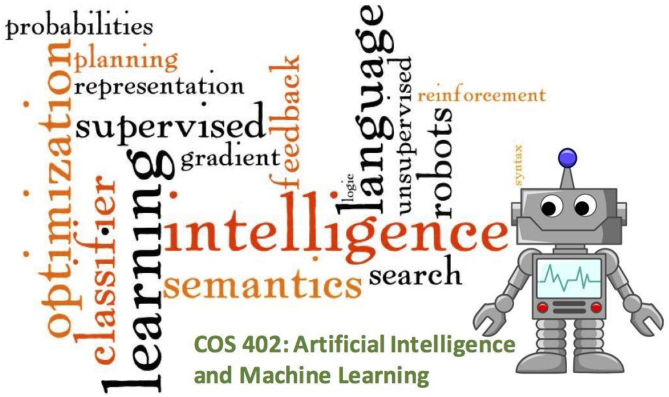


COS 402 – Machine Learning and Artificial Intelligence
Fall 2016

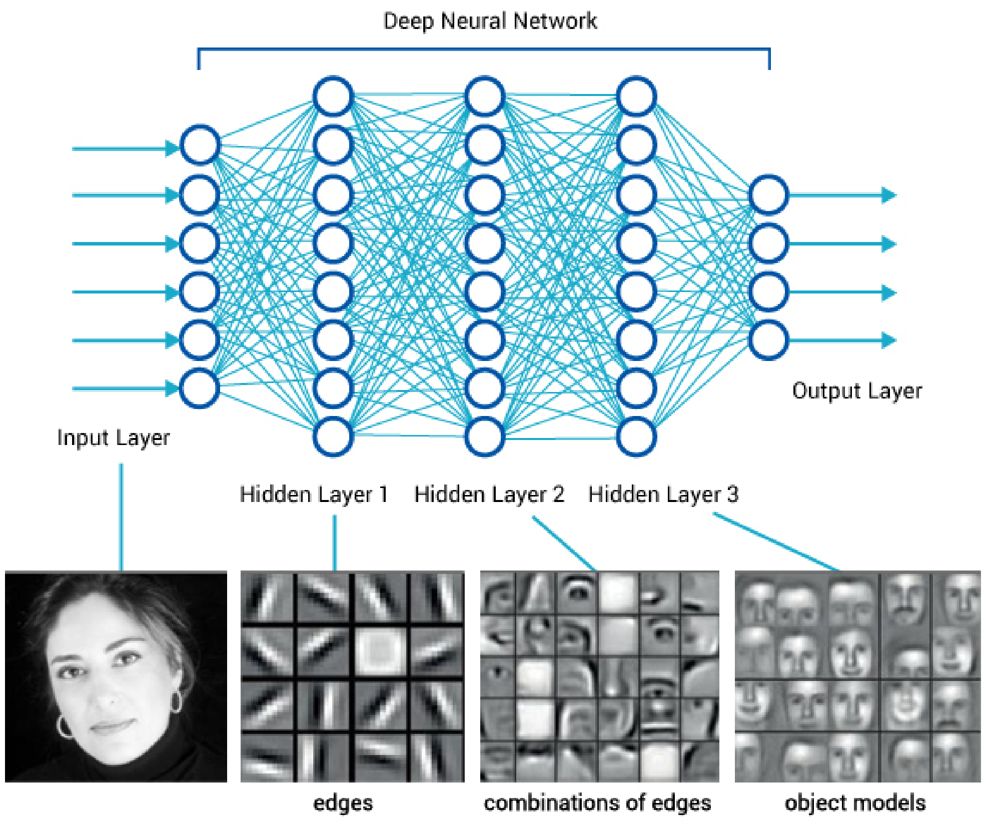
Lecture 7: Introduction to Deep Learning

Sanjeev Arora Elad Hazan

PRINCETON UNIVERSITY

(huge hype –some justified;
dozens of startups)

Deep learning: What is it?



