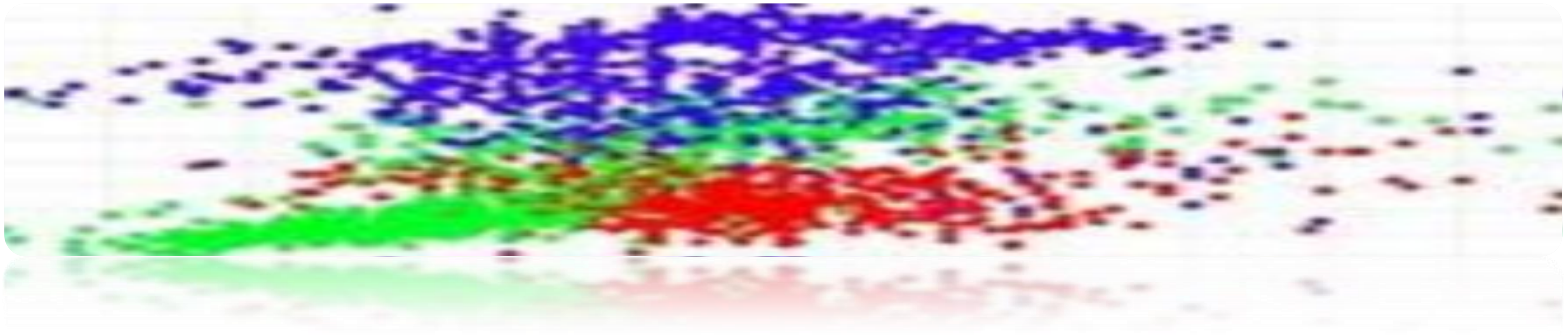


ARTIFICIAL NEURAL NETWORKS

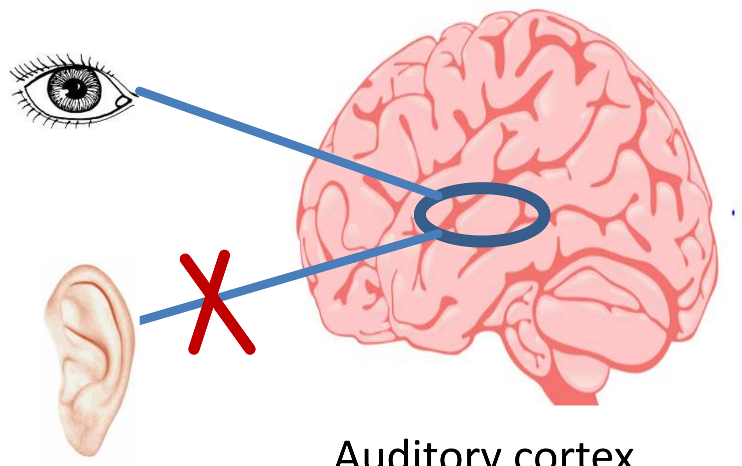
Outline

- Motivation from neuroscience
- Sparse coding
- Perceptron
- Logistic Regression
- Deep belief networks
- Backpropagation Algorithm
- Scalable inference with Artificial Neural Networks (ANNs)

Motivation from neurophysiology

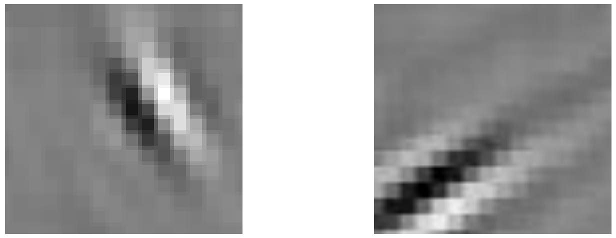
- Goal: Design a learning algorithm that emulates the brain
  - Brain consists of $\sim 10^{11}$ interconnected neurons, each connected to $\sim 10^{4}$ other neurons
  - Neuron switching time $\sim 10^{-3}$ seconds (relatively slow compared to number of operations a computer can do per second)

- Hypothesis: In biological learning systems, there is only one generic learning algorithm
  - The same brain tissue can be adapted to perform various tasks



Motivation: Sparse coding for visual processing (Olshausen & Field 1996)

- The visual cortex processes stimuli from the environment by encoding them most efficiently, e.g. by removing redundancies and representing only the "strongest" stimuli


Generic patterns occurring in natural images

- Only relatively "few" neurons are involved in this process
- Each neuron handles certain reoccurring patterns

Sparse coding: Formalization as linear model

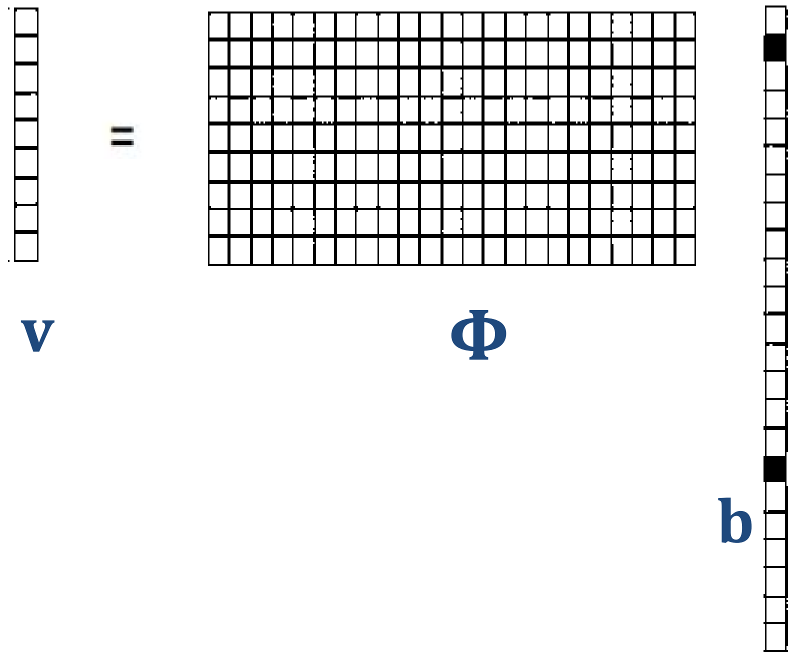
Target vector to be approximated

$$\mathbf{v} = \sum_i b_i \, \boldsymbol{\phi}_i$$

Basis function / vector (from a dictionary of basis functions)

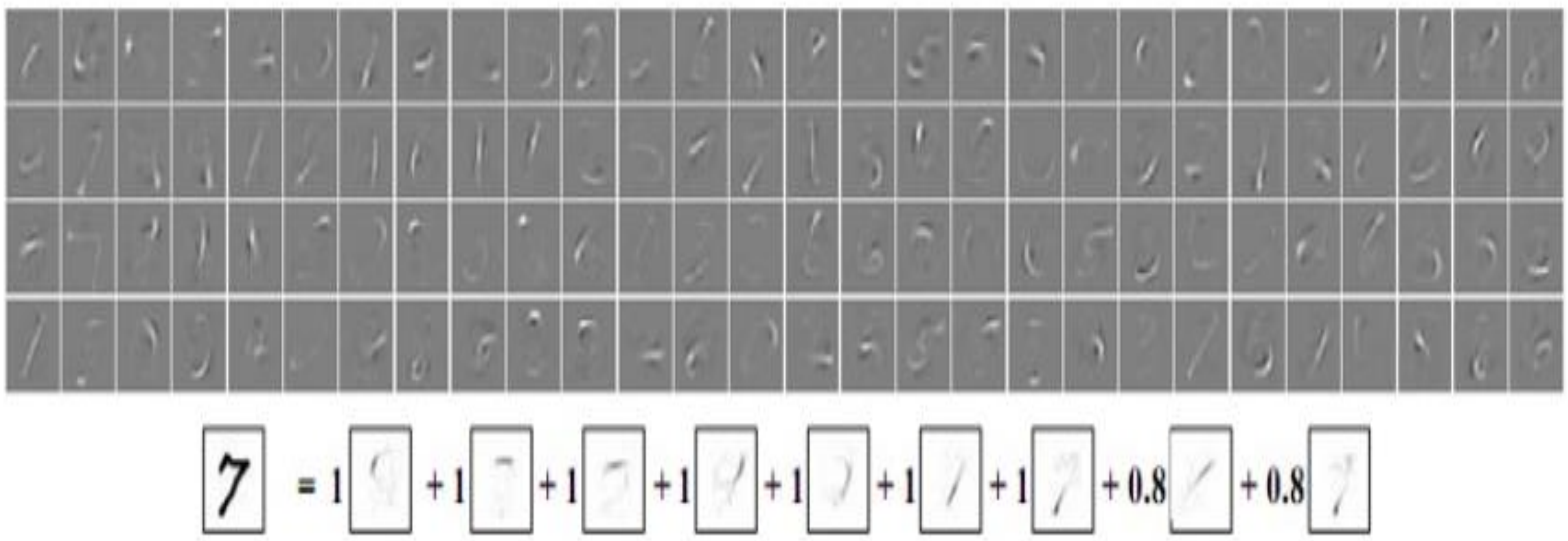
weight

In terms of matrix operations:



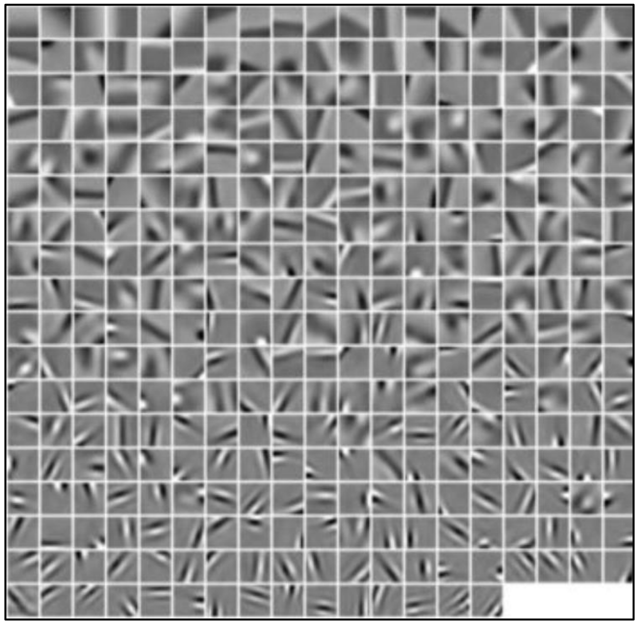
Sparse coding: Example 1

- Combination of dictionary entries to sparsely represent a hand-written number

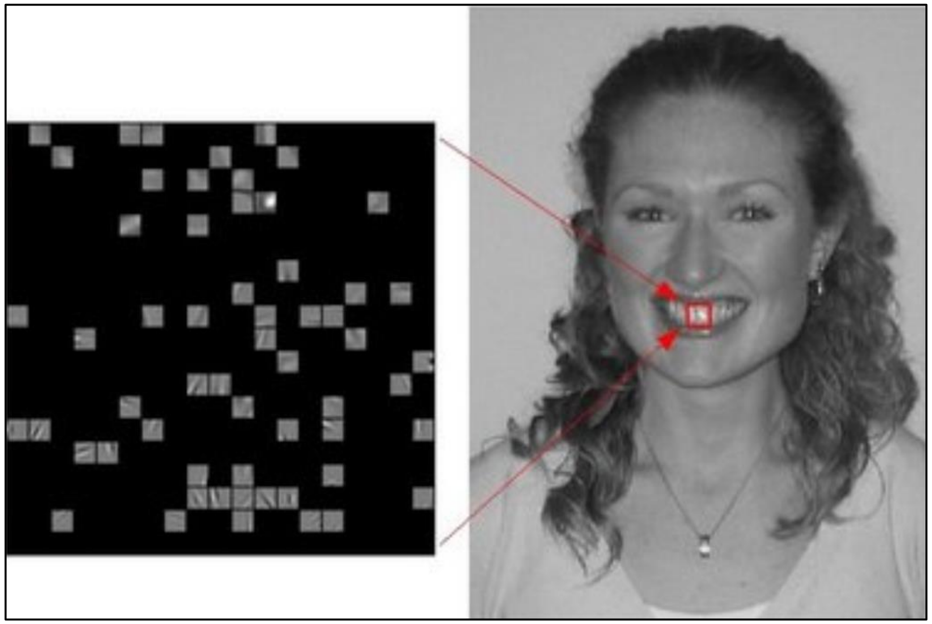


Source: http://wiki.ldv.ei.tum.de/Sparse%20Coding

Sparse coding: Example 2



Dictionary derived from natural images



Reconstruction by using dictionary entries

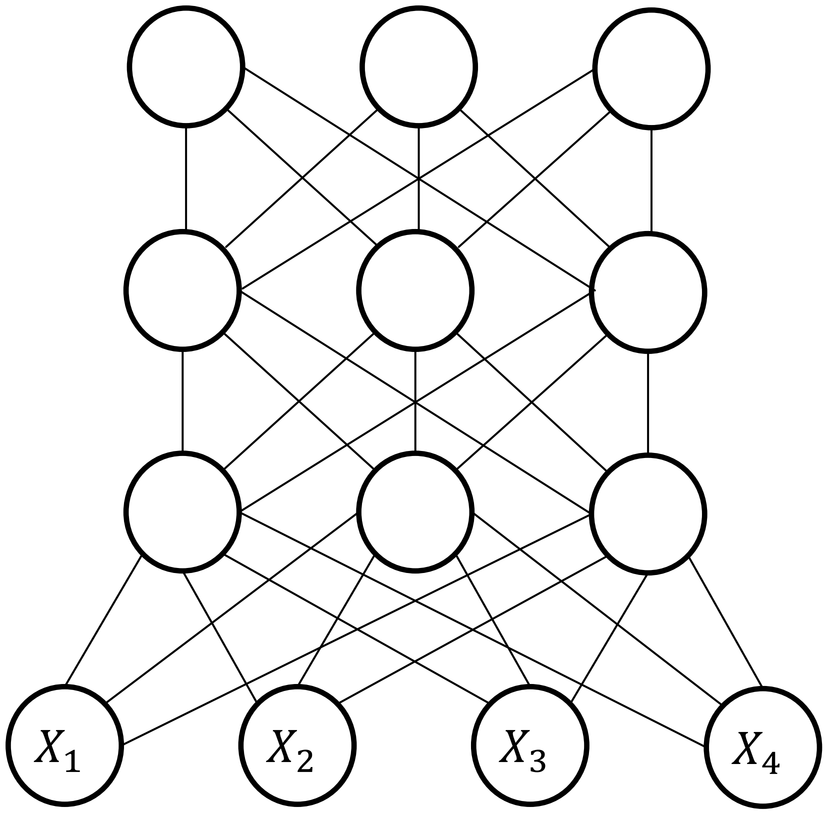
Source: http://wiki.ldv.ei.tum.de/Sparse%20Coding

Recap of sparse coding

- Generic features (i.e., dictionary entries) are combined to yield parts of the image
- The reconstructed parts (i.e. super features) can be combined to yield an even larger part of the image
- Seems like a hierarchical model …
- Is there a general formalization of this?

Deep Belief ANNs: Learning Feature Structures

- Multi-layer Artificial Neural Network (= Deep Belief Networks)



Model layer 3: Higher level concepts

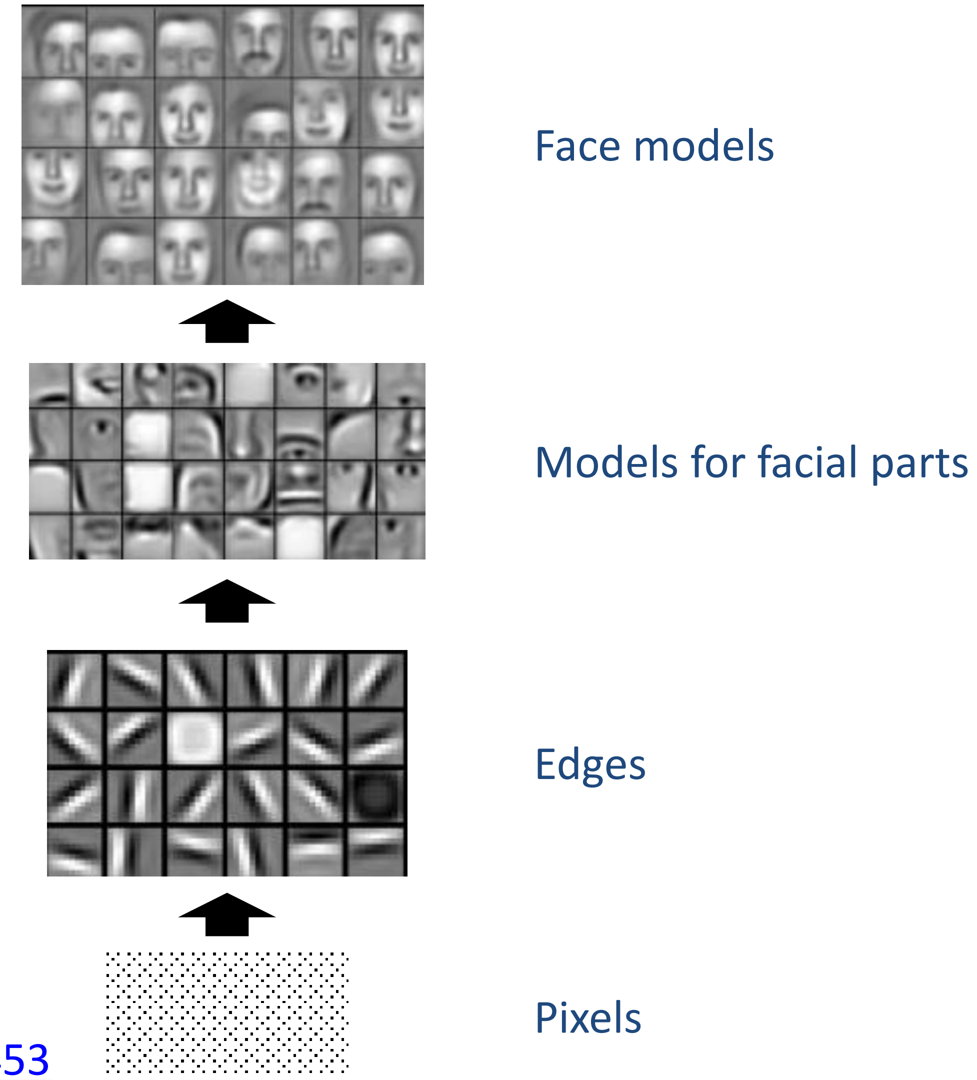
Model layer 2: Higher level concepts

Model layer 1: Higher level concepts

Input features

Example: Face Recognition (H. Lee et al., ICML'09)

