CS287 Advanced Robotics
Lecture 4 (Fall 2019)

Function Approximation

Pieter Abbeel
UC Berkeley EECS

Value Iteration

Algorithm:

Start with $V_0^*(s) = 0$ for all s.

Impractical for large state spaces

For i = 1, … , H

For all states s in S:

$$V_{i+1}^*(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_i^*(s')\right]$$

$$\pi_{i+1}^*(s) \leftarrow \arg\max_{a \in A} \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_i^*(s')\right]$$

This is called a value update or Bellman update/back-up

$V_i^*(s)$ = expected sum of rewards accumulated starting from state s, acting optimally for i steps

$\pi_i^*(s)$ = optimal action when in state s and getting to act for i steps

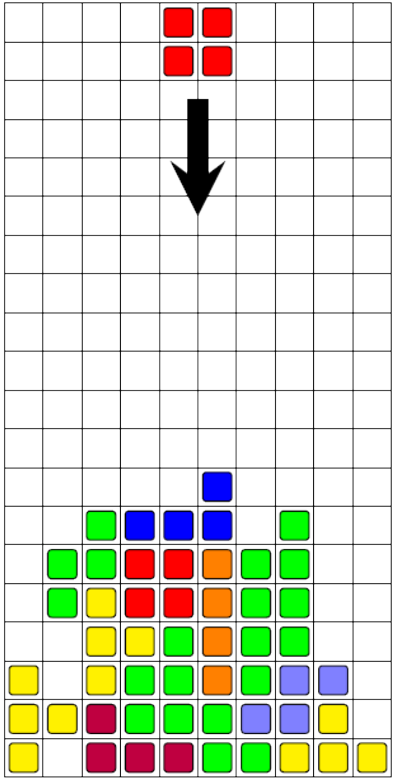
Similar issue for policy iteration and linear programming

Outline

- Function approximation
- Value iteration with function approximation
- Policy iteration with function approximation
- Linear programming with function approximation

Function Approximation Example 1 : Tetris

- state: board configuration + shape of the falling piece ~2^{200} states!
- action: rotation and translation applied to the falling piece



- 22 features aka basis functions ϕ_i
  - Ten basis functions, 0*, . . . , 9, mapping the state to the height h[k] of each* column.
  - Nine basis functions, 10*, . . . , 18, each mapping the state to the absolute difference* between heights of successive columns: |*h[k+1] − h[k]*|*, k = 1, . . . , 9.*
  - One basis function, 19, that maps state to the maximum column height: $\max_k h[k]$
  - One basis function, 20, that maps state to the number of 'holes' in the board.
  - One basis function, 21, that is equal to 1 in every state.

$$\hat{V}_\theta(s) = \sum_{i=0}^{21} \theta_i \phi_i(s) = \theta^\top \phi(s)$$

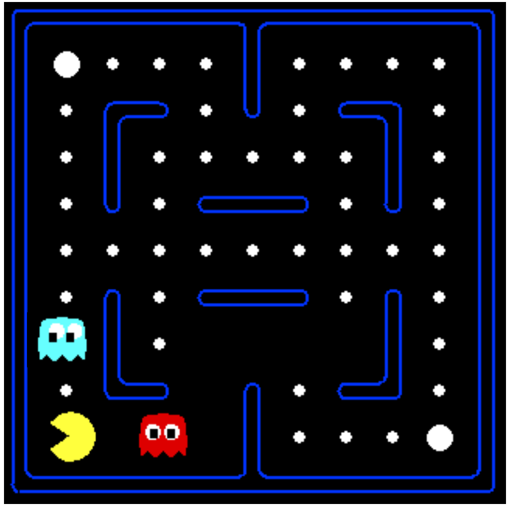
[Bertsekas & Ioffe, 1996 (TD); Bertsekas & Tsitsiklis 1996 (TD); Kakade 2002 (policy gradient); Farias & Van Roy, 2006 (approximate LP)]

Function Approximation Example 2: Pacman

V(s) = θ_0

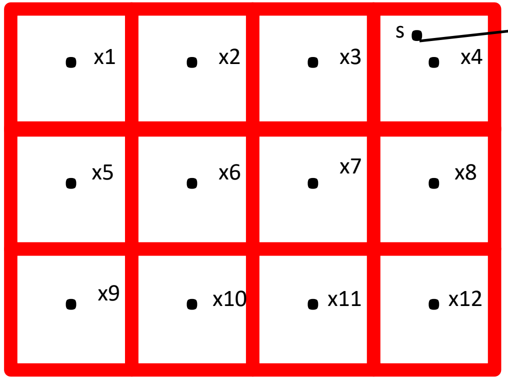
+ θ_1 "distance to closest ghost"
+ θ_2 "distance to closest power pellet"
+ θ_3 "in dead-end"
+ θ_4 "closer to power pellet than ghost"
+ ...

$$= \sum_{i=0}^{n} \theta_i \phi_i(s) = \theta^\top \phi(s)$$



Function Approximation Example 3: Nearest Neighbor

- 0'th order approximation (1-nearest neighbor):



$$\hat{V}(s) = \hat{V}(x4) = \theta_4$$

$$\phi(s) = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ \dots \\ 0 \end{pmatrix} \qquad \hat{V}(s) = \theta^\top \phi(s)$$

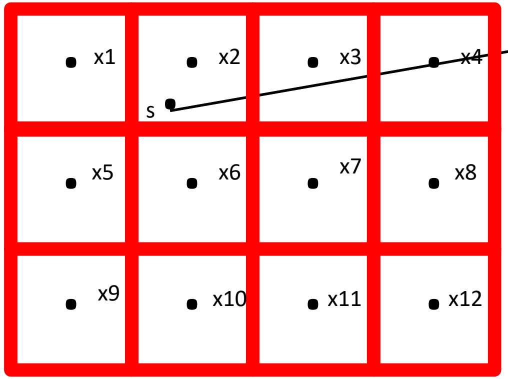
Only store values for x1, x2, …, x12

– call these values $\theta_1, \theta_2, \ldots, \theta_{12}$

Assign other states value of nearest "x" state

Function Approximation Example 4: k-Nearest Neighbor

- 1'th order approximation (k-nearest neighbor interpolation):



$$\hat{V}(s) = \phi_1(s)\theta_1 + \phi_2(s)\theta_2 + \phi_5(s)\theta_5 + \phi_6(s)\theta_6$$

$$\phi(s) = \begin{pmatrix} 0.2 \\ 0.6 \\ 0 \\ 0 \\ 0.05 \\ 0.15 \\ 0 \\ \dots \\ 0 \end{pmatrix} \qquad \hat{V}(s) = \theta^\top \phi(s)$$

Only store values for x1, x2, …, x12

– call these values $\theta_1, \theta_2, \ldots, \theta_{12}$

Assign other states interpolated value of nearest 4 "x" states

More Function Approximation Examples

- Examples:
  - $S = \mathbb{R}, \quad \hat{V}(s) = \theta_1 + \theta_2 s$
  - $S = \mathbb{R}, \quad \hat{V}(s) = \theta_1 + \theta_2 s + \theta_3 s^2$
  - $S = \mathbb{R}, \quad \hat{V}(s) = \sum_{i=0}^{n} \theta_i s^i$
  - $S = \mathbb{R}^n \quad \hat{V}(s) = f_\theta(s)$ (e.g. neural net)

Function Approximation

- Main idea:
  - Use approximation $\hat{V}_\theta$ of the true value function V^*
    - θ is a free parameter to be chosen from its domain Θ
    - Representation size: $|S|$ downto $|\Theta|$

+ : less parameters to estimate

- : less expressiveness,
because typically there exist many V* for which there is no θ such that $\hat{V}_\theta = V^*$

Supervised Learning

- Given:
  - set of examples $(s^{(1)}, V(s^{(1)})), (s^{(2)}, V(s^{(2)})), \ldots, (s^{(m)}, V(s^{(m)})),$

