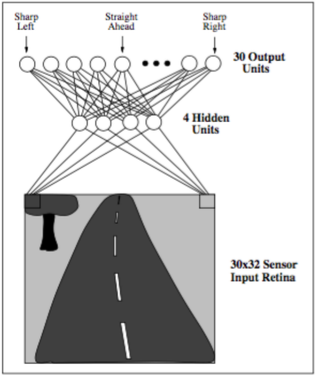
Multilayer Perceptrons

Nan Ye

School of Mathematics and Physics
The University of Queensland

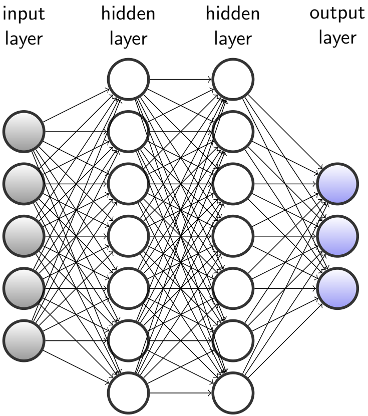
ALVINN Driving at 70 MPH (1993)



Pomerleau, Knowledge-based training of artificial neural networks for autonomous robot driving, 1993

- ALVINN (Autonomous Land Vehicle In Neural Networks) is an early autonomous driving system.
- It learns a neural network (specifically, a multilayer perceptron with a single hidden layer) to map a camera image to a steering decision.

Multilayer Perceptron (MLP)

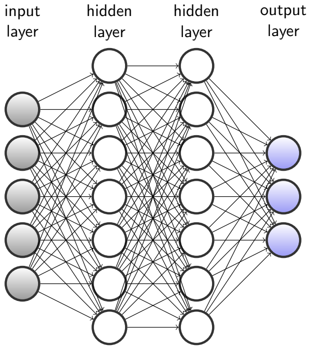


Structure of an MLP

- The perceptron and the Adaline are the simplest kinds of MLPs.
- An MLP is also known as a multilayer feedforward neural network
    - in a feedforward neural network, the connections do not form cycles (note that each connection points from the input neuron to the output neuron).
    - in a multilayer network, the neurons are grouped into different layers
- The depth or the number of layers is the number of all layers with tunable parameters (i.e. all layers except the input layer).
- An MLP can be seen as a series of complex transformations.

Naming the layers and neurons

- The input layer is also called the first/bottom layer, and neurons in it are called input neurons/units.
- The output layer is also called the last/top layer, and neurons in it are called output neurons/units.
- Layers between the input and the output layers are called hidden layers, and neurons in them are called hidden neurons/units.
- A neural with more than one hidden layer is called a deep neural network.



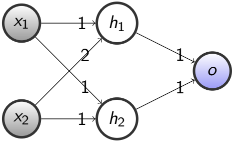
- This is a 3 layer MLP, or a 2 hidden layer MLP.
- There are 5 input units, 7 hidden units for each of the two hidden layers, and 3 output units.

Activation function

- Each neuron applies a function to transform the weighted input sum to an output.
- This function is called the transfer function or the activation function.
- The sigmoid activation function $\sigma(\cdot)$ is defined by $\sigma(u) = \frac{1}{1+e^{-u}}$.
  - for a vector $\mathbf{u} = (u_1, \ldots, u_d)$, we shall use $\sigma(\mathbf{u})$ to denote $(\sigma(u_1), \ldots, \sigma(u_d))$, that is, we apply the sigmoid function to each component of $\mathbf{u}$.
- Another commonly used activation function is the rectifier $(u)_+ = \max(0, u)$. A linear unit using the rectifier activation is called a ReLU (rectified linear unit).

An MLP Example

- Consider the following MLP, with sigmoid hidden units and identity output activation, and weights shown on the edges.



- Then the output o is obtained using the following computation

$$\begin{pmatrix} h_1 \\ h_2 \end{pmatrix} = \sigma\left(W_1 \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}\right), \qquad o = W_2 \begin{pmatrix} h_1 \\ h_2 \end{pmatrix},$$

where $W_1 = \begin{pmatrix} w_{1,11} & w_{1,12} \\ w_{1,21} & w_{1,22} \end{pmatrix} = \begin{pmatrix} 1 & 2 \\ 1 & 1 \end{pmatrix}$, and
$W_2 = \begin{pmatrix} w_{2,1} & w_{2,2} \end{pmatrix} = \begin{pmatrix} 1 & 1 \end{pmatrix}$.