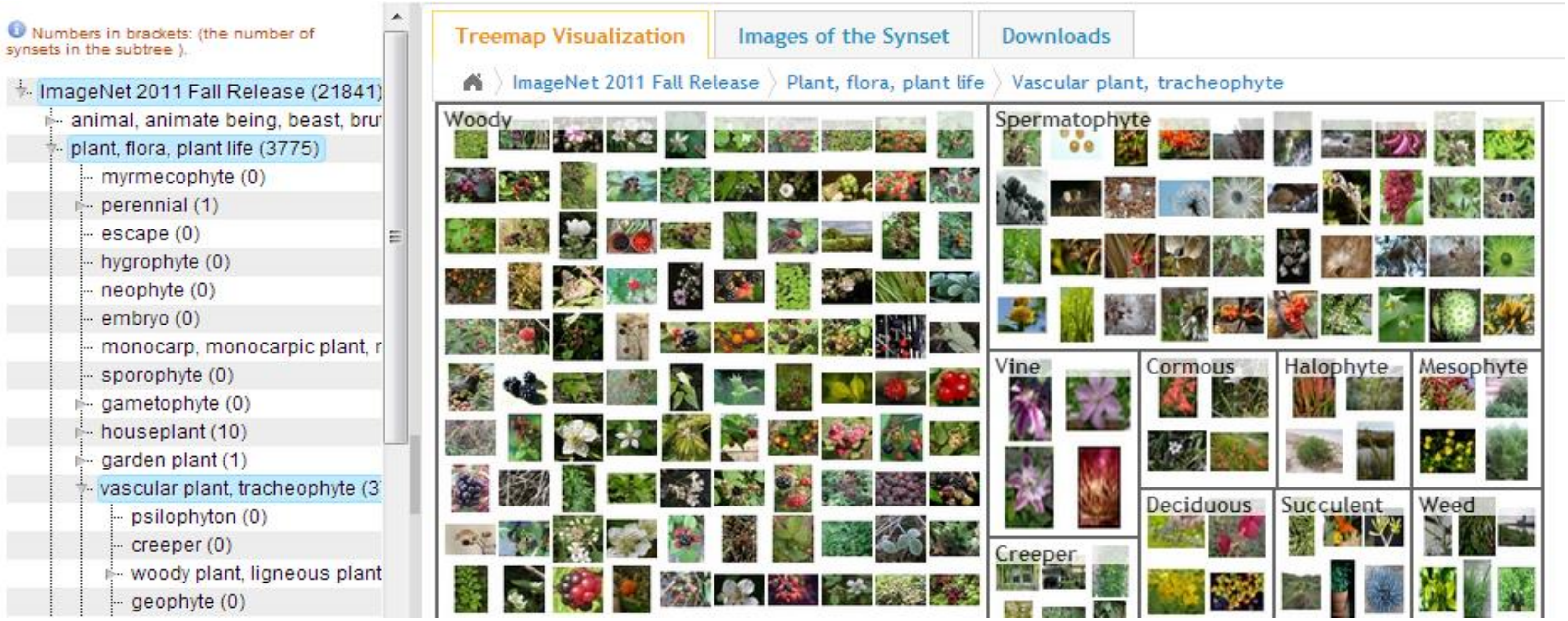
- Hierarchical sparse coding through deep belief networks
- Basic features are combined to more general features
- The final layer represents a model of a real-world object



For details see:
http://dl.acm.org/citation.cfm?id=1553453

Example: Categorization of Images (Le et al., ICML'12)

- ImageNet dataset: 20,000 Categories, 16,000,000 images
- Task: Automated assignment of images to corresponding categories
  - Probability of correct assignment through random guess: 0,005%
  - State-of-the-art ML techniques: 9.5%
  - Unsupervised learning of features from raw pixels: **19.2%** (see: http://icml.cc/discuss/2012/73.html)



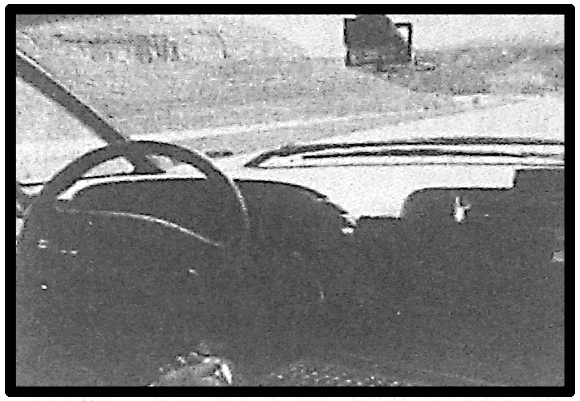
Source: http://www.image-net.org/explore

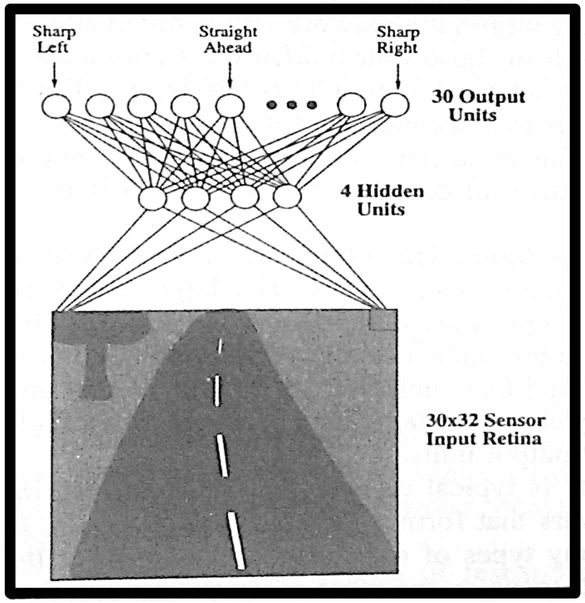
Common characteristics of problems to solve with ANNs

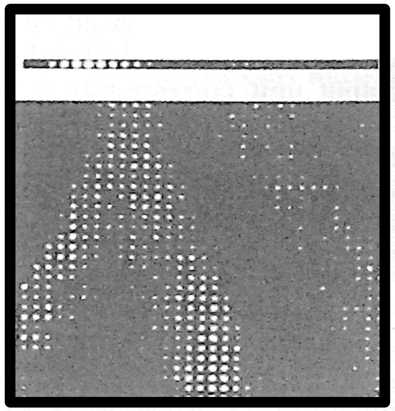
- Input instances can be represented as attribute-value pairs
  - E.g.: (pixel position, pixel value), (term id, frequency), (patient, age)…
  - Input attributes can be correlated or independent
  - Input values can be any real values
- Target function can be discrete- or real-valued or a vector of discrete or real-valued attributes
- Training examples may contain errors; ANNs are quite robust with respect to noise
- Long training times should be acceptable
  - Training time depends on number of weights in the network, the number of training examples, and the initial setting of the parameters
- Understanding the learned target function is not critical (or important)
  - Target function is general a non-linear mapping of the input data onto the output space

ALVINN: ANN system for autonomous driving

- 960 inputs connected to 4 hidden units, which in turn are connected to 30 output units
- Matrix on the left depicts weight values for one hidden unit (the brighter the cells the higher the values)
- Values from this hidden unit to the 30 output units are depicted on top of the matrix
- ALVINN uses Backpropagation to adjust the weights and enables autonomous driving at speeds up to 112 km/h



