MLP three times:

  1. use $\mathcal{D}_2 \cup \mathcal{D}_3$ for training, $\mathcal{D}_1$ for validation: average validation error e_1
  2. use $\mathcal{D}_1 \cup \mathcal{D}_3$ for training, $\mathcal{D}_2$ for validation: average validation error e_2
  3. use $\mathcal{D}_1 \cup \mathcal{D}_2$ for training, $\mathcal{D}_3$ for validation: average validation error e_3

- calculate average validation error: $e = \frac{e_1 \cdot |\mathcal{D}_1| + e_2 \cdot |\mathcal{D}_2| + e_3 \cdot |\mathcal{D}_3|}{|\mathcal{D}|} = \frac{6e_1 + 6e_2 + 5e_3}{17}$

Regularization techniques

- regularization techniques: approaches to improve generalization implementing some preference rules
- four basic principles:
  - smaller networks should be prefered (cf. Ockham's razor)
    the hypothesis set $\mathcal{H}$ of smaller networks is smaller than the hypothesis set of large networks
  - smaller weights should be prefered
    neurons with logistic activation function and small weights are almost linear, networks of linear neurons can be replaced by a single neuron
  - better description of the task
    better input features may simplify the learning task, e.g. make a nonlinear problem linear
  - ensemble techniques
    combine several MLPs after learning

Regularization techniques - Overview

techniques to prefer small weights:

- small initial weights
- early stopping
- weight decay
- Bayesian learning

techniques to reduce network size:

- weight pruning
- topology optimization
- weight sharing
- cascade correlation

techniques for better task description:

- provide more training data
- filter training data
- use more/less/other input features

ensemble techniques:

- bagging
- boosting

Regularization techniques - weight initialisation

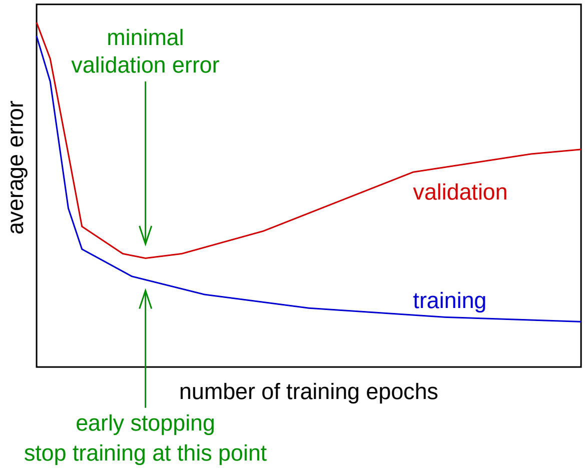
- small initial weights

  initialize all weights with small values. Since initial steplength in learning algorithms is small, the network has larger chance to converge to a local minimum that is characterized by small weights

  Per se not very powerful technique, useful especially when combined with early stopping.

Regularization techniques - Early stopping

early stopping



- stop learning when the error on the validation set has reached its minimum
- heuristic often used, needs perpetual observation of the validation error, typically combined with small initial weights
- often, training is already stopped after a few iterations (10-30 iterations)

Regularization techniques - Weight decay

- weight decay

  explicit preference of small weights. Modify the error term that is minimized during learning:

  $$E(\vec{w}; \mathcal{D}; \lambda) := E(\vec{w}; \mathcal{D}) + \frac{1}{2}\lambda||\vec{w}||^2$$

  $\lambda \geq 0$ controls the importance of the preference: for $\lambda = 0$ we get unregularized learning. Typical values for λ range between 0 and 10

  very powerful technique, often used

  disadvantage: λ must be adjusted manually

  – λ is too small $\Rightarrow$ overfitting
  – λ is too large $\Rightarrow$ underfitting

fits very smoothly into the gradient descent framework

Regularization techniques - Pruning

- weight pruning

  after training a MLP check which connections do not contribute much to the result. Remove these connections and retrain the MLP.

  several approaches:

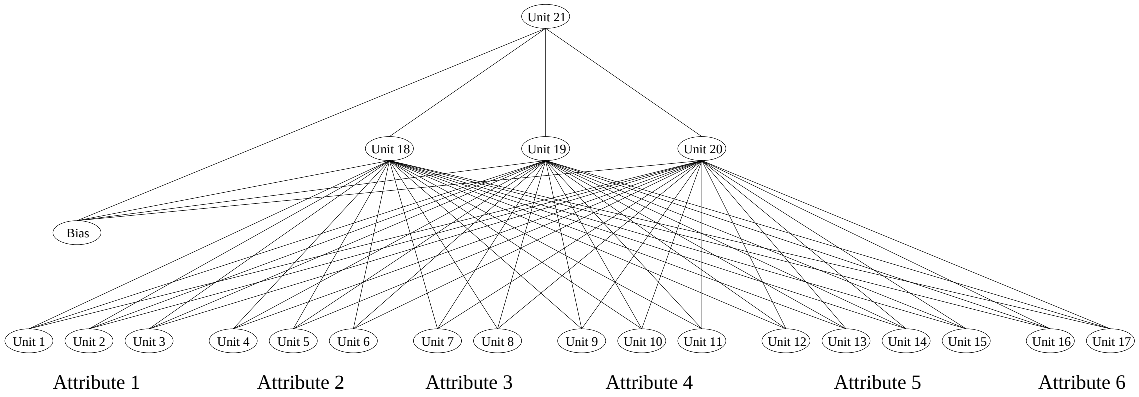
  - remove connections with the smallest weights (absolute value)
  - *optimal brain damage (OBD)* ((LeCun, Denker, Solla 1990)), *optimal brain surgeon (OBS)* ((Hassibi, Storck 1992)): remove weights depending on second order information $\frac{\partial^2 E}{(\partial w_{ij})^2}$. Background: remove weights whose removal has the smallest influence on the network output
  - evolutionary approaches

  approaches are sometimes used, but not as standard

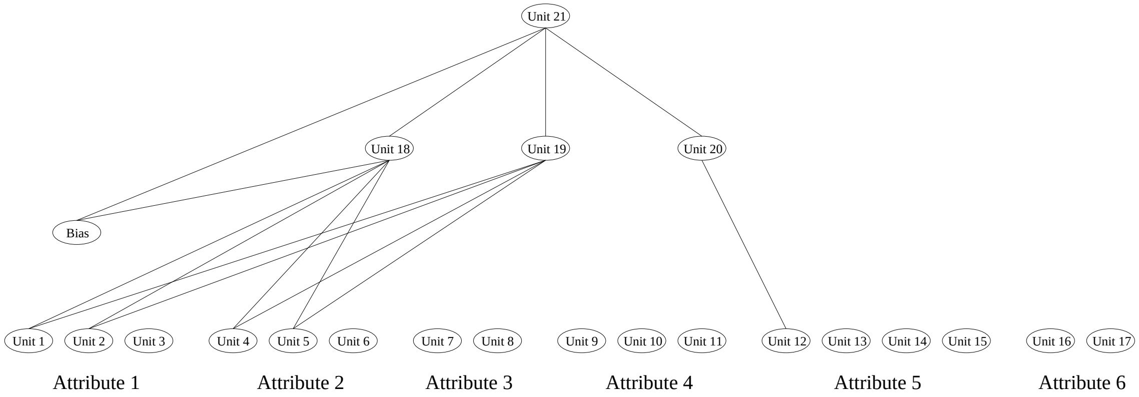
Pruning example: Monk's problem (Thrun, 1992)

- boolean function, depending on few attributes
- unit-obs never deleted useful units, 100 % correct classification

Original MLP:



Pruned MLP using Unit-OBS + OBS (Stahlberger and Riedmiller, 1996):



Pruning example: Thyroid

- three class classification problem
- originally 21 inputs

algorithm	# weights	topology	speedup	cpu-time	perf. test
no pruning	316	21-10-3	-	-	98.4 %
OBS	28	**8**-7-3	1.0	511 min.	98.5 %
Unit-OBS	41	**7**-2-3	7.8	76 min.	98.4 %
Unit-OBS + OBS	24	**7**-2-3	-	137 min.	98.5 %

