Machine Learning: Part I

UMaine COS 470/570 – Introduction to AI

Spring 2019

Introduction

What are ANNs?

Machine Learning: Part I
Introduction
What are ANNs?
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

Artificial neural networks

- Systems of simple computing elements: *neurons*
- Each neuron accepts inputs from others, produces *activation*
- Neurons connected via *weights* that modulate activation
- Can be viewed as:
  - Pattern-learning (inductive) systems
  - Statistical programs
  - Dimension/feature-changing systems
  - Search programs (in weight space)

Machine Learning: Part I
Introduction
What are ANNs?
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

Artificial Intelligence

What can they do?

- Image classification and labeling
- Word recognition
- Natural language systems
- Machine translation systems
- General pattern recognition
- Superhuman-level performance on games, other RL tasks
- Pattern generators (images, music, . . .)

Inspiration: Natural pattern recognition

- Pattern recognition in natural world:
  - Chemoreceptors
  - Immune system
  - Biological neural networks
    - Animal/human vision system
    - Auditory system
    - Neocortex
    - Etc.

Introduction
What are ANNs?
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

Neural systems

- Most flexible pattern recognizers:
- Biological computing elements: Neurons
- Neurons are *excitatory* cells
- Connections determine how activation spreads

Introduction
What are ANNs?
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

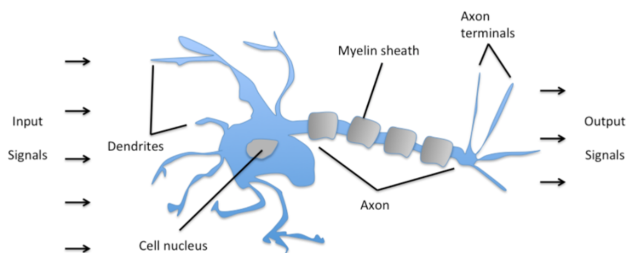
Problem: Complexity

- Neurons are *very* complex

Introduction
What are ANNs?
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

Problem: Complexity

- Neurons are *very* complex



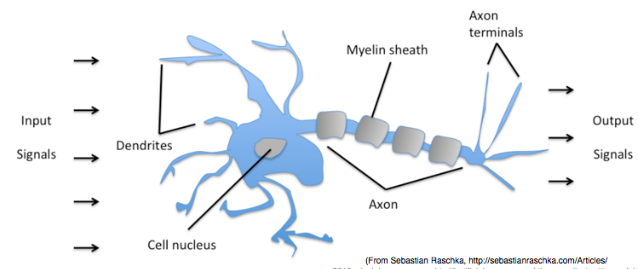
(From Sebastian Raschka, http://sebastianraschka.com/Articles/2015_singlelayer_neurons.html#artificial-neurons-and-the-mcculloch-pitts-model.)

Artificial Intelligence

Problem: Complexity

Machine Learning: Part I
Introduction
What are ANNs?
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

- Neurons are *very* complex



(From Sebastian Raschka, http://sebastianraschka.com/Articles/2015_singlelayer_neurons.html#artificial-neurons-and-the-mcculloch-pitts-model.)

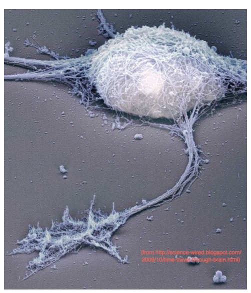
- Synapses: change potential across cell membrane
- Neuron effectively sums excitations, inhibitions
- At some point: potential at threshold and neuron *fires*
- Excitatory pulse down axon, release neurotransmitter at synapses
- *Lots* more to it than this!

Artificial Intelligence

Problem: Complexity

Machine Learning: Part I
Introduction
What are ANNs?
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

- Neurons are *very* complex



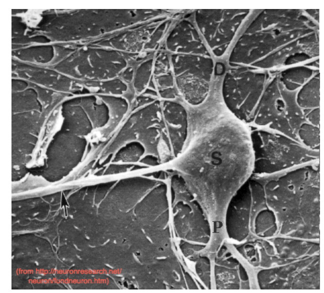
- Synapses: change potential across cell membrane
- Neuron effectively sums excitations, inhibitions
- At some point: potential at threshold and neuron *fires*
- Excitatory pulse down axon, release neurotransmitter at synapses
- *Lots* more to it than this!

Artificial Intelligence

Problem: Complexity

Machine Learning: Part I
Introduction
What are ANNs?
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

- Neurons are *very* complex



- Synapses: change potential across cell membrane
- Neuron effectively sums excitations, inhibitions
- At some point: potential at threshold and neuron *fires*
- Excitatory pulse down axon, release neurotransmitter at synapses
- *Lots* more to it than this!

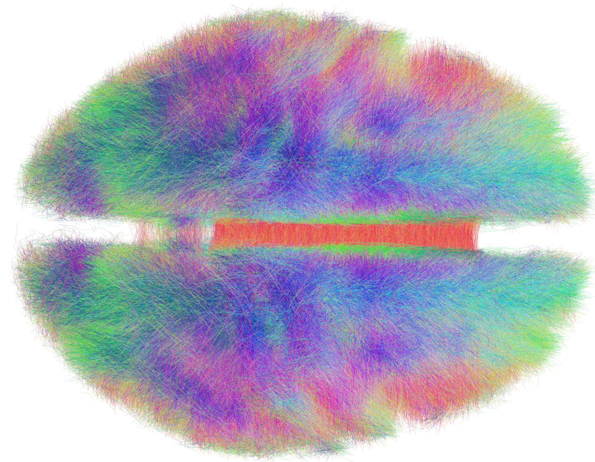
Artificial Intelligence

Problem: Complexity

- Connectsome is incredibly complex

Problem: Complexity

- Connectsome is incredibly complex



Andreashorn [CC BY-SA 4.0 (https://creativecommons.org/licenses/by-sa/4.0)]

Machine Learning: Part I

Introduction
What are ANNs?
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

Perceptrons

Perceptrons

Machine Learning: Part I

Introduction
Perceptrons
Perceptrons
Extending perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

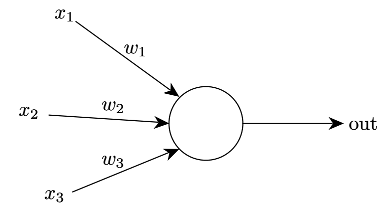
First simple artificial neuron: Perceptron

- McCulloch & Pitts
- *Very* simple model of a neuron

Artificial Intelligence

First simple artificial neuron: Perceptron

- McCulloch & Pitts
- *Very* simple model of a neuron



$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{threshold} \\ 1 & \text{otherwise} \end{cases}$$

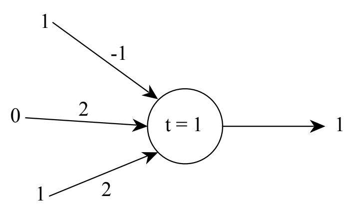
Machine Learning: Part I

Introduction
Perceptrons
Perceptrons
Extending perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

Artificial Intelligence

First simple artificial neuron: Perceptron

- McCulloch & Pitts
- *Very* simple model of a neuron



$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{threshold} \\ 1 & \text{otherwise} \end{cases}$$

Machine Learning: Part I

Introduction
Perceptrons
Perceptrons
Extending perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

Artificial Intelligence

First simple artificial neuron: Perceptron

- McCulloch & Pitts
- *Very* simple model of a neuron
- Usually change threshold to *bias* ($= -$threshold)

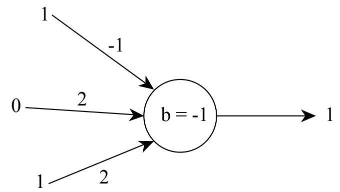
Machine Learning: Part I

Introduction
Perceptrons
Perceptrons
Extending perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

Artificial Intelligence

First simple artificial neuron: Perceptron

- McCulloch & Pitts
- *Very* simple model of a neuron
- Usually change threshold to *bias* (= −threshold)



$$\boxed{\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j + b \leq 0 \\ 1 & \text{otherwise} \end{cases}}$$

Artificial Intelligence

What can they do?

- "Weigh evidence" ⇒ decision

What can they do?

- "Weigh evidence" ⇒ decision
- E.g.:

What can they do?

- "Weigh evidence" ⇒ decision
- E.g.:
  - output = "study"

What can they do?

- "Weigh evidence" $\Rightarrow$ decision
- E.g.:
  - output = "study"
  - x_1 = test on Monday, x_2 = confident of material, x_3 = doing poorly in class

What can they do?

- "Weigh evidence" $\Rightarrow$ decision
- E.g.:
  - output = "study"
  - x_1 = test on Monday, x_2 = confident of material, x_3 = doing poorly in class
  - $w_1 = 1$, $w_2 = -1$, $w_3 = 2$

What can they do?

- "Weigh evidence" $\Rightarrow$ decision
- E.g.:
  - output = "study"
  - x_1 = test on Monday, x_2 = confident of material, x_3 = doing poorly in class
  - $w_1 = 1$, $w_2 = -1$, $w_3 = 2$
  - bias = 0

What can they do?

- "Weigh evidence" $\Rightarrow$ decision
- E.g.:
  - output = "study"
  - x_1 = test on Monday, x_2 = confident of material, x_3 = doing poorly in class
  - $w_1 = 1$, $w_2 = -1$, $w_3 = 2$
  - bias = 0
  - Test on Monday, confident, doing well in class $\Rightarrow$ output $= 0$

What can they do?

- "Weigh evidence" $\Rightarrow$ decision
- E.g.:
  - output = "study"
  - x_1 = test on Monday, x_2 = confident of material, x_3 = doing poorly in class
  - $w_1 = 1$, $w_2 = -1$, $w_3 = 2$
  - bias = 0
  - Test on Monday, confident, doing well in class $\Rightarrow$ output $= 0$
  - Test on Monday, not confident, doing well $\Rightarrow$ output $= 1$

What can they do?

Machine Learning: Part I

- "Weigh evidence" $\Rightarrow$ decision
- E.g.:
  - output = "study"
  - x_1 = test on Monday, x_2 = confident of material, x_3 = doing poorly in class
  - $w_1 = 1$, $w_2 = -1$, $w_3 = 2$
  - bias = 0
  - Test on Monday, confident, doing well in class $\Rightarrow$ output $= 0$
  - Test on Monday, not confident, doing well $\Rightarrow$ output $= 1$
  - Test on Monday, confident, doing poorly: $1 + (-1) + 2 = 2 \Rightarrow$ output $= 1$

Introduction
Perceptrons
Perceptrons
Extending perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

Copyright © 2019 UMaine School of Computing and Information Science

Artificial Intelligence

Learning the weights

Machine Learning: Part I

- Rosenblatt's perceptron algorithm

Introduction
Perceptrons
Perceptrons
Extending perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

Copyright © 2019 UMaine School of Computing and Information Science

Artificial Intelligence

Learning the weights

Machine Learning: Part I

- Rosenblatt's perceptron algorithm
- Use *training examples*

Introduction
Perceptrons
Perceptrons
Extending perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

Copyright © 2019 UMaine School of Computing and Information Science

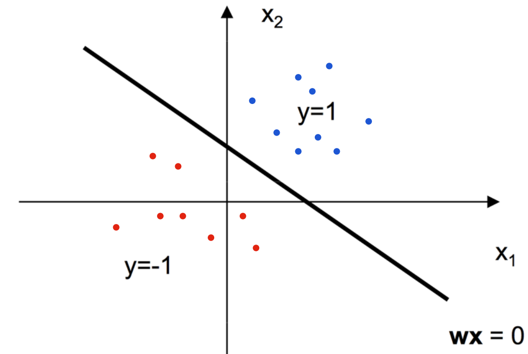
Artificial Intelligence

Learning the weights

- Rosenblatt's perceptron algorithm
- Use *training examples*
- Modify weights such that output error is minimized

Learning the weights

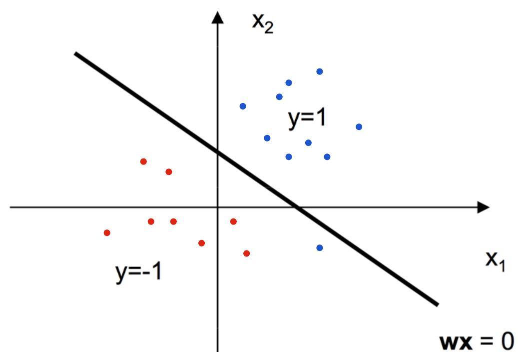
- Rosenblatt's perceptron algorithm
- Use *training examples*
- Modify weights such that output error is minimized



