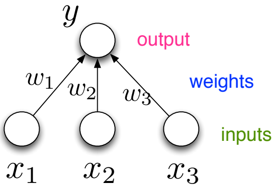
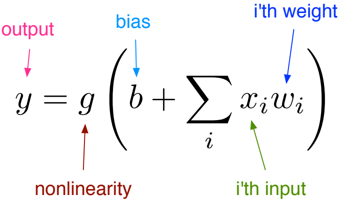
CSC321 Lecture 5: Multilayer Perceptrons

Roger Grosse

Overview

- Recall the simple neuron-like unit:





$$y = g\left(b + \sum_i x_i w_i\right)$$

- These units are much more powerful if we connect many of them into a neural network.

Overview

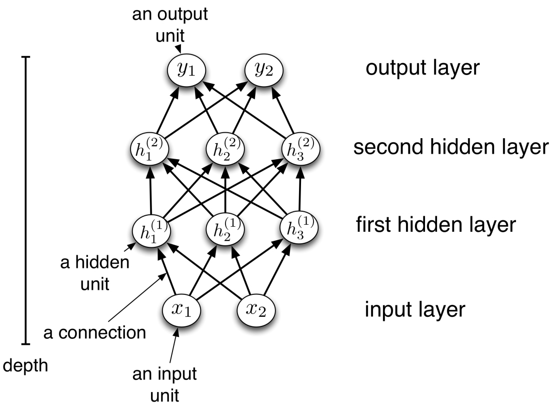
Design choices so far

- **Task:** regression, binary classification, multiway classification
- **Model/Architecture:** linear, log-linear, feed-forward neural network
- **Loss function:** squared error, 0–1 loss, cross-entropy, hinge loss
- **Optimization algorithm:** direct solution, gradient descent, perceptron

Multilayer Perceptrons

- We can connect lots of units together into a directed acyclic graph.
- This gives a feed-forward neural network. That's in contrast to recurrent neural networks, which can have cycles. (We'll talk about those later.)
- Typically, units are grouped together into layers.

hidden: learnt



width of layer: number of unit in a layer

Multilayer Perceptrons

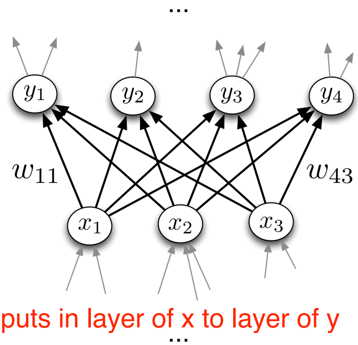
- Each layer connects N input units to M output units.
- In the simplest case, all input units are connected to all output units. We call this a fully connected layer. We'll consider other layer types later.
- Note: the inputs and outputs for a layer are distinct from the inputs and outputs to the network.

1 entry in weight matrix for each unit pairs

- Recall from multiway logistic regression: this means we need an $M \times N$ weight matrix.
- The output units are a function of the input units:

$$\mathbf{y} = f(\mathbf{x}) = \phi(\mathbf{W}\mathbf{x} + \mathbf{b})$$

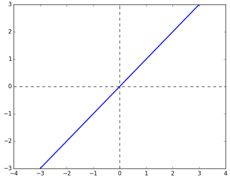
- A multilayer network consisting of fully connected layers is called a multilayer perceptron. Despite the name, it has nothing to do with perceptrons!



W is 4x3, inputs in layer of x to layer of y

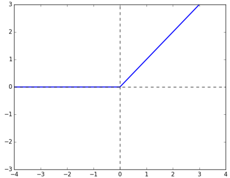
Some activation functions:

equivalent to no activation
linear regression



Linear

$y = z$

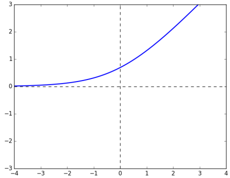


clip -> no negative

Rectified Linear Unit (ReLU)

$y = \max(0, z)$

make ReLU differentiable at 0
same form as logistic + CE



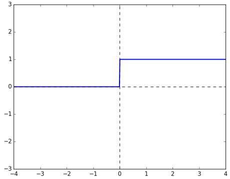
Soft ReLU

$y = \log 1 + e^z$

Multilayer Perceptrons

Some activation functions:

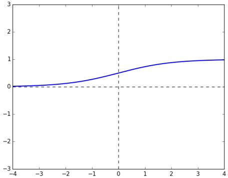
equivalent to binary classification



Hard Threshold

$$y = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{if } z \leq 0 \end{cases}$$