Table 1: The thyroid benchmark

Regularization techniques - Topology optimization

- topology optimization

  vary the network size and structure, check the validation error, use automated search engines (e.g. evolutionary algorithm)

- System based on learning + evolution: ENZO (Weisbrod, Braun, 1992)

Regularization techniques - weight sharing

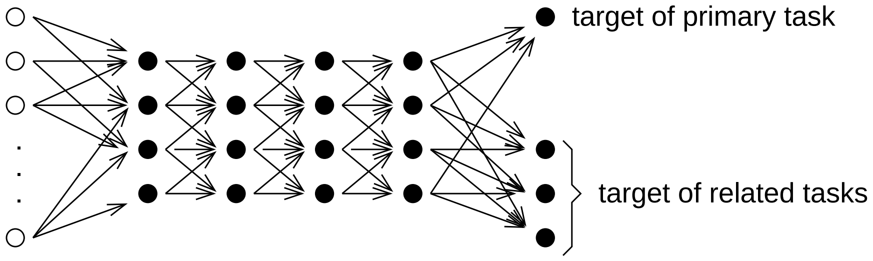
- weight sharing

  neurons of several subnetworks use the same weights ⇒ reduction in the number of parameters and model complexity.

  Useful to generate masks e.g. to detect objects in pictures etc. (see example on next slides)

- Multi-Task learning

  combine learning the primary task with learning related tasks within the same neural network.



  typical examples: timeseries prediction

Regularization techniques

- cascade correlation ((Fahlmann 1990))

  start with a small MLP and successively add neurons. Similar to Tower-algorithm. Weights of additional neurons are trained to maximize the covariance with current error on training set and then fixed

  - constructive method.
  - danger to overfit.

Regularization techniques - more data

- provide more data

  data analysts' fundamental slogan:

there's no data like more data!

  try to get more data; if not possible directly, think about related sources of similar data

  although trivial, one of the most important techniques to improve ML models

Regularization techniques - filter training data (cont.)

- training data often contain a subset of data that help much learning the task and a subset that does not contribute much. Filtering means, reducing the training set to the really important patterns that help adjusting the classification boundary/regression curve.
- techniques: oversampling, subsampling, outlier rejection, jitering
- appropriate technique must be chosen problem-specific.
- frequent problem: unbalanced data in classification

Regularization techniques - input features

- use more/less/other input features
- removing features may reduce overfitting. Helps to avoid that MLP concentrates on pseudo relationships
- adding features may improve generalization if it is related to desired output
- performing non-linear transformations on the features may also be appropriate
- there are techniques for semi-automated feature selection and dimensionality reduction (principle component analysis, independent component analysis, mutual information)
- appropriate techniques must be chosen problem-specific.

Regularization techniques - Committees

- committee/ensemble approaches:
- if you ask an expert, expert may fail
- ask a committee of experts: the majority has a better chance to be right. premise: experts are experienced and diverse.

Committees -Bagging (Breimann, 1996)

