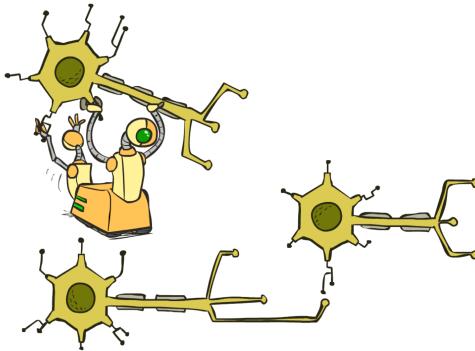


CS 188: Artificial Intelligence

Optimization and Neural Nets

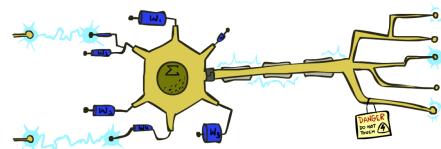


Instructors: Pieter Abbeel and Dan Klein --- University of California, Berkeley

[These slides were created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley. All CS188 materials are available at <http://ai.berkeley.edu>.]

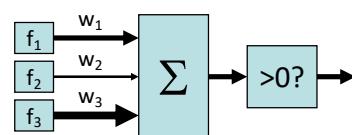
Reminder: Linear Classifiers

- Inputs are **feature values**
- Each feature has a **weight**
- Sum is the **activation**



$$\text{activation}_w(x) = \sum_i w_i \cdot f_i(x) = w \cdot f(x)$$

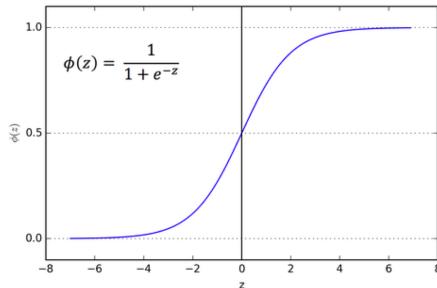
- If the activation is:
 - Positive, output +1
 - Negative, output -1



How to get probabilistic decisions?

- Activation: $z = w \cdot f(x)$
- If $z = w \cdot f(x)$ very positive → want probability going to 1
- If $z = w \cdot f(x)$ very negative → want probability going to 0
- Sigmoid function

$$\phi(z) = \frac{1}{1 + e^{-z}}$$



Best w?

- Maximum likelihood estimation:

$$\max_w ll(w) = \max_w \sum_i \log P(y^{(i)} | x^{(i)}; w)$$

with:

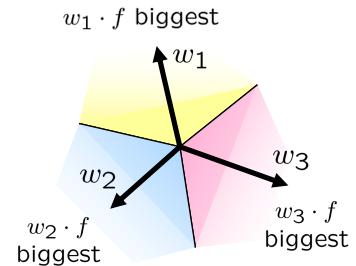
$$P(y^{(i)} = +1 | x^{(i)}; w) = \frac{1}{1 + e^{-w \cdot f(x^{(i)})}}$$
$$P(y^{(i)} = -1 | x^{(i)}; w) = 1 - \frac{1}{1 + e^{-w \cdot f(x^{(i)})}}$$

= Logistic Regression

Multiclass Logistic Regression

- Multi-class linear classification

- A weight vector for each class: w_y
- Score (activation) of a class y : $w_y \cdot f(x)$
- Prediction w/highest score wins: $y = \arg \max_y w_y \cdot f(x)$



- How to make the scores into probabilities?

$$z_1, z_2, z_3 \rightarrow \underbrace{\frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3}}, \frac{e^{z_2}}{e^{z_1} + e^{z_2} + e^{z_3}}, \frac{e^{z_3}}{e^{z_1} + e^{z_2} + e^{z_3}}}_{\text{softmax activations}}$$

original activations

Best w ?

- Maximum likelihood estimation:

$$\max_w ll(w) = \max_w \sum_i \log P(y^{(i)} | x^{(i)}; w)$$

with:

$$P(y^{(i)} | x^{(i)}; w) = \frac{e^{w_{y^{(i)}} \cdot f(x^{(i)})}}{\sum_y e^{w_y \cdot f(x^{(i)})}}$$

= Multi-Class Logistic Regression

This Lecture

- Optimization

- i.e., how do we solve:

$$\max_w \text{ll}(w) = \max_w \sum_i \log P(y^{(i)}|x^{(i)}; w)$$

Hill Climbing

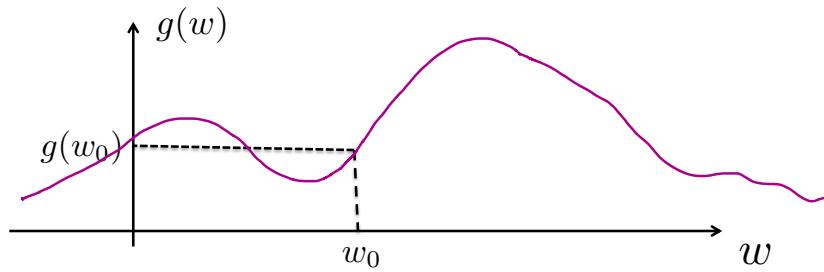
- Recall from CSPs lecture: simple, general idea

- Start wherever
 - Repeat: move to the best neighboring state
 - If no neighbors better than current, quit



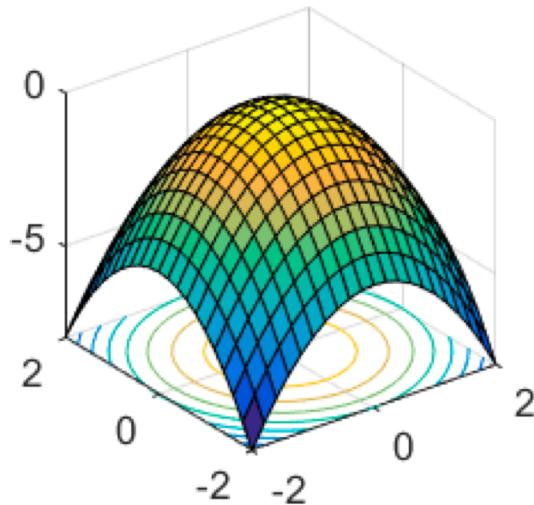
- What's particularly tricky when hill-climbing for multiclass logistic regression?
 - Optimization over a continuous space
 - Infinitely many neighbors!
 - How to do this efficiently?