Motivation: Each node learns a “feature”; depends upon lower level nodes.

- Trained using backpropagation algorithm.
- Training leverages highly parallel (vector) operations possible on modern GPUs.

- Features learnt from data turn out superior to hand-crafted features (“learning from data” vs ”introspection” again)

Structure of a deep net:

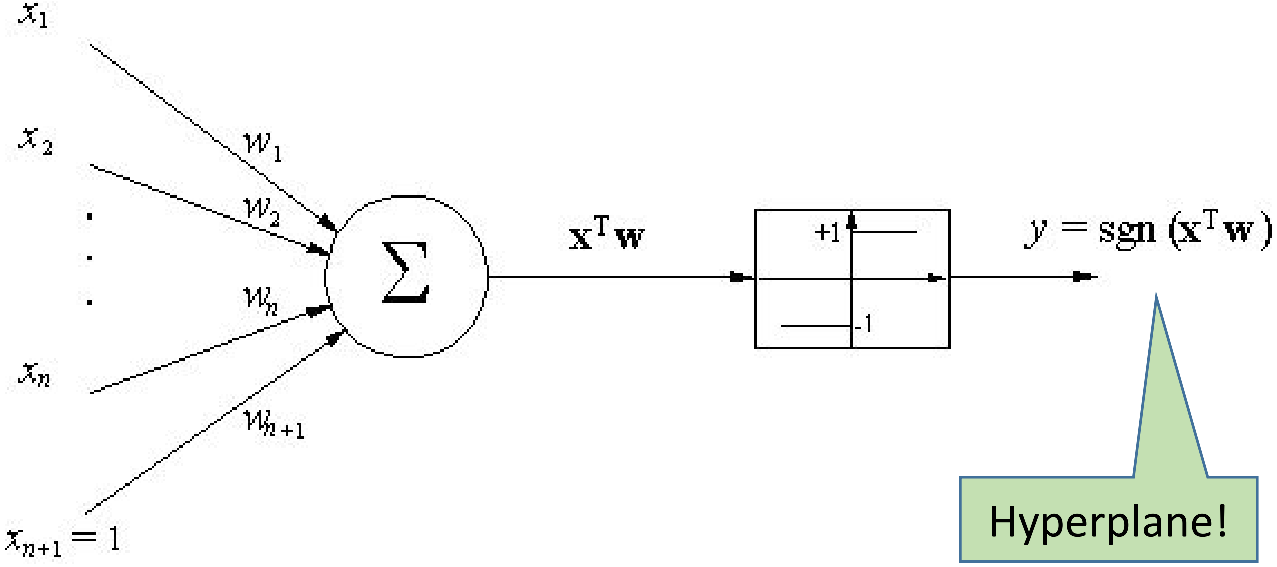
- ”Circuit” of gates connected by wires.
- Each wire has a weight on it.
- Each gate first computes weighted sum of incoming signals, then applies some function on it.

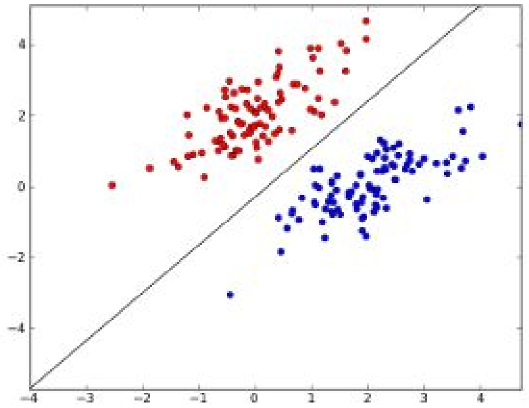
Examples: Quadratic: $f(s) = s^2$.

Threshold function [Mccullough-Pitt 1943]
$T_a(s) = 1$ if s is positive;
$= -1$ else.

Reminds you of something? What is a net with a single threshold gate?