(from www.cs.stir.ac.uk/courses/CSC9YF/lectures/)

Learning the weights

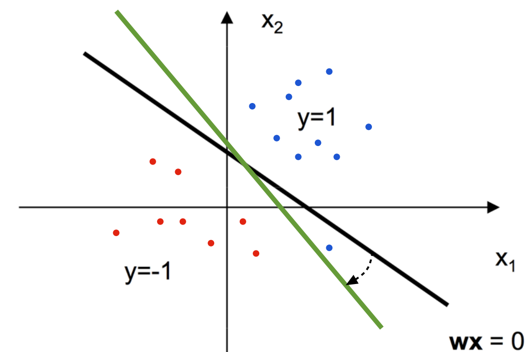
- Rosenblatt's perceptron algorithm
- Use *training examples*
- Modify weights such that output error is minimized



(from www.cs.stir.ac.uk/courses/CSC9YF/lectures/)

Learning the weights

- Rosenblatt's perceptron algorithm
- Use *training examples*
- Modify weights such that output error is minimized



(from www.cs.stir.ac.uk/courses/CSC9YF/lectures/)

Learning the weights

- Let:
  - y_k = desired output for example k
  - a_k = actual output for example k
- Error on example $k = y_k - a_k$
- Define an error function E_k for example k

$$E = \sum_k E_k = \frac{1}{2}\sum_k (y_k - a_k)^2$$

- Why?
  - Squaring make error always positive (parabola)
  - The 1/2 "makes the math easier" (as we'll see)

Learning the weights

Machine Learning: Part I

- Goal: minimize E by minimizing each E_k

Introduction
Perceptrons
Perceptrons
Extending perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

Copyright © 2019 UMaine School of Computing and Information Science

Learning the weights

Machine Learning: Part I

- Goal: minimize E by minimizing each E_k
- E_k is a function of the weights

Introduction
Perceptrons
Perceptrons
Extending perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

Copyright © 2019 UMaine School of Computing and Information Science

Learning the weights

Machine Learning: Part I

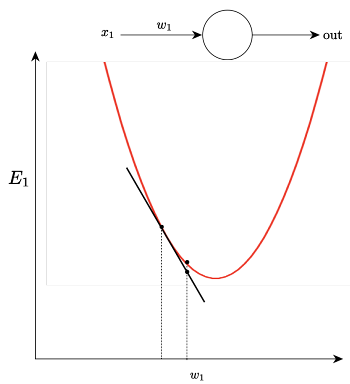
- Goal: minimize E by minimizing each E_k
- E_k is a function of the weights
- Use *gradient descent* instead

Introduction
Perceptrons
Perceptrons
Extending perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

Copyright © 2019 UMaine School of Computing and Information Science

Learning the weights

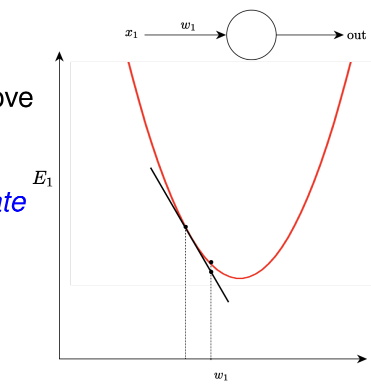
- Goal: minimize E by minimizing each E_k
- E_k is a function of the weights
- Use *gradient descent* instead
- With one weight:



Artificial Intelligence

Learning the weights

- Goal: minimize E by minimizing each E_k
- E_k is a function of the weights
- Use *gradient descent* instead
- With one weight:
- Slope at point: $\dfrac{dE_k}{dx_i}$
  tells which direction to move
  $$w_1' = w_1 - \alpha \frac{dE_k}{dx_i}$$
  where α is the *learning rate*



Artificial Intelligence

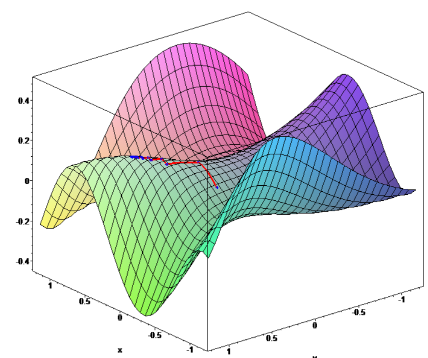
Learning the weights

- Suppose there are 2 weights, x and y

Artificial Intelligence

Learning the weights

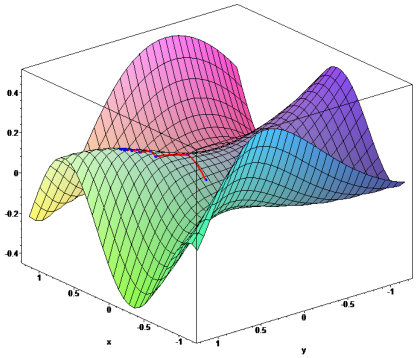
- Suppose there are 2 weights, x and y



Artificial Intelligence

Learning the weights

- Suppose there are 2 weights, x and y

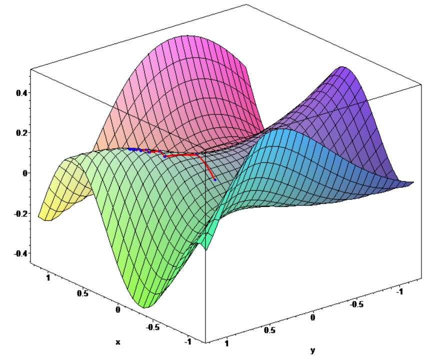


- Now "slope" is really the *gradient* ∇ at (x, y)

Artificial Intelligence

Learning the weights

- Suppose there are 2 weights, x and y



- Now "slope" is really the *gradient* ∇ at (x, y)

$$\nabla(w_i) = \frac{\partial E_k}{\partial w_i} \text{ and } w_{i,t+1} = w_{i,t} - \alpha \frac{\partial E_k}{\partial w_{i,t}}$$

- Gradient descent: hill-climbing in multiple dimensions

Artificial Intelligence

Learning the weights

- What is $\frac{\partial E_k}{\partial w_i}$?

Artificial Intelligence

Learning the weights

- What is $\frac{\partial E_k}{\partial w_i}$?
- We know that the output for k^{th} example $a_k = \sum_i w_i x_i$

Artificial Intelligence

Learning the weights

- What is $\frac{\partial E_k}{\partial w_i}$?
- We know that the output for k^{th} example $a_k = \sum_i w_i x_i$
- *Chain rule*:

$$\begin{aligned}\frac{\partial E_k}{\partial w_i} &= \frac{\partial E_k}{\partial a_k}\frac{\partial a_k}{\partial w_i} \\ &= \frac{\partial \frac{1}{2}(y_k - a_k)^2}{\partial a_k}\frac{\partial (w_1x_1 + w_2x_2 + \cdots w_nx_n)}{\partial w_i} \\ &= -(y_k - a_k)x_i\end{aligned}$$

Artificial Intelligence

Learning the weights

- What is $\frac{\partial E_k}{\partial w_i}$?
- We know that the output for k^{th} example $a_k = \sum_i w_i x_i$
- *Chain rule*:

$$\begin{aligned}\frac{\partial E_k}{\partial w_i} &= \frac{\partial E_k}{\partial a_k}\frac{\partial a_k}{\partial w_i} \\ &= \frac{\partial \frac{1}{2}(y_k - a_k)^2}{\partial a_k}\frac{\partial (w_1x_1 + w_2x_2 + \cdots w_nx_n)}{\partial w_i} \\ &= -(y_k - a_k)x_i\end{aligned}$$

- Since $\Delta w_i = \alpha \frac{\partial E_k}{\partial w_i}$, then

$$w_{i,t+1} = w_{i,t} - \alpha(-(y_k - a_k)x_i) = w_{i,t} + \alpha(y_k - a_k)x_i$$

Introduction
Perceptrons
Perceptrons
Extending perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

Copyright © 2019 UMaine School of Computing and Information Science

Artificial Intelligence

Learning the weights

- So for each example $< (x_1, x_2, \cdots, x_n), y >$
  - Compute output a
  - Adjust weights:

$$w_{i,t+1} = w_{i,t} + \alpha(y - a)x_i$$

for all weights weights w_i

Introduction
Perceptrons
Perceptrons
Extending perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

Copyright © 2019 UMaine School of Computing and Information Science

Artificial Intelligence

Implementations

- Algorithm first in IBM 704 in late 50s

Introduction
Perceptrons
Perceptrons
Extending perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

Copyright © 2019 UMaine School of Computing and Information Science

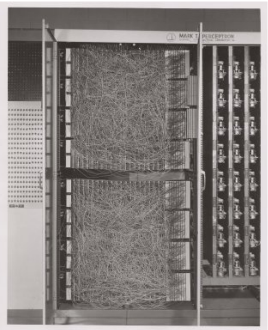
Artificial Intelligence

Implementations

- Algorithm first in IBM 704 in late 50s
- Then:

Implementations

- Algorithm first in IBM 704 in late 50s
- Then:



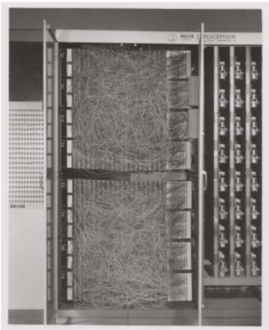
  - Mark I Perceptron Machine (Wikipedia)

Machine Learning: Part I
Introduction
Perceptrons
Perceptrons
Extending perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

Copyright © 2019 UMaine School of Computing and Information Science

Implementations

- Algorithm first in IBM 704 in late 50s
- Then:



  - Mark I Perceptron Machine (Wikipedia)
  - Image recognition: 20×20 photocell array
  - Potentiometers: weights
  - Pots adjusted by motors from learning

Machine Learning: Part I
Introduction
Perceptrons
Perceptrons
Extending perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

Copyright © 2019 UMaine School of Computing and Information Science

Example

```
(defclass perceptron ()
  ((num-inputs :initarg :num-inputs :initform 3
               :accessor num-inputs)
   (inputs :initarg :inputs :initform nil :accessor inputs)
   (weights :initarg :weights :initform nil
            :accessor weights)
   (bias :initarg :bias :initform 0 :accessor bias)
   (output :initarg :output :initform nil :accessor output)
   (target :initarg :target :initform nil :accessor target)
   (alpha :initarg :alpha :initform 1.0 :accessor alpha)
   )
  )

(defmethod initialize-instance :after ((self perceptron)
                                       &rest l)
  (declare (ignore l))
  (with-slots (num-inputs weights) self
    (setq weights
      (loop for i from 1 to num-inputs
          collect (random 1.0)))))
```

Machine Learning: Part I
Introduction
Perceptrons
Perceptrons
Extending perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

Copyright © 2019 UMaine School of Computing and Information Science

Example


```
(defmethod compute-output ((self perceptron))
  (with-slots (output bias inputs weights) self
    (setq output (if (> (+ bias
                           (apply #'+
                                  (mapcar #'* inputs
                                          weights)))
                        0.0)
                     1
                     0))))

(defmethod adjust-weights ((self perceptron))
  (with-slots (inputs weights target output alpha) self
    (compute-output self)
    (let ((delta (loop for weight in weights
     for input in inputs
                       collect (* alpha (- target output)
input))))
      (format t
      "~s -> ~s (desired = ~s), weights=~s, delta=~s~%"
              inputs output target weights delta)
      (setq weights (mapcar #'+ weights delta))
      (format t "     new weights=~s~%" weights))))
```


Artificial Intelligence

Example


```
(defmethod train ((self perceptron) examples)
  (with-slots (inputs target output weights) self
    (loop for count from 1 to (length examples)
        for example in examples
        do (setf inputs (car example)
                 target (cadr example))
           (compute-output self)
           (adjust-weights self)
           (compute-output self)
           )))
```


Artificial Intelligence

Example


```
(defvar *perceptron* nil)

(defun train-for-tt (&key new? examples (bias -6) (inputs 3))
  (when new?
    (setq *perceptron* (make-instance 'perceptron
 :bias bias :num-inputs inputs)))
  (train *perceptron* examples)
  ;; now check it:
  (loop for thing in examples
      do (setf (inputs *perceptron*) (car thing))
 (compute-output *perceptron*)
 (format t "~s => ~s~%" (car thing)
 (output *perceptron*))))
```


Artificial Intelligence

Example


```
(defvar *and-tt* '(((0 0) 0)
   ((0 1) 0)
   ((1 0) 0)
   ((1 1) 1)))

(defvar *or-tt* '(((0 0) 0)
  ((0 1) 1)
  ((1 0) 1)
  ((1 1) 1)))

(defvar *xor-tt* '(((0 0) 0)
   ((0 1) 1)
   ((1 0) 1)
   ((1 1) 0)))
```


Artificial Intelligence

What can it do?