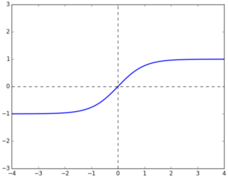
equivalent to logistic regression



soft threshold

Logistic

$$y = \frac{1}{1 + e^{-z}}$$



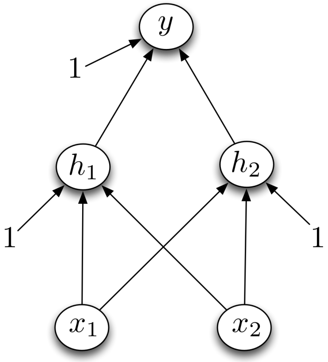
similar to logistic

Hyperbolic Tangent (tanh)

$$y = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

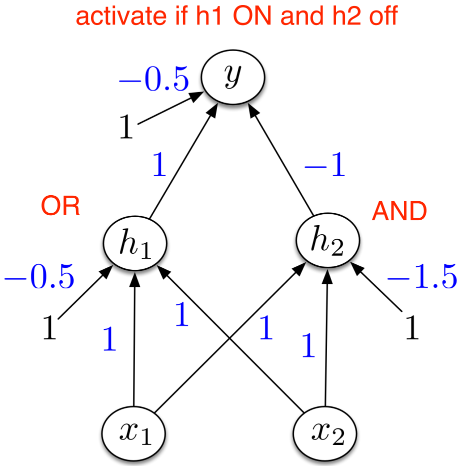
Designing a network to compute XOR:

Assume hard threshold activation function



h1: AND
h2: OR

Multilayer Perceptrons



Multilayer Perceptrons

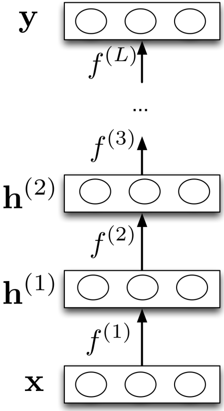
- Each layer computes a function, so the network computes a composition of functions:

$$
\begin{aligned}
\mathbf{h}^{(1)} &= f^{(1)}(\mathbf{x}) \\
\mathbf{h}^{(2)} &= f^{(2)}(\mathbf{h}^{(1)}) \\
&\vdots \\
\mathbf{y} &= f^{(L)}(\mathbf{h}^{(L-1)})
\end{aligned}
$$

- Or more simply:

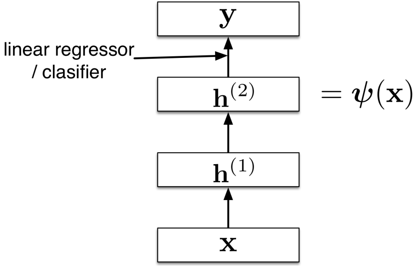
$$
\mathbf{y} = f^{(L)} \circ \cdots \circ f^{(1)}(\mathbf{x}).
$$

- Neural nets provide modularity: we can implement each layer's computations as a black box.



Feature Learning

- Neural nets can be viewed as a way of learning features:

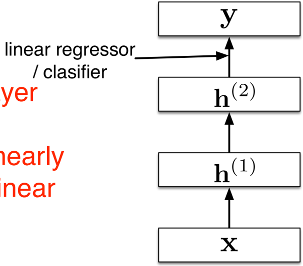


Feature Learning

map input space to feature space
such that the problem is now linearly separable

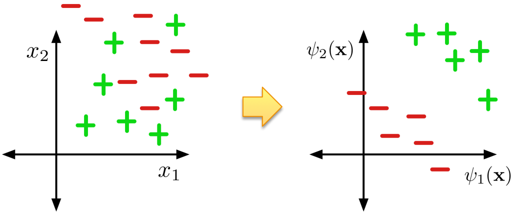
- Neural nets can be viewed as a way of learning features:

idea is the last hidden layer
are functions of inputs,
we the activations are linearly
separate using the last linear
regressor / classifier



$= \psi(\mathbf{x})$ the feature layer

- The goal:



Feature Learning

Input representation of a digit : 784 dimensional vector.

0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	221.0	115.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	51.0	254.0	148.0	0.0	0.0	0.0	0.0	22.0	230.0	134.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	168.0	254.0	115.0	0.0	0.0	0.0	0.0	24.0	253.0	184.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	51.0	234.0	254.0	81.0	0.0	0.0	0.0	0.0	91.0	253.0	184.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	13.0	221.0	254.0	160.0	0.0	0.0	0.0	0.0	0.0	141.0	254.0	127.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	114.0	253.0	253.0	76.0	0.0	0.0	0.0	0.0	0.0	207.0	253.0	93.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	15.0	232.0	253.0	102.0	0.0	0.0	0.0	0.0	0.0	0.0	207.0	253.0	17.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	229.0	253.0	202.0	19.0	0.0	0.0	0.0	0.0	0.0	34.0	240.0	253.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	7.0	170.0	254.0	254.0	46.0	0.0	0.0	0.0	0.0	0.0	0.0	47.0	254.0	254.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	24.0	253.0	254.0	253.0	234.0	163.0	47.0	47.0	26.0	0.0	0.0	130.0	253.0	253.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	21.0	246.0	254.0	253.0	253.0	253.0	254.0	253.0	232.0	174.0	208.0	232.0	253.0	177.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	84.0	161.0	211.0	219.0	219.0	254.0	253.0	253.0	253.0	254.0	253.0	244.0	69.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	93.0	142.0	142.0	93.0	170.0	254.0	230.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	153.0	253.0	213.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	170.0	253.0	137.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	220.0	253.0	137.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	68.0	255.0	254.0	80.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	93.0	254.0	253.0	46.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	51.0	254.0	215.0	9.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	186.0	186.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Feature Learning

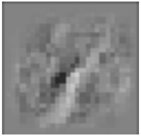
Each first-layer hidden unit computes $\sigma(\mathbf{w}_i^T \mathbf{x})$

Here is one of the weight vectors (also called a feature).