1-D Optimization



- Could evaluate $g(w_0 + h)$ and $g(w_0 - h)$
 - Then step in best direction
- Or, evaluate derivative: $\frac{\partial g(w_0)}{\partial w} = \lim_{h \rightarrow 0} \frac{g(w_0 + h) - g(w_0 - h)}{2h}$
 - Tells which direction to step into

2-D Optimization



Source: offconvex.org

Gradient Ascent

- Perform update in uphill direction for each coordinate
- The steeper the slope (i.e. the higher the derivative) the bigger the step for that coordinate
- E.g., consider: $g(w_1, w_2)$

- Updates:

$$w_1 \leftarrow w_1 + \alpha * \frac{\partial g}{\partial w_1}(w_1, w_2)$$

$$w_2 \leftarrow w_2 + \alpha * \frac{\partial g}{\partial w_2}(w_1, w_2)$$

- Updates in vector notation:

$$w \leftarrow w + \alpha * \nabla_w g(w)$$

with: $\nabla_w g(w) = \begin{bmatrix} \frac{\partial g}{\partial w_1}(w) \\ \frac{\partial g}{\partial w_2}(w) \end{bmatrix}$ = **gradient**

Gradient Ascent

- Idea:
 - Start somewhere
 - Repeat: Take a step in the gradient direction

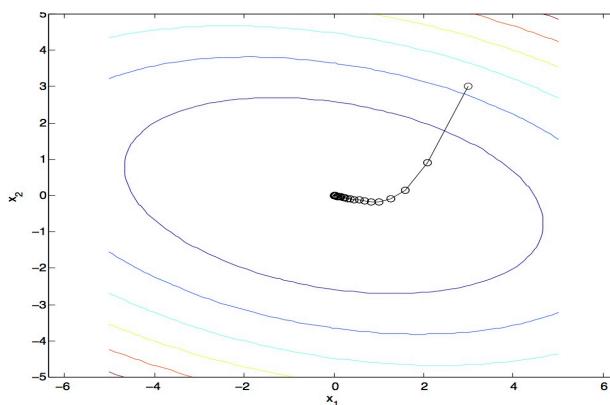


Figure source: Mathworks

What is the Steepest Direction?

$$\max_{\Delta: \Delta_1^2 + \Delta_2^2 \leq \varepsilon} g(w + \Delta)$$



■ First-Order Taylor Expansion: $g(w + \Delta) \approx g(w) + \frac{\partial g}{\partial w_1} \Delta_1 + \frac{\partial g}{\partial w_2} \Delta_2$

■ Steepest Descent Direction: $\max_{\Delta: \Delta_1^2 + \Delta_2^2 \leq \varepsilon} g(w) + \frac{\partial g}{\partial w_1} \Delta_1 + \frac{\partial g}{\partial w_2} \Delta_2$

■ Recall: $\max_{\Delta: \|\Delta\| \leq \varepsilon} \Delta^\top a \rightarrow \Delta = \varepsilon \frac{a}{\|a\|}$

■ Hence, solution: $\Delta = \varepsilon \frac{\nabla g}{\|\nabla g\|}$ **Gradient direction = steepest direction!** $\nabla g = \begin{bmatrix} \frac{\partial g}{\partial w_1} \\ \frac{\partial g}{\partial w_2} \end{bmatrix}$

Gradient in n dimensions

$$\nabla g = \begin{bmatrix} \frac{\partial g}{\partial w_1} \\ \frac{\partial g}{\partial w_2} \\ \vdots \\ \frac{\partial g}{\partial w_n} \end{bmatrix}$$

Optimization Procedure: Gradient Ascent

```
▪ init  $w$ 
▪ for iter = 1, 2, ...
 $w \leftarrow w + \alpha * \nabla g(w)$ 
```

- α : learning rate --- tweaking parameter that needs to be chosen carefully
- How? Try multiple choices
 - Crude rule of thumb: update changes w about 0.1 – 1 %

Batch Gradient Ascent on the Log Likelihood Objective

$$\max_w ll(w) = \max_w \underbrace{\sum_i \log P(y^{(i)}|x^{(i)}; w)}_{g(w)}$$

```
▪ init  $w$ 
▪ for iter = 1, 2, ...
 $w \leftarrow w + \alpha * \sum_i \nabla \log P(y^{(i)}|x^{(i)}; w)$ 
```

Stochastic Gradient Ascent on the Log Likelihood Objective

$$\max_w \text{ll}(w) = \max_w \sum_i \log P(y^{(i)}|x^{(i)}; w)$$

Observation: once gradient on one training example has been computed, might as well incorporate before computing next one

```
■ init w
■ for iter = 1, 2, ...
  ■ pick random j
     $w \leftarrow w + \alpha * \nabla \log P(y^{(j)}|x^{(j)}; w)$ 
```