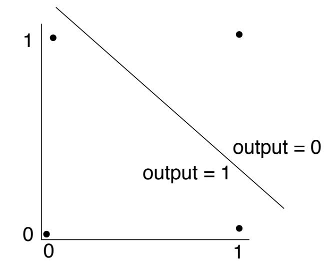
- *Linear classifier*:
  - Finds line/plane/hyperplane separating class 1 from class 2
    - 2 inputs ⇒ line between sets
    - 3 inputs ⇒ plane, etc.
  - Sets can be separated by hyperplane ⇒ *linearly-separable*
- Training set linearly-separable, algorithm converges
- Example: can learn NAND function

What can it do?

Machine Learning: Part I

Introduction
Perceptrons
Perceptrons
Extending perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

- *Linear classifier*:
  - Finds line/plane/hyperplane separating class 1 from class 2
    - 2 inputs ⇒ line between sets
    - 3 inputs ⇒ plane, etc.
  - Sets can be separated by hyperplane ⇒ *linearly-separable*
- Training set linearly-separable, algorithm converges
- Example: can learn NAND function



Problems

Machine Learning: Part I

Introduction
Perceptrons
Perceptrons
Extending perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

- May not be a unique solution

Problems

Machine Learning: Part I

Introduction
Perceptrons
Perceptrons
Extending perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

- May not be a unique solution
  - Thus may have suboptimal learning
  - *Support vector machine* (SM): "perceptron of optimal stability"

Problems

- May not be a unique solution
  - Thus may have suboptimal learning
  - *Support vector machine* (SM): “perceptron of optimal stability”
- Worse problem: can’t learn even simple non-linearly-separable function

Artificial Intelligence

Machine Learning: Part I

Problems

- May not be a unique solution
  - Thus may have suboptimal learning
  - *Support vector machine* (SM): “perceptron of optimal stability”
- Worse problem: can’t learn even simple non-linearly-separable function
  - Minsky & Papert (1960): *Perceptrons* book
  - Perceptron can’t learn XOR function

Introduction
Perceptrons
Perceptrons
Extending perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

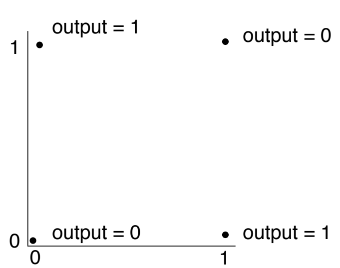
Copyright © 2019 UMaine School of Computing and Information Science

Artificial Intelligence

Machine Learning: Part I

Problems

- May not be a unique solution
  - Thus may have suboptimal learning
  - *Support vector machine* (SM): “perceptron of optimal stability”
- Worse problem: can’t learn even simple non-linearly-separable function
  - Minsky & Papert (1960): *Perceptrons* book
  - Perceptron can’t learn XOR function



Introduction
Perceptrons
Perceptrons
Extending perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

Copyright © 2019 UMaine School of Computing and Information Science

Artificial Intelligence

Machine Learning: Part I

Problems

- May not be a unique solution
  - Thus may have suboptimal learning
  - *Support vector machine* (SM): “perceptron of optimal stability”
- Worse problem: can’t learn even simple non-linearly-separable function
  - Minsky & Papert (1960): *Perceptrons* book
  - Perceptron can’t learn XOR function
  - Killed perceptron research for a while

Introduction
Perceptrons
Perceptrons
Extending perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

Copyright © 2019 UMaine School of Computing and Information Science

Artificial Intelligence

Extending perceptrons

Perceptron networks

- Single perceptron: very limited

Machine Learning: Part I
Introduction
Perceptrons
Perceptrons
Extending perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

Artificial Intelligence

Perceptron networks

- Single perceptron: very limited
- Idea: hook a bunch together in a network

Machine Learning: Part I
Introduction
Perceptrons
Perceptrons
Extending perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

Artificial Intelligence

Perceptron networks

- Single perceptron: very limited
- Idea: hook a bunch together in a network
- What can a perceptron network do?

Machine Learning: Part I
Introduction
Perceptrons
Perceptrons
Extending perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

Artificial Intelligence

Perceptron networks

- Single perceptron: very limited
- Idea: hook a bunch together in a network
- What can a perceptron network do?
  - Based on what you know, what do *you* think?

Perceptron networks

Machine Learning: Part I

- Single perceptron: very limited
- Idea: hook a bunch together in a network
- What can a perceptron network do?
  - Based on what you know, what do *you* think?
  - Hint: perceptron can implement NAND function

Introduction
Perceptrons
Perceptrons
Extending perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

Copyright © 2019 UMaine School of Computing and Information Science

Perceptron networks

Machine Learning: Part I

- Single perceptron: very limited
- Idea: hook a bunch together in a network
- What can a perceptron network do?
  - Based on what you know, what do *you* think?
  - Hint: perceptron can implement NAND function
  - NAND forms complete gate/boolean function set

Introduction
Perceptrons
Perceptrons
Extending perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

Copyright © 2019 UMaine School of Computing and Information Science

Perceptron networks

Machine Learning: Part I

- Single perceptron: very limited
- Idea: hook a bunch together in a network
- What can a perceptron network do?
  - Based on what you know, what do *you* think?
  - Hint: perceptron can implement NAND function
  - NAND forms complete gate/boolean function set
  - $\therefore \Rightarrow$ any binary function, $\Rightarrow$ Turing-equivalent

Introduction
Perceptrons
Perceptrons
Extending perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

Copyright © 2019 UMaine School of Computing and Information Science

Perceptron networks

- Single perceptron: very limited
- Idea: hook a bunch together in a network
- What can a perceptron network do?
  - Based on what you know, what do *you* think?
  - Hint: perceptron can implement NAND function
  - NAND forms complete gate/boolean function set
  - $\therefore \Rightarrow$ any binary function, $\Rightarrow$ Turing-equivalent
- E.g., a half-adder

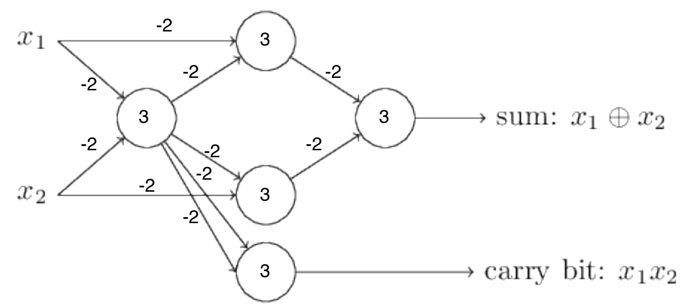
Artificial Intelligence

Perceptron networks

Machine Learning: Part I

Introduction
Perceptrons
Perceptrons
Extending perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

- Single perceptron: very limited
- Idea: hook a bunch together in a network
- What can a perceptron network do?
  - Based on what you know, what do *you* think?
  - Hint: perceptron can implement NAND function
  - NAND forms complete gate/boolean function set
  - $\therefore \Rightarrow$ any binary function, $\Rightarrow$ Turing-equivalent
- E.g., a half-adder:



Cc

Artificial Intelligence

Learning

Machine Learning: Part I

Introduction
Perceptrons
Perceptrons
Extending perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

- So we're done, right?

Artificial Intelligence

Learning

Machine Learning: Part I

Introduction
Perceptrons
Perceptrons
Extending perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

- So we're done, right?
- Not quite

Artificial Intelligence

Learning

- So we're done, right?
- Not quite
- Want property: small $\Delta w \Rightarrow$ small Δoutput

Learning

Machine Learning: Part I

Introduction
Perceptrons
Perceptrons
Extending perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

- So we're done, right?
- Not quite
- Want property: small $\Delta w \Rightarrow$ small Δoutput
- But perceptrons have *step function*

Copyright © 2019 UMaine School of Computing and Information Science

Artificial Intelligence

Learning

Machine Learning: Part I

Introduction
Perceptrons
Perceptrons
Extending perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

- So we're done, right?
- Not quite
- Want property: small $\Delta w \Rightarrow$ small Δoutput
- But perceptrons have *step function*
- Small input change can give completely different output

Copyright © 2019 UMaine School of Computing and Information Science

Artificial Intelligence

Learning

Machine Learning: Part I

Introduction
Perceptrons
Perceptrons
Extending perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

- So we're done, right?
- Not quite
- Want property: small $\Delta w \Rightarrow$ small Δoutput
- But perceptrons have *step function*
- Small input change can give completely different output
- Step function isn't *differentiable*

Copyright © 2019 UMaine School of Computing and Information Science

Artificial Intelligence

Learning

- So we're done, right?
- Not quite
- Want property: small $\Delta w \Rightarrow$ small Δoutput
- But perceptrons have *step function*
- Small input change can give completely different output
- Step function isn't *differentiable*
- Can't easily find weights $\Rightarrow$ minimum error

Nonlinear neurons

Nonlinear neurons

- Instead of step function, use differentiable function
- E.g.: use *sigmoid neurons* (*logistic neurons*)

Machine Learning: Part I
Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

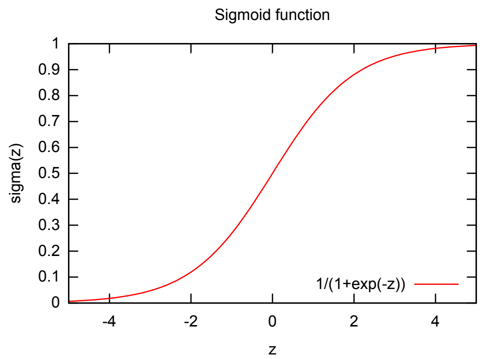
Nonlinear neurons

- Instead of step function, use differentiable function
- E.g.: use *sigmoid neurons* (*logistic neurons*)
- Output is sigmoid function $\sigma(z) = \dfrac{1}{1 + e^{-z}}$

Machine Learning: Part I
Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

Nonlinear neurons

- Instead of step function, use differentiable function
- E.g.: use *sigmoid neurons* (*logistic neurons*)
- Output is sigmoid function $\sigma(z) = \dfrac{1}{1+e^{-z}}$



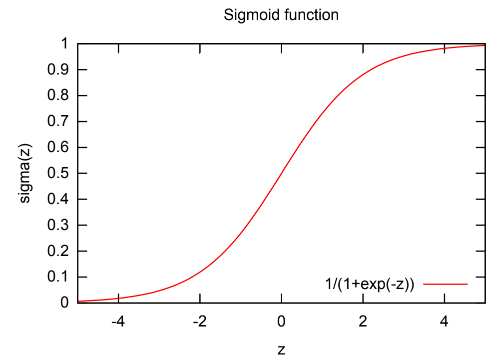
Artificial Intelligence

Nonlinear neurons

Machine Learning: Part I

Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

- Instead of step function, use differentiable function
- E.g.: use *sigmoid neurons* (*logistic neurons*)
- Output is sigmoid function $\sigma(z) = \dfrac{1}{1+e^{-z}}$



- What is z in this case?

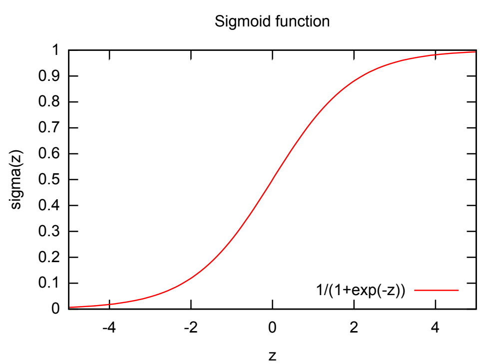
Artificial Intelligence

Nonlinear neurons

Machine Learning: Part I

Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

- Instead of step function, use differentiable function
- E.g.: use *sigmoid neurons* (*logistic neurons*)
- Output is sigmoid function $\sigma(z) = \dfrac{1}{1+e^{-z}}$



- What is z in this case?$\Rightarrow$ weighted input, bias sum

Artificial Intelligence

Notation changes

Machine Learning: Part I

Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

- Perceptron output function: only output 1 if

$$\sum_j w_j x_j + b > 0$$

Artificial Intelligence

Notation changes

- Perceptron output function: only output 1 if

$$\sum_j w_j x_j + b > 0$$

- Let's represent all the w_j, x_j as *vectors* $\mathbf{w}$, $\mathbf{x}$

Notation changes

- Perceptron output function: only output 1 if

$$\sum_j w_j x_j + b > 0$$

- Let's represent all the w_j, x_j as *vectors* $\mathbf{w}$, $\mathbf{x}$
- Now we can use *dot product* instead of summation

Notation changes

- Perceptron output function: only output 1 if

$$\sum_j w_j x_j + b > 0$$

- Let's represent all the w_j, x_j as *vectors* $\mathbf{w}$, $\mathbf{x}$
- Now we can use *dot product* instead of summation:

$$[w_1 \ w_2 \ \cdots w_n] \cdot [x_1 \ x_2 \ \cdots x_n] = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$$

- So perceptron outputs 1 when $\mathbf{w} \cdot \mathbf{x} + b > 0$

Sigmoid neurons

- Sigmoid:

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

- What is z?