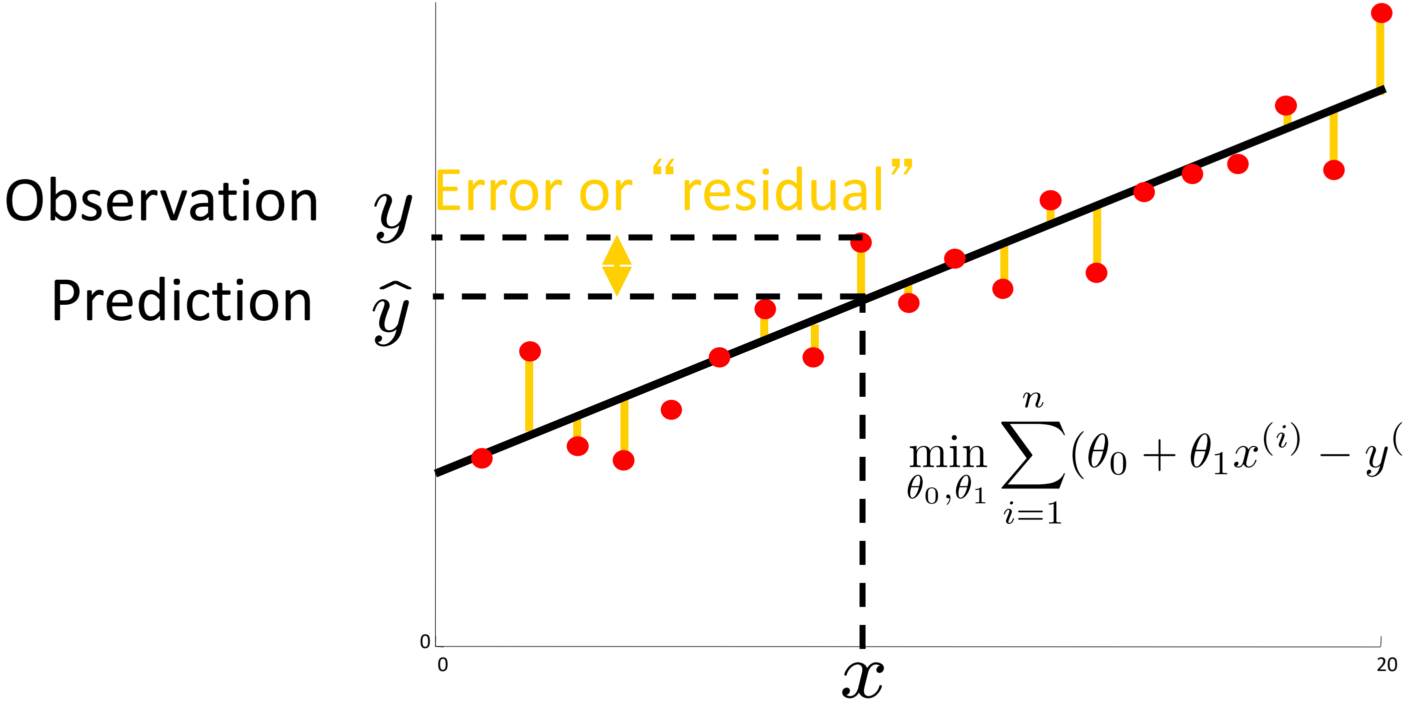
- Asked for:
  - "best" $\hat{V}_\theta$

- Representative approach: find θ through least squares

$$\min_{\theta \in \Theta} \sum_{i=1}^{m} (\hat{V}_\theta(s^{(i)}) - V(s^{(i)}))^2$$

Supervised Learning Example

- Linear regression

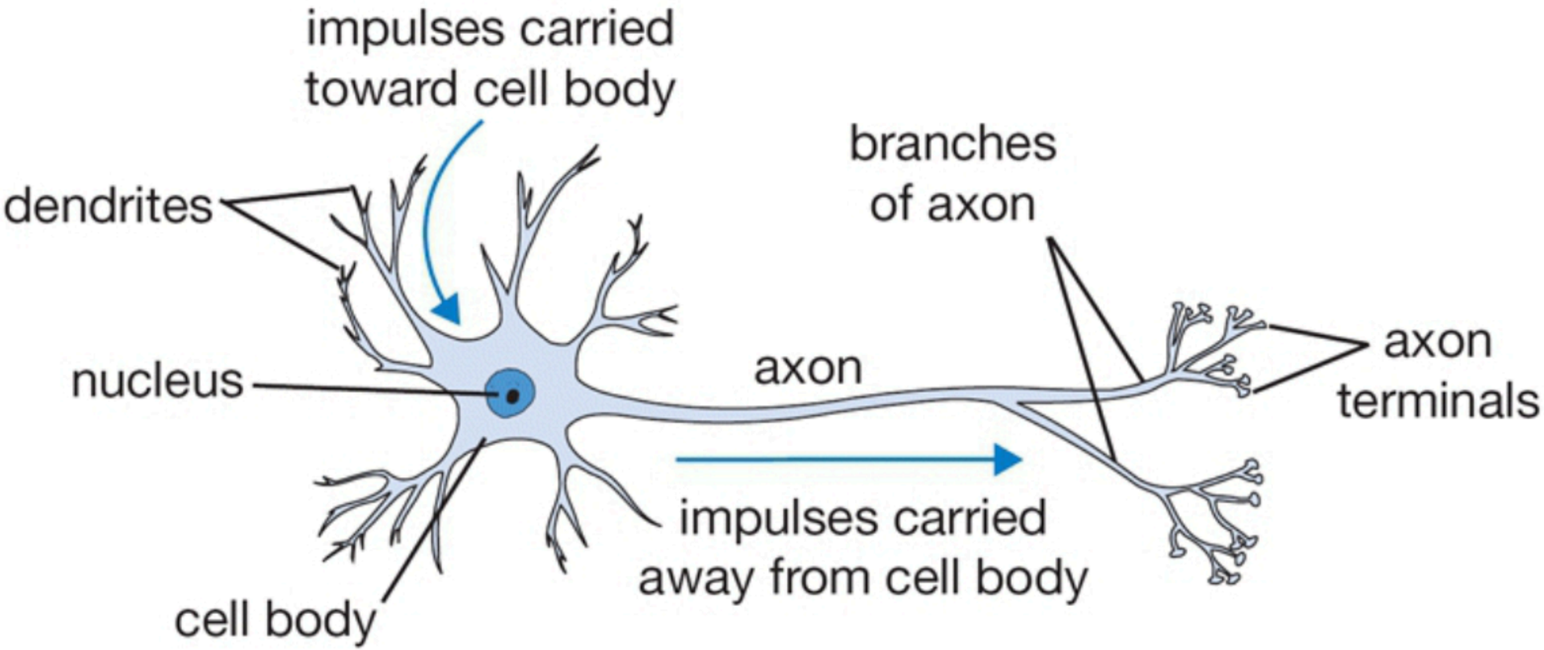


$$\min_{\theta_0,\theta_1} \sum_{i=1}^{n} (\theta_0 + \theta_1 x^{(i)} - y^{(i)})^2$$