It's reshaped into an image, with gray = 0, white = +, black = -.

To compute $\mathbf{w}_i^T \mathbf{x}$, multiply the corresponding pixels, and sum the result.

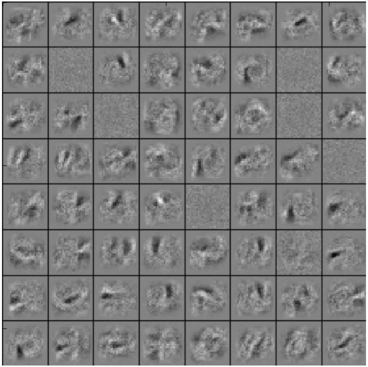
dot product of weights and input image x



Feature Learning

There are 256 first-level features total. Here are some of them.

features are localized



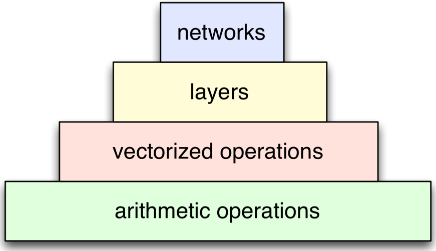
Levels of Abstraction

The psychological profiling [of a programmer] is mostly the ability to shift levels of abstraction, from low level to high level. To see something in the small and to see something in the large.

– Don Knuth

Levels of Abstraction

When you design neural networks and machine learning algorithms, you'll need to think at multiple levels of abstraction.



Expressive Power

- We've seen that there are some functions that linear classifiers can't represent. Are deep networks any better?
- Any sequence of *linear* layers can be equivalently represented with a single linear layer.

$$\mathbf{y} = \underbrace{\mathbf{W}^{(3)}\mathbf{W}^{(2)}\mathbf{W}^{(1)}}_{\triangleq \mathbf{W}'}\mathbf{x}$$

  - Deep linear networks are no more expressive than linear regression!
  - Linear layers do have their uses — stay tuned!

Expressive Power

- Multilayer feed-forward neural nets with *nonlinear* activation functions are universal approximators: they can approximate any function arbitrarily well.
- This has been shown for various activation functions (thresholds, logistic, ReLU, etc.)
    - Even though ReLU is "almost" linear, it's nonlinear enough!

Expressive Power

bias=-.25
output 1 only for this configuration
only 1 hidden unit activates

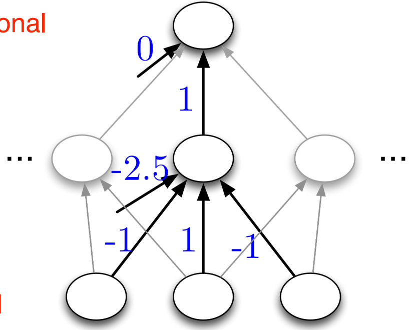
Universality for binary inputs and targets:

- Hard threshold hidden units, linear output
- Strategy: 2^D hidden units, each of which responds to one particular input configuration

requires just 1 really wide hidden year

where input vector is D-dimensional

x_1	x_2	x_3	t
	$\vdots$		$\vdots$
-1	-1	1	-1
-1	1	-1	1
-1	1	1	1
	$\vdots$		$\vdots$

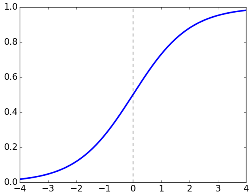


hard threshold

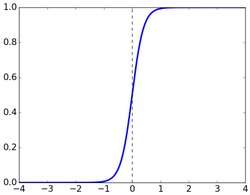
- Only requires one hidden layer, though it needs to be extremely wide!

Expressive Power

- What about the logistic activation function?
- You can approximate a hard threshold by scaling up the weights and biases: so logistic is equally universal as hard threshold with approximation



$$y = \sigma(x)$$



$$y = \sigma(5x)$$

- This is good: logistic units are differentiable, so we can tune them with gradient descent. (Stay tuned!)

Expressive Power

- Limits of universality
  - You may need to represent an exponentially large network.
  - If you can learn any function, you'll just overfit.
  - Really, we desire a *compact* representation!

Expressive Power

- Limits of universality
  - You may need to represent an exponentially large network.
  - If you can learn any function, you'll just overfit.
  - Really, we desire a *compact* representation!
- We've derived units which compute the functions AND, OR, and NOT. Therefore, any Boolean circuit can be translated into a feed-forward neural net.
  - This suggests you might be able to learn *compact* representations of some complicated functions
  - The view of neural nets as "differentiable computers" is starting to take hold. More about this when we talk about recurrent neural nets.