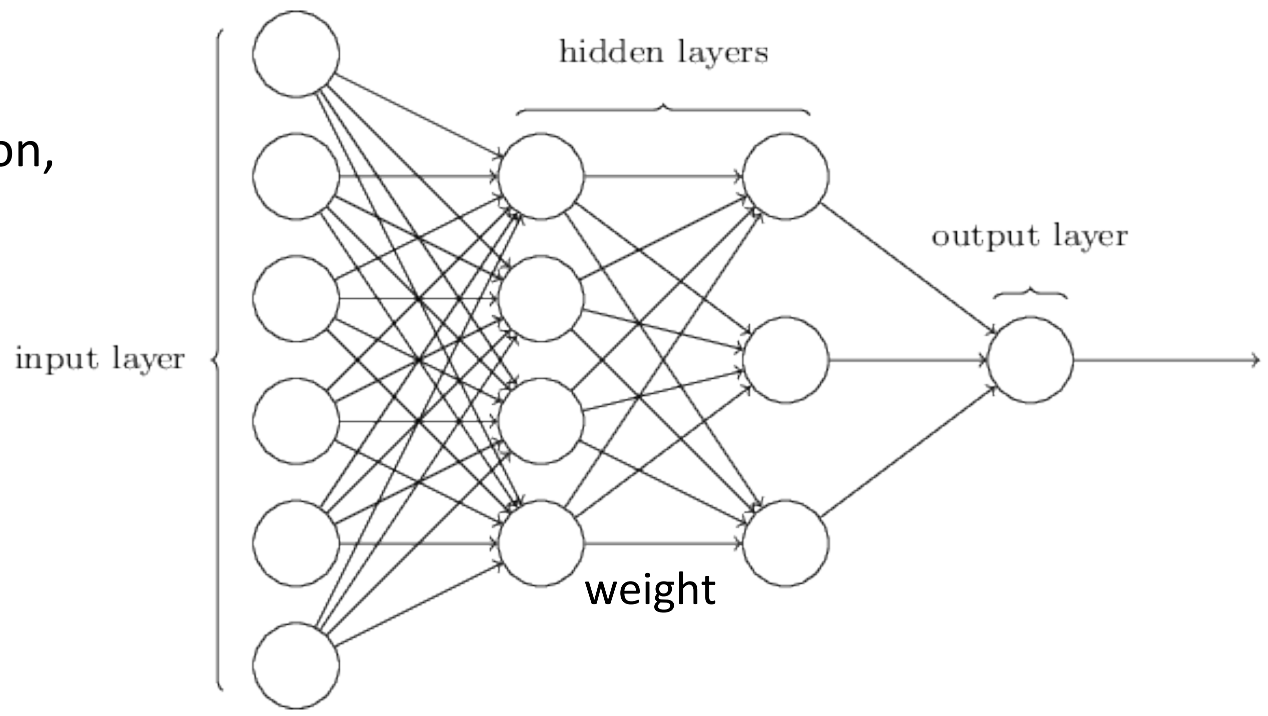
Network with one Threshold Gate.





Structure of a deep net (contd)