Supervised Learning Example

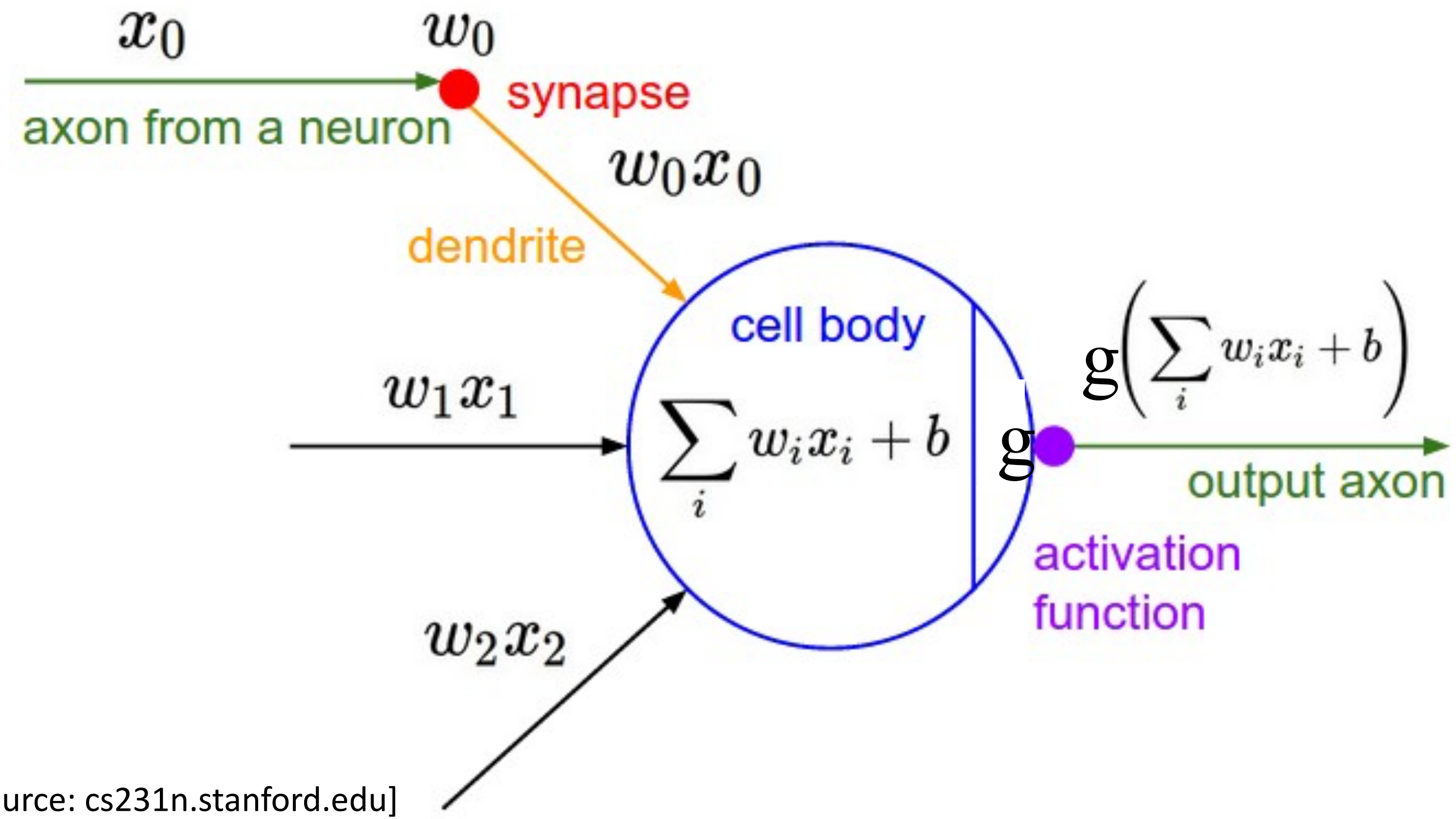
- Neural Nets

Single (Biological) Neuron



[image source: cs231n.stanford.edu]

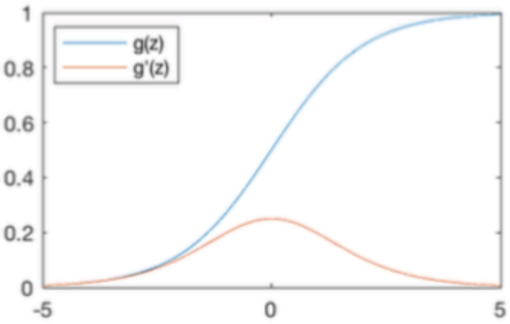
Single (Artificial) Neuron



[image source: cs231n.stanford.edu]

Common Activation Functions

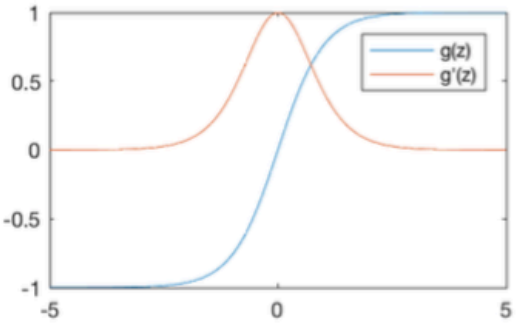
Sigmoid Function



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

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

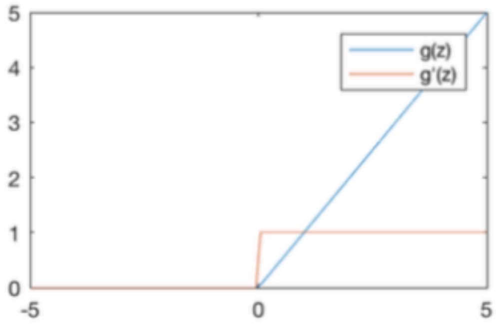
Hyperbolic Tangent



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

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

Rectified Linear Unit (ReLU)

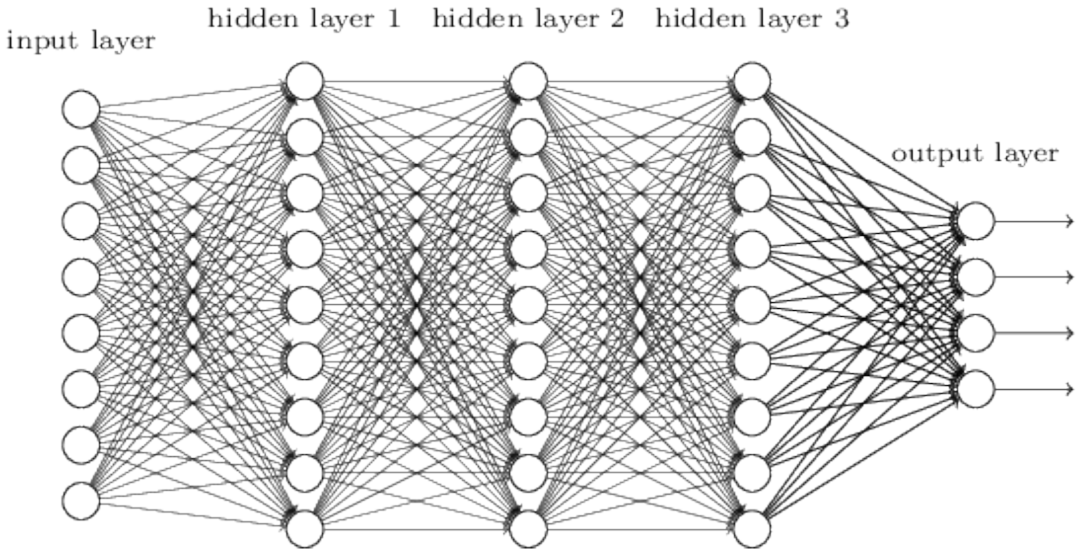


$$g(z) = \max(0, z)$$

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

[source: MIT 6.S191 introtodeeplearning.com]

