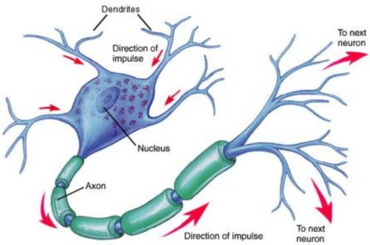
Topic 11: ARTIFICIAL NEURAL NETWORKS

STAT 37710/CMSC 25400 Machine Learning

Neural networks

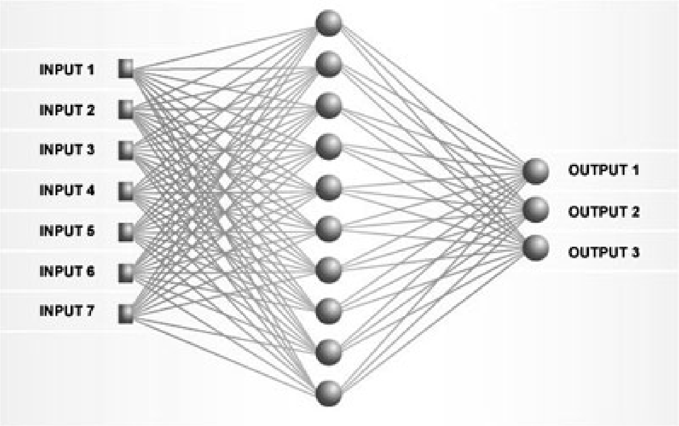


- The human brain has $\sim 10^{11}$ neurons, each connected to $\sim 10^{4}$ others.
- Inputs come from the dendrites, are aggregated in the soma, if the neuron starts firing impulses propagated to other neurons via axons.

- Neurons learn by changing the connection strengths of their synapses.
- Information storage is the nervous system is “distributed”.
- The response time of the brain is quite fast, so the “depth” of the network can’t be very great. (clear layer by layer organization in the visual system).

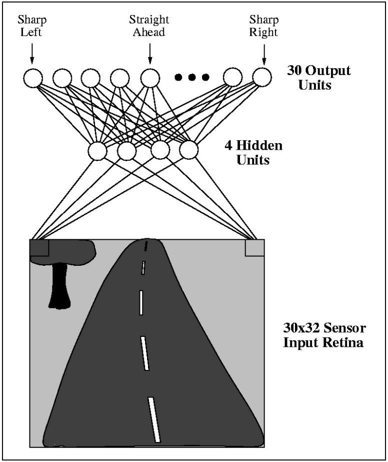
IDEA: Humans seem to be okay at learning, so why not try to replicate this in a computer? Goes back to the early days of AI, many successes and failures.

Multilayer artificial neural net



Question: But what should the individual neurons do and how should they learn?

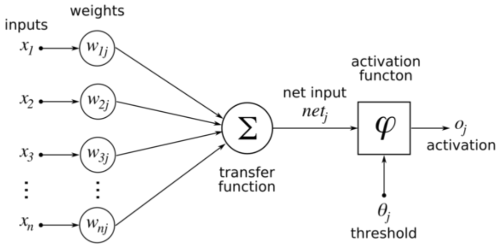
A success: ALVINN [Pomerleau '95]



Drove unassisted from Pittsburgh to NYC on the highway.

A model neuron

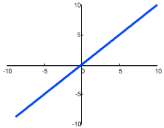
$$o = \phi\Big(\theta + \sum_p w_p x_p\Big),$$



Notation:

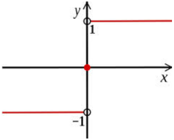
- x_i : the i'th input
- w_i : the corresponding synaptic weight
- θ : the bias (can be eliminated as in the perceptron)
- o : the output

Activation functions



Linear: $\phi(t) = t$

Question: What is the problem with this? Linear functions composed with each other are still linear, so no point in having a multilayer network.



