Deep Feedforward Networks: Overview

Sargur N. Srihari
srihari@cedar.buffalo.edu

Topics

- Overview

1. Example: Learning XOR
2. Gradient-Based Learning
3. Hidden Units
4. Architecture Design
5. Backpropagation and Other Differentiation
6. Historical Notes

Goal of a feedforward network

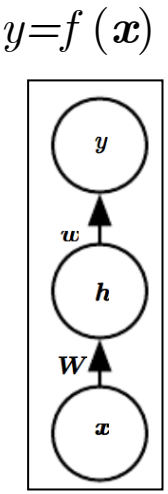
- Feedforward Nets are quintessential deep learning models
- Deep Feedforward Networks are also called as
  - Feedforward neural networks or
  - Multilayer Perceptrons (MLPs)
- Their Goal is to approximate some function f^*
  - E.g., classifier $y = f^*(\boldsymbol{x})$ maps input $\boldsymbol{x}$ to category y
  - Feedforward Network defines a mapping

$$\boldsymbol{y} = f^*(\boldsymbol{x}\ ;\ \boldsymbol{\theta}\ )$$

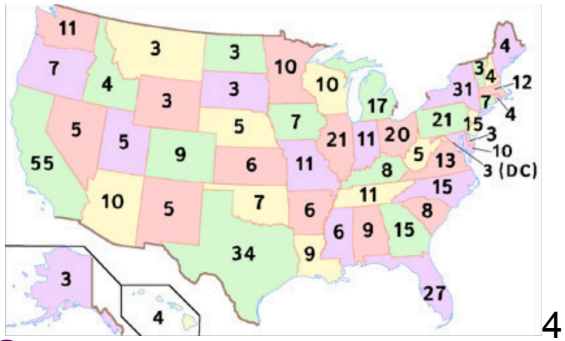
    - and learns the values of the parameters $\boldsymbol{\theta}$ that result in the best function approximation

Flow of Information

- Models are called *Feedforward* because:
  - To evaluate $f(\boldsymbol{x})$: information flows one-way from $\boldsymbol{x}$ through computations defining f to output y
- There are no feedback connections
  - No outputs of model are fed back into itself
- US Presidential Election
  - Input: $\boldsymbol{x}=\{x_1,..x_{50}\}$
    - are votes cast for a candidate
  - $\boldsymbol{h}$ is electoral college
    - Each state has fixed no of electors
  - Output: y
    - votes of electoral college for candidate



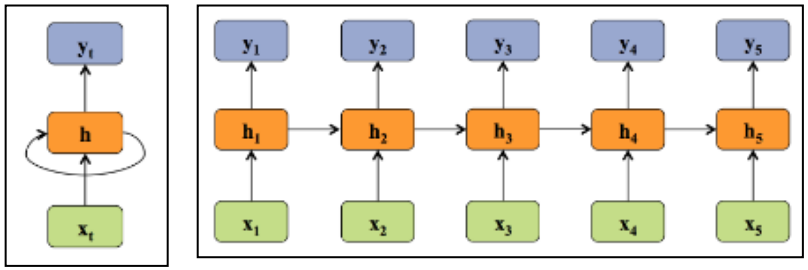
$\boldsymbol{h}$ is defined for each state as shown in map



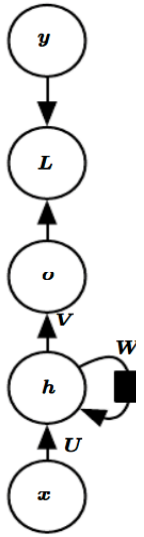
Feedforward vs. Recurrent

- When feedforward neural networks are extended to include feedback connections they are called *Recurrent Neural Networks* (*RNNs*)

RNN Unrolled RNN



RNN with learning component



Importance of Feedforward Networks

- They are extremely important to ML practice
- Form basis for many commercial applications
  1. CNNs are a special kind of feedforward networks
     - They are used for recognizing objects from photos
  2. They are a conceptual stepping stones to RNNs
     - RNNs power many NLP applications