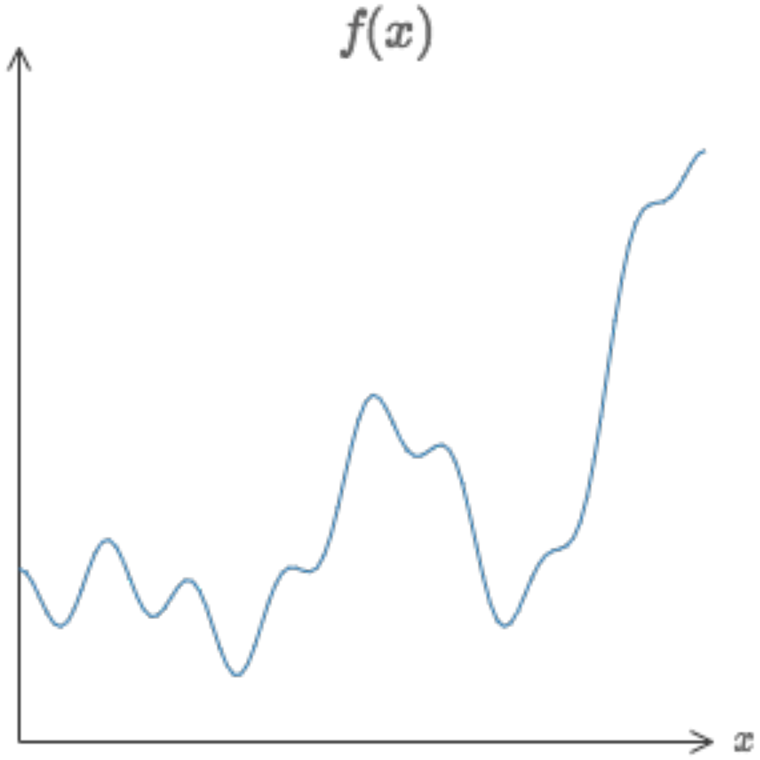
Neural Network



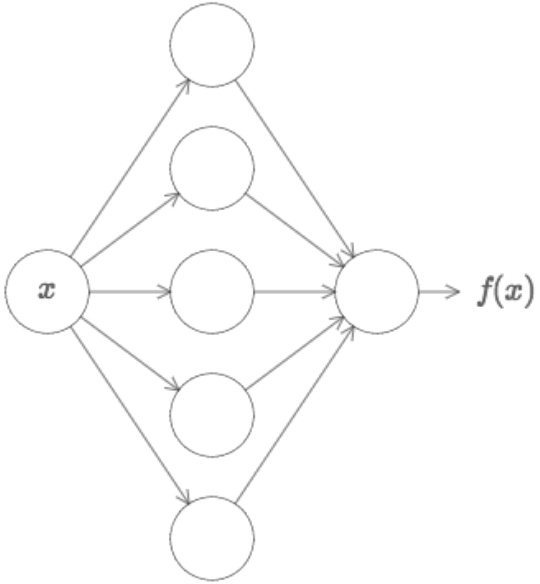
Notation: x $z^{(1)}$ $z^{(2)}$ $z^{(3)}$ $y = f(x, w)$

Choice of w determines the function from x --> y

What Functions Can a Neural Net Represent?



Does there exist a choice for w to make this work?



[images source: neuralnetworksanddeeplearning.com]

Universal Function Approximation Theorem

<u>**Hornik theorem 1:**</u> 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.

<u>**Hornik theorem 2:**</u> 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.

- <u>In words:</u> 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 Schocken (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

Approximation by Superpositions of a Sigmoidal Function*

G. Cybenko†

Abstract. In this paper we demonstrate that finite linear combinations of compositions of a fixed, univariate function and a set of affine functionals can uniformly approximate any continuous function of n real variables with support in the unit hypercube; only mild conditions are imposed on the univariate function. Our results settle an open question about representability in the class of single hidden layer neural networks. In particular, we show that arbitrary decision regions can be arbitrarily well approximated by continuous feedforward neural networks with only a single internal, hidden layer and any continuous sigmoidal nonlinearity. The paper discusses approximation properties of other possible types of nonlinearities that might be implemented by artificial neural networks.

Key words. Neural networks, Approximation, Completeness.

1. Introduction

A number of diverse application 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

$$\sum_{j=1}^{N} \alpha_j \sigma(y_j^T x + \theta_j), \tag{1}$$

where $y_j \in \mathbb{R}^n$ and $\alpha_j, \theta \in \mathbb{R}$ are fixed. (y^T is the transpose of y so that $y^T x$ is the inner product of y and x.) Here the univariate function σ depends heavily on the context of the application. Our major concern is with so-called sigmoidal σ's:

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

Such functions arise naturally in neural network theory as the activation function of a neural node (or *unit* as is becoming the preferred term) [L1], [RHM]. The main result of this paper is a demonstration of the fact that sums of the form (1) are dense in the space of continuous functions on the unit cube if σ is any continuous sigmoidal

* Date received: October 21, 1988. Date revised: February 17, 1989. This research was supported in part by NSF Grant DCR-8619103, ONR Contract N000-86-G-0202 and DOE Grant DE-FG02-85ER25001.

† Center for Supercomputing Research and Development and Department of Electrical and Computer Engineering, University of Illinois, Urbana, Illinois 61801, U.S.A.

Neural Networks, Vol. 4, pp. 251–257, 1991
Printed in the USA. All rights reserved.
0893-6080/91 $3.00 + .00
Copyright © 1991 Pergamon Press plc

ORIGINAL CONTRIBUTION

Approximation Capabilities of Multilayer Feedforward Networks

Kurt Hornik
Technische Universität Wien, Vienna, Austria

(Received 30 January 1990; revised and accepted 25 October 1990)

Abstract—*We show that standard multilayer feedforward networks with as few as a single hidden layer and arbitrary bounded and nonconstant activation function are universal approximators with respect to $L^p(\mu)$ performance criteria, for arbitrary finite input environment measures μ, provided only that sufficiently many hidden units are available. If the activation function is continuous, bounded and nonconstant, then continuous mappings can be learned uniformly over compact input sets. We also give very general conditions ensuring that networks 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 capabilities, Input environment measure, $L^p(\mu)$ approximation, Uniform approximation, Sobolev spaces, Smooth approximation.

1. INTRODUCTION

The approximation capabilities of neural network architectures have recently been investigated by many authors, including Carroll and Dickinson (1989), Cybenko (1989), Funahashi (1989), Gallant and White (1988), Hecht-Nielsen (1989), Hornik, Stinchcombe, and White (1989, 1990), Irie and Miyake (1988), Lapedes and Farber (1988), Stinchcombe and White (1989, 1990). (This list is by no means complete.)

If we think of the network architecture as a rule for computing values at l output units given values at k input units, hence implementing a class of mappings from R^k to R^l, we can ask how well arbitrary mappings from R^k to R^l can be approximated by the network, in particular, if as many hidden units as required for internal representation and computation may be employed.

How to measure the accuracy of approximation depends on how we measure closeness between functions, which in turn varies significantly with the specific problem to be dealt with. In many applications, it is necessary to have the network perform *simultaneously* well on all input samples taken from some compact input set X in R^k. In this case, closeness is measured by the uniform distance between functions on X, that is,

$$\rho_{\mu,X}(f, g) = \sup_{x \in X} |f(x) - g(x)|.$$

In other applications, we think of the inputs as random variables and are interested in the *average performance* where the average is taken with respect to the input environment measure μ, where $\mu(R^k) < \infty$. In this case, closeness is measured by the $L^p(\mu)$ distances

$$\rho_{p,\mu}(f, g) = \left[\int_{R^k} |f(x) - g(x)|^p \, d\mu(x)\right]^{1/p},$$

$1 \le p < \infty$, the most popular choice being $p = 2$, corresponding to mean square error.

Of course, there are many more ways of measuring closeness of functions. In particular, in many applications, it is also necessary that the *derivatives* of the approximating function implemented by the network closely resemble those of the function to be approximated, up to some order. This issue was first taken up in Hornik et al. (1990), who discuss the sources of need of smooth functional approximation in more detail. Typical examples arise in robotics (learning of smooth movements) and signal processing (analysis of chaotic time series); for a recent application to problems of nonparametric inference in statistics and econometrics, see Gallant and White (1989).

All papers establishing certain approximation ca-

Requests for reprints should be sent to Kurt Hornik, Institut für Statistik und Wahrscheinlichkeitstheorie, Technische Universität Wien, Wiedner Hauptstraße 8-10/107, A-1040 Wien, Austria.