Mini-Batch Gradient Ascent on the Log Likelihood Objective

$$\max_w \text{ll}(w) = \max_w \sum_i \log P(y^{(i)}|x^{(i)}; w)$$

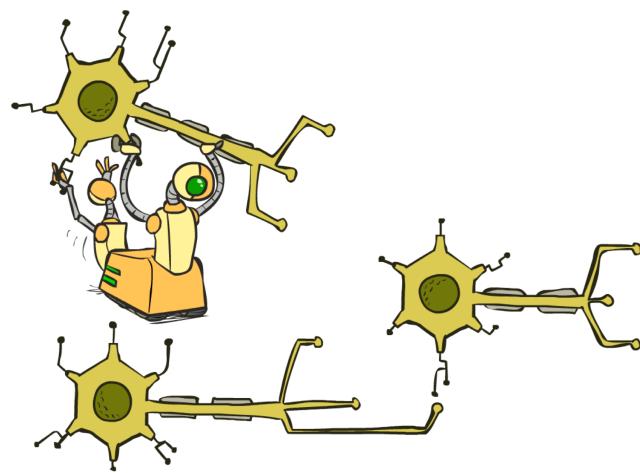
Observation: gradient over small set of training examples (=mini-batch) can be computed in parallel, might as well do that instead of a single one

```
■ init w
■ for iter = 1, 2, ...
  ■ pick random subset of training examples J
     $w \leftarrow w + \alpha * \sum_{j \in J} \nabla \log P(y^{(j)}|x^{(j)}; w)$ 
```

How about computing all the derivatives?

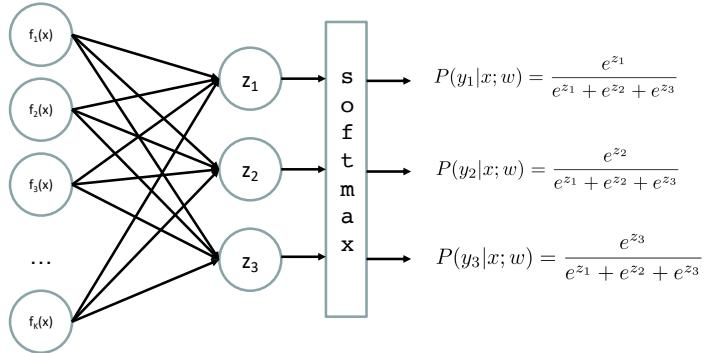
- We'll talk about that once we covered neural networks, which are a generalization of logistic regression

Neural Networks

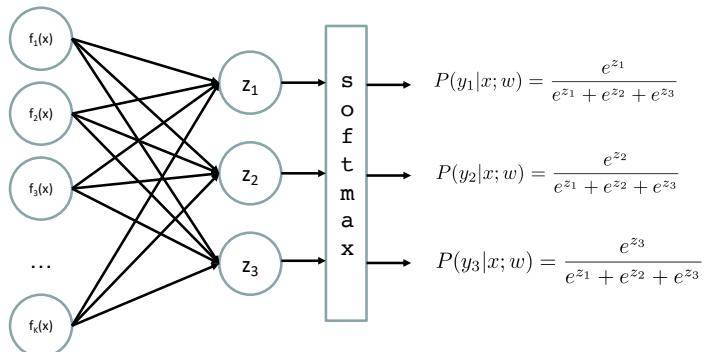


Multi-class Logistic Regression

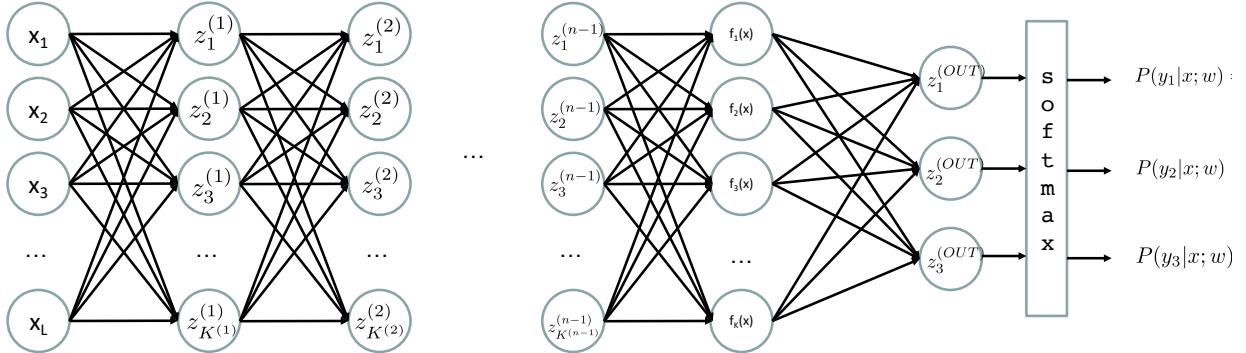
- = special case of neural network



Deep Neural Network = Also learn the features!