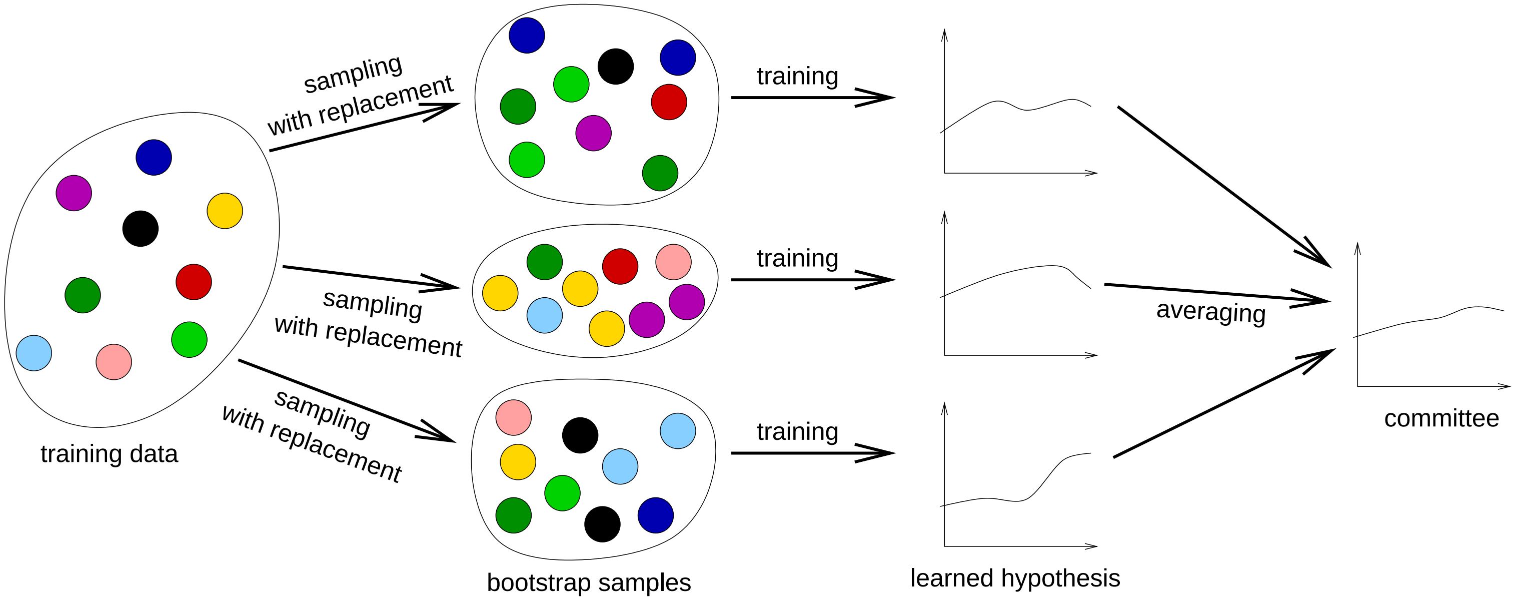
- building a committee:

- train several MLPs on bootstrap samples of the training data

- data is drawn randomly with replacements ⇒ some patterns may occur twice or more, others don't occur at all

- average the ouput of all the MLPs

- single members of the committee might produce a higher test-set error; however in general the diversity of the committee compensates for this effect and therefore the committee error improves over the error of the individuals


training data
sampling with replacement
sampling with replacement
sampling with replacement
bootstrap samples
training
training
training
learned hypothesis
averaging
committee

Committees - Boosting (Freund & Schapire, 1995)

- ensemble technique like bagging, but MLPs are not trained independently
  - second MLP is learned on the training data that are not well learned by the first MLP
  - third MLP is learned on the training data that are not well learned by the first and second MLP
  - ...
- step by step, we get better commitees on the training set
- boosting is successfully used in practice
- good generalisation capabilities on low-noise data
- overfitting might occur on noisy data

Recipe for learning

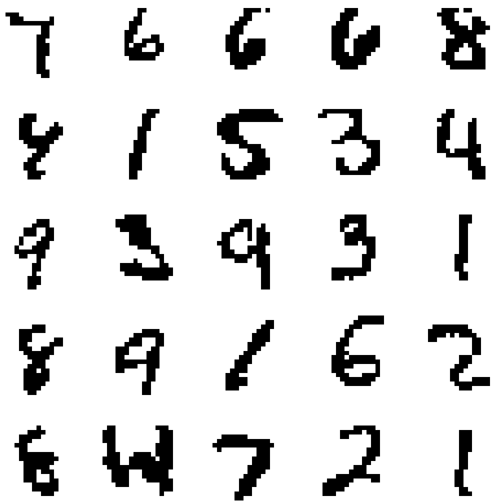
given a real world problem:

- decide on the principle modeling (classification, regression, unsupervised approach)
- collect data
- generate and select features
- create training patterns
- train MLP
- change regularization parameters (weight decay), net topology and repeat training
- eventually change features, create training patterns again and repeat training
- test MLP on new data (test data) and report results

Example: digit recognition

- refers to work of Yann LeCun et. al. (1995)

- background:
  US postal service is interested in automated recognition of zip codes, they provided a large database of digits observed with a digital camera from letterns and postcards. The images are labeled manually. Each image consists of a 20×20 pixel array with black and white pixels.