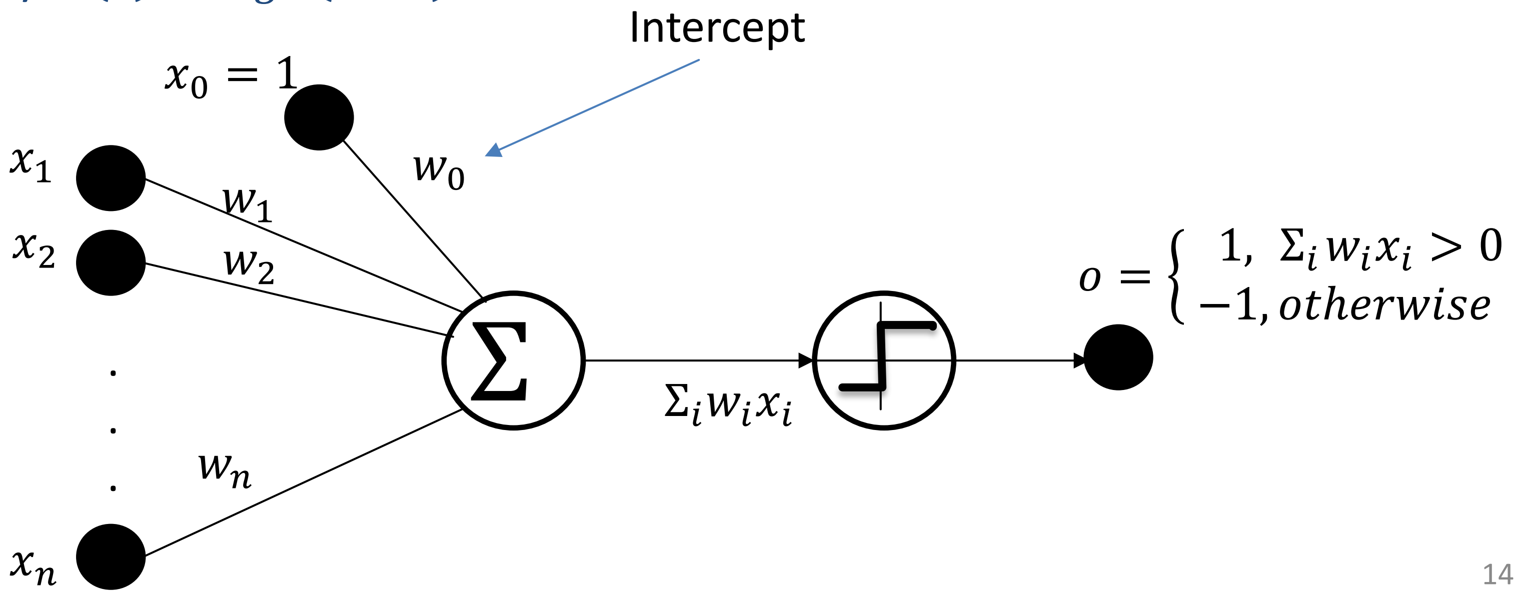




Source: Machine Learning by T. Mitchell

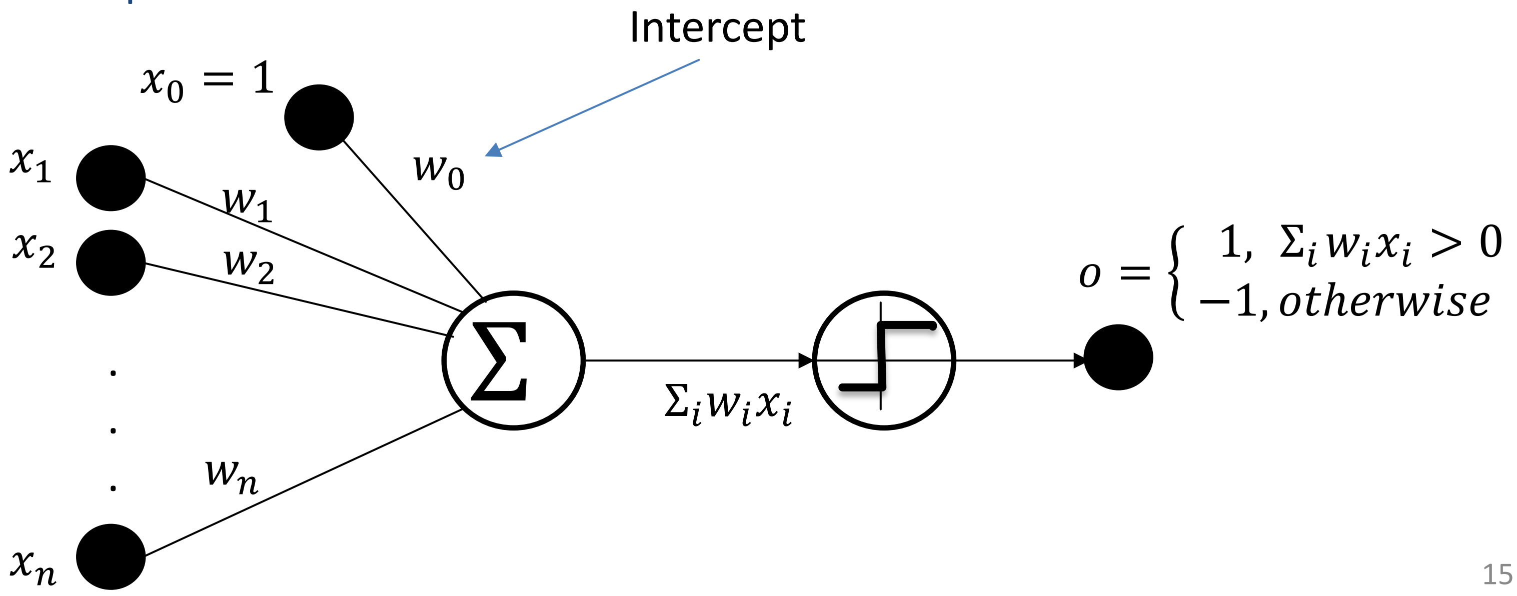
Simplest ANN: The perceptron

- Input is a vector of features $\mathbf{x} = (x_1, \ldots, x_n)$ of real values, e.g., a persons age, weight, height, blood pressure, heart rate, gender …
- Predict risk for heart attack
- Model: $o(x_1, \ldots, x_n) = \begin{cases} 1, & w_0 + w_1 x_1 + \cdots + w_n x_n > 0 \\ -1, & otherwise \end{cases}$
- Or simply: $o(\mathbf{x}) = sign(\mathbf{w} \cdot \mathbf{x})$



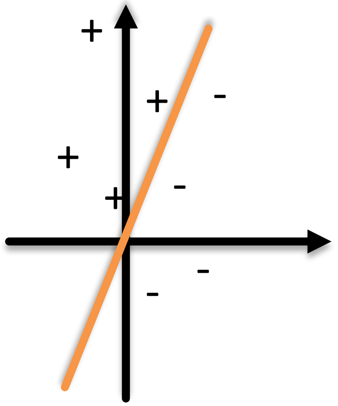
Simplest ANN: The perceptron

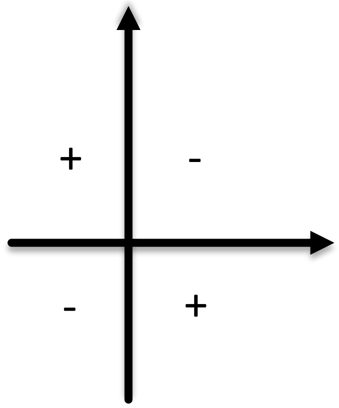
- Model: $o(x_1, \dots, x_n) = \begin{cases} 1, & w_0 + w_1 x_1 + \cdots + w_n x_n > 0 \\ -1, & otherwise \end{cases}$
- Or simply: $o(\mathbf{x}) = sign(\mathbf{w} \cdot \mathbf{x})$
- Boolean AND, OR, NAND, and NOR can be represented
- How?
- Why is this important?



Limitations of perceptrons

- XOR function cannot be represented
- Can not separate training data that is not linearly separable
- Note that a two-layer perceptron can represent any Boolean function





Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

The perceptron training algorithm

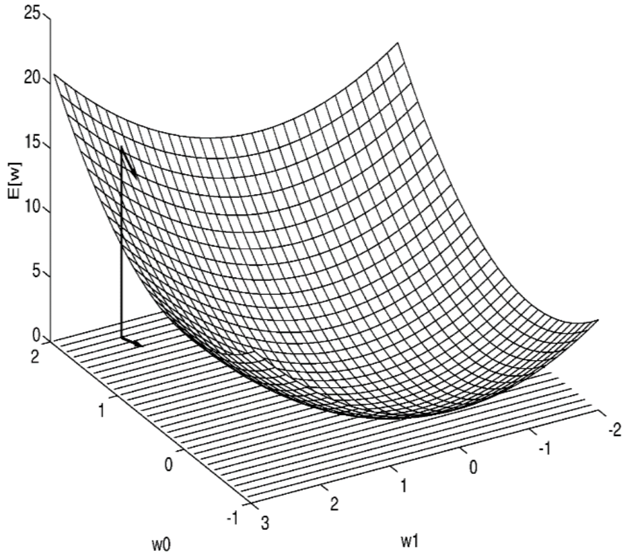
```
Start with random weights $w_1, \dots w_n$
Until the perceptron classifies all training examples correctly
    For each training example $(\mathbf{x} = (x_1, \dots, x_n), l(\mathbf{x}))$
        For each $w_i$
            Compute $\Delta w_i = \eta(l(\mathbf{x}) - o(\mathbf{x}))x_i$,    $w_i \leftarrow w_i + \Delta w_i$
```

Learning rate, e.g., 0.1   True value of $\mathbf{x}$

What if training data is not separable?