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

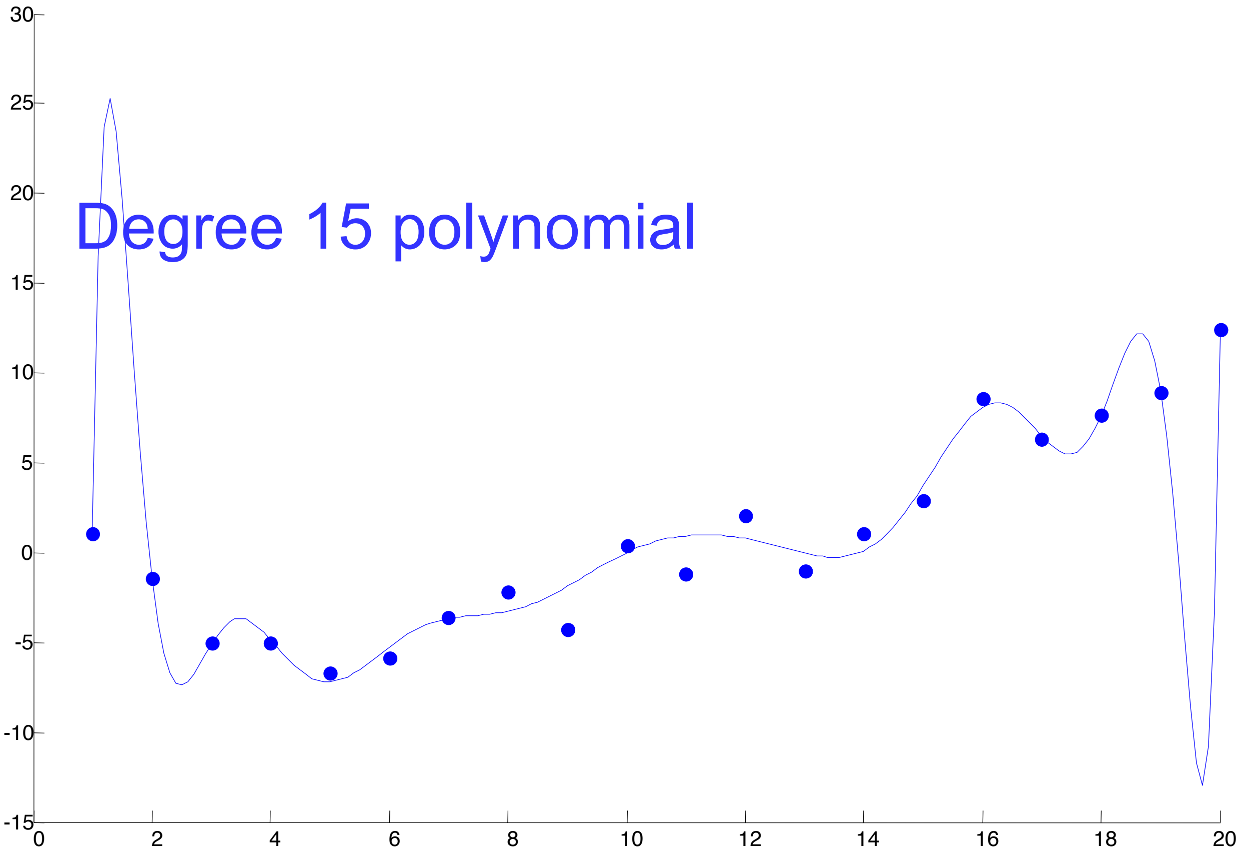
Working Paper Series

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”

Overfitting



Avoiding Overfitting

- Reduce number of features or size of the network
- Regularize θ
- Early stopping: stop training updates once loss increases on hold-out data

Status

- Function approximation through supervised learning

BUT: where do the supervised examples come from?

Value Iteration with Function Approximation

- Initialize by choosing some setting for $\theta^{(0)}$
- Iterate for i = 0, 1, 2, …, H:
  - Step 0: Pick some $S' \subseteq S$ (typically $|S'| << |S|$)
  - Step 1: Bellman back-ups

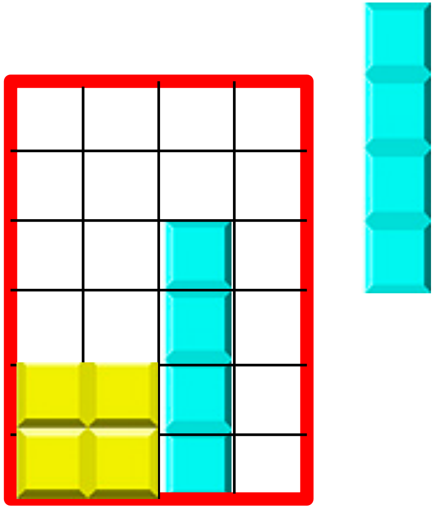
$$\forall s \in S': \quad \bar{V}_{i+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \hat{V}_{\theta^{(i)}}(s') \right]$$

  - Step 2: Supervised learning

find $\theta^{(i+1)}$ as the solution of: $\min_\theta \sum_{s \in S'} \left(\hat{V}_{\theta^{(i+1)}}(s) - \bar{V}_{i+1}(s) \right)^2$

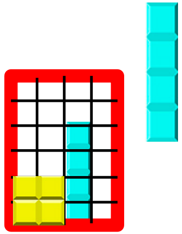
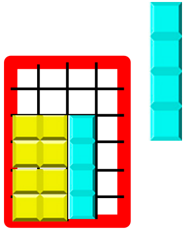
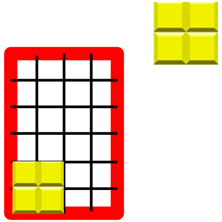
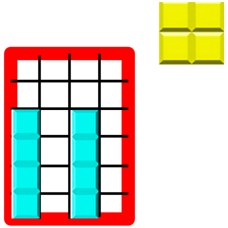
Value Iteration w/Function Approximation --- Example

- Mini-tetris: two types of blocks, can only choose translation (not rotation)
  - Example state:

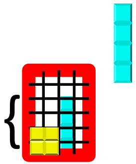
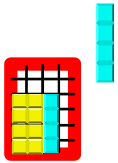
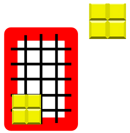
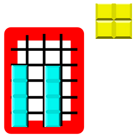


  - Reward = 1 for placing a block
  - Sink state / Game over is reached when block is placed such that part of it extends above the red rectangle
  - If you have a complete row, it gets cleared

Value Iteration w/Function Approximation --- Example

S’ = {  ,  ,  ,  }

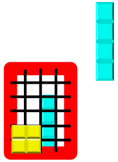
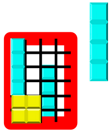
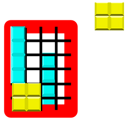
Value Iteration w/Function Approximation --- Example

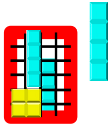
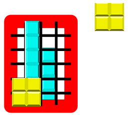
S’ = {  ,  ,  ,  }

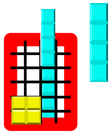
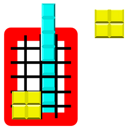
- 10 features (also called basis functions) φ_i
  - Four basis functions, 0, . . . , 3, *mapping the state to the height h[k] of each of the four* columns.
  - Three basis functions, 4, . . . , 6, *each mapping the state to the absolute difference* between heights of successive columns: $|h[k+1] - h[k]|$, $k = 1, \ldots, 3$.
  - One basis function, 7, that maps state to the maximum column height: $\max_k h[k]$
  - One basis function, 8, that maps state to the number of ’holes’ in the board.
  - One basis function, 9, that is equal to 1 in every state.
- Init with $\vartheta^{(0)}$ = (-1, -1, -1, -1, -2, -2, -2, -3, -2, 10)

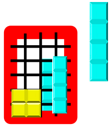
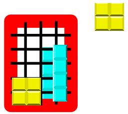
Value Iteration w/Function Approximation --- Example

- Bellman back-ups for the states in S':