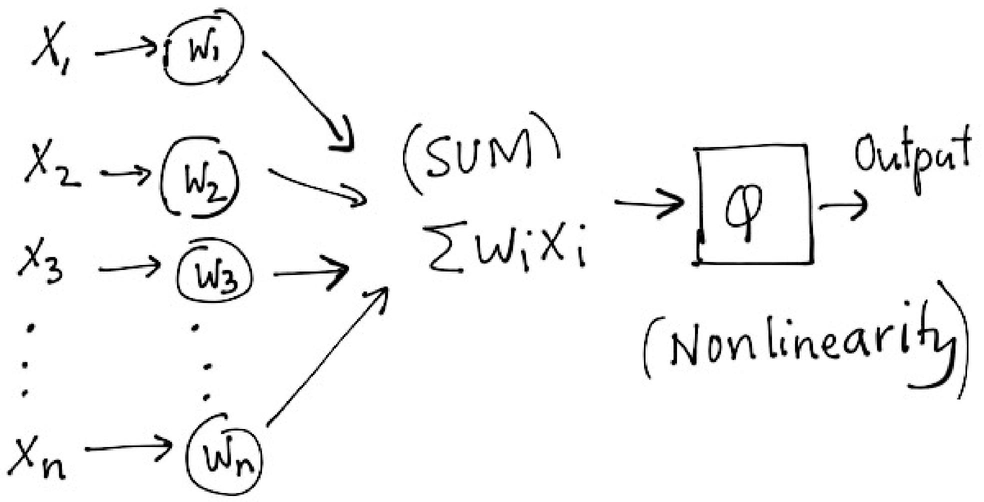
- ”Circuit” of gates connected by wires.
- Each wire has a weight on it.
- Each gate computes a simple nonlinear function, which is applied to weighted sum of incoming signals.



Each gate first computes weighted sum of incoming signals, then applies nonlinear function on it.

Basic structure of a deep net (contd)

- ”Circuit” of gates connected by wires.
- Each wire has a weight on it.
- Each gate computes a simple nonlinear function, which is applied to weighted sum of incoming signals.

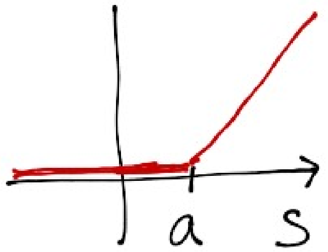


More popular nonlinearities:

- Rectifier Linear Unit (“RELU”).
- Sigmoid (soft threshold)

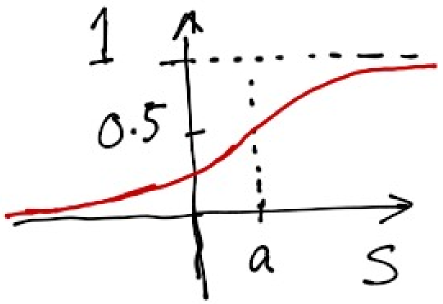
RELU with bias a

$$RELU_a(s) = \max\{0, s-a\}$$

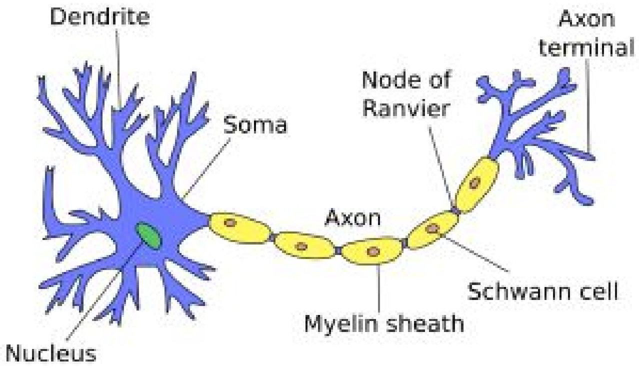


SIGMOID with bias a

$$\sigma_a(s) = \frac{1}{1+e^{-(s-a)}}$$



Brief history of Deep Nets (aka “neural nets)



- Mccullough-Pitt 1943. Threshold gates as simple model for neurons. (Today considered **very** simplistic.)
- Perceptron = network with single threshold gate.
- Backpropagation training algorithm rediscovered independently in many fields starting 1960s. (popularized for Neural net training by Rumelhart, Hinton, Williams 1986)
- Neural nets find some uses in 1970s and 1980s.
- Achieve human level ability to read handwritten digits in 1990s.
- Dominant paradigm for computer vision by 2013 (exceeds human performance 2015)
- Little formal understanding why they work, but their success in various domains (vision, language, speech etc.) has caused a frenzy (tech bubble? AI fears?)