The Gradient Descent Rule (1)

- Lets suppose we aim to minimize
- $\mathrm{Er}(\mathbf{w}) = \frac{1}{2}\sum_{\mathbf{x}\in TrainSet}\left(l(\mathbf{x}) - o(\mathbf{x})\right)^2$
- If we knew the gradient of Er, $\nabla \mathrm{Er}(\mathbf{w}) = \left[\frac{\partial \mathrm{Er}}{\partial w_1}, \ldots, \frac{\partial \mathrm{Er}}{\partial w_n}\right]$ we would have an algorithm to minimize it
- The negated gradient indicates the direction of the steepest descent
- We would only need to iteratively re-compute the gradient and follow it



Source: Machine Learning by T. Mitchell

The Gradient Descent Rule (2)

- The partial derivative of E for a w_i is

$$\frac{\partial \mathrm{Er}}{\partial w_i} = \frac{1}{2} \sum_{\mathbf{x} \in TrainSet} 2\big(l(\mathbf{x}) - o(\mathbf{x})\big) \frac{\partial \mathrm{Er}}{\partial w_i} \big(l(\mathbf{x}) - \mathbf{w} \cdot \mathbf{x}\big)$$

$$= \sum_{\mathbf{x} \in TrainSet} \big(l(\mathbf{x}) - o(\mathbf{x})\big)(-x_i)$$

- Set $\Delta w_i = -\eta\left(\sum_{\mathbf{x} \in TrainSet} \big(l(\mathbf{x}) - o(\mathbf{x})\big)(-x_i)\right)$

- Then by following the gradient we get $w_i \leftarrow w_i + \Delta w_i$

Gradient Descent Algorithm

```
Start with random weights $w_1, \dots w_n$
Until the error is smaller than some threshold
        Initialize each $\Delta w_i$ to zero
        For each training example $\left(\mathbf{x} = (x_1, \dots, x_n), l(\mathbf{x})\right)$
                For each $w_i$
                        Compute $\Delta w_i = \Delta w_i + \eta\left(l(\mathbf{x}) - o(\mathbf{x})\right)x_i$,
        For each $w_i$
                $w_i \leftarrow w_i + \Delta w_i$
```

Stochastic Gradient Descent Algorithm

```
Start with random weights $w_1, \dots w_n$
Until the error is smaller than some threshold
        ~~Initialize each $\Delta w_i$ to zero~~
        For each training example $(\mathbf{x} = (x_1, \dots, x_n), l(\mathbf{x}))$
                For each $w_i$
                        Compute ~~$\Delta w_i = \Delta w_i + \eta(l(\mathbf{x}) - o(\mathbf{x}))x_i$~~
                                $w_i = w_i + \eta(l(\mathbf{x}) - o(\mathbf{x}))x_i$
        ~~For each $w_i$~~
                ~~$w_i \leftarrow w_i + \Delta w_i$~~
```

Delta Rule
or Least-Mean-Square Rule

Remarks to Stochastic Gradient Descent

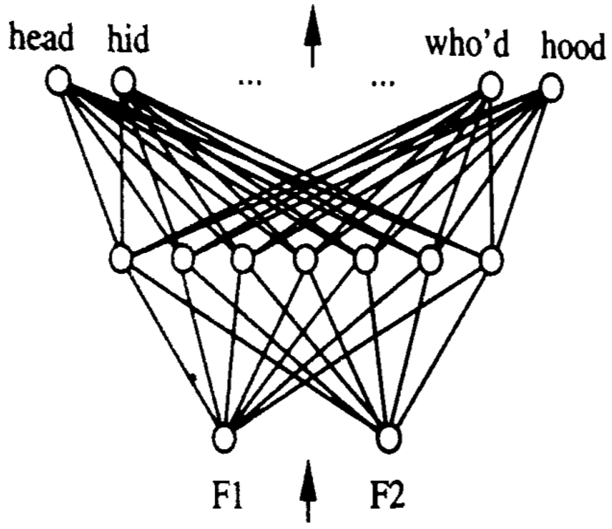
- It is an approximation to Gradient Descent
- The actual error function we aimed to minimize is

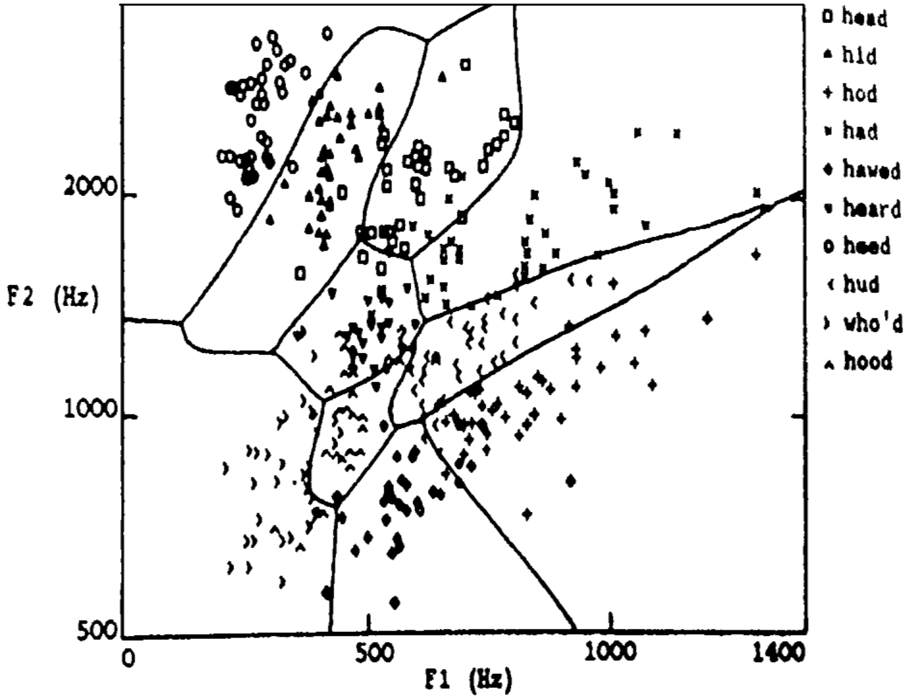
$$\mathrm{Er}(\mathbf{w}) = \frac{1}{2} \sum_{\mathbf{x} \in TrainSet} \left(l(\mathbf{x}) - o(\mathbf{x}) \right)^2$$

- Can be applied whenever the solution space is continuously parameterized and the error function can be differentiated
- Should be applied whenever there are many possible solutions and the training data is too large (because gradient descent is not guaranteed to reach the global minimum)
- By making η sufficiently small, true gradient descent can be approximated arbitrarily closely

Multi-Layer Networks: Example

- Recognize one of ten vowel sounds of the form “h_d”
- From spectral analysis we can get the first and the second principal component F1 and F2

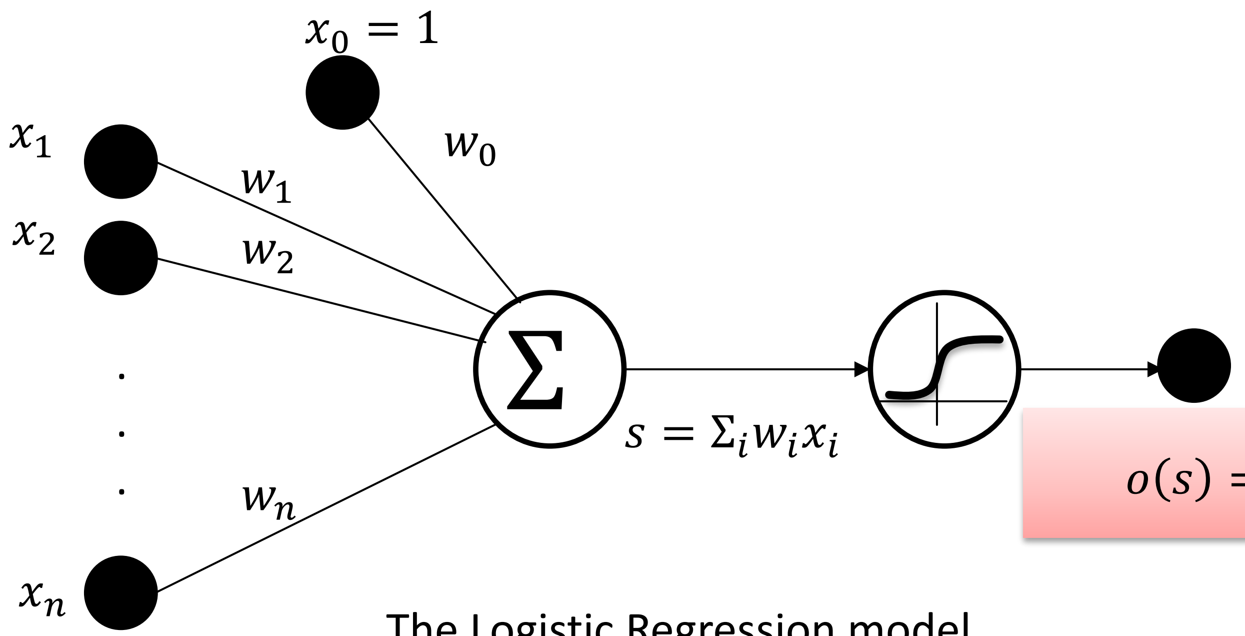




Source: “Machine Learning” by T. Mitchell

The sigmoid unit: A differentiable threshold unit

- The cascaded combination of multiple linear threshold units can only produce (piece-wise) linear functions
- As in the previous example, we are interested in representing highly non-linear functions