Sigmoid neurons

- Sigmoid:
$$\sigma(z) = \frac{1}{1+e^{-z}}$$
- What is z?
It is the *logit*: $z = \mathbf{w} \cdot \mathbf{x} + b$

Sigmoid neurons

- Sigmoid:
$$\sigma(z) = \frac{1}{1+e^{-z}}$$
- What is z?
It is the *logit*: $z = \mathbf{w} \cdot \mathbf{x} + b$
- Small change in w or $b \Rightarrow$ small Δ in z and $\sigma(z)$

Sigmoid neurons

- Sigmoid:
$$\sigma(z) = \frac{1}{1+e^{-z}}$$
- What is z?
It is the *logit*: $z = \mathbf{w} \cdot \mathbf{x} + b$
- Small change in w or $b \Rightarrow$ small Δ in z and $\sigma(z)$
- $\sigma(z)$ is differentiable

Sigmoid neurons

- Sigmoid:
$$\sigma(z) = \frac{1}{1+e^{-z}}$$
- What is z?
It is the *logit*: $z = \mathbf{w} \cdot \mathbf{x} + b$
- Small change in w or $b \Rightarrow$ small Δ in z and $\sigma(z)$
- $\sigma(z)$ is differentiable
- Δoutput approximated by derivative of function at point:

Sigmoid neurons

- Sigmoid:
$$\sigma(z) = \frac{1}{1+e^{-z}}$$
- What is z?
  It is the *logit*: $z = \mathbf{w} \cdot \mathbf{x} + b$
- Small change in w or $b \Rightarrow$ small Δ in z and $\sigma(z)$
- $\sigma(z)$ is differentiable
- Δoutput approximated by derivative of function at point:
$$\Delta\text{output} \approx \Sigma_j \frac{\partial\,\text{output}}{\partial w_j}\Delta w_j + \frac{\partial\,\text{output}}{\partial b}\Delta b$$

Sigmoid neurons

Machine Learning: Part I
Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

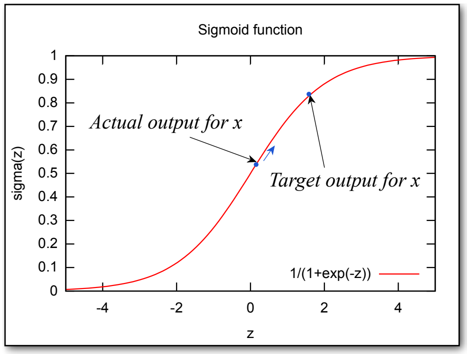
- Sigmoid:
$$\sigma(z) = \frac{1}{1+e^{-z}}$$
- What is z?
  It is the *logit*: $z = \mathbf{w} \cdot \mathbf{x} + b$
- Small change in w or $b \Rightarrow$ small Δ in z and $\sigma(z)$
- $\sigma(z)$ is differentiable
- Δoutput approximated by derivative of function at point:
$$\Delta\text{output} \approx \Sigma_j \frac{\partial\,\text{output}}{\partial w_j}\Delta w_j + \frac{\partial\,\text{output}}{\partial b}\Delta b$$
- Now Δoutput is a *linear* function of changes of weights & bias

Incorporating the bias

- Neuron has inputs x_i and weights w_i, $i = 1, 2, \ldots n$
- Sometimes add x_0, w_0 to replace *bias*:
  - Bias = $x_0 w_0$
  - $x_0 = 1$, w_0 is learned
- $\mathbf{x} = [x_0\ x_1 \ldots x_n]^T$, $w = [w_0\ w_1 \ldots w_n]^T$
- $z = \sum_{i=0}^{n} w_i x_i = \mathbf{w} \cdot \mathbf{x}$ is the *activation* of the neuron
- $y = f(z) = \dfrac{1}{1+e^{-z}}$ is the output (“activity”) of neuron

Training a sigmoid neuron

Same basic idea as in training a perceptron:



- Want to choose Δz to move output toward target
- Determine slope at z
- Move in direction of increasing slope
- Problem: z isn’t a variable: it’s a dot product!
- Vector x is fixed (for an example)
- So we need to change vector w to move z to move toward target for same x

Derivatives of logistic neuron

- Derivative of logit z wrt. weights, inputs:

$$z = b + \sum_i w_i x_i$$

$$\frac{\partial z}{\partial w_i} = x_i, \quad \frac{\partial z}{\partial x_i} = w_i$$

- Derivative of logistic equation:

$$\begin{aligned} y &= \frac{1}{1+e^{-z}} \\ \frac{dy}{dz} &= \frac{1}{1+e^{-z}}\left(1 - \frac{1}{1+e^{-z}}\right) \\ &= y(1-y) \end{aligned}$$

Derivatives of logistic neuron

Machine Learning: Part I
Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

- Use chain rule to differentiate y wrt w_i:

$$\frac{\partial y}{\partial w_i} = \frac{\partial z}{\partial w_i}\frac{dy}{dz} = x_i y(1-y)$$

- Can get derivative of error wrt w_i:

$$\frac{\partial E}{\partial w_i} = \sum_n \frac{\partial y^n}{\partial w_i}\frac{\partial E}{\partial y^n} = -\sum_n x_i^n y^n (1-y^n)(a^n - y^n)$$

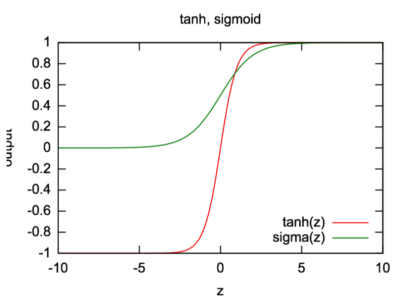
  where a^n means "a from training example n"
- First, last term ⇒ delta rule
- Middle term: slope of logistic equation

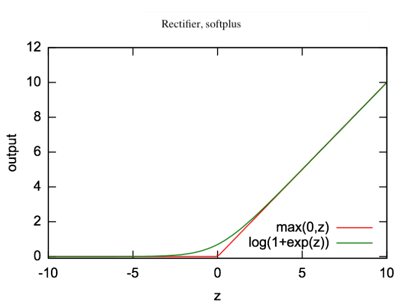
Copyright © 2019 UMaine School of Computing and Information Science
Artificial Intelligence

Other non-linear neurons

Machine Learning: Part I
Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

- Other non-perceptron neurons possible, often used
- tanh(z), rectifier, softplus, . . .





Copyright © 2019 UMaine School of Computing and Information Science
Artificial Intelligence

Feedforward neural networks

Machine Learning: Part I
Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

Feedforward networks

- Networks of sigmoid (or other non-perceptron) neurons
- Multiple *layers*
  - Input layer
  - Output layer
  - 1 or more *hidden layers*
- Sometimes: *multilayer perceptrons* (MLP) – though not perceptrons
- In *feedforward* net: inputs → hidden layers → outputs
- Often *dense* networks (fully-connected between layers)

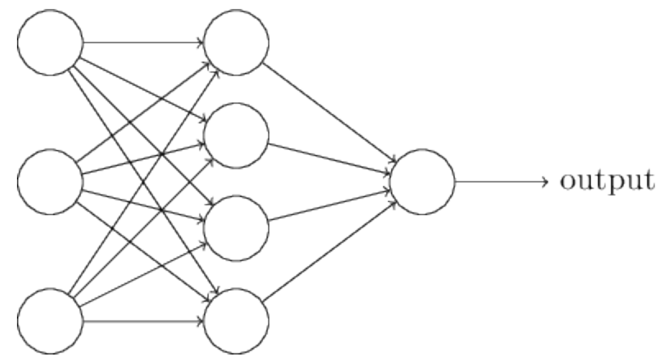
Artificial Intelligence

FFN

Machine Learning: Part I

Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

- Could be simple, moderately complex, very complex



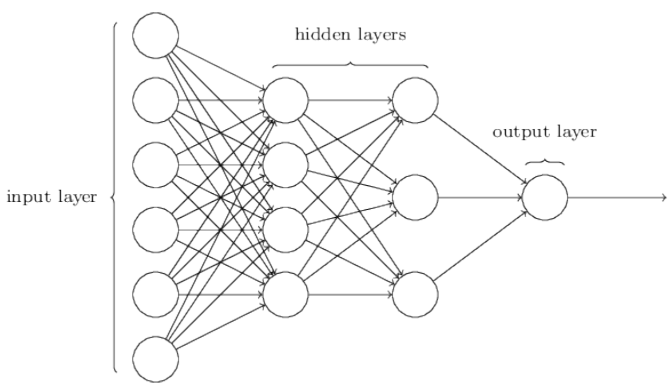
Artificial Intelligence

FFN

Machine Learning: Part I

Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

- Could be simple, moderately complex, very complex



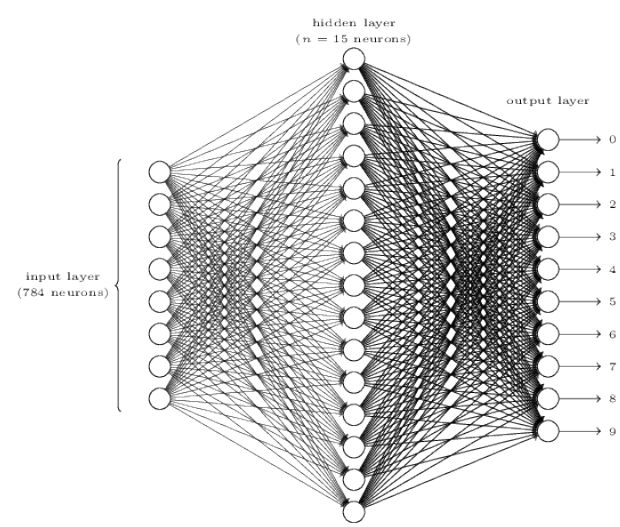
Artificial Intelligence

FFN

Machine Learning: Part I

Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

- Could be simple, moderately complex, very complex



Artificial Intelligence

Inputs, outputs

- Inputs?
  - “Clamped” to some activation
  - Some “natural” representation
- Outputs?
  - Classification or encoding?
  - E.g., numeral recognition:
    - Neuron for each numeral
    - Why not binary coding?

What can FFNs learn?

Machine Learning: Part I

Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

- Output is composition of multiple “soft” thresholds

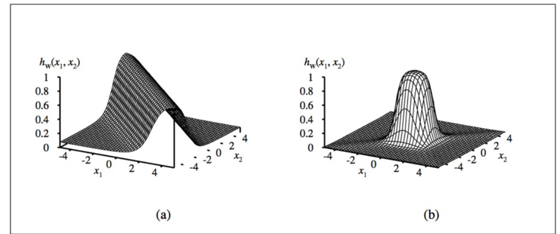


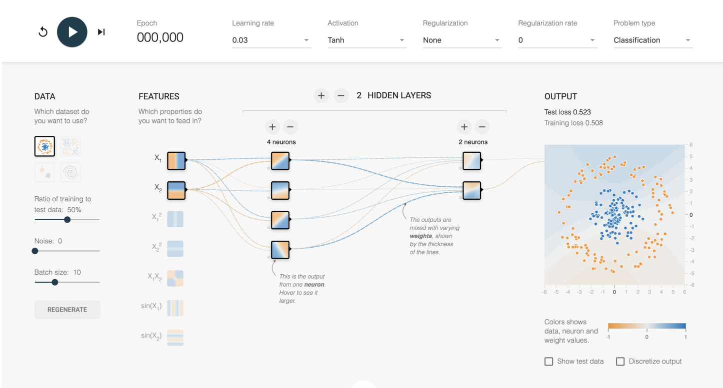
Figure 18.23 (a) The result of combining two opposite-facing soft threshold functions to produce a ridge. (b) The result of combining two ridges to produce a bump.