How to train your deep net

Statistical Learning: Recap

N inputs $x_1, x_2,.., x_N$ in R^d, labeled with values $y_1, y_2,..., y_N$ in $\{0,1\}$

SAMPLING

Algorithm designer has in mind a class of models/classifiers
His Algorithm Fits best model in this class to the labeled data.

Under appropriate conditions, this model will "generalize" to random unseen images from Distrib. D.

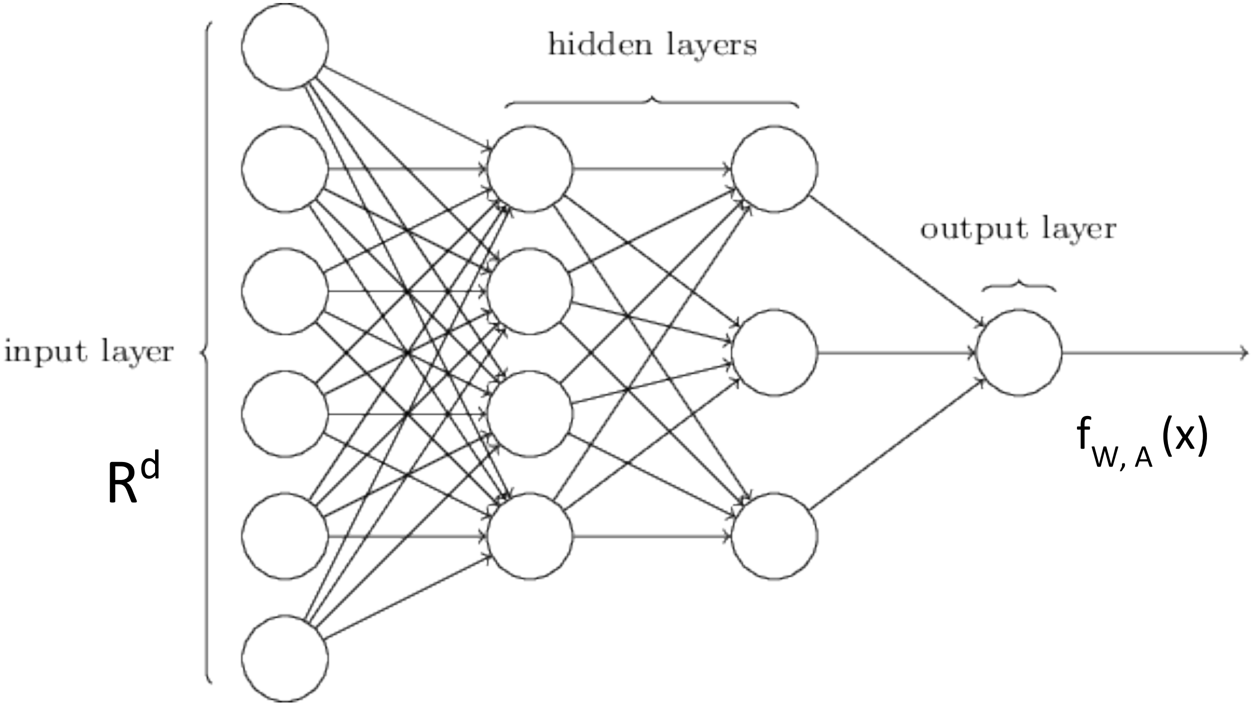
Distribution D on all possible inputs (in this case, images)

The optimization problem

- N inputs $x_1, x_2,.., x_N$ in R^d, labeled with values $y_1, y_2,..., y_N$ in {0,1}
- Experimenter decides on # of layers, # of nodes in each, and the nonlinearity type.
- (W, A) = Vector of unknowns. (Weight of each wire, and "bias" of each node.)

$f_{W, A}(x)$ = output of this net on input x.

