Neuronale Netze (SS 2002), 29.4.

Simple perceptron networks:
Easy extensions of the perceptron algorithm where neurons are trained iteratively with the perceptron learning algorithm:

- **Tower algorithm**

  $P := P_0$; $f := (\vec{x} \mapsto \mathbf{0})$;
  repeat, while f does not classify P correctly:
  $P := \{(\vec{x}, f(\vec{x})); y) \mid (\vec{x}; y) \in P_0\}$;
  train a perceptron p for P.
  $f := (\vec{x} \mapsto p(\vec{x}, f(\vec{x})))$;

  I.e. in each run the new neuron may benefit from the correct classification of the previous one.

- The tower algorithm can correctly classify at least one more point in each iterative run if $P \subset \{-1, 1\}^n \times \{0, 1\}$.

- **Upstart algorithm**

  Training(P; f)
  {
  train a perceptron p for P, assume $(\vec{w}, \theta)$are the weights and bias.
  If P is not yet correct:
  $P_1 := \{(\vec{x}; 1) \mid (\vec{x}; 1) \in P, p(\vec{x}) = 0\} \cup \{(\vec{x}; 0) \mid (\vec{x}; 0) \in P\}$
  $P_2 := \{(\vec{x}; 1) \mid (\vec{x}; 0) \in P, p(\vec{x}) = 1\} \cup \{(\vec{x}; 0) \mid (\vec{x}; 1) \in P\}$
  Training(P_1, f_1)
  Training(P_2, f_2)
  $f(\vec{x}) := \mathrm{H}(\vec{w}^t\vec{x} + \lambda f_1(\vec{x}) - \lambda f_2(\vec{x}) - \theta)$
  otherwise: $f := p$
  }

  I.e. the top neuron recruits specialists for wrong positive/negative points in each run.

- If in the above algorithm f_1 and f_2 are correct, then also f.

- **Ensemples**

  Train perceptrons $p_1, \ldots, p_e$ on P,

  afterwards, train a perceptron p on the outputs of the p_i,
  i.e. $\{(p_1(\vec{x}), \ldots, p_e(\vec{x}); y) \mid (\vec{x}, y) \in P\}$.

- Nice ideas, but there are problems:
  - when to stop the single perceptron runs?
  - how to divide the task?
  - very slow decrease of the classification error for an increasing number of neurons if complicated tasks are dealt with.
  - wasted neurons.
  - strong overfitting can be observed.
- We would like to obtain a model for automatic learning
  - with efficient training algorithms
  - with the universal approximation capability
  - with guaranteed generalization capability
  - and as few tricks as necessary ;)

$\Rightarrow$ Feedforward networks and backprop.