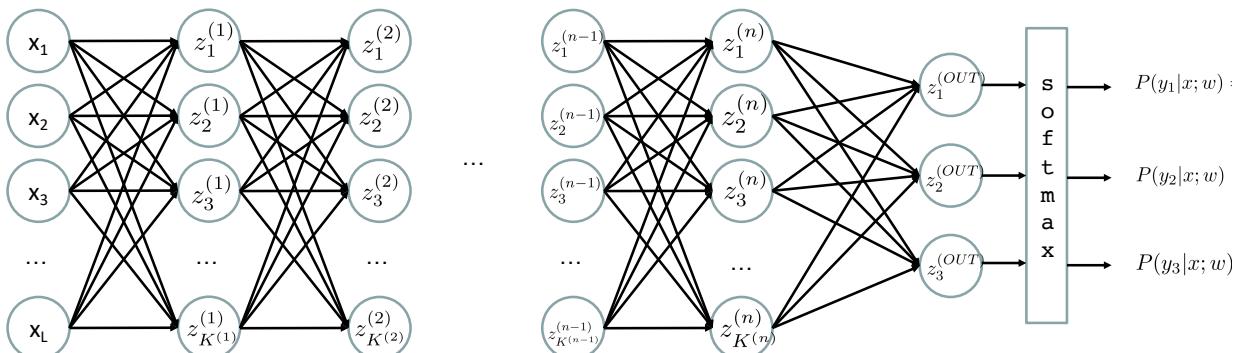


Deep Neural Network = Also learn the features!



$$z_i^{(k)} = g(\sum_j W_{i,j}^{(k-1,k)} z_j^{(k-1)}) \quad \text{g = nonlinear activation function}$$

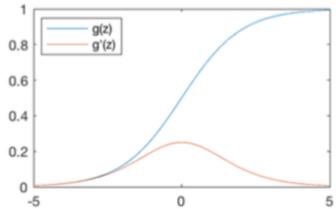
Deep Neural Network = Also learn the features!



$$z_i^{(k)} = g(\sum_j W_{i,j}^{(k-1,k)} z_j^{(k-1)}) \quad \text{g = nonlinear activation function}$$

Common Activation Functions

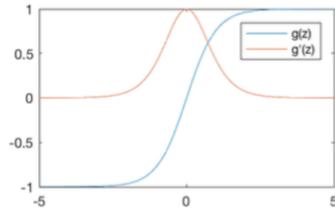
Sigmoid Function



$$g(z) = \frac{1}{1 + e^{-z}}$$

$$g'(z) = g(z)(1 - g(z))$$

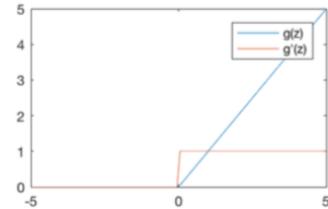
Hyperbolic Tangent



$$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

$$g'(z) = 1 - g(z)^2$$

Rectified Linear Unit (ReLU)



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

$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$

[source: MIT 6.S191 intro to deep learning.com]

Deep Neural Network: Also Learn the Features!

- Training the deep neural network is just like logistic regression:

$$\max_w ll(w) = \max_w \sum_i \log P(y^{(i)} | x^{(i)}; w)$$

just w tends to be a much, much larger vector ☺

→ just run gradient ascent

+ stop when log likelihood of hold-out data starts to decrease

Neural Networks Properties

- Theorem (Universal Function Approximators). A two-layer neural network with a sufficient number of neurons can approximate any continuous function to any desired accuracy.
- Practical considerations
 - Can be seen as learning the features
 - Large number of neurons
 - Danger for overfitting
 - (hence early stopping!)

Universal Function Approximation Theorem*

Hornik theorem 1: Whenever the activation function is *bounded and nonconstant*, then, for any finite measure μ , standard multilayer feedforward networks can approximate any function in $L^p(\mu)$ (the space of all functions on R^k such that $\int_{R^k} |f(x)|^p d\mu(x) < \infty$) arbitrarily well, provided that sufficiently many hidden units are available.

Hornik theorem 2: Whenever the activation function is *continuous, bounded and nonconstant*, then, for arbitrary compact subsets $X \subseteq R^k$, standard multilayer feedforward networks can approximate any continuous function on X arbitrarily well with respect to uniform distance, provided that sufficiently many hidden units are available.

- In words: Given any continuous function $f(x)$, if a 2-layer neural network has enough hidden units, then there is a choice of weights that allow it to closely approximate $f(x)$.

Cybenko (1989) "Approximations by superpositions of sigmoidal functions"
Hornik (1991) "Approximation Capabilities of Multilayer Feedforward Networks"
Leshno and Shtokhen (1991) "Multilayer Feedforward Networks with Non-Polynomial Activation Functions Can Approximate Any Function"

Universal Function Approximation Theorem*

Math. Control Signals Systems (1989) 2: 303–314

Mathematics of Control, Signals, and Systems
© 1989 Springer-Verlag New York Inc.

Approximation by Superpositions of a Sigmoidal Function*
G. Cybenko

Abstract. In this paper we demonstrate that finite linear combinations of compositions of a fixed, invertible sigmoidal function and its translates suffice for universal approximation under very general conditions. Only mild conditions are imposed on the sigmoidal function. Our results apply to a wide variety of sigmoidal functions, including the logistic (sigmoid), hyperbolic tangent, and their relatives. They also apply to piecewise smooth functions, and, in particular, to piecewise constant functions, and thus to single hidden layer neural networks. In particular, we show that arbitrary decision regions can be obtained. It follows that every function which is continuous and piecewise smooth with only a single jump discontinuity and any continuous sigmoidal nonlinearity. The paper also contains a brief discussion of the relevance of this type of approximation that might be implemented by artificial neural networks.

Key words. Neural networks, Approximation, Completeness.