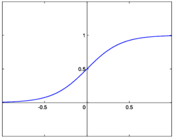
Hard threshold: $\phi(t) = \text{sgn}(t)$

"Threshold Logic Unit" [McCulloch & Pitts, 1943]

$\rightarrow$ Perceptron [Rosenblatt, 1958]

Question: What is the problem with this?

Not differentiable.



(log-)sigmoid: $\phi(t) = 1/(1 + e^{-t})$

Also called the logistic function.

This is what we will use.

Notation

Consider a feed forward architecture with L layers:

- Set of neurons in layer ℓ: $\mathcal{N}_\ell$
- Weight of connection from neuron s to neuron t: $w_{s\to t}$
- Output of a neuron t in layer ℓ:

$$x_t = \phi\Big(\sum_{s\in\mathcal{I}(t)} w_{s\to t}\, x_s\Big) = \phi(a_t),$$

  where $\mathcal{I}_t \subseteq \mathcal{N}_{\ell-1}$ is the set of neurons feeding into t (in a fully connected feedfoward network $\mathcal{I}_t = \mathcal{N}_{\ell-1}$), and

- $a_t = \sum_{s\in\mathcal{I}(t)} w_{s\to t}\, x_s$.

General principle of training NNs

- Present training examples one by one (online learning).
- The error on an example $(\chi, \mathbf{y})$ is some function of the difference between the desired and actual output of the last layer, e.g., for a multivariate regression task

$$\mathcal{E}((\chi, \mathbf{y})) = \frac{1}{2} \sum_{i=1}^{d} (x_{\tau(i)} - y_i)^2,$$

  where $\tau(i)$ is the index of the output neuron that is supposed to predict y_i (the unusual notation χ is because now the x_i's are the ***outputs*** of neurons).
- Adjust each weight of each neuron in each layer by gradient descent

$$w_{s \to t} \leftarrow w_{s \to t} - \eta \, \frac{\partial \mathcal{E}}{\partial w_{s \to t}},$$

  where η is a parameter called the **learning rate**.

Training a neuron in the last layer

Assuming the squared error loss function for regression,

$$\frac{\partial \mathcal{E}}{\partial w_{s\to t}} = \underbrace{\frac{\partial \mathcal{E}}{\partial a_t}}_{\delta_t} \underbrace{\frac{\partial a_t}{\partial w_{s\to t}}}_{x_s} \qquad \text{where} \qquad \delta_t = (x_t - y_i)\,\phi'(a_t)$$

where $t = \tau(i)$, and ϕ' is the derivative of ϕ. So the update rule is

$$w_{s\to t} \leftarrow w_{s\to t} - \eta\,\delta_t\,x_s.$$

Similar to the percepetron. Note that for $\phi(u) = 1/(1 + e^{-u})$,

$$\phi'(u) = \frac{d}{du}\frac{1}{1+e^{-u}} = -\frac{-e^{-u}}{(1+e^{-u})^2} = \phi(u)\,(1 - \phi(u)).$$

Training a neuron in another layer

Once again, for a neuron t in some layer $\ell < L$,

$$\frac{\partial \mathcal{E}}{\partial w_{s\to t}} = \delta_t \, x_s \qquad \text{where} \qquad \delta_t = \frac{\partial \mathcal{E}}{\partial a_t}$$

The key observation is that letting $\mathcal{O}(t) = \{\, u \in \mathcal{N}_{\ell+1} \mid t \in \mathcal{I}(u) \,\}$ the set of neurons that t feeds into,

$$\delta_t = \frac{\partial \mathcal{E}}{\partial a_t} = \sum_{u\in\mathcal{O}(t)} \underbrace{\frac{\partial \mathcal{E}}{\partial a_u}}_{\delta_u} \frac{\partial a_u}{\partial a_t} = \sum_{u\in\mathcal{O}(t)} \delta_u \, w_{t\to u} \, \phi'(a_t) = \phi'(a_t) \sum_{u\in\mathcal{O}(t)} w_{t\to u} \, \delta_u$$