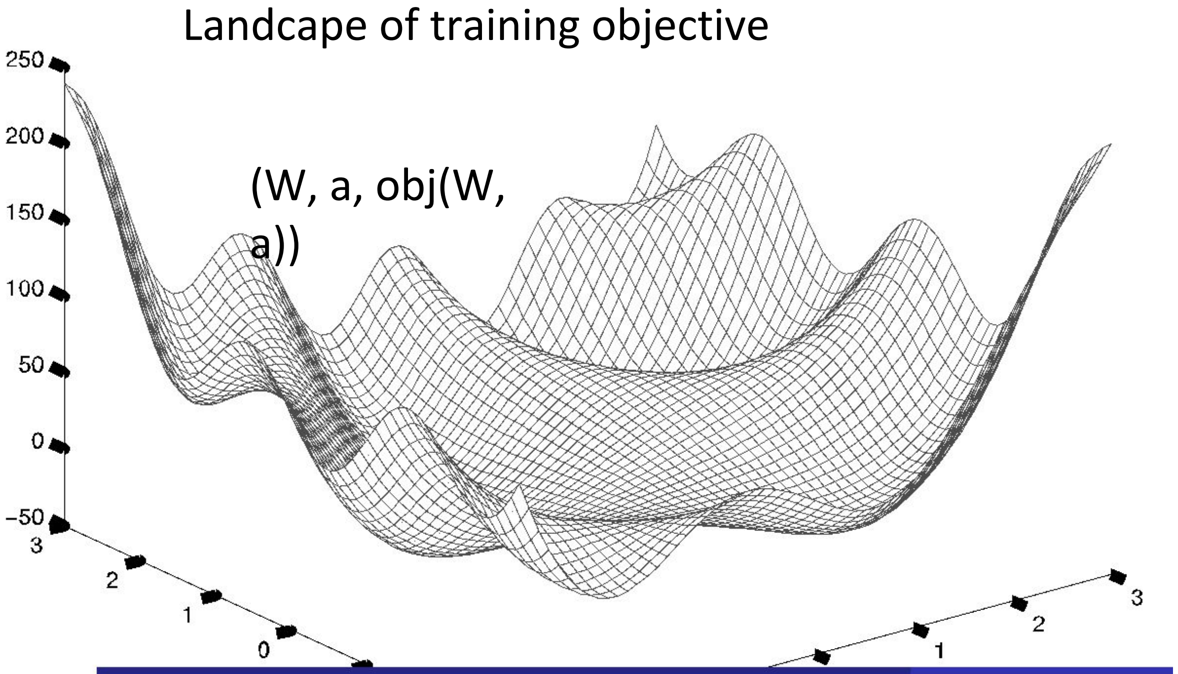
Minimize over (W, A):
$\sum_i (f_{W, A}(x_i) - y_i)^2$ + Regularizer(W, A)



Typical choice of regularizer = sum of squares of entries of W.

Distribution over vectors
$\{x\} \in R^n$

input lay



Chair/car

Output is $f_{W,a}(x)$

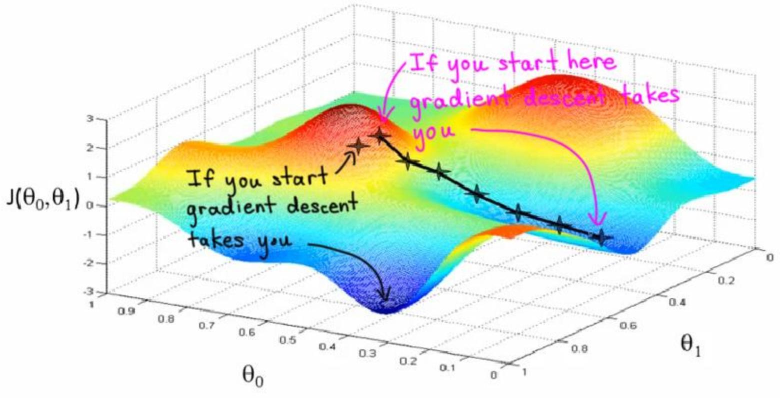
(W, a)= Weights and Biases of network nodes (in vector form)