1. Introduction

A number of diverse applications areas are concerned with the representation of general functions of an n -dimensional real variable, $x \in \mathbb{R}^n$, by finite linear combinations of the form

$$\sum_{j=1}^k \sigma_j(x^T \theta_j) \quad (1)$$

where $\sigma_j \in \mathbb{R}^m$ and $\theta_j \in \mathbb{R}^m$. If σ is the transpose of σ or that $y^T x$ is the inner product of y and x then the univariate function σ depends heavily on the context of the application. Our major concern is with so-called sigmoidal σ :

$$\sigma(t) = \begin{cases} 1 & \text{as } t \rightarrow +\infty, \\ 0 & \text{as } t \rightarrow -\infty. \end{cases}$$

Such functions arise naturally in neural network theory as the activation function of a neural node (or unit as is becoming the preferred term) [1], [2]. The main result of this paper is that every function which is continuous and piecewise smooth in the space of continuous functions on the unit cube if it is any continuous sigmoidal

* This research was funded by the National Science Foundation under Grant No. DMS-8703520. It was presented at the Conference on Supercomputing Research and Development and Department of Electrical and Computer Engineering, University of Illinois, Urbana, Illinois 61801, U.S.A.

303

Neural Networks, Vol. 4, pp. 251–271, 1991
Printed in Great Britain © 1991 Pergamon Press Ltd.

ORIGINAL CONTRIBUTION

Approximation Capabilities of Multilayer Feedforward Networks
KURT HORNIK

Abstract. We show that standard multilayer feedforward networks with a few as one hidden layer and arbitrary bounded and monotone activation functions are universal approximators with respect to L_2 -norm bounded functions on compact sets. The number of neurons in the hidden units are available. If the activation function is continuous, bounded and nonconstant, then continuous mappings from \mathbb{R}^n to \mathbb{R}^m can be approximated uniformly with respect to L_p -norm, $1 \leq p \leq \infty$. Functions with sufficiently smooth activation functions are capable of arbitrarily accurate approximation to a function and its derivatives.

Keywords. Multilayer feedforward networks, Activation function, Universal approximation, L_p -norm, Input environment measure, L_p -approximation, Uniform approximation, Sobolev spaces, Smooth approximation.

1. INTRODUCTION

The approximation capabilities of feedforward architectures have recently been investigated by many authors (cf., e.g., [1]–[10]). In particular, Cybenko (1989), Funahashi (1989), Gelfand and White (1989), Hornik (1989), Hornik, Stinchcombe, and White (1989), Jang (1989), Lin and Lee (1989), Lapedes and Farber (1989), Stinchcombe and White (1989), and White (1989).

If we think of the network architecture as a rule for combining information from several different input units, hence implementing a class of step functions, then it is clear that every function which is a continuous mapping from \mathbb{R}^n to \mathbb{R}^m can be approximated by the network if the number of hidden units is sufficiently large and if the required rules for internal representation and computation may be employed.

How accurately the accuracy of approximation depends on how we measure closeness of functions, however, is a question which must be addressed in any specific problem to be dealt with. In many applications, it is a requirement that the function be approximated reasonably well on all input samples taken from some compact input set $X \subset \mathbb{R}^n$. In this case, closeness is measured by the L_2 -norm difference

$$J(f, f') = \left(\int_X |f(x) - f'(x)|^2 dx \right)^{1/2}.$$

Or if $p \neq \infty$, the most popular choice being $p = 2$, closeness is measured by the L_p -norm of the approximating function implemented by the network, where the average is taken with respect to the input environment measure μ , where $\mathcal{X}(\mu) \subset \mathcal{X}$. In this case, closeness is measured by the L_p -norm difference

$$J(f, f') = \left(\int_X |f(x) - f'(x)|^p d\mu(x) \right)^{1/p}.$$

Of course, there are many more ways of measuring closeness of functions. In particular, in many applications, the closeness of two functions is measured by the error of the approximating function implemented by the network, when the function is approximated, up to some order. This issue was first addressed by Hornik (1989) in the context of the neural network approximation of smooth functions. He also discusses the need of smooth functional approximation in more detail. Typical examples are in robotics (e.g., trajectory planning), control theory, and signal processing (analysis of chaotic time series); for a recent application of neural networks in control theory see the work of Statman and Sontact, see Gallant and White (1989).

All papers establishing certain approximation ca-

Requests for reprints should be sent to Kurt Hornik, Institut für Statistik und Wirtschaftsmathematik, Technische Universität Wien, Wiedner Hauptstraße 8/172a, A-1040 Wien, Austria.

271

MULTILAYER FEEDFORWARD NETWORKS
WITH NON-POLYNOMIAL ACTIVATION
FUNCTIONS CAN APPROXIMATE ANY FUNCTION

by
Moshe Leshno
Faculty of Management
Tel Aviv University
Tel Aviv, Israel 69978

and

Shimon Schocken
Leonard N. Stern School of Business
New York University
New York, NY 10003

September 1991

Center for Research on Information Systems
Information Systems Department
Leonard N. Stern School of Business
New York University

Working Paper Series
STERN IS-91-26

Appeared previously as Working Paper No. 21/91 at The Israel Institute Of Business Research

Cybenko (1989) "Approximations by superpositions of sigmoidal functions"
 Hornik (1991) "Approximation Capabilities of Multilayer Feedforward Networks"
 Leshno and Schocken (1991) "Multilayer Feedforward Networks with Non-Polynomial Activation Functions Can Approximate Any Function"

Fun Neural Net Demo Site

- **Demo-site:**
- <http://playground.tensorflow.org/>

How about computing all the derivatives?