- task: multi-class classification (10 classes)

- preprocessing: finding correct clipping of image, rescaling number to equal size, centering digits within 20×20 clipping, deslanting digits

- input coding: 400 binary features

- output coding: 1-of-10 vector

Example: digit recognition (cont.)

- baseline approach: learning a linear model (no hidden neurons)

  $400 \cdot 10 + 10 = 4010$ weights, $60\,000$ training patterns

  8.4% misclassified test patterns out of $10\,000$ test patterns
- second approach: MLP with one hidden layer, 400-300-10 topology

  $300 \cdot (400 + 1) + 10 \cdot (300 + 1) = 123\,310$ weights
- best result: 1.6% misclassified test patterns

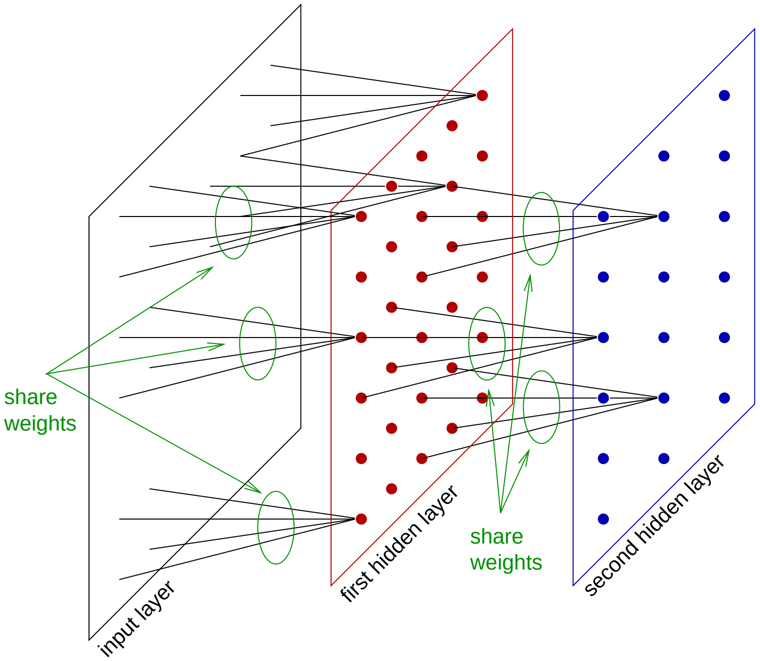
  learned with gradient descent.

Yann LeCun et al. (1995):

> *It remains somewhat of a mystery that networks with such a large number of free parameters manage to achieve reasonably low error rates on the test set, even though comparing their size to the number of training samples makes them appear grossly over-parameterized.*

Example: digit recognition (cont.)

- third approach: weight sharing large MLP, 4 hidden layers, several subnetworks share their weights ⇒ reduces number of free parameters
- LeNet1: 3000 free parameters, test error: 1.7%
- LeNet4: $17\,000$ free parameters, test error: 1.1%
- LeNet5: $60\,000$ free parameters, test error: 0.9%



Example: digit recognition

- fourth approach: boosting
- using principle approach of LeNet4, use boosting algorithm to improve results
- test error of boosted LeNet4: 0.7%

Example: digit recognition

- boosted LeNet4 gave the overall best results in 1995, even compared to other machine learning approaches
- in 2002, DeCoste and Schölkopf could improve the performance using a support vector machine (SVM): test error: 0.56%
- human performance: approx. 0.2%
- training time (on Sun Sparc 10):
  - linear model: half a day
  - LeNet1: 3 days
  - LeNet4: 14 days
  - boosted LeNet4: 35 days
- do you like to compete?
  here are the data: http://yann.lecun.com/exdb/mnist/

Summary

- important insight: you cannot learn without inductive bias (preference rule, incomplete hypothesis sets)
- regularization techniques: realizations of preference rule
- most important ideas for regularization:
  - prefer small weights
  - prefer small networks
  - improve task description
  - ask several networks, not only one (ensembles)