$$o(s) = \sigma(s) = \frac{1}{1 + e^{-s}}$$

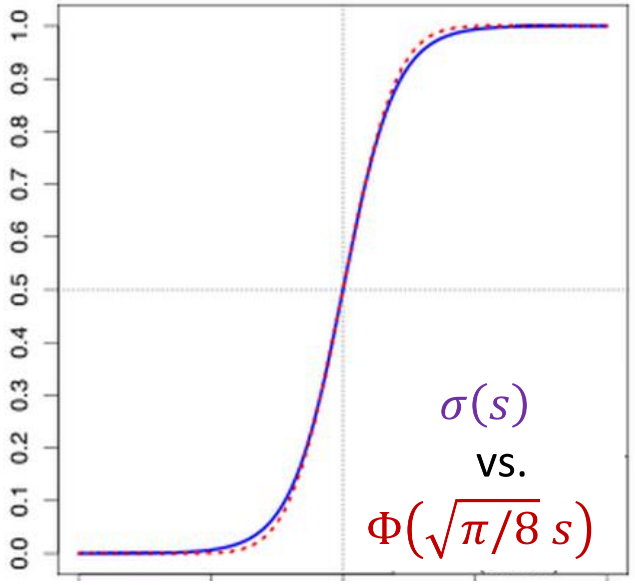
The Logistic Regression model

Properties of the Logistic unit

- Can be interpreted as probability
- Easy to differentiate (i.e., gradients can be easily computed)

$$\sigma(s) = \frac{1}{1+e^{-s}}, \quad \frac{\partial\sigma(s)}{\partial s} = \sigma(s)\big(1-\sigma(s)\big)$$

- Can be replaced by other similar so-called sigmoid functions e.g.,
  - $\sigma_k(s) = \frac{i}{1+e^{-ks}}$
  - $\Phi(s) = \int_{-\infty}^{s} \mathcal{N}(t; 0,1)\ \mathrm{d}t$



Logistic Regression: a probabilistic perspective

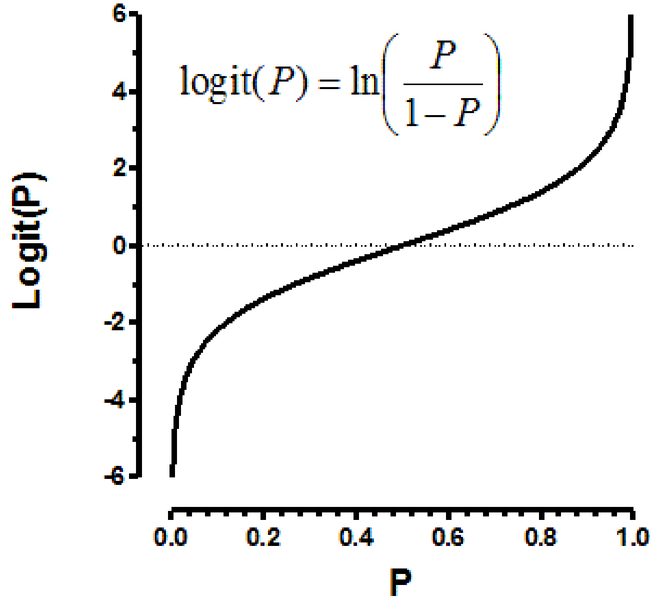
- Example task: How likely is a stroke for patient $\mathbf{x}_i = (x_{i1}, \ldots, x_{ik})$, e.g., with features age=57y, height=178cm, weight=95kg, gender=m, …?

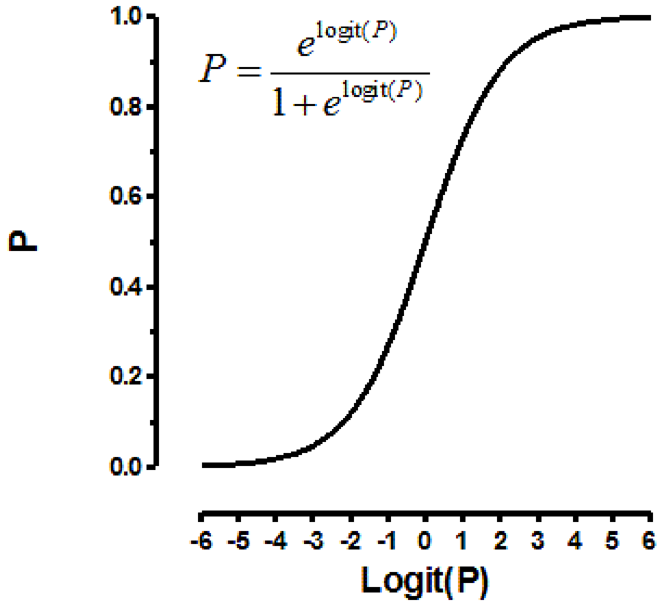
- For large number n_i of people with same feature values as $\mathbf{x}_i$ report fraction of stroke cases

- But we are interested in general importance of the various features

- Formally:
  - $Y_i \sim Bin(n_i, p_i)$, i.e., binomially distributed variable (#strokes in n_i observations)
  - Then $p_i = E(Y_i / n_i | \mathbf{x}_i)$
  - Check log odds by using $logit(p_i) = \log\left(\frac{p_i}{1-p_i}\right) \to \infty, p_i \to 1$ $(\to -\infty, p_i \to 0)$
  - Set $w_0 + \mathbf{w} \cdot \mathbf{x}_i = logit(p_i) = \log\left(\frac{p_i}{1-p_i}\right)$ (i.e., linear combination of feature values)
  - If we are interested in p_i, we need to compute $logit^{-1}(logit(p_i)) = \frac{1}{1+e^{-logit(p_i)}}$

Logit vs. logistic function



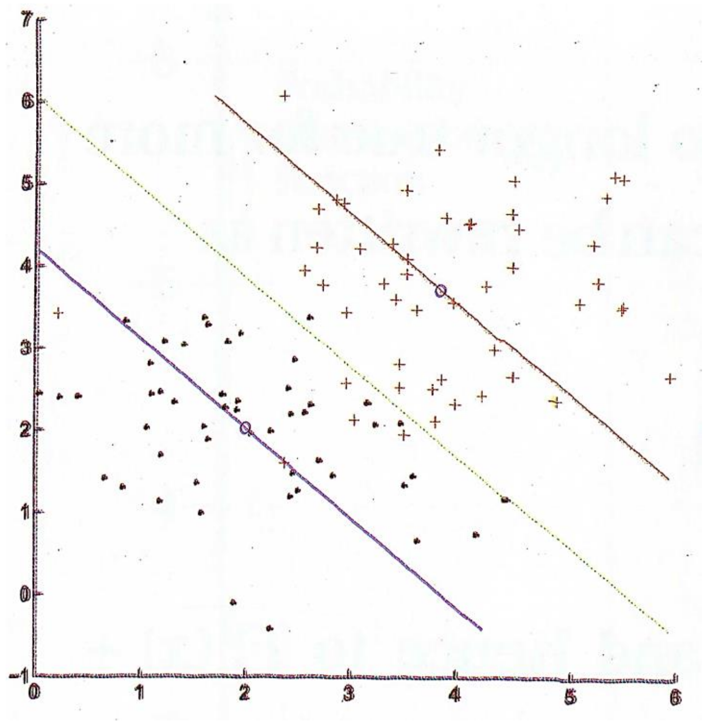


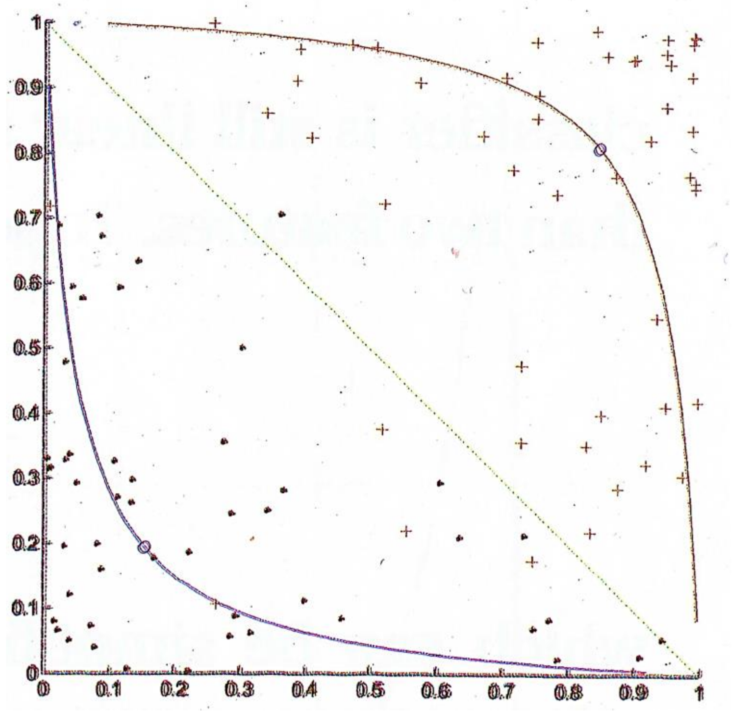
- Properties of logistic regression
  - The smaller the training set, the worse the estimation of log odds
  - Few observations per ***explanatory variable*** $\mathbf{x}_i$ may be enough to enable reliable predictions
  - In case of sparse data, discretization of the feature domains can be considered
  - Goodness-of-fit can be used to validate the model
  - Decision threshold can be adjusted later through calibration