Recap: Gradient Descent

$$f : \Re \to \Re$$

$$f(x+\delta) \approx f(x) + \delta \frac{df}{dx}.$$

$$(Taylor\ approximation)$$



$$f : \Re^m \to \Re$$

$$f\left((x_1, x_2, \ldots, x_m) + (\delta_1, \delta_2, \ldots, \delta_m)\right) \approx f(x_1, x_2, \ldots, x_m) + \sum_i \delta_i \frac{\partial f}{\partial x_i}$$

$$= f(\vec{x}) + \nabla(f) \cdot \vec{\delta} \quad (in\ vector\ notation)$$

➔ To reduce f, take a tiny step along direction $-\nabla(f)$

If f is nonconvex, not guaranteed to reach global minimum!

Multilayer net: how to compute gradient

Minimize over (W, A):
$\sum_i (f_{W,A}(x_i) - y_i)^2$ + Regularizer(W, A)

Want: Gradient of f() with respect to W, A

Main idea: Chain rule from calculus.

$$\frac{d}{dx} f(g(x)) = f'(g(x)) \frac{d}{dx} g(x)$$

Recall: $f'(g(x)) = \frac{d}{dy} f(y) \text{ evaluated at } y = g(x)$

Example: $\frac{d}{dx} e^{x^2+3x} = e^{x^2+3x}(2x+3)$

Backpropagation

Key points before we dive in (writeups on internet are very confusing!!)

(i) Want computation time O(1) per edge; i.e., O(Network Size) total.

(ii) Reason about correctness using induction.

(iii) We first see the trivial algorithm that runs in $O((\text{Network})^2)$ time. Backprop is more efficient version that uses special form of nonlinearity.

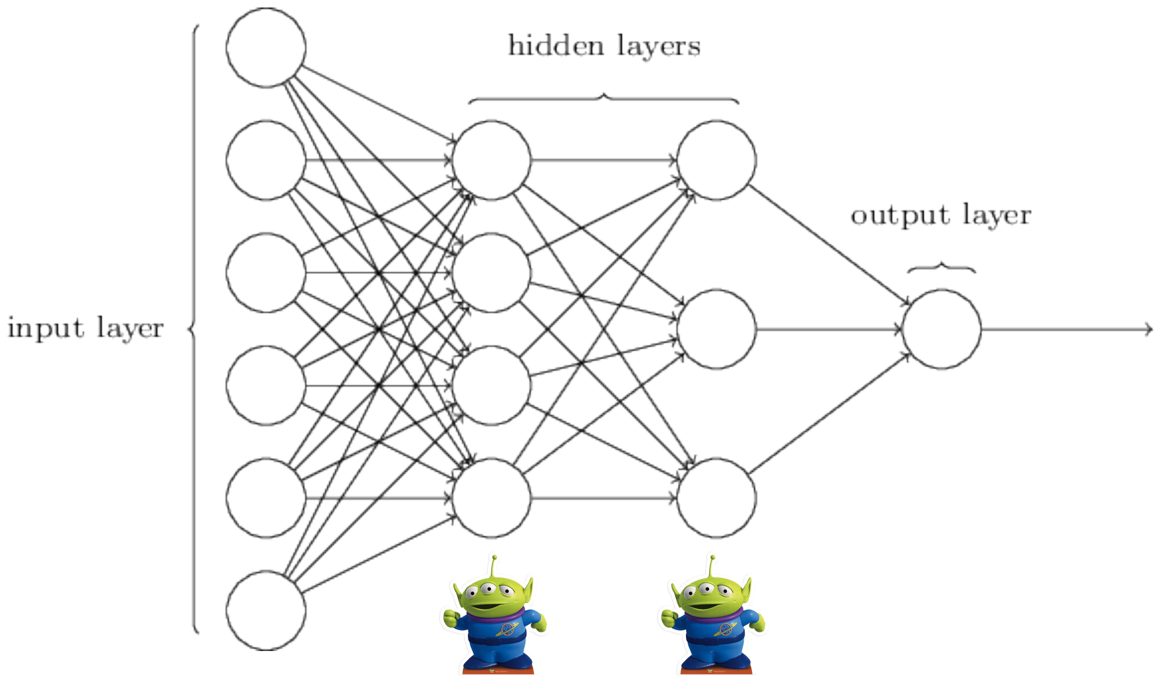
Gradient calculation as message passing

Imagine: On each node, one little green man doing some computation.