- Derivatives tables:

$$\begin{aligned}\frac{d}{dx}(a) &= 0 & \frac{d}{dx}[\ln u] &= \frac{d}{dx}[\log_e u] = \frac{1}{u} \frac{du}{dx} \\ \frac{d}{dx}(x) &= 1 & \frac{d}{dx}[\log_a u] &= \log_a e \frac{1}{u} \frac{du}{dx} \\ \frac{d}{dx}(au) &= a \frac{du}{dx} & \frac{d}{dx}e^u &= e^u \frac{du}{dx} \\ \frac{d}{dx}(u+v-w) &= \frac{du}{dx} + \frac{dv}{dx} - \frac{dw}{dx} & \frac{d}{dx}a^u &= a^u \ln a \frac{du}{dx} \\ \frac{d}{dx}(uv) &= u \frac{dv}{dx} + v \frac{du}{dx} & \frac{d}{dx}(u^v) &= vu^{v-1} \frac{du}{dx} + \ln u \cdot u^v \frac{dv}{dx} \\ \frac{d}{dx}\left(\frac{u}{v}\right) &= \frac{1}{v} \frac{du}{dx} - \frac{u}{v^2} \frac{dv}{dx} & \frac{d}{dx}\sin u &= \cos u \frac{du}{dx} \\ \frac{d}{dx}(u^n) &= nu^{n-1} \frac{du}{dx} & \frac{d}{dx}\cos u &= -\sin u \frac{du}{dx} \\ \frac{d}{dx}(\sqrt{u}) &= \frac{1}{2\sqrt{u}} \frac{du}{dx} & \frac{d}{dx}\tan u &= \sec^2 u \frac{du}{dx} \\ \frac{d}{dx}\left(\frac{1}{u}\right) &= -\frac{1}{u^2} \frac{du}{dx} & \frac{d}{dx}\cot u &= -\csc^2 u \frac{du}{dx} \\ \frac{d}{dx}\left(\frac{1}{u^n}\right) &= -\frac{n}{u^{n+1}} \frac{du}{dx} & \frac{d}{dx}\sec u &= \sec u \tan u \frac{du}{dx} \\ \frac{d}{dx}[f(u)] &= \frac{d}{du}[f(u)] \frac{du}{dx} & \frac{d}{dx}\csc u &= -\csc u \cot u \frac{du}{dx}\end{aligned}$$

[source: <http://hyperphysics.phy-astr.gsu.edu/hbase/Math/derfunc.html>]

How about computing all the derivatives?

- But neural net f is never one of those?

- No problem: CHAIN RULE:

$$\text{If } f(x) = g(h(x))$$

$$\text{Then } f'(x) = g'(h(x))h'(x)$$

→ Derivatives can be computed by following well-defined procedures

Automatic Differentiation

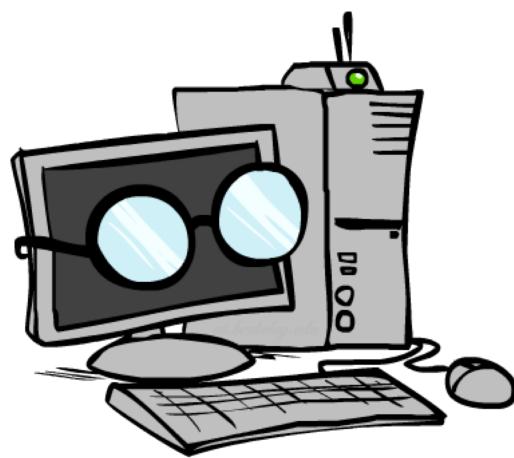
- Automatic differentiation software
 - e.g. Theano, TensorFlow, PyTorch, Chainer
 - Only need to program the function $g(x,y,w)$
 - Can automatically compute all derivatives w.r.t. all entries in w
 - This is typically done by caching info during forward computation pass of f , and then doing a backward pass = “backpropagation”
 - Autodiff / Backpropagation can often be done at computational cost comparable to the forward pass
- Need to know this exists
- How this is done? -- outside of scope of CS188

Summary of Key Ideas

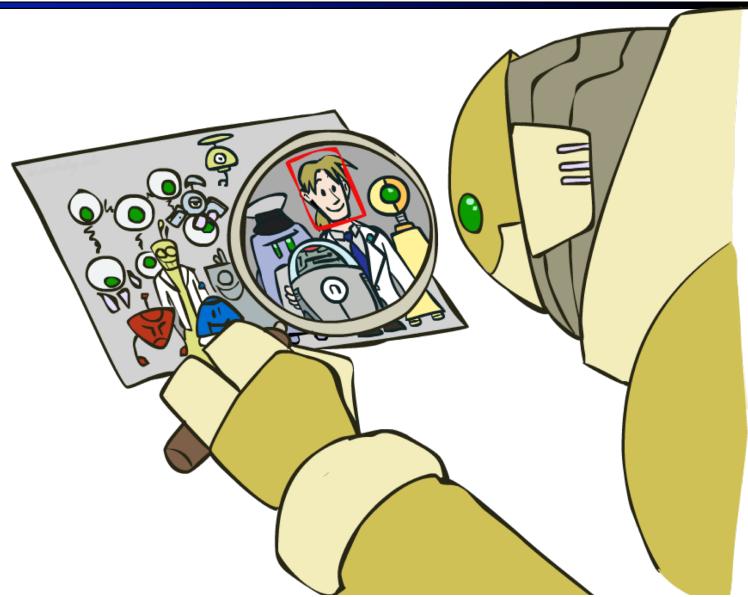
- Optimize probability of label given input $\max_w ll(w) = \max_w \sum_i \log P(y^{(i)}|x^{(i)}; w)$
- Continuous optimization
 - Gradient ascent:
 - Compute steepest uphill direction = gradient (= just vector of partial derivatives)
 - Take step in the gradient direction
 - Repeat (until held-out data accuracy starts to drop = “early stopping”)
- Deep neural nets
 - Last layer = still logistic regression
 - Now also many more layers before this last layer
 - = computing the features
 - → the features are learned rather than hand-designed
 - Universal function approximation theorem
 - If neural net is large enough
 - Then neural net can represent any continuous mapping from input to output with arbitrary accuracy
 - But remember: need to avoid overfitting / memorizing the training data → early stopping!
 - Automatic differentiation gives the derivatives efficiently (how? = outside of scope of 188)

How well does it work?