Logistic calibration

- For linearly separated data and weight vector $\mathbf{w}$
  - Compute class mean scores μ^+ and μ^- and the standard deviation s with respect to the score $(w_0 + \mathbf{w} \cdot \mathbf{x}_i)/\|\mathbf{w}\|$
  - For each $\mathbf{x}_i$, compute: $$\alpha_i = \underbrace{\frac{(\mu^+ - \mu^-)}{s}}_{\text{\# standard deviations between the score mean of the positive and the negative class}} \left(\underbrace{\frac{(w_0 + \mathbf{w} \cdot \mathbf{x}_i)/\|\mathbf{w}\|}{s} - \frac{(\mu^+ - \mu^-)}{2s}}_{\text{Distance (in terms of standard deviations) between actual sore and the score mean}} \right)$$
  - Compute $P(+|\mathbf{x}_i) = \frac{1}{1+e^{-\alpha_i}}$
- Very effective calibration method that can also be used for feature calibration (i.e., for continuous features)
- Underlying assumptions
  - Data is approximately normally distributed in each class
  - Similar variance in both classes

Visualization of logistic calibration



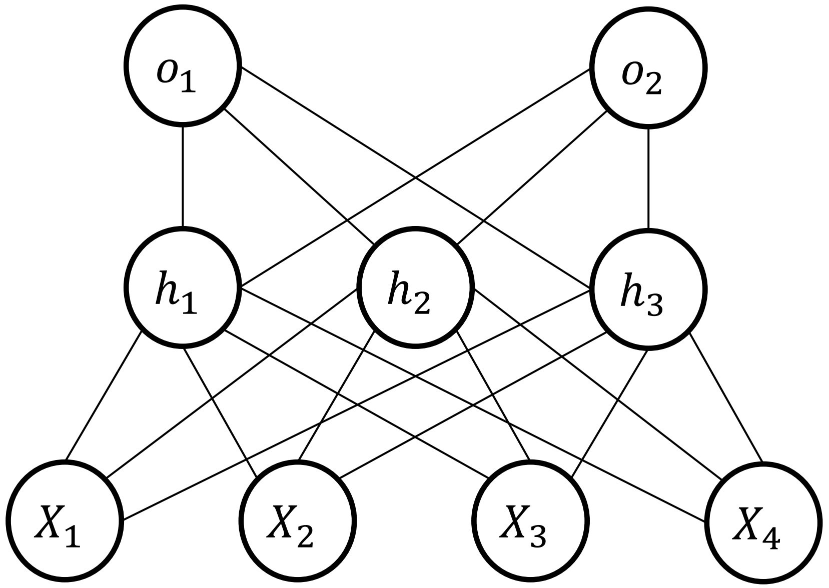


Linearly separated classes with specific weight vector $\mathbf{w}$

Source: Machine Learning by P. Flach

Logistic calibration of data based on weight vector $\mathbf{w}$ corresponds to non-linear transformation that pushes data away from decision boundary

Feed-forward ANNs



- There are no cycles in terms of information processing (i.e., the output of a node is always forwarded to the nodes in the layer above)
- Here we will consider the logistic function as differentiable threshold unit

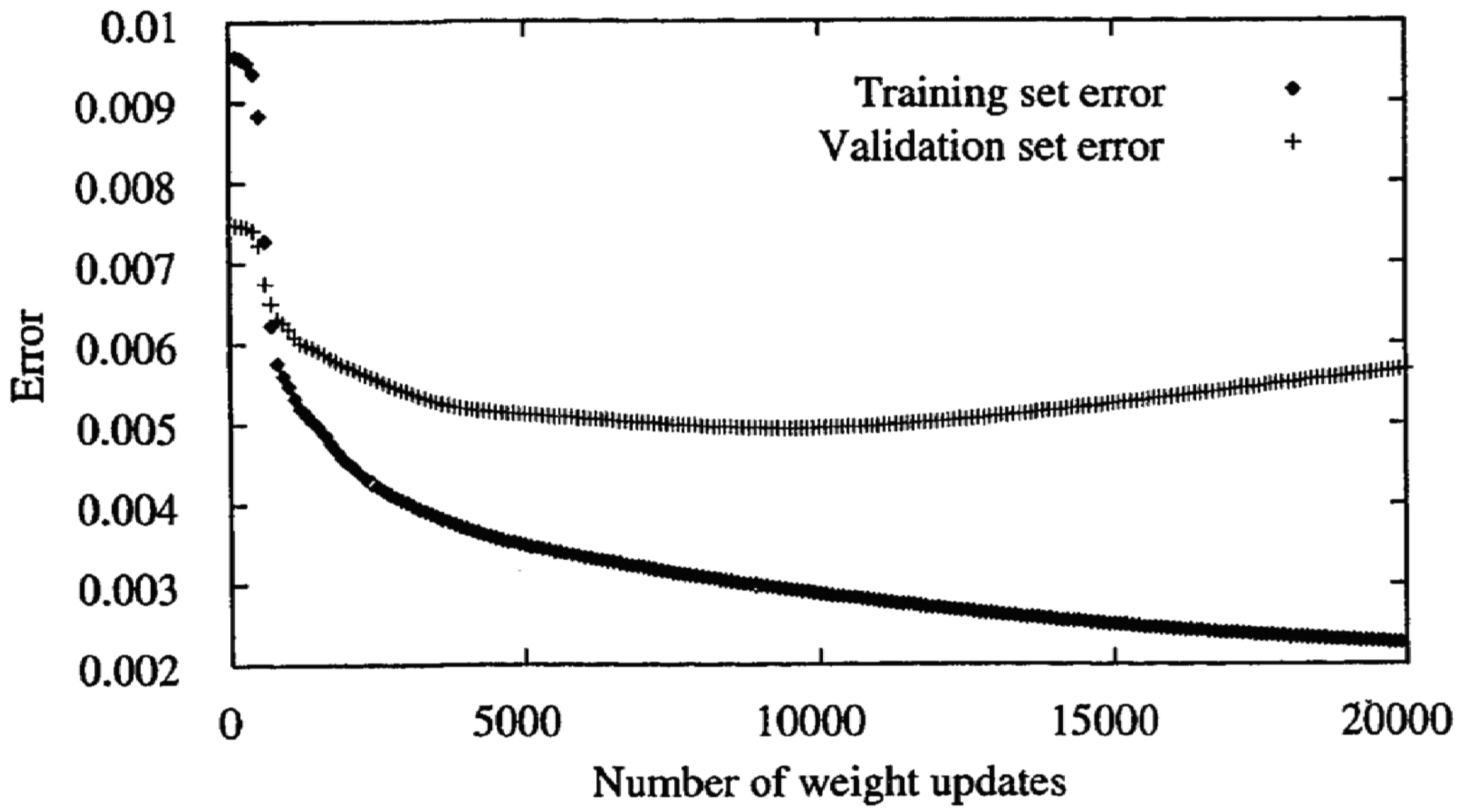
Backpropagation Algorithm for multilayer ANNs

```
Initialize all network weights to random values from
[-0.05, 0.05]
Until error is smaller than some threshold
    For each training pair $(\mathbf{x}_i, l(\mathbf{x}_i))$
        Forward the instance through the network
        and compute output $o_k$ for each $k$
    For each output $o_k$
        $\delta_k \leftarrow o_k(1 - o_k)(l_k(\mathbf{x}_i) - o_k)$
    For each hidden unit $h$
        $\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in out(h)} w_{kh}\delta_k$
    Update each network weight $w_{ji}$
        $w_{ji} \leftarrow w_{ji} + \eta\delta_j x_{ji}$,
        where $x_{ji}$ is the input from unit $i$ to $j$
```