- The function computed by the network can be written as

$$o = W_2\sigma\left(W_1\begin{pmatrix}x_1\\x_2\end{pmatrix}\right) = \frac{1}{1+e^{-x_1-2x_2}} + \frac{1}{1+e^{-x_1-x_2}}.$$

- When $x_1 = 1$, $x_2 = 1$, we have

$$\begin{pmatrix}h_1\\h_2\end{pmatrix} = \begin{pmatrix}\sigma(3)\\\sigma(2)\end{pmatrix}, \quad o = h_1 + h_2 = \sigma(3) + \sigma(2) \approx 1.83$$

- Assume that the observed output for the input $(1,1)$ is $y = 2$, and we want to minimize the squared error $L = (o - y)^2$.
- For gradient-based learning, we want to compute the gradient of L wrt the network weights W_1 and W_2.
- For $\frac{\partial L}{\partial w_{2,1}}$, using the chain rule

$$\frac{\partial L}{\partial w_{2,1}} = \frac{\partial L}{\partial o} \frac{\partial o}{\partial w_{2,1}} = 2(o - y) h_1.$$

- Derivatives like $\frac{\partial L}{\partial w_{1,11}}$ are much more complex.
- We see that even for this small MLP, it is tedious to compute the gradient of the error function.

Backpropagation

- The backpropagation algorithm provides an efficient way to compute the gradient of the error function of a feedforward neural net, which is essential in gradient-based learning.
- The algorithm performs a forward pass and a backward pass through the neural net
  - the forward pass propagates information from the input neurons to the output neurons to compute the outputs of all neurons
  - the backward pass propagates information from the output neurons to the input neurons to compute derivatives

- We illustrate the backpropagation algorithm on an MLP $f(\mathbf{x};\mathbf{w})$
  - all hidden units are sigmoid units
  - there is one output neuron with identity activation function
  - the loss is the squared error (strictly speaking, 1/2 squared error)

$$L = \sum_i \frac{1}{2}(f(\mathbf{x}_i;\mathbf{w}) - y_i)^2.$$

- Notations
  - $P(j)$: the set of parents of unit j.
  - o_i: the output of unit i. For an input neuron, o_i denotes its input.
  - w_{ij}: weight on the connection from unit i to unit j.

Forward propagation

For each neuron j,

$$o_j \leftarrow \begin{cases} \sigma(\sum_{i \in P(j)} w_{ij} o_i), & \text{if } j \text{ is not the output neuron.} \\ \sum_{i \in P(j)} w_{ij} o_i, & \text{if } j \text{ is the output neuron.} \end{cases}$$

when all input o_i's have been computed.

- we don't need to keep the neurons waiting for their inputs to be ready.
- instead, we compute the outputs one layer at a time from the input layer to the output layer (as illustrated in the small numerical example).

The backpropagation algorithm

- We need to compute the derivative g_{ij} of the error function wrt to each weight w_{ij}
- We only need to figure out how to do this for one example $(\mathbf{x}, y)$
  - if there are multiple examples, the gradient is the sum of the individual gradients computed on these examples

1: Compute all o_i's.

2: For the output unit k,

$$\delta_k \leftarrow (o_k - y).$$

3: For each hidden unit i,

$$\delta_i \leftarrow o_i(1 - o_i) \sum_{j \in C(i)} w_{ij}\delta_j$$

when all input δ_j's have been computed.

4: For each connection (i, j),

$$g_{ij} \leftarrow \delta_j o_i.$$

Derivation (optional)

- Notations
  - $L(o_k, y) = \frac{1}{2}(o_k - y)^2$ is the loss function (neuron k is output).
  - $s_j = \sum_{i \in P(j)} w_{ij} o_i$ is the weighted input sum for neuron j.
  - $\delta_i = \partial L / \partial s_i$.
- For the output unit k,

$$\delta_k = \frac{\partial L}{\partial s_k} = \frac{\partial L}{\partial o_k} \frac{\partial o_k}{\partial s_k} = (o_k - y).$$

This is because $o_k = s_k$ (identity activation).