Desired: At the end, each edge knows $\frac{\partial f}{\partial w}$ where w is its weight, and f is the function at the last layer.

Ultimate Goal: Work done per node is O(# of adjacent edges).

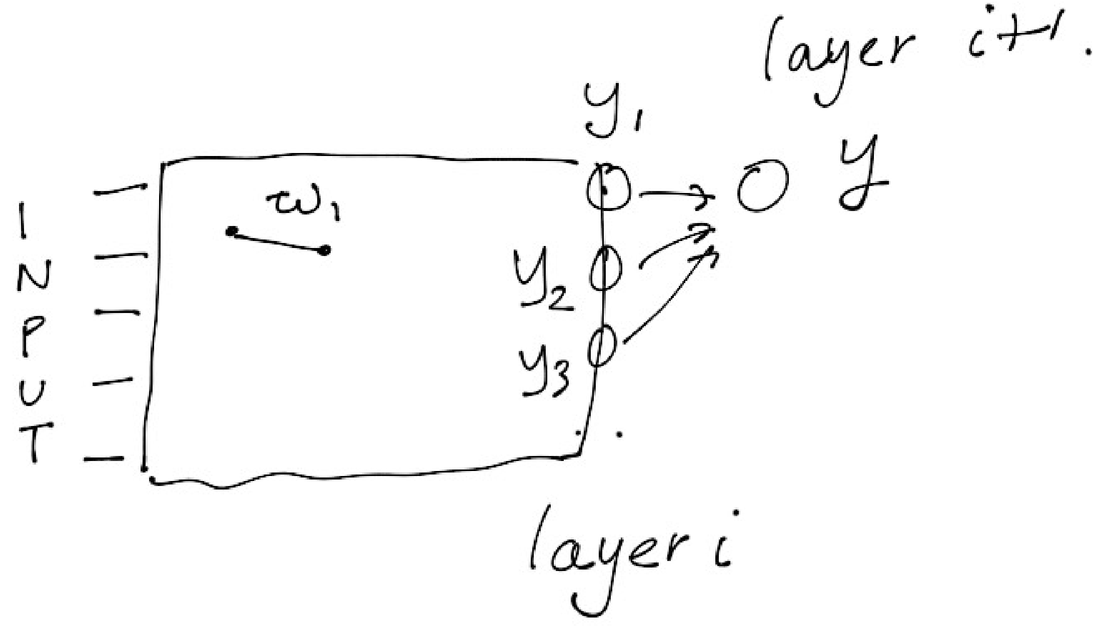
➔ Total work by all green men = O(Network Size).



Simple inductive algorithm to compute $\partial y / \partial w_1$ for all nodes y in the network.

Running time is O(network size).

To compute gradient, need to do this for all network parameters w_1, hence running time is $O((\text{network size})^2)$



$$y = \sigma(\alpha_1 y_1 + \alpha_2 y_2 + \ldots \alpha_m y_m)$$

$$\Rightarrow \frac{\partial y}{\partial w_1} = \sigma'(\alpha_1 y_1 + \ldots + \alpha_m y_m) \times \left(\alpha_1 \frac{\partial y_1}{\partial w_1} + \ldots + \alpha_m \frac{\partial y_m}{\partial w_1}\right)$$

Main idea to improve:
Lots of identical operations in above; consolidate!

Next lecture: (i) Finish backprop and details of training.
(ii) Using deep nets for computer vision (image recognition)