Expressiveness of ANNs

- Theorem
  - Every Boolean function can be represented by a two-layer ANN
  - Every bounded continuous function can be approximated with arbitrary small error by a network with two layers
  - Any arbitrary function can be approximated to arbitrary accuracy by a network with three layers

Overfitting with ANNS



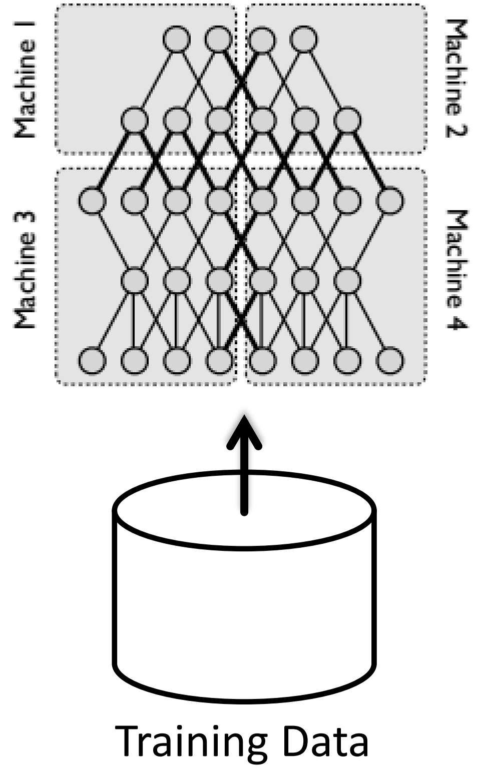
Source: Machine Learning by T. M. Mitchell

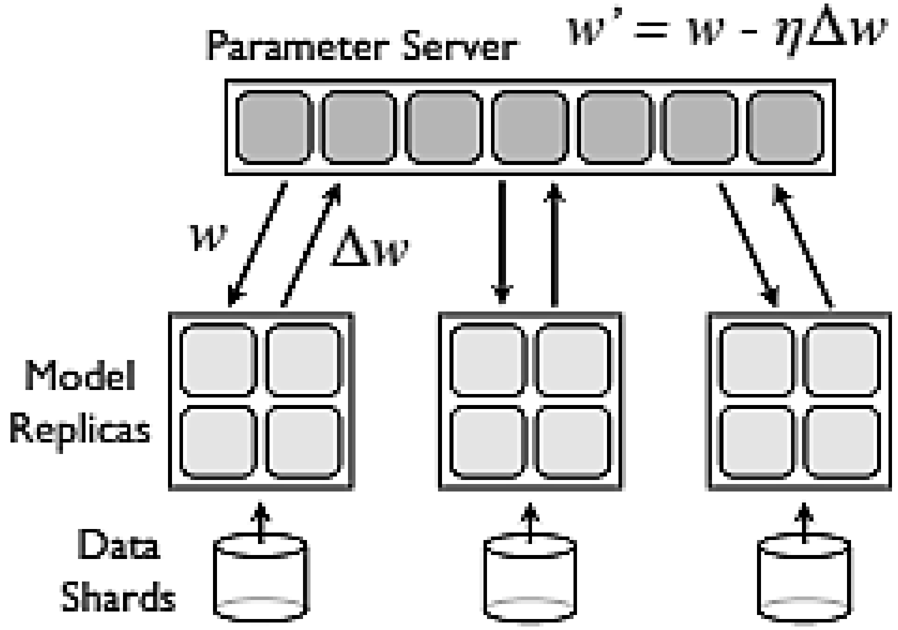
Common issues of ANNs

- Overfitting can be mitigated with more variability in the examples of the training data
- Training data that covers many different examples is invaluable for deep ANNs (e.g., see recent projects like Google Brain Project)
- To learn structure from basic features or accurate classification functions, millions of parameters have to be learned
- Training can be extremely slow (if not parallelized)
- If parallelized all machines need to know the current weights

Large-scale deep learning (J. Dean et al., 2012)

- Networks with up to 1.7B parameters
- Distributed over hundreds of machines and thousands of cores

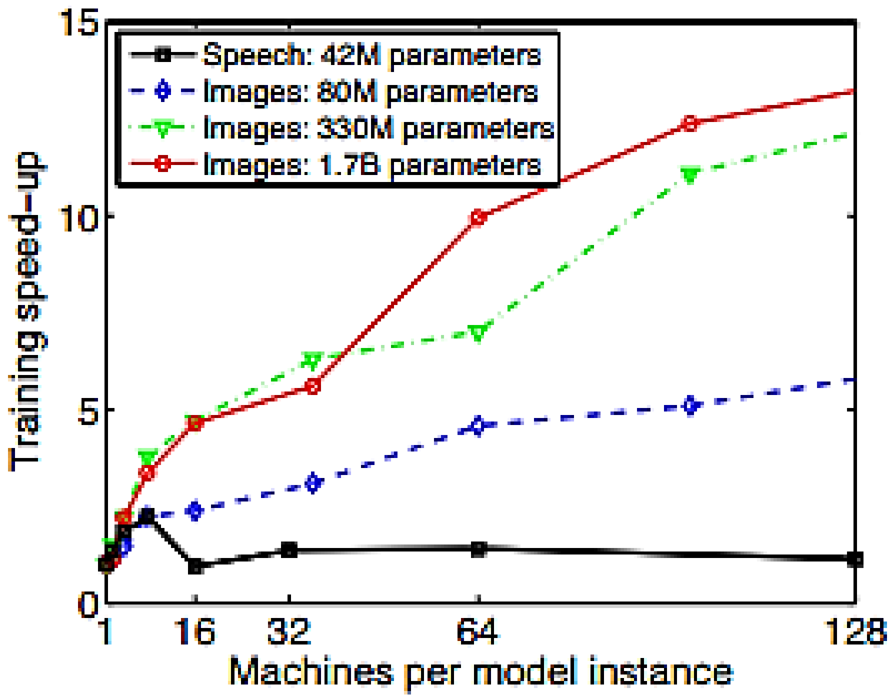


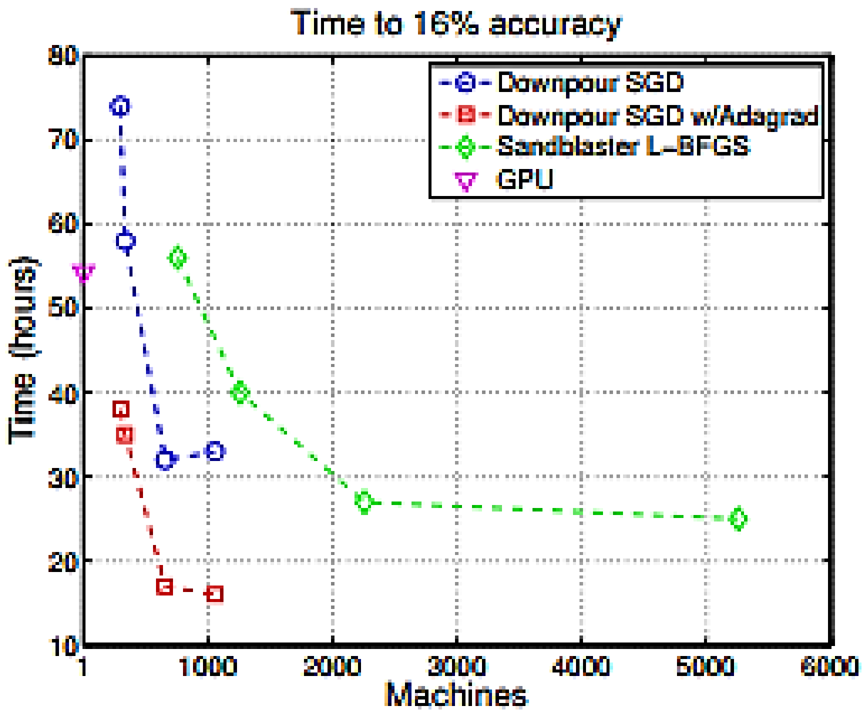


Stochastic Gradient Descent: Model replicas asynchronously fetch parameters ***w*** and push gradients Δ***w*** to the parameter server

Source of figures: J. Dean et al. NIPS 2012

Large-scale deep learning (J. Dean et al., 2012)





Models with more parameters benefit more from additional machines

Time to reach a fixed accuracy (16%) for different optimization strategies as a function of number of machines

Source: J. Dean et al. NIPS 2012

Other types of ANNs

- Recurrent ANNs
  - E.g., for applications to time series data
  - Output at time t is used as input of time $t + 1$
- Dynamic ANNs
  - E.g., start with network that has no (or only few) hidden units and add units as needed (in order to minimize some error)
  - E.g., remove or add interconnections between units
- Bayesian Networks and Markov Random Fields
  - Directly model logical dependencies between variables
  - Model interrelations between variables
- Self-Organizing Maps
  - Learn structure in the data
  - Non-linear mapping of data to lower-dimensional space by preserving original neighborhood topology