- Using the chain rule, we have

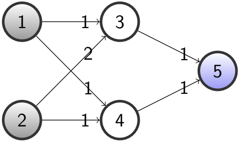
$$\delta_i = \frac{\partial L}{\partial s_i} = \sum_{j \in C(i)} \frac{\partial L}{\partial s_j} \frac{\partial s_j}{\partial o_i} \frac{\partial o_i}{\partial s_i} = \sum_{j \in C(i)} \delta_j w_{ij} o_i (1 - o_i).$$

  This is because $o_i = \sigma(s_i)$ and $\sigma'(s_i) = o_i(1 - o_i)$.
- In addition, we have

$$\frac{\partial L}{\partial w_{ij}} = \frac{\partial L}{\partial s_j} \frac{\partial s_j}{\partial w_{ij}} = \delta_j o_i.$$

Finishing off the small MLP example

- We label the nodes as follows



- We first compute the o_i's, then δ_i's, and finally g_{ij}.

i	o_i	δ_i
1	1	-
2	1	-
3	$\sigma(3)$	$\delta_5 w_{35} o_3(1-o_3)$
4	$\sigma(2)$	$\delta_5 w_{45} o_4(1-o_4)$
5	$\sigma(3)+\sigma(2)$	$o_5 - 1$

(i,j)	g_{ij}
$(1,3)$	$o_1\delta_3$
$(2,3)$	$o_2\delta_3$
$(1,4)$	$o_1\delta_4$
$(2,4)$	$o_2\delta_4$
$(3,5)$	$o_3\delta_5$
$(4,5)$	$o_4\delta_5$

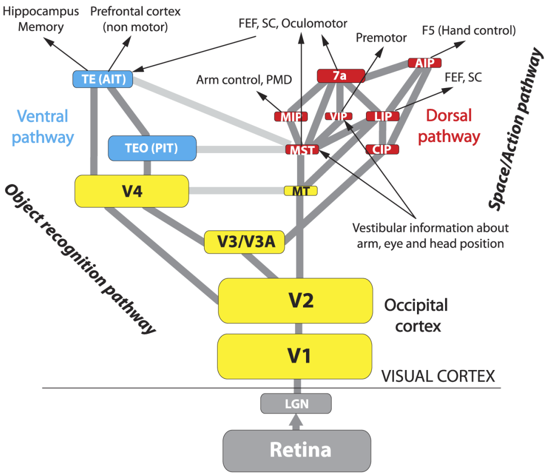
Extensions

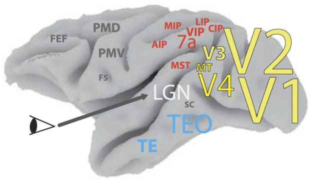
- We can extend the backpropagation algorithm to handle different loss functions, activation functions and multiple output units.
- By choosing different loss functions and using multiple output neurons, we can train an MLP for classification and density estimation.

Why Deep Architectures?

- It is known that any function can be approximated arbitrarily well by a single hidden layer MLP (universal approximation theorems).
- Why do we still need to care about deep neural networks?

Inspiration from Nature





The primate visual cortex is hierarchical

Kruger, Janssen, Kalkan, Lappe, Leonardis, Piater, Rodriguez-Sanchez, and Wiskott, Deep hierarchies in the primate visual cortex: What can we learn for computer vision?, 2013

Deeper Can Be More Compact

Representational power of neural nets

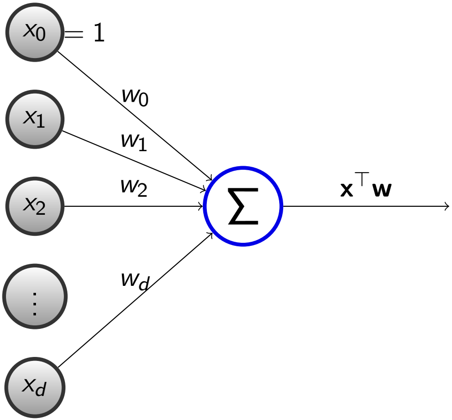
- Every boolean function can be represented by network with single hidden layer but might require exponential (in number of inputs) hidden units.
- Every bounded continuous function can be approximated with arbitrarily small error, by network with one hidden layer.
- Any function can be approximated to arbitrary accuracy by a network with two hidden layers.

Deeper can be more compact

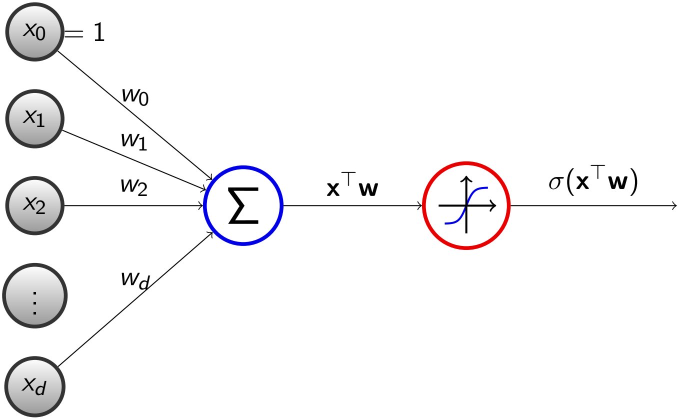
- When a function can be compactly represented by a deep network, it may need a very large shallow network to represent it.
- *E.g. There are functions computable with a depth k network consisting of a polynomially many perceptron units that require exponentially many perceptron units when using a depth $k-1$ network.*