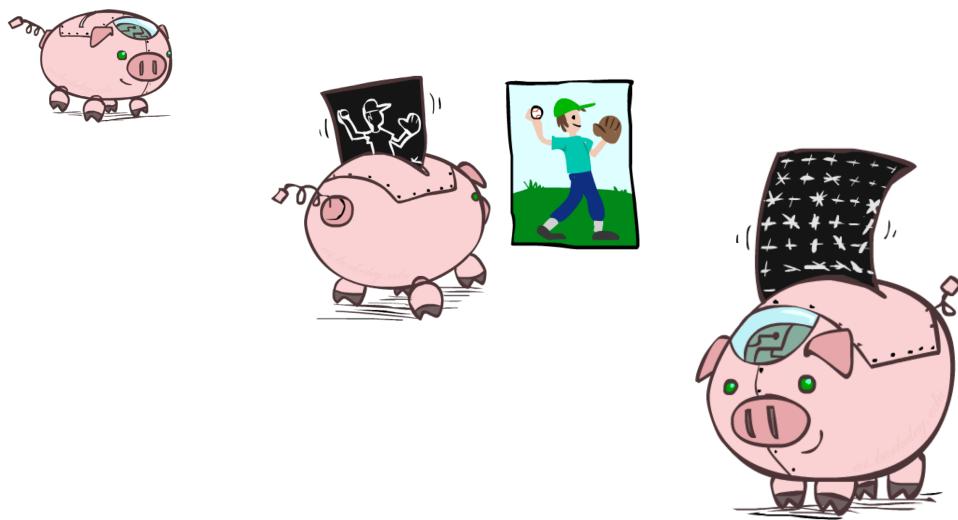
Computer Vision



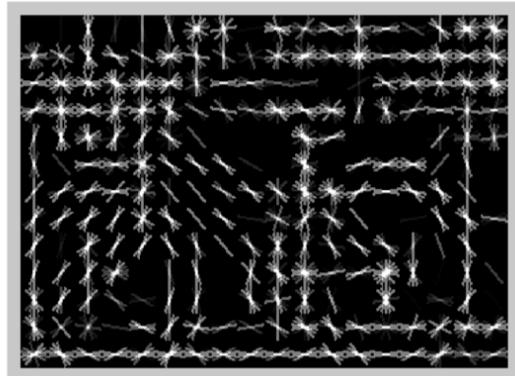
Object Detection



Manual Feature Design



Features and Generalization

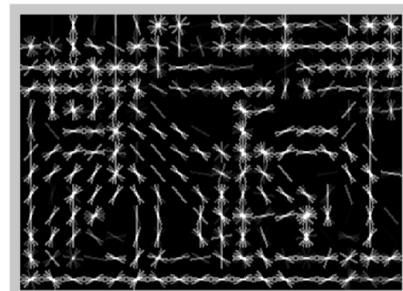


[HoG: Dalal and Triggs, 2005]

Features and Generalization



Image



HoG

Performance

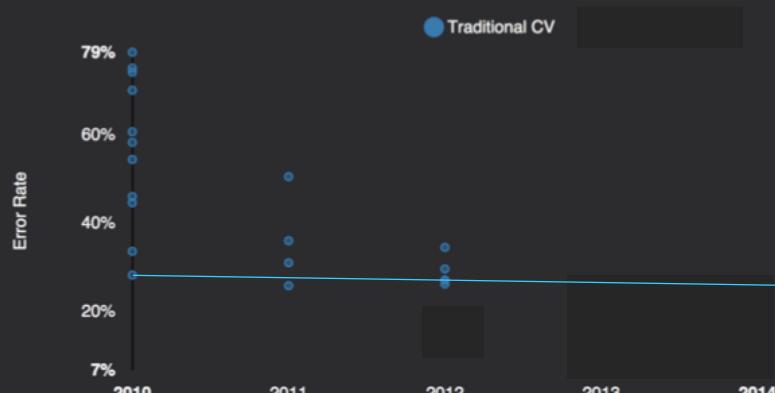
ImageNet Error Rate 2010-2014



graph credit Matt
Zeiler, Clarifai

Performance

ImageNet Error Rate 2010-2014



graph credit Matt
Zeiler, Clarifai

Performance

ImageNet Error Rate 2010-2014



graph credit Matt
Zeiler, Clarifai

Performance

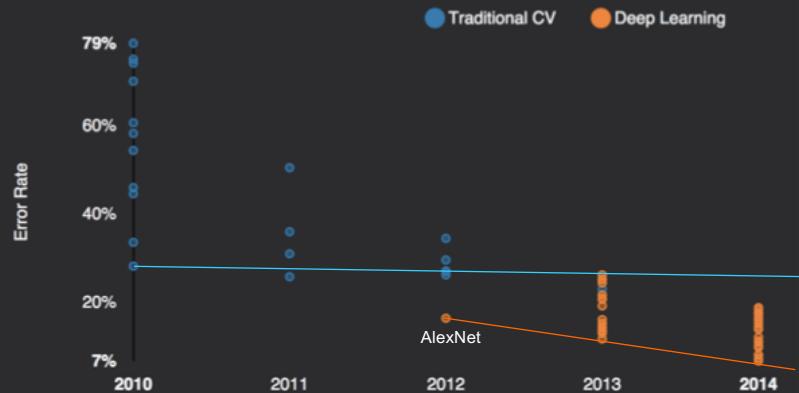
ImageNet Error Rate 2010-2014



graph credit Matt
Zeiler, Clarifai

Performance

ImageNet Error Rate 2010-2014

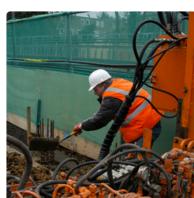


graph credit Matt
Zeiler, Clarifai

MS COCO Image Captioning Challenge



"man in black shirt is playing guitar."



