

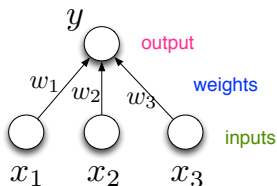
# Multilayer Perceptrons

CS 114  
Spring 2019

Slides from Roger Grosse

# Overview

- Recall the simple neuron-like unit:



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

The equation is annotated with colored arrows: a pink arrow points to  $y$  labeled "output"; a blue arrow points to  $b$  labeled "bias"; a blue arrow points to  $w_i$  labeled "i'th weight"; a green arrow points to  $x_i$  labeled "i'th input"; and a red arrow points to  $g$  labeled "nonlinearity".

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

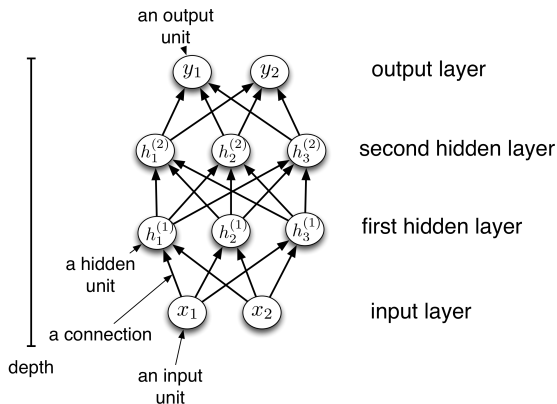
# Overview

## 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

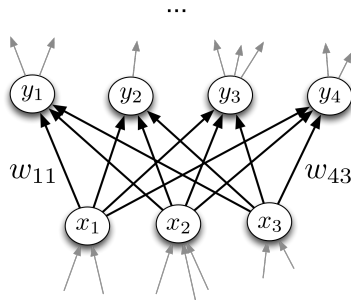
# Multilayer Perceptrons

- 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**.



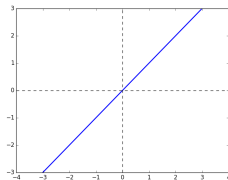
# Multilayer Perceptrons

- 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.
- 
- 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!



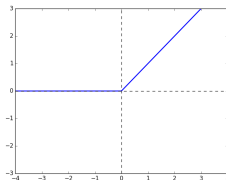
# Multilayer Perceptrons

## Some activation functions:



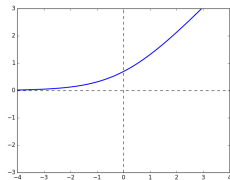
**Linear**

$$y = z$$



**Rectified Linear Unit  
(ReLU)**

$$y = \max(0, z)$$

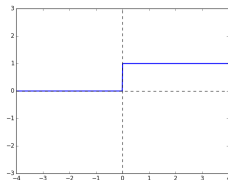


**Soft ReLU**

$$y = \log(1 + e^z)$$

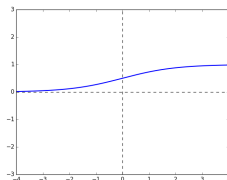
# Multilayer Perceptrons

## Some activation functions:



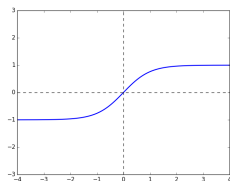
**Hard Threshold**

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



**Logistic**

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



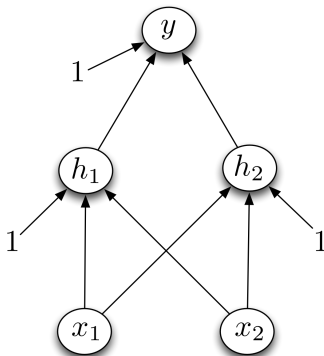
**Hyperbolic Tangent  
(tanh)**

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

# Multilayer Perceptrons

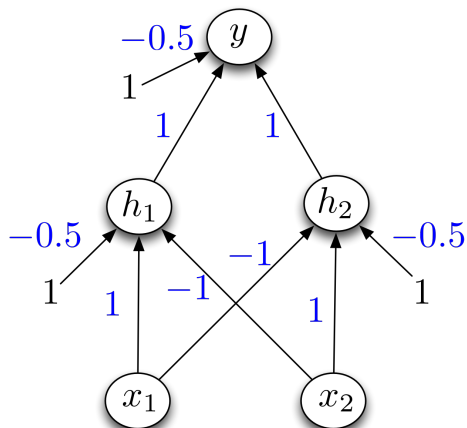
## Designing a network to compute XOR:

Assume hard threshold activation function





# Multilayer Perceptrons



# Multilayer Perceptrons

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

$$\mathbf{h}^{(1)} = f^{(1)}(\mathbf{x})$$

$$\mathbf{h}^{(2)} = f^{(2)}(\mathbf{h}^{(1)})$$

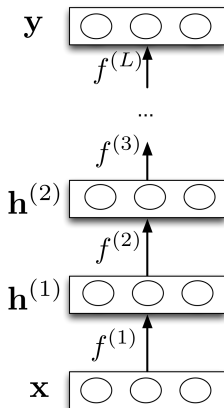
$$\vdots$$

$$\mathbf{y} = f^{(L)}(\mathbf{h}^{(L-1)})$$

- Or more simply:

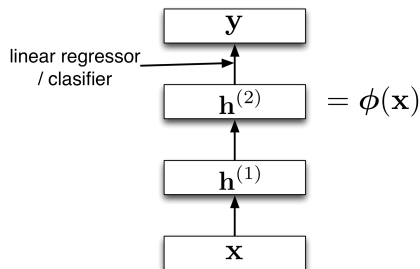
$$\mathbf{y} = f^{(L)} \circ \dots \circ f^{(1)}(\mathbf{x}).$$

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



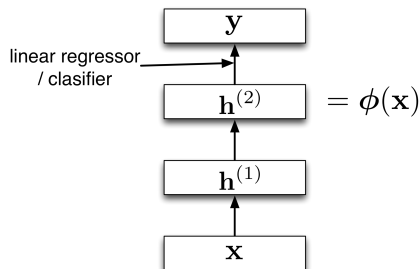
# Feature Learning

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

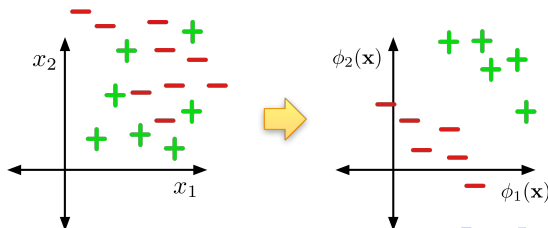


# Feature Learning

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



- The goal:



# Feature Learning

Input representation of a digit : 784 dimensional vector.

[illegible]

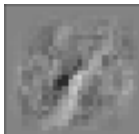
# Feature Learning

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

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

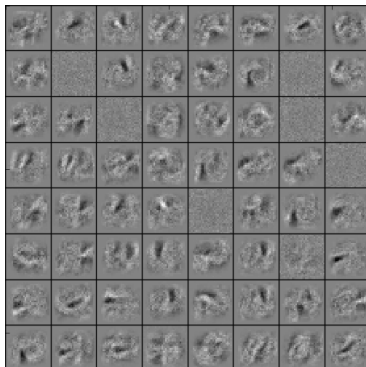
It's reshaped into an image, with gray = 0, white = +, black = -.

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



# Feature Learning

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



# Levels of Abstraction

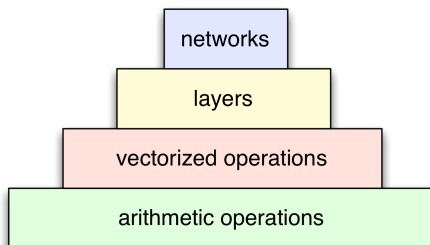
The psychological profiling [of a programmer] is mostly the ability to shift levels of abstraction, from low level to high level. To see something in the small and to see something in the large.

– 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

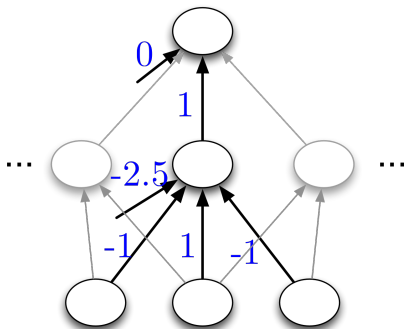
- Multilayer feed-forward neural nets with *nonlinear* activation functions are **universal approximators**: they can approximate any function arbitrarily well.
- This has been shown for various activation functions (thresholds, logistic, ReLU, etc.)
  - Even though ReLU is “almost” linear, it’s nonlinear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

# Expressive Power

## 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

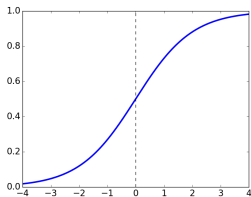
$x_1$	$x_2$	$x_3$	$t$
	$\vdots$		$\vdots$
-1	-1	1	-1
-1	1	-1	1
-1	1	1	1
	$\vdots$		$\vdots$



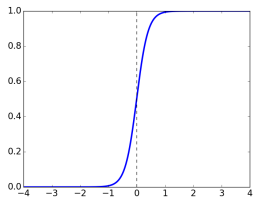
- Only requires one hidden layer, though it needs to be extremely wide!

# Expressive Power

- What about the logistic activation function?
- You can approximate a hard threshold by scaling up the weights and biases:



$$y = \sigma(x)$$



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

- 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.