V() = max {0.5 *(1+ γ V()) + 0.5*(1 + γ V()) ,

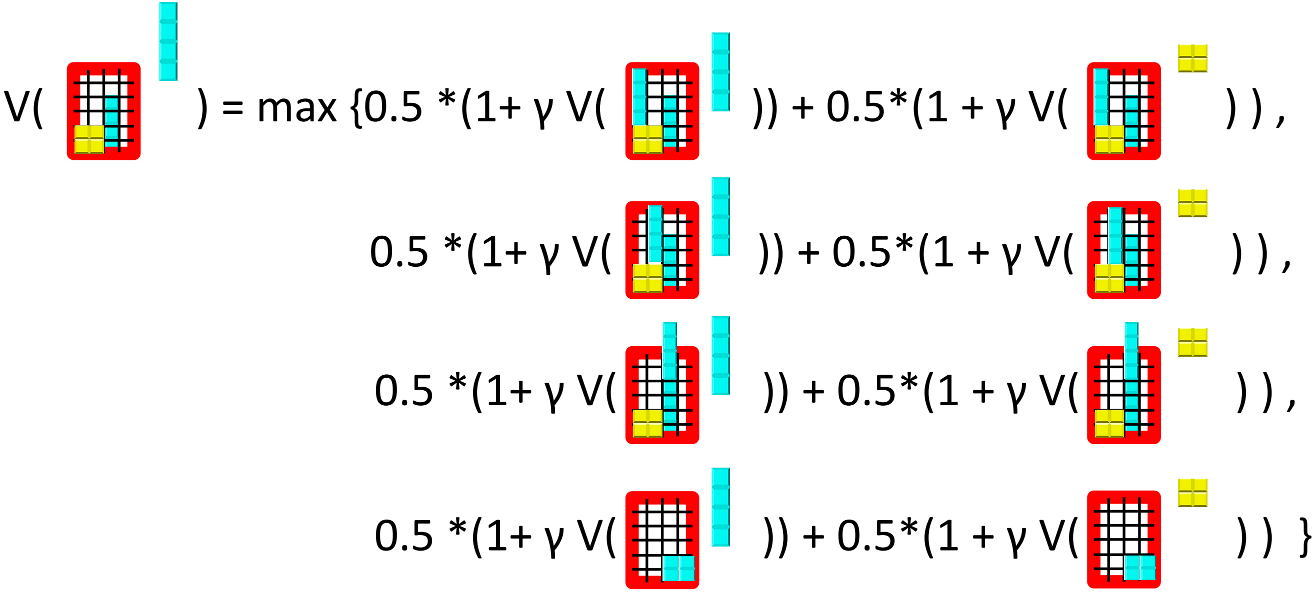
0.5 *(1+ γ V()) + 0.5*(1 + γ V()) ,

0.5 *(1+ γ V()) + 0.5*(1 + γ V()) ,

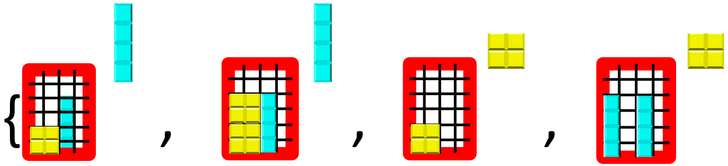
0.5 *(1+ γ V()) + 0.5*(1 + γ V()) }

Value Iteration w/Function Approximation --- Example

- Bellman back-ups for the states in S':



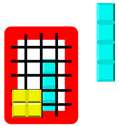
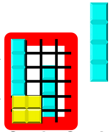
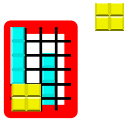
Value Iteration w/Function Approximation --- Example

S’ = {  }

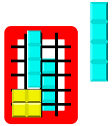
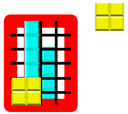
- 10 features aka basis functions φ_i
  - Four basis functions, 0*, . . . , 3, mapping the state to the height h[k] of each of the four* columns.
  - Three basis functions, 4*, . . . , 6, each mapping the state to the absolute difference* between heights of successive columns: $|h[k+1] - h[k]|$, *k = 1, . . . , 3.*
  - One basis function, 7, that maps state to the maximum column height: $\max_k h[k]$
  - One basis function, 8, that maps state to the number of ’holes’ in the board.
  - One basis function, 9, that is equal to 1 in every state.
- Init with $\vartheta^{(0)}$ = (-1, -1, -1, -1, -2, -2, -2, -3, -2, 10)

Value Iteration w/Function Approximation --- Example

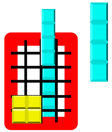
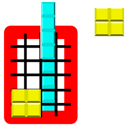
- Bellman back-ups for the states in S':

$V($  $) = \max \{\; 0.5 * (1 + \gamma\; \theta^\top \phi\; ($  $)) + 0.5 * (1 + \gamma\; \theta^\top \phi\; ($  $))\,,$

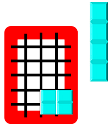
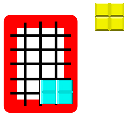
(6,2,4,0, 4, 2, 4, 6, 0, 1) (6,2,4,0, 4, 2, 4, 6, 0, 1)

$0.5 * (1 + \gamma\; \theta^\top \phi\; ($  $)) + 0.5 * (1 + \gamma\; \theta^\top \phi\; ($  $))\,,$

(2,6,4,0, 4, 2, 4, 6, 0, 1) (2,6,4,0, 4, 2, 4, 6, 0, 1)

$0.5 * (1 + \gamma\; V\, ($  $)) + 0.5 * (1 + \gamma\; V\, ($  $))\,,$

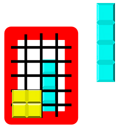
(sink-state, V=0) (sink-state, V=0)

$0.5 * (1 + \gamma\; \theta^\top \phi\; ($  $)) + 0.5 * (1 + \gamma\; \theta^\top \phi\; ($  $))\; \}$

(0,0,2,2, 0,2,0, 2, 0, 1) (0,0,2,2, 0,2,0, 2, 0, 1)

Value Iteration w/Function Approximation --- Example

- Bellman back-ups for the states in S':

V() $= \max \{\, 0.5 * (1 + \gamma\ (-30)) + 0.5 * (1 + \gamma\ (-30))\,,$

$0.5 * (1 + \gamma\ (-30)) + 0.5 * (1 + \gamma\ (-30))\,,$

$0.5 * (1 + \gamma\ (0)) + 0.5 * (1 + \gamma\ (0))\,,$

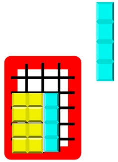
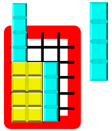
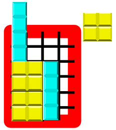
$0.5 * (1 + \gamma\ (6)) + 0.5 * (1 + \gamma\ (6)) \,\}$

$= 6.4 \qquad (\text{for } \gamma = 0.9)$

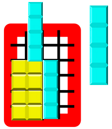
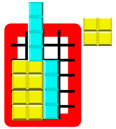
Value Iteration w/Function Approximation --- Example

$\theta^{(0)} = (-1, -1, -1, -1, -2, -2, -2, -3, -2, 20)$

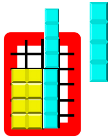
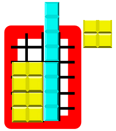
- Bellman back-ups for the second state in S':

V() = max { 0.5 *(1 + γ V ()) + 0.5 *(1 + γ V ()) ,

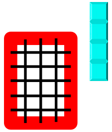
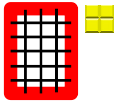
(sink-state, V=0) (sink-state, V=0)

0.5 *(1 + γ V ()) + 0.5 *(1 + γ V ()) ,

(sink-state, V=0) (sink-state, V=0)

0.5 *(1 + γ V ()) + 0.5*(1 + γ V ()) ,

(sink-state, V=0) (sink-state, V=0)

0.5 *(1 + γ $\theta^\top \phi$ ()) + 0.5*(1 + γ $\theta^\top \phi$ ()) }

(0,0,0,0, 0,0,0, 0, 0, 1) (0,0,0,0, 0,0,0, 0, 0, 1)

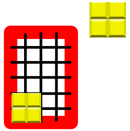
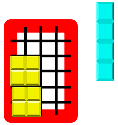
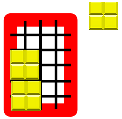
-> V = 20 -> V = 20

= 19

Value Iteration w/Function Approximation --- Example

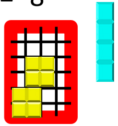
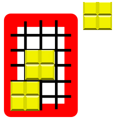
$$\theta^{(0)} = (-1, -1, -1, -1, -2, -2, -2, -3, -2, 20)$$

- Bellman back-ups for the third state in S':

V() = max {0.5 * (1 + γ $\theta^\top \phi$ ()) + 0.5 * (1 + γ $\theta^\top \phi$ ()) ,

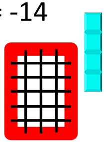
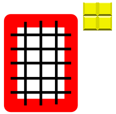
(4,4,0,0, 0,4,0, 4, 0, 1) (4,4,0,0, 0,4,0, 4, 0, 1)

-> V = -8 -> V = -8

0.5 *(1 + γ $\theta^\top \phi$ ()) + 0.5 * (1 + γ $\theta^\top \phi$ ()) ,

(2,4,4,0, 2,0,4, 4, 0, 1) (2,4,4,0, 2,0,4, 4, 0, 1)

-> V = -14 -> V = -14

0.5 *(1 + γ $\theta^\top \phi$ ()) + 0.5 * (1 + γ $\theta^\top \phi$ ()) }

(0,0,0,0, 0,0,0, 0, 0, 1) (0,0,0,0, 0,0,0, 0, 0, 1)

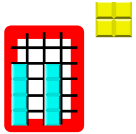
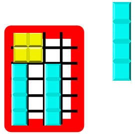
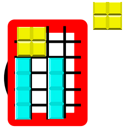
-> V = 20 -> V = 20