Features: Engineering to Learning

Traditional models as neural nets



Least squares regression $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x}$

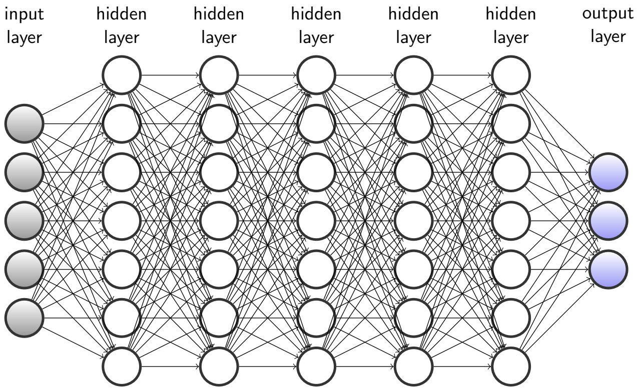


Logistic regression $f(\mathbf{x}) = p(y = 1 \mid \mathbf{x}) = \frac{1}{1+e^{-\mathbf{w}^\top \mathbf{x}}}$

Traditional learning: handcrafted features + classifier learning

- Many other traditional learning algorithms can be seen as neural networks.
- They build classifiers using *handcrafted* features.

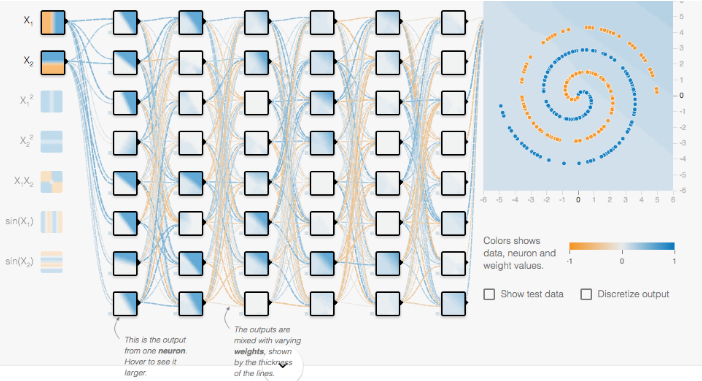
Deep learning: feature learning + classifer learning



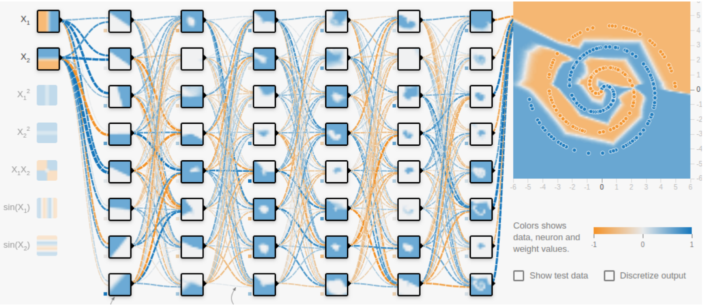
- Deep learning uses deep architectures to additionally learn features.
- Deeper layers build abstract representations of previous layers.

  e.g. pixels $\rightarrow$ *edges* $\rightarrow$ *noses, eyes, ears* $\rightarrow$ *face*

A Demo



- We want to distinguish points on two spirals.
- Each unit can be visualized by drawing a heat map for its output.
- Try different # of hidden layers: 1, 2, 3, 4, 5, 6.



- This trained 6-layer MLP is able to learn fairly complex decision boundaries.
- While neurons in shallow layers represent simple features (e.g. straight lines), neurons in deeper layers pick up useful high-level features (e.g. parts of the spirals).

Your Turn

Which of the following statement is correct? (Multiple choice)

(a) In a multilayer perceptrons, neurons are organized into several layers with connections between adjacent layers only.

(b) Backpropagation allows efficient computation of the gradient of the loss function of an MLP wrt the network parameters in a recursive manner.

(c) Deep neural nets can possibly learn complex features.

What You Need to Know...

- Multilayer perceptrons (aka multilayer feedforward networks)
  - Specifying an MLP: structure and activation function
  - Forward propagation (compute output for a given input)
  - Backpropagation for gradient computation
- Motivations for deep networks
  - Inspiration from nature
  - Deeper can be more compact
  - Replacing feature engineering by feature learning