- FFN w/ single hidden layer: any continuous function, any desired precision (w/ enough neurons)
- ≥ 2 layers: discontinuous, too
- How many neurons?
  - Exponential in the inputs
  - Need $\mathcal{O}(2^n/n)$ for all Boolean functions of n inputs

Example: NN simulator

Machine Learning: Part I

Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary



Machine Learning: Part I

Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

Matrix form of NN

Notation (from Nielsen)

- Assume a multilayer FF network
- w^l_{jk}: wt from neuron k in layer $l-1$ to neuron j in layer l
- Subscript: jk for ease of calculation (later)
- b^l_j: bias of neuron j in layer l

Artificial Intelligence

Notation (from Nielsen)

Machine Learning: Part I

Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

- Assume a multilayer FF network
- w^l_{jk}: wt from neuron k in layer $l-1$ to neuron j in layer l
- Subscript: jk for ease of calculation (later)
- b^l_j: bias of neuron j in layer l
- a^l_j: activation (output) of neuron j in layer l

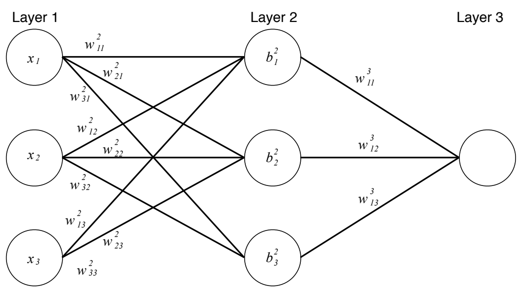
$$a^l_j = \sigma\left(\sum_k w^l_{jk} a^{l-1}_k + b^l_j\right)$$

Artificial Intelligence

Matrix form of NN

Machine Learning: Part I

Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

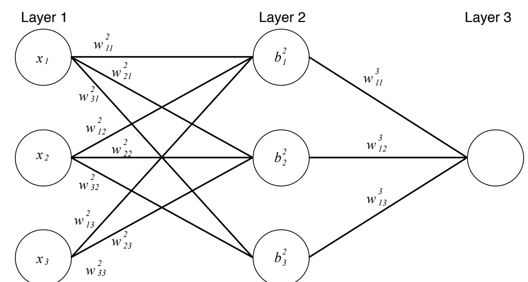


Artificial Intelligence

Matrix form of NN

Machine Learning: Part I

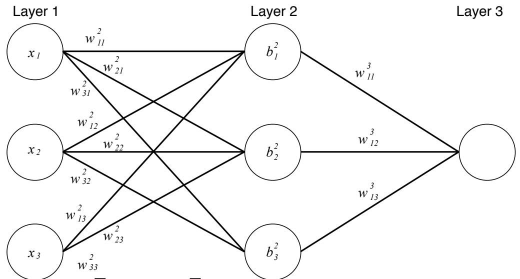
Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary



$$\mathbf{w}^2 = \begin{bmatrix} w^2_{11} & w^2_{12} & w^2_{13} \\ w^2_{21} & w^2_{22} & w^2_{23} \\ w^2_{31} & w^2_{32} & w^2_{33} \end{bmatrix} \qquad \mathbf{w}^3 = \begin{bmatrix} w^3_{11} & w^3_{12} & w^3_{13} \end{bmatrix}$$

Artificial Intelligence

Matrix form of NN



$$\mathbf{w^2} = \begin{bmatrix} w^2_{11} & w^2_{12} & w^2_{13} \\ w^2_{21} & w^2_{22} & w^2_{23} \\ w^2_{31} & w^2_{32} & w^2_{33} \end{bmatrix} \qquad \mathbf{w^3} = \begin{bmatrix} w^3_{11} & w^3_{12} & w^3_{13} \end{bmatrix}$$

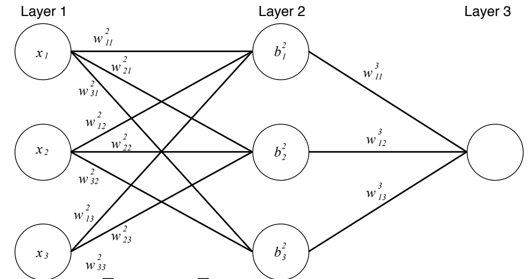
$$\mathbf{z^2} = \begin{bmatrix} (w^2_{11}x_1 + w^2_{12}x_2 + w^2_{13}x_3) + b^2_1 \\ (w^2_{21}x_1 + w^2_{22}x_2 + w^2_{23}x_3) + b^2_2 \\ (w^2_{31}x_1 + w^2_{32}x_2 + w^2_{33}x_3) + b^2_3 \end{bmatrix} = \begin{bmatrix} (w^2_{11}x_1 + w^2_{12}x_2 + w^2_{13}x_3) \\ (w^2_{21}x_1 + w^2_{22}x_2 + w^2_{23}x_3) \\ (w^2_{31}x_1 + w^2_{32}x_2 + w^2_{33}x_3) \end{bmatrix} + \begin{bmatrix} b^2_1 \\ b^2_2 \\ b^2_3 \end{bmatrix} = \begin{bmatrix} w^2_{11} & w^2_{12} & w^2_{13} \\ w^2_{21} & w^2_{22} & w^2_{23} \\ w^2_{31} & w^2_{32} & w^2_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b^2_1 \\ b^2_2 \\ b^2_3 \end{bmatrix} = \mathbf{w^2 x + b^2}$$

Artificial Intelligence

Matrix form of NN

Machine Learning: Part I

Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary



$$\mathbf{w^2} = \begin{bmatrix} w^2_{11} & w^2_{12} & w^2_{13} \\ w^2_{21} & w^2_{22} & w^2_{23} \\ w^2_{31} & w^2_{32} & w^2_{33} \end{bmatrix} \qquad \mathbf{w^3} = \begin{bmatrix} w^3_{11} & w^3_{12} & w^3_{13} \end{bmatrix}$$

$$\mathbf{z^2} = \begin{bmatrix} (w^2_{11}x_1 + w^2_{12}x_2 + w^2_{13}x_3) + b^2_1 \\ (w^2_{21}x_1 + w^2_{22}x_2 + w^2_{23}x_3) + b^2_2 \\ (w^2_{31}x_1 + w^2_{32}x_2 + w^2_{33}x_3) + b^2_3 \end{bmatrix} = \begin{bmatrix} (w^2_{11}x_1 + w^2_{12}x_2 + w^2_{13}x_3) \\ (w^2_{21}x_1 + w^2_{22}x_2 + w^2_{23}x_3) \\ (w^2_{31}x_1 + w^2_{32}x_2 + w^2_{33}x_3) \end{bmatrix} + \begin{bmatrix} b^2_1 \\ b^2_2 \\ b^2_3 \end{bmatrix} = \begin{bmatrix} w^2_{11} & w^2_{12} & w^2_{13} \\ w^2_{21} & w^2_{22} & w^2_{23} \\ w^2_{31} & w^2_{32} & w^2_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b^2_1 \\ b^2_2 \\ b^2_3 \end{bmatrix} = \mathbf{w^2 x + b^2}$$

$$\boxed{\mathbf{a^2} = \sigma(\mathbf{w^2 x + b^2})}$$

Copyright © 2019 UMaine School of Computing and Information Science

Artificial Intelligence

Matrix form

Machine Learning: Part I

Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

- General equation:

$$a^l = \sigma(w^l a^{l-1} + b^l)$$

- σ is said to be "vectorized"
- Logit (weighted input) vector z^l is important, too

$$z^l = w^l a^{l-1} + b^l$$

- So $a^l = \sigma(z^l)$

Copyright © 2019 UMaine School of Computing and Information Science

Artificial Intelligence

Machine Learning: Part I

Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

Gradient descent learning in FFNs

What are we learning?

- Network computes function of inputs
- Single output, n inputs $\mathbf{w}$: $h_{\mathbf{w}}(\mathbf{X})$
- What if $m > 1$ outputs?
  - Single layer net: separate into m nets, train separately
  - Multilayer: all outputs depend on hidden layer weights
  - $\Rightarrow$ vector function
- Output function $\mathbf{h}_{\mathbf{w}}(\mathbf{x})$:

Machine Learning: Part I

What are we learning?

- Network computes function of inputs
- Single output, n inputs $\mathbf{w}$: $h_{\mathbf{w}}(\mathbf{X})$
- What if $m > 1$ outputs?
  - Single layer net: separate into m nets, train separately
  - Multilayer: all outputs depend on hidden layer weights
  - $\Rightarrow$ vector function
- Output function $\mathbf{h}_{\mathbf{w}}(\mathbf{x})$:

$$\mathbf{h}_{\mathbf{w}}(\mathbf{x}) = \mathbf{a}^L = \sigma(\mathbf{w}^L \mathbf{a}^{l-1} + \mathbf{b}^L)$$

Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

Copyright © 2019 UMaine School of Computing and Information Science

Machine Learning: Part I

What are we learning?

- Network computes function of inputs
- Single output, n inputs $\mathbf{w}$: $h_{\mathbf{w}}(\mathbf{X})$
- What if $m > 1$ outputs?
  - Single layer net: separate into m nets, train separately
  - Multilayer: all outputs depend on hidden layer weights
  - $\Rightarrow$ vector function
- Output function $\mathbf{h}_{\mathbf{w}}(\mathbf{x})$:

$$\begin{aligned}
\mathbf{h}_{\mathbf{w}}(\mathbf{x}) &= \mathbf{a}^L = \sigma(\mathbf{w}^L \mathbf{a}^{l-1} + \mathbf{b}^L) \\
&= \sigma(\mathbf{w}^L(\sigma(\mathbf{w}^{l-1}\mathbf{a}^{l-2} + \mathbf{b}^{l-1}) + \mathbf{b}^L)
\end{aligned}$$

Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

Copyright © 2019 UMaine School of Computing and Information Science

Machine Learning: Part I

What are we learning?

- Network computes function of inputs
- Single output, n inputs $\mathbf{w}$: $h_{\mathbf{w}}(\mathbf{X})$
- What if $m > 1$ outputs?
  - Single layer net: separate into m nets, train separately
  - Multilayer: all outputs depend on hidden layer weights
  - $\Rightarrow$ vector function
- Output function $\mathbf{h}_{\mathbf{w}}(\mathbf{x})$:

$$\begin{aligned}
\mathbf{h}_{\mathbf{w}}(\mathbf{x}) &= \mathbf{a}^L = \sigma(\mathbf{w}^L \mathbf{a}^{l-1} + \mathbf{b}^L) \\
&= \sigma(\mathbf{w}^L(\sigma(\mathbf{w}^{l-1}\mathbf{a}^{l-2} + \mathbf{b}^{l-1}) + \mathbf{b}^L) \\
&\cdots \\
&= \sigma(\mathbf{w}^L(\sigma(\cdots\sigma(\mathbf{w}^2\mathbf{x} + b^2)\cdots)) + \mathbf{b}^L)
\end{aligned}$$

Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

Copyright © 2019 UMaine School of Computing and Information Science

Error function

- First, let's eliminate **b** $\Rightarrow$ into **x**
- Error of network:
  - Let **b** = desired output
  - Error on training example **x**:

$$\mathbf{E}_\mathbf{w}(\mathbf{x}) = \mathbf{y} - \mathbf{h}_\mathbf{w}(\mathbf{x})$$

- But:
  - $\mathbf{E}_\mathbf{w}(\mathbf{x})$: positive/negative
  - We don't want any particular error element: want *average* error
  - Want to learn weights, so want a function of weights

Artificial Intelligence

Machine Learning: Part I

Cost (loss) function

- Define a *cost* (loss, objective) function:

$$C_\mathbf{x}(\mathbf{w}) = \frac{1}{2}||(\mathbf{E}_\mathbf{w}(\mathbf{x}))||^2$$

Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

Copyright © 2019 UMaine School of Computing and Information Science

Artificial Intelligence

Machine Learning: Part I

Cost (loss) function

- Define a *cost* (loss, objective) function:

$$\begin{aligned} C_\mathbf{x}(\mathbf{w}) &= \frac{1}{2}||(\mathbf{E}_\mathbf{w}(\mathbf{x}))||^2 \\ &= \frac{1}{2}||\mathbf{y} - \mathbf{h}_\mathbf{w}(\mathbf{x})||^2 \end{aligned}$$

Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

Copyright © 2019 UMaine School of Computing and Information Science

Artificial Intelligence

Machine Learning: Part I

Cost (loss) function

- Define a *cost* (loss, objective) function:

$$\begin{aligned} C_\mathbf{x}(\mathbf{w}) &= \frac{1}{2}||(\mathbf{E}_\mathbf{w}(\mathbf{x}))||^2 \\ &= \frac{1}{2}||\mathbf{y} - \mathbf{h}_\mathbf{w}(\mathbf{x})||^2 \\ &= \frac{1}{2}\sum_m (y_m - a_m^L)^2 \end{aligned}$$

Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

Copyright © 2019 UMaine School of Computing and Information Science

Artificial Intelligence

Cost (loss) function

- Define a *cost* (loss, objective) function:

$$\begin{aligned} C_{\mathbf{x}}(\mathbf{w}) &= \frac{1}{2}||(\mathbf{E}_{\mathbf{w}}(\mathbf{x}))||^2 \\ &= \frac{1}{2}||\mathbf{y} - \mathbf{h}_{\mathbf{w}}(\mathbf{x})||^2 \\ &= \frac{1}{2}\sum_m (y_m - a_m^L)^2 \end{aligned}$$

- $C_{\mathbf{x}}(\mathbf{w})$: *quadratic cost (MSE) function*
- Entire cost function: average over all $\mathbf{x}_i$:

Cost (loss) function

Machine Learning: Part I

Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

- Define a *cost* (loss, objective) function:

$$\begin{aligned} C_{\mathbf{x}}(\mathbf{w}) &= \frac{1}{2}||(\mathbf{E}_{\mathbf{w}}(\mathbf{x}))||^2 \\ &= \frac{1}{2}||\mathbf{y} - \mathbf{h}_{\mathbf{w}}(\mathbf{x})||^2 \\ &= \frac{1}{2}\sum_m (y_m - a_m^L)^2 \end{aligned}$$

- $C_{\mathbf{x}}(\mathbf{w})$: *quadratic cost (MSE) function*
- Entire cost function: average over all $\mathbf{x}_i$:

$$C(\mathbf{w}) = \frac{1}{n}\sum_i C_{\mathbf{x}_i}(\mathbf{w})$$

Cost (loss) function

Machine Learning: Part I

Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

- Define a *cost* (loss, objective) function:

$$\begin{aligned} C_{\mathbf{x}}(\mathbf{w}) &= \frac{1}{2}||(\mathbf{E}_{\mathbf{w}}(\mathbf{x}))||^2 \\ &= \frac{1}{2}||\mathbf{y} - \mathbf{h}_{\mathbf{w}}(\mathbf{x})||^2 \\ &= \frac{1}{2}\sum_m (y_m - a_m^L)^2 \end{aligned}$$

- $C_{\mathbf{x}}(\mathbf{w})$: *quadratic cost (MSE) function*
- Entire cost function: average over all $\mathbf{x}_i$:

$$C(\mathbf{w}) = \frac{1}{n}\sum_i C_{\mathbf{x}_i}(\mathbf{w})$$

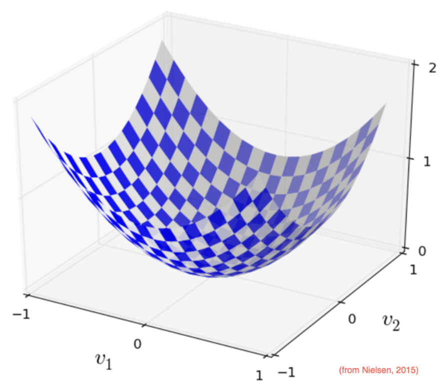
- Always positive, $\rightarrow 0$ as output $\rightarrow \mathbf{y}$

Minimizing cost function

Machine Learning: Part I

Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

- If we minimize C, minimize $||\mathbf{E}||$
- Using calculus, can find analytical solution
- But with n weights, $n+1$-dimensional curve
- E.g., two dimension:



(from Nielsen, 2015)

- Largest nets: *billions* of weights

Gradient descent search

- *Gradient descent search* instead of analytical solution
- Find *local gradients* wrt weights
- $\Rightarrow n$ *partial derivatives* of C
- Take a small step in direction of decrease in *all* the derivatives
- Repeat until close enough to minimum

What is the local gradient?

Machine Learning: Part I

Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

- For simplicity: two variables, v_1, v_2

Copyright © 2019 UMaine School of Computing and Information Science

Artificial Intelligence

What is the local gradient?

Machine Learning: Part I

Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

- For simplicity: two variables, v_1, v_2
- Then:
$$\Delta C \approx \frac{\partial C}{\partial v_1}\Delta v_1 + \frac{\partial C}{\partial v_2}\Delta v_2$$

Copyright © 2019 UMaine School of Computing and Information Science

Artificial Intelligence

What is the local gradient?

Machine Learning: Part I

Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

- For simplicity: two variables, v_1, v_2
- Then:
$$\Delta C \approx \frac{\partial C}{\partial v_1}\Delta v_1 + \frac{\partial C}{\partial v_2}\Delta v_2$$
- Let $\Delta \mathbf{v} = [\Delta v_1 \ \Delta v_2]^T$

Copyright © 2019 UMaine School of Computing and Information Science

Artificial Intelligence

What is the local gradient?

- For simplicity: two variables, v_1, v_2
- Then:

$$\Delta C \approx \frac{\partial C}{\partial v_1}\Delta v_1 + \frac{\partial C}{\partial v_2}\Delta v_2$$

- Let $\Delta \mathbf{v} = [\Delta v_1 \ \Delta v_2]^T$
- Then *gradient* of C is:

$$\nabla C = \left[\frac{\partial C}{\partial v_1} \ \frac{\partial C}{\partial v_2}\right]^T$$

What is the local gradient?

Machine Learning: Part I

Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

- For simplicity: two variables, v_1, v_2
- Then:

$$\Delta C \approx \frac{\partial C}{\partial v_1}\Delta v_1 + \frac{\partial C}{\partial v_2}\Delta v_2$$

- Let $\Delta \mathbf{v} = [\Delta v_1 \ \Delta v_2]^T$
- Then *gradient* of C is:

$$\nabla C = \left[\frac{\partial C}{\partial v_1} \ \frac{\partial C}{\partial v_2}\right]^T$$

- Thus $\Delta C \approx \nabla C \cdot \Delta \mathbf{v}$

Copyright © 2019 UMaine School of Computing and Information Science

Artificial Intelligence

Updating the variables

Machine Learning: Part I

Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

- Must choose Δv s.t. ΔC is negative
- Let $\Delta v = -\eta \nabla C$
  - For $||\Delta v|| \leq \epsilon$, minimizes $\nabla C \cdot \Delta v$
  - Cost function now:

$$\Delta C \approx \nabla C \cdot -\eta \nabla C = -\eta ||\nabla C||^2$$

  - Always negative
- η is learning rate (or α; depends on author)
- New variable vector $\mathbf{v}$: $\mathbf{v}_{t+1} = \mathbf{v}_t - \eta \nabla C$
- Now generalize $\mathbf{v} \rightarrow \mathbf{w}$ (including $\mathbf{b}$)
- BTW: Other gradient descent functions have been tried

Copyright © 2019 UMaine School of Computing and Information Science

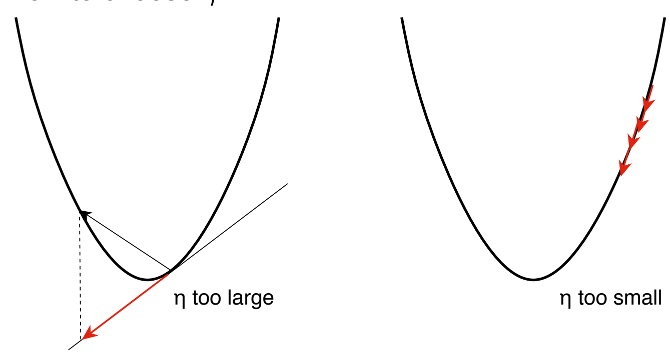
Artificial Intelligence

Choosing learning rate

Machine Learning: Part I

Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

- How to choose η?



- If too large $\Rightarrow$ may overshoot minimum
- If too small $\Rightarrow$ will take a very long time to find minimum

Copyright © 2019 UMaine School of Computing and Information Science

Artificial Intelligence

Computing gradient

- Difficult
- Cost function: Must compute all C_x then average

$$C = \frac{1}{n}\sum_x C_x = \frac{1}{n}\sum_x \frac{||y(x) - a||^2}{2}$$

- To find overall gradient ∇C:

$$\nabla C = \frac{1}{n}\sum_x \nabla C_x$$

- With many training examples, costly $\Rightarrow$ slow learning

Stochastic gradient descent

- Speeds up learning
- Estimate ∇C:
  - Choose small sample of inputs randomly: a *mini-batch*
  - Compute ∇C_x for these to estimate ∇C
- If batch size is large enough, average $\approx \nabla C$
- Idea:
  - Randomly partition training examples into mini-batches
  - Train with each mini-batch
- Doing this: *epoch*
- Repeat until error is satisfactory

Machine Learning: Part I
Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

Stochastic gradient descent

- Speeds up learning
- Estimate ∇C:
  - Choose small sample of inputs randomly: a *mini-batch*
  - Compute ∇C_x for these to estimate ∇C
- If batch size is large enough, average $\approx \nabla C$
- Idea:
  - Randomly partition training examples into mini-batches
  - Train with each mini-batch
- Doing this: *epoch*
- Repeat until error is satisfactory
- Problem: Don't know how to calculate ∇C with hidden layers!

Machine Learning: Part I
Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

Backpropagation

Machine Learning: Part I
Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

Gradient descent

- Computing the gradient ∇C of the cost function:
  - Composed of $\frac{\partial C}{\partial \mathbf{w}}, \frac{\partial C}{\partial \mathbf{b}}$ – where w, b are vectors
  - May be very difficult to compute

Artificial Intelligence

Backpropagation

- *Backpropagation* algorithm (Rumelhart, Hinton, & Williams, 1986)
- Rather than trying to adjust all weights at once, do it by layers
- Compare output layer with target

Artificial Intelligence

Backpropagation

- *Backpropagation* algorithm (Rumelhart, Hinton, & Williams, 1986)
- Rather than trying to adjust all weights at once, do it by layers
- Compare output layer with target
- Compute error, use it to update weights from previous hidden layer to output layer

Artificial Intelligence

Backpropagation

- *Backpropagation* algorithm (Rumelhart, Hinton, & Williams, 1986)
- Rather than trying to adjust all weights at once, do it by layers
- Compare output layer with target
- Compute error, use it to update weights from previous hidden layer to output layer
- Now propagate error in expected outputs of hidden layer backward, etc.

Artificial Intelligence

Backpropagation

- *Backpropagation* algorithm (Rumelhart, Hinton, & Williams, 1986)
- Rather than trying to adjust all weights at once, do it by layers
- Compare output layer with target
- Compute error, use it to update weights from previous hidden layer to output layer
- Now propagate error in expected outputs of hidden layer backward, etc.
- Propagate by dividing responsibility for error at neuron in l according to contribution from each neuron in $l-1$

Error in output layer

- First define vector δ^L, where for element j:

$$\delta_j^L = \frac{\partial C}{\partial a_j^L}\sigma'(z_j^L)$$

where:

Error in output layer

- First define vector δ^L, where for element j:

$$\delta_j^L = \frac{\partial C}{\partial a_j^L}\sigma'(z_j^L)$$

where:
  - $\frac{\partial C}{\partial a_j^L}$: how fast the cost function is changing due to j's output

Error in output layer

- First define vector δ^L, where for element j:

$$\delta_j^L = \frac{\partial C}{\partial a_j^L}\sigma'(z_j^L)$$

where:
  - $\frac{\partial C}{\partial a_j^L}$: how fast the cost function is changing due to j's output
  - $\sigma'(\cdot)$: 1st deriv. of $\sigma(\cdot)$

Error in output layer

- First define vector δ^L, where for element j:

$$\delta_j^L = \frac{\partial C}{\partial a_j^L}\sigma'(z_j^L)$$

where:
  - $\frac{\partial C}{\partial a_j^L}$: how fast the cost function is changing due to j's output
  - $\sigma'(\cdot)$: 1st deriv. of $\sigma(\cdot)$
  - z_j^L: weighted input to j

Error in output layer

Machine Learning: Part I

- First define vector δ^L, where for element j:

$$\delta_j^L = \frac{\partial C}{\partial a_j^L}\sigma'(z_j^L)$$

where:
  - $\frac{\partial C}{\partial a_j^L}$: how fast the cost function is changing due to j's output
  - $\sigma'(\cdot)$: 1st deriv. of $\sigma(\cdot)$
  - z_j^L: weighted input to j
  - Thus $\sigma'(z_j^L)$ is how fast σ is changing at z_j^L

Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

Copyright © 2019 UMaine School of Computing and Information Science

Error in output layer

Machine Learning: Part I

- First define vector δ^L, where for element j:

$$\delta_j^L = \frac{\partial C}{\partial a_j^L}\sigma'(z_j^L)$$

where:
  - $\frac{\partial C}{\partial a_j^L}$: how fast the cost function is changing due to j's output
  - $\sigma'(\cdot)$: 1st deriv. of $\sigma(\cdot)$
  - z_j^L: weighted input to j
  - Thus $\sigma'(z_j^L)$ is how fast σ is changing at z_j^L
- δ^L is a measure of error at L

Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

Copyright © 2019 UMaine School of Computing and Information Science

Error in output layer

Machine Learning: Part I

- First define vector δ^L, where for element j:

$$\delta_j^L = \frac{\partial C}{\partial a_j^L}\sigma'(z_j^L)$$

where:
  - $\frac{\partial C}{\partial a_j^L}$: how fast the cost function is changing due to j's output
  - $\sigma'(\cdot)$: 1st deriv. of $\sigma(\cdot)$
  - z_j^L: weighted input to j
  - Thus $\sigma'(z_j^L)$ is how fast σ is changing at z_j^L
- δ^L is a measure of error at L
- z_j^L already computed, $\sigma'(z_j^L)$ easy to compute

Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

Copyright © 2019 UMaine School of Computing and Information Science

Error in output layer

- First define vector δ^L, where for element j:

$$\delta_j^L = \frac{\partial C}{\partial a_j^L}\sigma'(z_j^L)$$

where:
  - $\frac{\partial C}{\partial a_j^L}$: how fast the cost function is changing due to j's output
  - $\sigma'(\cdot)$: 1st deriv. of $\sigma(\cdot)$
  - z_j^L: weighted input to j
  - Thus $\sigma'(z_j^L)$ is how fast σ is changing at z_j^L
- δ^L is a measure of error at L
- z_j^L already computed, $\sigma'(z_j^L)$ easy to compute
- $\frac{\partial C}{\partial a_j^L}$ for quadratic cost function: $\frac{\partial C}{\partial a_j^L} = (a_j^L - y_j)$

Artificial Intelligence

Error in output layer

- First define vector δ^L, where for element j:

$$\delta_j^L = \frac{\partial C}{\partial a_j^L}\sigma'(z_j^L)$$

where:
  - $\frac{\partial C}{\partial a_j^L}$: how fast the cost function is changing due to j's output
  - $\sigma'(\cdot)$: 1st deriv. of $\sigma(\cdot)$
  - z_j^L: weighted input to j
  - Thus $\sigma'(z_j^L)$ is how fast σ is changing at z_j^L
- δ^L is a measure of error at L
- z_j^L already computed, $\sigma'(z_j^L)$ easy to compute
- $\frac{\partial C}{\partial a_j^L}$ for quadratic cost function: $\frac{\partial C}{\partial a_j^L} = (a_j^L - y_j)$
- So for quadratic: $\delta_j^L = (a_j^L - y_j)\sigma'(z_j^L)$

Artificial Intelligence

Hadamard product

- Need a new operator to simplify expressions
- Define *Hadamard product* as: $\mathbf{s} \odot \mathbf{t} = \mathbf{h}$ s.t. $h_j = s_j \times t_j$
- I.e., elementwise product – e.g.:

$$\begin{bmatrix}-2\\20\\3\end{bmatrix} \odot \begin{bmatrix}3\\2\\1\end{bmatrix} = \begin{bmatrix}-6\\40\\3\end{bmatrix}$$

Artificial Intelligence

Error in output layer

- $\delta_j^L = \frac{\partial C}{\partial a_j^L}\sigma'(z_j^L)$
- Can be rewritten as:

$$\delta^L = \nabla_a C \odot \sigma'(\mathbf{z}^L)$$

- Or

$$\delta^L = (\mathbf{a}^L - \mathbf{y}) \odot \sigma'(\mathbf{z}^L)$$

Artificial Intelligence

Finding previous layer's error

- If we know δ^{l+1}, can we find δ^{l}?
- $(\mathbf{w}^{l+1})^T$ = transpose of weight matrix into $l+1$
- $(\mathbf{w}^{l+1})^T \delta^{l+1}$:
  - Moves error backward
  - Gives measure of error at layer l
- Then

$$\delta^l = ((\mathbf{w}^{l+1})^T \delta^{l+1}) \odot \sigma'(\mathbf{z}^l)$$

- Can now compute the error at any layer

Rate of change for biases, weights

Machine Learning: Part I

- For any weight in the network:

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j$$

- For any bias in the network:

$$\frac{\partial C}{\partial b^l_j} = \delta^L_j$$

since "activation" for any bias is just 1

Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

Copyright © 2019 UMaine School of Computing and Information Science

Artificial Intelligence

Backpropagation & gradient descent

Machine Learning: Part I

- For each x $\in m$ training examples:

Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

Copyright © 2019 UMaine School of Computing and Information Science

Artificial Intelligence

Backpropagation & gradient descent

Machine Learning: Part I

- For each x $\in m$ training examples:
  - Feedforward: for each layer l, compute:

Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

Copyright © 2019 UMaine School of Computing and Information Science

Artificial Intelligence

Backpropagation & gradient descent

- For each x $\in$ *m* training examples:
  - Feedforward: for each layer *l*, compute:
    - $\mathbf{z}^{\mathbf{x},\mathbf{l}} = \mathbf{w}^l \mathbf{a}^{x,l-1} + \mathbf{b}^l$

Artificial Intelligence

Backpropagation & gradient descent

- For each x $\in$ *m* training examples:
  - Feedforward: for each layer *l*, compute:
    - $\mathbf{z}^{\mathbf{x},\mathbf{l}} = \mathbf{w}^l \mathbf{a}^{x,l-1} + \mathbf{b}^l$
    - $\mathbf{a}^{x,l} = \sigma(\mathbf{z}^{x,l})$

Artificial Intelligence

Backpropagation & gradient descent

- For each x $\in$ *m* training examples:
  - Feedforward: for each layer *l*, compute:
    - $\mathbf{z}^{\mathbf{x},\mathbf{l}} = \mathbf{w}^l \mathbf{a}^{x,l-1} + \mathbf{b}^l$
    - $\mathbf{a}^{x,l} = \sigma(\mathbf{z}^{x,l})$
  - Compute the output error:

Artificial Intelligence

Backpropagation & gradient descent

- For each x $\in$ *m* training examples:
  - Feedforward: for each layer *l*, compute:
    - $\mathbf{z}^{\mathbf{x},\mathbf{l}} = \mathbf{w}^l \mathbf{a}^{x,l-1} + \mathbf{b}^l$
    - $\mathbf{a}^{x,l} = \sigma(\mathbf{z}^{x,l})$
  - Compute the output error:
    - $\delta^{x,L} = \nabla_a C_x \odot \sigma'(\mathbf{z}^{x,L})$

Artificial Intelligence

Backpropagation & gradient descent

- For each x $\in m$ training examples:
  - Feedforward: for each layer l, compute:
    - $\mathbf{z}^{\mathbf{x},\mathbf{l}} = \mathbf{w}^l \mathbf{a}^{x,l-1} + \mathbf{b}^l$
    - $\mathbf{a}^{x,l} = \sigma(\mathbf{z}^{x,l})$
  - Compute the output error:
    - $\delta^{x,L} = \nabla_a C_x \odot \sigma'(\mathbf{z}^{x,L})$
  - Backpropagate error for each layer l:

Artificial Intelligence

Backpropagation & gradient descent

Machine Learning: Part I

- For each x $\in m$ training examples:
  - Feedforward: for each layer l, compute:
    - $\mathbf{z}^{\mathbf{x},\mathbf{l}} = \mathbf{w}^l \mathbf{a}^{x,l-1} + \mathbf{b}^l$
    - $\mathbf{a}^{x,l} = \sigma(\mathbf{z}^{x,l})$
  - Compute the output error:
    - $\delta^{x,L} = \nabla_a C_x \odot \sigma'(\mathbf{z}^{x,L})$
  - Backpropagate error for each layer l:
    - $\delta^{x,l} = ((\mathbf{w}^{l+1})^T \delta^{x,l+1}) \odot \sigma'(\mathbf{z}^{x,l})$

Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

Copyright © 2019 UMaine School of Computing and Information Science

Artificial Intelligence

Backpropagation & gradient descent

Machine Learning: Part I

- For each x $\in m$ training examples:
  - Feedforward: for each layer l, compute:
    - $\mathbf{z}^{\mathbf{x},\mathbf{l}} = \mathbf{w}^l \mathbf{a}^{x,l-1} + \mathbf{b}^l$
    - $\mathbf{a}^{x,l} = \sigma(\mathbf{z}^{x,l})$
  - Compute the output error:
    - $\delta^{x,L} = \nabla_a C_x \odot \sigma'(\mathbf{z}^{x,L})$
  - Backpropagate error for each layer l:
    - $\delta^{x,l} = ((\mathbf{w}^{l+1})^T \delta^{x,l+1}) \odot \sigma'(\mathbf{z}^{x,l})$
- Gradient descent: For each layer from L $\to$ 2:

Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

Copyright © 2019 UMaine School of Computing and Information Science

Artificial Intelligence

Backpropagation & gradient descent

Machine Learning: Part I

- For each x $\in m$ training examples:
  - Feedforward: for each layer l, compute:
    - $\mathbf{z}^{\mathbf{x},\mathbf{l}} = \mathbf{w}^l \mathbf{a}^{x,l-1} + \mathbf{b}^l$
    - $\mathbf{a}^{x,l} = \sigma(\mathbf{z}^{x,l})$
  - Compute the output error:
    - $\delta^{x,L} = \nabla_a C_x \odot \sigma'(\mathbf{z}^{x,L})$
  - Backpropagate error for each layer l:
    - $\delta^{x,l} = ((\mathbf{w}^{l+1})^T \delta^{x,l+1}) \odot \sigma'(\mathbf{z}^{x,l})$
- Gradient descent: For each layer from L $\to$ 2:
  - Next $\mathbf{w}^l = \mathbf{w}^l - \frac{\eta}{m} \sum_x \delta^{x,l} (\mathbf{a}^{x,l-1})^T$

Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

Copyright © 2019 UMaine School of Computing and Information Science

Artificial Intelligence

Backpropagation & gradient descent

- For each $x \in m$ training examples:
  - Feedforward: for each layer l, compute:
    - $\mathbf{z}^{x,l} = \mathbf{w}^l \mathbf{a}^{x,l-1} + \mathbf{b}^l$
    - $\mathbf{a}^{x,l} = \sigma(\mathbf{z}^{x,l})$
  - Compute the output error:
    - $\delta^{x,L} = \nabla_a C_x \odot \sigma'(\mathbf{z}^{x,L})$
  - Backpropagate error for each layer l:
    - $\delta^{x,l} = ((\mathbf{w}^{l+1})^T \delta^{x,l+1}) \odot \sigma'(\mathbf{z}^{x,l})$
- Gradient descent: For each layer from L $\rightarrow$ 2:
  - Next $\mathbf{w}^l = \mathbf{w}^l - \frac{\eta}{m} \sum_x \delta^{x,l} (\mathbf{a}^{x,l-1})^T$
  - Next $\mathbf{b}^l = \mathbf{b}^l - \frac{\eta}{m} \sum_x \delta^{x,l}$

Backpropagation & gradient descent

Machine Learning: Part I

Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

- For each $x \in m$ training examples:
  - Feedforward: for each layer l, compute:
    - $\mathbf{z}^{x,l} = \mathbf{w}^l \mathbf{a}^{x,l-1} + \mathbf{b}^l$
    - $\mathbf{a}^{x,l} = \sigma(\mathbf{z}^{x,l})$
  - Compute the output error:
    - $\delta^{x,L} = \nabla_a C_x \odot \sigma'(\mathbf{z}^{x,L})$
  - Backpropagate error for each layer l:
    - $\delta^{x,l} = ((\mathbf{w}^{l+1})^T \delta^{x,l+1}) \odot \sigma'(\mathbf{z}^{x,l})$
- Gradient descent: For each layer from L $\rightarrow$ 2:
  - Next $\mathbf{w}^l = \mathbf{w}^l - \frac{\eta}{m} \sum_x \delta^{x,l} (\mathbf{a}^{x,l-1})^T$
  - Next $\mathbf{b}^l = \mathbf{b}^l - \frac{\eta}{m} \sum_x \delta^{x,l}$

Copyright © 2019 UMaine School of Computing and Information Science

Artificial Intelligence

Backpropagation & gradient descent

- For each $x \in m$ training examples:
  - Feedforward: for each layer l, compute:
    - $\mathbf{z}^{x,l} = \mathbf{w}^l \mathbf{a}^{x,l-1} + \mathbf{b}^l$
    - $\mathbf{a}^{x,l} = \sigma(\mathbf{z}^{x,l})$
  - Compute the output error:
    - $\delta^{x,L} = \nabla_a C_x \odot \sigma'(\mathbf{z}^{x,L})$
  - Backpropagate error for each layer l:
    - $\delta^{x,l} = ((\mathbf{w}^{l+1})^T \delta^{x,l+1}) \odot \sigma'(\mathbf{z}^{x,l})$
- Gradient descent: For each layer from L $\rightarrow$ 2:
  - Next $\mathbf{w}^l = \mathbf{w}^l - \frac{\eta}{m} \sum_x \delta^{x,l} (\mathbf{a}^{x,l-1})^T$
  - Next $\mathbf{b}^l = \mathbf{b}^l - \frac{\eta}{m} \sum_x \delta^{x,l}$

Do for some # of epochs, some # mini-batches each.