= 19

Value Iteration w/Function Approximation --- Example

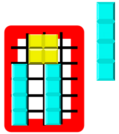
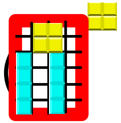
$$\theta^{(0)} = (-1, -1, -1, -1, -2, -2, -2, -3, -2, 20)$$

- Bellman back-ups for the fourth state in S':

V() = max { 0.5 * (1 + γ $\theta^\top \phi$ ()) + 0.5 * (1 + γ $\theta^\top \phi$ ()) ,

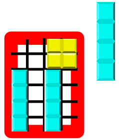
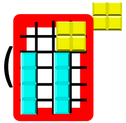
(6,6,4,0, 0,2,4, 6, 4, 1) (6,6,4,0, 0,2,4, 6, 4, 1)

-> V = -34 -> V = -34

0.5 * (1 + γ $\theta^\top \phi$ ()) + 0.5 * (1 + γ $\theta^\top \phi$ ()) ,

(4,6,6,0, 2,0,6, 6, 4, 1) (4,6.6,0, 2,0,6, 6, 4, 1)

-> V = -38 -> V = -38

0.5 * (1 + γ $\theta^\top \phi$ ()) + 0.5 * (1 + γ $\theta^\top \phi$ ()) }

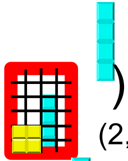
(4,0,6,6, 4,6,0, 6, 4, 1) (4,0,6,6, 4,6,0, 6, 4, 1)

-> V = -42 -> V = -42

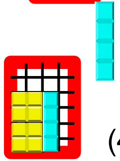
= -29.6

Value Iteration w/Function Approximation --- Example

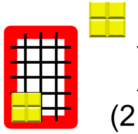
- After running the Bellman back-ups for all 4 states in S' we have:

V()= 6.4

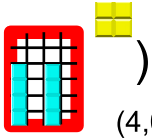
(2,2,4,0, 0,2,4, 4, 0, 1)

V()= 19

(4,4,4,0, 0,0,4, 4, 0, 1)

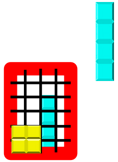
V()= 19

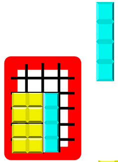
(2,2,0,0, 0,2,0, 2, 0, 1)

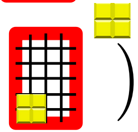
V()= -29.6

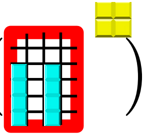
(4,0,4,0, 4,4,4, 4, 0, 1)

- We now run supervised learning on these 4 examples to find a new θ:

$\min_\theta (6.4 - \theta^\top \phi($  $))^2$

$+(19 - \theta^\top \phi($  $))^2$

$+(19 - \theta^\top \phi($  $))^2$

$+((-29.6) - \theta^\top \phi($  $))^2$

Running least squares gives:

$$\theta^{(1)} = (0.195, 6.24, -2.11, 0, -6.05, 0.13, -2.11, 2.13, 0, 1.59)$$

Value Iteration with ***Neural Net*** Function Approximation

- Initialize by choosing some setting for $\theta^{(0)}$
- Iterate for i = 0, 1, 2, …, H:
  - Step 0: Pick some $S' \subseteq S$ (typically $|S'| << |S|$)
  - Step 1: Bellman back-ups

$$\forall s \in S' : \quad \bar{V}_{i+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \hat{V}_{\theta^{(i)}}(s') \right]$$

  - Step 2: Supervised learning

find $\theta^{(i+1)}$ as the solution of: $\min_\theta \sum_{s \in S'} \left(\hat{V}_{\theta^{(i+1)}}(s) - \bar{V}_{i+1}(s) \right)^2$

To avoid overfitting: only small number of gradient updates on objective or early stopping based on hold-out set

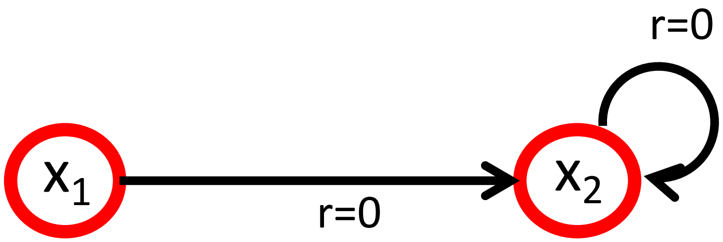
Potential Guarantees?

Theoretical Analysis of Value Iteration + Function Approximation

- We'll consider the following varation on the algorithm:
  - Assume we iterate over:
    - VI back-up for ALL states
    - Function approximation

Note: For ALL states is not practical (that's why we do function approximation). But (i) it's helpful to theoretically think through things; (ii) if we have a negative result, it's an even stronger negative result

Simple Example



θ

2θ

Function approximator: [1 2] * θ

Simple Example

$$\bar{J}_\theta = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \theta$$

$$\begin{aligned} \bar{J}^{(1)}(x_1) &= 0 + \gamma \hat{J}_{\theta^{(0)}}(x_2) = 2\gamma\theta^{(0)} \\ \bar{J}^{(1)}(x_2) &= 0 + \gamma \hat{J}_{\theta^{(0)}}(x_2) = 2\gamma\theta^{(0)} \end{aligned}$$

Function approximation with least squares fit:

$$\begin{bmatrix} 1 \\ 2 \end{bmatrix} \theta^{(1)} \approx \begin{bmatrix} 2\gamma\theta^{(0)} \\ 2\gamma\theta^{(0)} \end{bmatrix}$$

Least squares fit results in:

$$\theta^{(1)} = \frac{6}{5}\gamma\theta^{(0)}$$

Repeated back-ups and function approximations result in:

$$\theta^{(i)} = \left(\frac{6}{5}\gamma\right)^i \theta^{(0)}$$

which diverges if $\gamma > \frac{5}{6}$ even though the function approximation class can represent the true value function.]

Composing Operators

- **Definition.** An operator G is a *non-expansion* with respect to a norm || . || if $\|GJ_1 - GJ_2\| \leq \|J_1 - J_2\|$

- **Fact.** If the operator F is a γ-contraction with respect to a norm || . || and the operator G is a non-expansion with respect to the same norm, then the sequential application of the operators G and F is a γ-contraction, i.e., $\|GFJ_1 - GFJ_2\| \leq \gamma \|J_1 - J_2\|$

- **Corollary.** If the supervised learning step is a non-expansion, then value iteration with function approximation is a γ-contraction, and in this case we have a convergence guarantee.

Averager Function Approximators Are Non-Expansions

DEFINITION: A real-valued function approximation scheme is an *averager* if every fitted value is the weighted average of zero or more target values and possibly some predetermined constants. The weights involved in calculating the fitted value $\hat{Y}_i$ may depend on the sample vector X_0, but may not depend on the target values Y. More precisely, for a fixed X_0, if Y has n elements, there must exist n real numbers k_i, n^2 nonnegative real numbers β_{ij}, and n nonnegative real numbers β_i, so that for each i we have $\beta_i + \sum_j \beta_{ij} = 1$ and $\hat{Y}_i = \beta_i k_i + \sum_j \beta_{ij} Y_j$.