Backpropagation

The general scheme is as follows:

1. For $\ell = L$, set

$$\delta_t = (x_{\tau(i)} - y_i)\,\phi'(a_t) \qquad \forall\, t \in \mathcal{N}_\ell.$$

2. For $\ell = L-1,\ L-2,\ \ldots, 1$ set

$$\delta_t = \phi'(a_t) \sum_{u \in \mathcal{O}(t)} w_{t\to u}\,\delta_u \qquad \forall\, t \in \mathcal{N}_\ell.$$

3. Update the weights

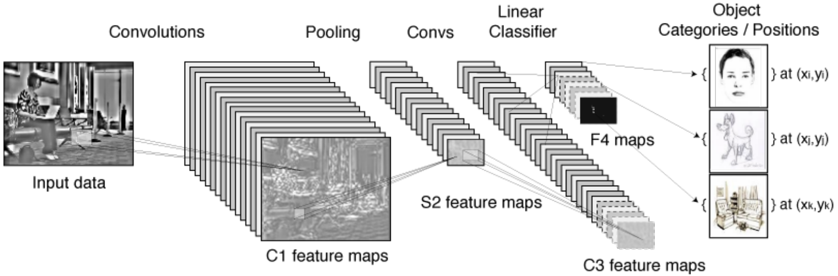
$$w_{s\to t} \leftarrow w_{s\to t} - \eta\,\delta_t\,x_s.$$

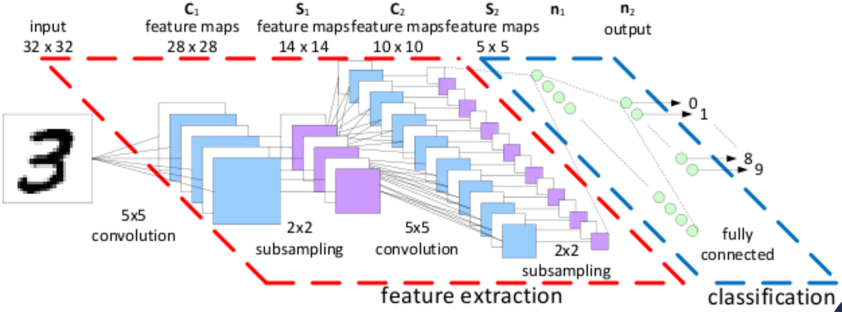
Super simple! [Rumelhart & Hinton, 1986]

Pros and cons of multilayer ANNs

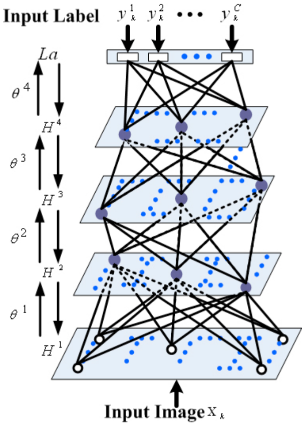
- Any Boolean fun can be learned with an ANN with a single hidden layer, but might need an exponential number of hidden units.
- Any bounded continuous function can be approximated with arbitrarily small error by one hidden layer.
- Any function can be approximated with arbitrarily small error by a network with two hidden layers.
- ANNs are not immune to overfitting.
- Learning rate can be hard to fit.
- Backpropagation is prone to local minima.

Convolutional Neural Nets





Deep neural networks



- backpropagation requires labeled training data and can get stuck in poor local optima
- learning time does not scale well

Solution: adjust the weights to maximize the probability that a generative model would have produced the input (Y. LeCun, G. Hinton & many more working on this problem)

Summary

- ANNs were one of the earliest ML algorithms.
- They went out of favor in the 90's because algorithms like SVMs can do as well as ANNs but also have a much more clear theoretical basis.
- Still, in highly engineered domains ANNs can perform very well and are used in various applications (reading checks, controlling robots, etc).
- ANNs have made a comeback recently in the form of deep learning.