"construction worker in orange safety vest is working on road."



"two young girls are playing with lego toy."



"boy is doing backflip on wakeboard."



"girl in pink dress is jumping in air."



"black and white dog jumps over bar."



"young girl in pink shirt is swinging on swing."



"man in blue wetsuit is surfing on wave."

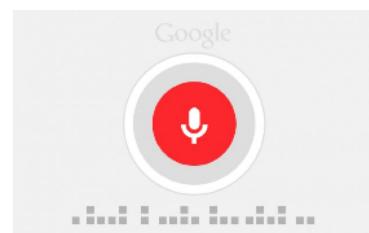
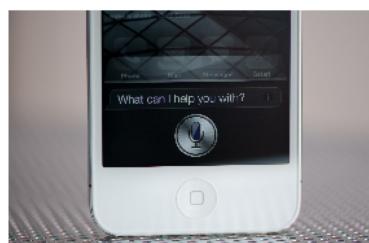
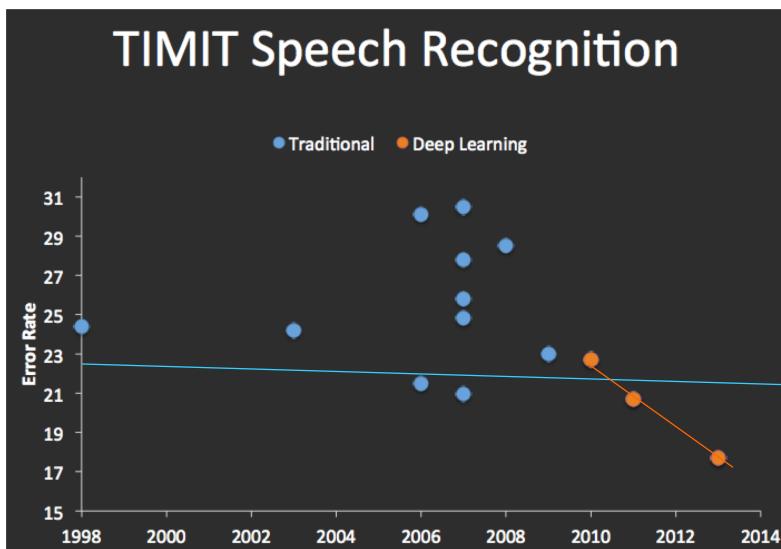
Karpathy & Fei-Fei, 2015; Donahue et al., 2015; Xu et al, 2015; many more

Visual QA Challenge

Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C. Lawrence Zitnick, Devi Parikh

			
What vegetable is on the plate? Neural Net: broccoli Ground Truth: broccoli	What color are the shoes on the person's feet ? Neural Net: brown Ground Truth: brown	How many school busses are there? Neural Net: 2 Ground Truth: 2	What sport is this? Neural Net: baseball Ground Truth: baseball
			
What is on top of the refrigerator? Neural Net: magnets Ground Truth: cereal	What uniform is she wearing? Neural Net: shorts Ground Truth: girl scout	What is the table number? Neural Net: 4 Ground Truth: 40	What are people sitting under in the back? Neural Net: bench Ground Truth: tent

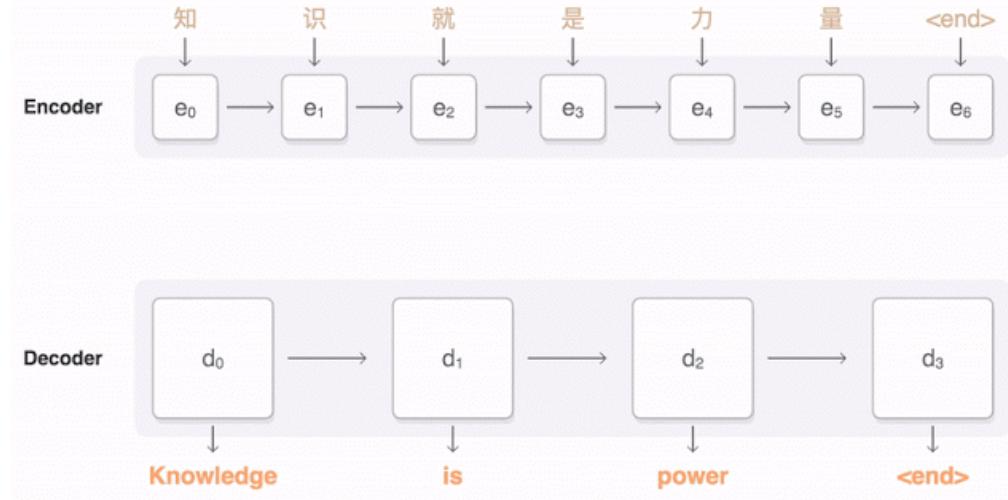
Speech Recognition



graph credit Matt Zeiler, Clarifai

Machine Translation

Google Neural Machine Translation (in production)



Next: More Neural Net Applications!