Backprop algorithm


```
function BACK-PROP-LEARNING(examples, network) returns a neural network
  inputs: examples, a set of examples, each with input vector x and output vector y
          network, a multilayer network with L layers, weights w_{i,j}, activation function g
  local variables: Δ, a vector of errors, indexed by network node

  repeat
      for each weight w_{i,j} in network do
          w_{i,j} ← a small random number
      for each example (x, y) in examples do
          /* Propagate the inputs forward to compute the outputs */
          for each node i in the input layer do
              a_i ← x_i
          for ℓ = 2 to L do
              for each node j in layer ℓ do
                  in_j ← Σ_i w_{i,j} a_i
                  a_j ← g(in_j)
          /* Propagate deltas backward from output layer to input layer */
          for each node j in the output layer do
              Δ[j] ← g'(in_j) × (y_j − a_j)
          for ℓ = L − 1 to 1 do
              for each node i in layer ℓ do
                  Δ[i] ← g'(in_i) Σ_j w_{i,j} Δ[j]
          /* Update every weight in network using deltas */
          for each weight w_{i,j} in network do
              w_{i,j} ← w_{i,j} + α × a_i × Δ[j]
  until some stopping criterion is satisfied
  return network
```

Speed of backprop

- What if we had instead done what we did in HC?
  - For each timestep, look at small changes in the weights
  - Pick set that decreases error
- Could do this for each weight separately, too
- If we have millions of weights, requires *millions* of passes through network
- With backprop: one forward, one backward pass, no matter how many weights

Machine Learning: Part I

Deep learning

Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

Deep learning

Machine Learning: Part I

- Networks with ≥ 2 hidden layers are *deep networks*

Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

Copyright © 2019 UMaine School of Computing and Information Science

Artificial Intelligence

Deep learning

Machine Learning: Part I

- Networks with ≥ 2 hidden layers are *deep networks*
- Backprop will still work for them

Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

Copyright © 2019 UMaine School of Computing and Information Science

Artificial Intelligence

Deep learning

- Networks with ≥ 2 hidden layers are *deep networks*
- Backprop will still work for them
- But:

Deep learning

Machine Learning: Part I

- Networks with ≥ 2 hidden layers are *deep networks*
- Backprop will still work for them
- But:
  - Tend to lose the error “signal” as propagate back through network

Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

Copyright © 2019 UMaine School of Computing and Information Science

Artificial Intelligence

Deep learning

Machine Learning: Part I

- Networks with ≥ 2 hidden layers are *deep networks*
- Backprop will still work for them
- But:
  - Tend to lose the error “signal” as propagate back through network
  - Each neuron in earlier layers have less and less impact on output error

Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

Copyright © 2019 UMaine School of Computing and Information Science

Artificial Intelligence

Deep learning

Machine Learning: Part I

- Networks with ≥ 2 hidden layers are *deep networks*
- Backprop will still work for them
- But:
  - Tend to lose the error “signal” as propagate back through network
  - Each neuron in earlier layers have less and less impact on output error
  - *Vanishing gradient problem*

Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

Copyright © 2019 UMaine School of Computing and Information Science

Artificial Intelligence

Deep learning

- Networks with ≥ 2 hidden layers are *deep networks*
- Backprop will still work for them
- But:
  - Tend to lose the error "signal" as propagate back through network
  - Each neuron in earlier layers have less and less impact on output error
  - *Vanishing gradient problem*
- $\Rightarrow$ *extremely* slow learning rate

Deep learning

Machine Learning: Part I
Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

- Networks with ≥ 2 hidden layers are *deep networks*
- Backprop will still work for them
- But:
  - Tend to lose the error "signal" as propagate back through network
  - Each neuron in earlier layers have less and less impact on output error
  - *Vanishing gradient problem*
- $\Rightarrow$ *extremely* slow learning rate
- Can have opposite problem, depending on net: *exploding gradient problem*

Deep learning

Machine Learning: Part I
Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

- Networks with ≥ 2 hidden layers are *deep networks*
- Backprop will still work for them
- But:
  - Tend to lose the error "signal" as propagate back through network
  - Each neuron in earlier layers have less and less impact on output error
  - *Vanishing gradient problem*
- $\Rightarrow$ *extremely* slow learning rate
- Can have opposite problem, depending on net: *exploding gradient problem*
- Stymied researchers for many years – until

Deep learning

Machine Learning: Part I
Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

- Networks with ≥ 2 hidden layers are *deep networks*
- Backprop will still work for them
- But:
  - Tend to lose the error "signal" as propagate back through network
  - Each neuron in earlier layers have less and less impact on output error
  - *Vanishing gradient problem*
- $\Rightarrow$ *extremely* slow learning rate
- Can have opposite problem, depending on net: *exploding gradient problem*
- Stymied researchers for many years – until
  - Faster machines

Deep learning

- Networks with ≥ 2 hidden layers are *deep networks*
- Backprop will still work for them
- But:
  - Tend to lose the error "signal" as propagate back through network
  - Each neuron in earlier layers have less and less impact on output error
  - *Vanishing gradient problem*
- $\Rightarrow$ *extremely* slow learning rate
- Can have opposite problem, depending on net: *exploding gradient problem*
- Stymied researchers for many years – until
  - Faster machines
  - Better versions of backprop-ish algorithms invented

Deep learning

Machine Learning: Part I

- Networks with ≥ 2 hidden layers are *deep networks*
- Backprop will still work for them
- But:
  - Tend to lose the error "signal" as propagate back through network
  - Each neuron in earlier layers have less and less impact on output error
  - *Vanishing gradient problem*
- $\Rightarrow$ *extremely* slow learning rate
- Can have opposite problem, depending on net: *exploding gradient problem*
- Stymied researchers for many years – until
  - Faster machines
  - Better versions of backprop-ish algorithms invented
- $\Rightarrow$ tremendous increase in deep learning research, applications

Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

Deep learning

Machine Learning: Part I

- Networks with ≥ 2 hidden layers are *deep networks*
- Backprop will still work for them
- But:
  - Tend to lose the error "signal" as propagate back through network
  - Each neuron in earlier layers have less and less impact on output error
  - *Vanishing gradient problem*
- $\Rightarrow$ *extremely* slow learning rate
- Can have opposite problem, depending on net: *exploding gradient problem*
- Stymied researchers for many years – until
  - Faster machines
  - Better versions of backprop-ish algorithms invented
- $\Rightarrow$ tremendous increase in deep learning research, applications
- We'll come back to this later in course

Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary

Summary

Machine Learning: Part I

Neural network training:

"You process a lot of data in a quiet way, don't you, Larry!"

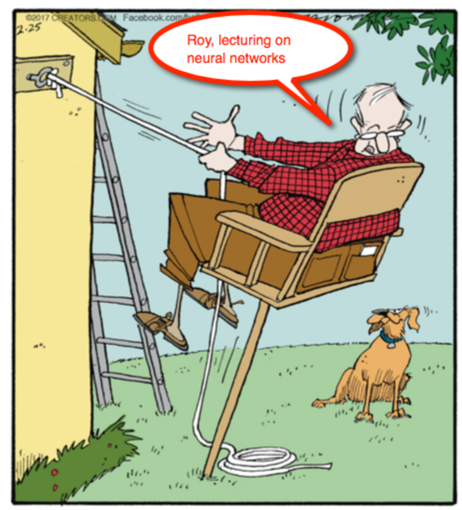
Machine Learning: Part I

Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary
Neural network training:
. . . and. . .
What can they do?
How do they work?
Autoencoders
Restricted Boltzmann machines
Feed-forward NN
Deep learning nets
Building them: Keras, TensorFlow, PyTorch, etc.

. . . and. . .

Machine Learning: Part I

Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary
Neural network training:
. . . and. . .
What can they do?
How do they work?
Autoencoders
Restricted Boltzmann machines
Feed-forward NN
Deep learning nets
Building them: Keras, TensorFlow, PyTorch, etc.



You'd like to ask Roy if he's really thought this through.

Machine Learning: Part I

Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary
Neural network training:
. . . and. . .
What can they do?
How do they work?
Autoencoders
Restricted Boltzmann machines
Feed-forward NN
Deep learning nets
Building them: Keras, TensorFlow, PyTorch, etc.

What can they do?

What can they do?

Machine Learning: Part I

Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary
Neural network training:
...and...
What can they do?
How do they work?
Autoencoders
Restricted Boltzmann machines
Feed-forward NN
Deep learning nets
Building them: Keras, TensorFlow, PyTorch, etc.

How do they work?

Machine Learning: Part I

Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary
Neural network training:
...and...
What can they do?
How do they work?
Autoencoders
Restricted Boltzmann machines
Feed-forward NN
Deep learning nets
Building them: Keras, TensorFlow, PyTorch, etc.

How do they work?

Machine Learning: Part I

Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary
Neural network training:
...and...
What can they do?
How do they work?
Autoencoders
Restricted Boltzmann machines
Feed-forward NN
Deep learning nets
Building them: Keras, TensorFlow, PyTorch, etc.

Autoencoders

Autoencoders

Machine Learning: Part I

Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary
Neural network training:
. . . and. . .
What can they do?
How do they work?
Autoencoders
Restricted Boltzmann machines
Feed-forward NN
Deep learning nets
Building them: Keras, TensorFlow, PyTorch, etc.

Artificial Intelligence

Restricted Boltzmann machines

Machine Learning: Part I

Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary
Neural network training:
. . . and. . .
What can they do?
How do they work?
Autoencoders
Restricted Boltzmann machines
Feed-forward NN
Deep learning nets
Building them: Keras, TensorFlow, PyTorch, etc.

Restricted Boltzmann machines

Machine Learning: Part I

Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary
Neural network training:
. . . and. . .
What can they do?
How do they work?
Autoencoders
Restricted Boltzmann machines
Feed-forward NN
Deep learning nets
Building them: Keras, TensorFlow, PyTorch, etc.

Artificial Intelligence

Feed-forward NN

Feed-forward NN

Machine Learning: Part I

Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary
Neural network training:
...and...
What can they do?
How do they work?
Autoencoders
Restricted Boltzmann machines
Feed-forward NN
Deep learning nets
Building them: Keras, TensorFlow, PyTorch, etc.

Artificial Intelligence

Deep learning nets

Machine Learning: Part I

Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary
Neural network training:
...and...
What can they do?
How do they work?
Autoencoders
Restricted Boltzmann machines
Feed-forward NN
Deep learning nets
Building them: Keras, TensorFlow, PyTorch, etc.

Deep learning nets

Machine Learning: Part I

Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary
Neural network training:
...and...
What can they do?
How do they work?
Autoencoders
Restricted Boltzmann machines
Feed-forward NN
Deep learning nets
Building them: Keras, TensorFlow, PyTorch, etc.

Artificial Intelligence

Building them: Keras, TensorFlow, PyTorch, etc.

Building them: Keras, TensorFlow, PyTorch, etc.

Machine Learning: Part I

Introduction
Perceptrons
Nonlinear neurons
Feedforward neural networks
Matrix form of NN
Gradient descent learning in FFNs
Backpropagation
Deep learning
Summary
Neural network training:
. . . and. . .
What can they do?
How do they work?
Autoencoders
Restricted Boltzmann machines
Feed-forward NN
Deep learning nets
Building them: Keras, TensorFlow, PyTorch, etc.