- Examples:
  - nearest neighbor (aka state aggregation)
  - linear interpolation over triangles (tetrahedrons, …)

Averager Function Approximators Are Non-Expansions

Proof: Let J_1 and J_2 be two vectors in $\Re^n$. Consider a particular entry s of ΠJ_1 and ΠJ_2:

$$\begin{aligned}
|(\Pi J_1)(s) - (\Pi J_2)(s)| &= |\beta_{s0} + \sum_{s'} \beta_{ss'} J_1(s') - \beta_{s0} + \sum_{s'} \beta_{ss'} J_2(s')| \\
&= |\sum_{s'} \beta_{ss'} (J_1(s') - J_2(s'))| \\
&\leq \max_{s'} |J_1(s') - J_2(s')| \\
&= \|J_1 - J_2\|_\infty
\end{aligned}$$

This holds true for all s, hence we have

$$\|\Pi J_1 - \Pi J_2\|_\infty \leq \|J_1 - J_2\|_\infty$$

Guarantees for Fixed Point

Theorem. Let J^* be the optimal value function for a finite MDP with discount factor γ. Let the projection operator Π be a non-expansion w.r.t. the infinity norm and let $\tilde{J}$ be any fixed point of Π. Suppose $\|\tilde{J} - J^*\|_\infty \leq \epsilon$. Then ΠT converges to a value function $\bar{J}$ such that:

$$\|\bar{J} - J^*\| \leq 2\epsilon + \frac{2\gamma\epsilon}{1-\gamma}$$

- I.e., if we pick a non-expansion function approximator which can approximate J* well, then we obtain a good value function estimate.
- To apply to discretization: use continuity assumptions to show that J* can be approximated well by chosen discretization scheme

Linear Regression ☹

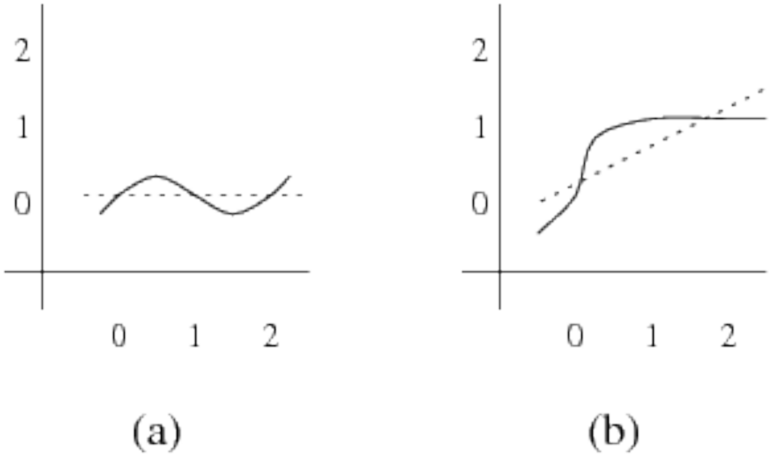


Figure 2: The mapping associated with linear regression when samples are taken at the points $x = 0, 1, 2$. In (a) we see a target value function (solid line) and its corresponding fitted value function (dotted line). In (b) we see another target function and another fitted function. The first target function has values $y = 0, 0, 0$ at the sample points; the second has values $y = 0, 1, 1$. Regression exaggerates the difference between the two functions: the largest difference between the two target functions at a sample point is 1 (at $x = 1$ and $x = 2$), but the largest difference between the two fitted functions at a sample point is $\frac{7}{6}$ (at $x = 2$).

Example taken from Gordon, 1995

Outline

- ✓ Function approximation
- ✓ Value iteration with function approximation
- Policy iteration with function approximation
- Linear programming with function approximation

Policy Iteration

One iteration of policy iteration:

- Policy evaluation: with fixed current policy π, find values with simplified Bellman updates:
  - Iterate until values converge

$$V_{i+1}^{\pi_k}(s) \leftarrow \sum_{s'} T(s, \pi_k(s), s') \left[R(s, \pi_k(s), s') + \gamma V_i^{\pi_k}(s')\right]$$

- Policy improvement: with fixed utilities, find the best action according to one-step look-ahead

$$\pi_{k+1}(s) = \arg\max_a \sum_{s'} T(s, a, s') \left[R(s, a, s') + \gamma V^{\pi_k}(s')\right]$$

Insert Function Approximation Here

- Repeat until policy converges
- At convergence: optimal policy; and converges faster under some conditions

Approximate Policy Evaluation *is* a Contraction!

- IF we do weighted linear regression, weighted by the state visitation frequencies under the current policy

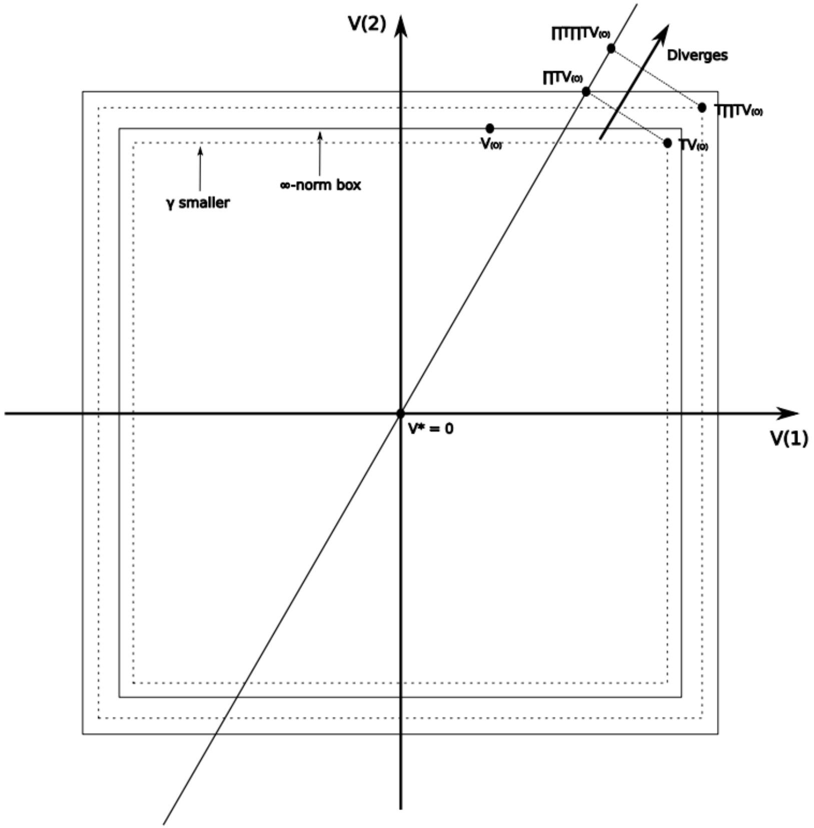
- THEN the resulting projection is a contraction w.r.t. the weighted 2-norm

- Policy Evaluation Bellman update is a contraction w.r.t. the same norm

  → Guaranteed convergence ☺ ☺ ☺

- Want to see the math: https://inst.eecs.berkeley.edu/~cs294-40/fa08/scribes/lecture5.pdf

Extra Intermezzo on Incompatible Norms



Recent Related Paper**

- **Towards Characterizing Divergence in Deep Q-Learning,** Joshua Achiam, Ethan Knight, Pieter Abbeel. arXiv 1903.08894

Outline

- ✓ Function approximation
- ✓ Value iteration with function approximation
- ✓ Policy iteration with function approximation
- Linear programming with function approximation

Infinite Horizon Linear Program**

$$\min_V \sum_{s \in S} \mu_0(s) V(s)$$

$$\text{s.t.} \quad V(s) \geq \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V(s') \right], \quad \forall s \in S, a \in A$$

Theorem. V^* is the solution to the above LP.

μ_0 is a probability distribution over S, with $\mu_0(s) > 0$ for all s in S.

Infinite Horizon Linear Program**

$$\min_V \sum_{s \in S} \mu_0(s) V(s)$$

$$\text{s.t.} \quad V(s) \geq \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V(s') \right], \quad \forall s \in S, a \in A$$

Let $V(s) = \theta^\top \phi(s)$, and consider S' rather than S:

$$\min_\theta \sum_{s \in S'} \mu_0(s) \theta^\top \phi(s)$$

$$\text{s.t.} \quad \theta^\top \phi(s) \geq \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \theta^\top \phi(s') \right], \quad \forall s \in S', a \in A$$

We find approximate value function $\hat{V}_\theta(s) = \theta^\top \phi(s)$

Approximate Linear Program – Guarantees**

$$\min_{\theta} \sum_{s \in S'} \mu_0(s) \theta^\top \phi(s)$$

$$\text{s.t.} \quad \theta^\top \phi(s) \geq \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \theta^\top \phi(s') \right], \quad \forall s \in S', a \in A$$

- LP solver will converge
- Solution quality: [de Farias and Van Roy, 2002]

Assuming one of the features is the feature that is equal to one for all states, and assuming S'=S we have that:

$$\|V^* - \Phi\theta\|_{1,\mu_0} \leq \frac{2}{1-\gamma} \min_{\theta} \|V^* - \Phi\theta\|_\infty$$

(slightly weaker, probabilistic guarantees hold for S' not equal to S, these guarantees require size of S' to grow as the number of features grows)