Feedforward Neural Network Structures

- They are called networks because they are composed of many different functions
- Model is associated with a directed acyclic graph describing how functions composed
  - E.g., functions $f^{(1)}, f^{(2)}, f^{(3)}$ connected in a chain to form $f(\boldsymbol{x}) = f^{(3)}[\, f^{(2)}[\, f^{(1)}(\boldsymbol{x})]]$
    - $f^{(1)}$ is called the first layer of network (which is a vector)
    - $f^{(2)}$ is called the second layer, etc
- These chain structures are the most commonly used structures of neural networks

Definition of Depth

- Overall length of the chain is the *depth* of the model
  - Ex: the composite function $f(\boldsymbol{x}) = f^{(3)}[\, f^{(2)}[\, f^{(1)}(\boldsymbol{x})]]$ has depth of 3
- The name *deep learning* arises from this terminology
- Final layer of a feedforward network, ex $f^{(3)}$, is called the *output layer*

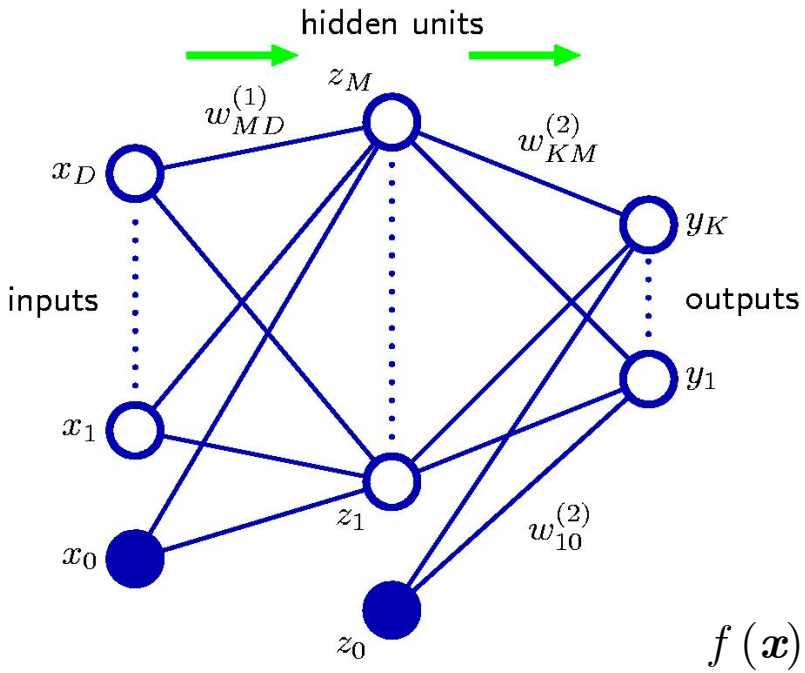
Training the Network

- In network training we drive $f(\boldsymbol{x})$ to match $f^*(\boldsymbol{x})$
- Training data provides us with noisy, approximate examples of $f^*(\boldsymbol{x})$ evaluated at different training points
- Each example accompanied by label $y \approx f^*(\boldsymbol{x})$
- Training examples specify directly what the output layer must do at each point $\boldsymbol{x}$
  - It must produce a value that is close to y

Definition of Hidden Layer

- Behavior of other layers is not directly specified by the data
- Learning algorithm must decide how to use those layers to produce value that is close to y
- Training data does not say what individual layers should do
- Since the desired output for these layers is not shown, they are called *hidden layers*

A net with depth 2: one hidden layer



K outputs $y_1,..y_K$ for a given input x
Hidden layer consists of M units

$$y_k(\mathrm{x},\mathrm{w}) = \sigma\left(\sum_{j=1}^{M} w^{(2)}_{kj} h\left(\sum_{i=1}^{D} w^{(1)}_{ji} x_i + w^{(1)}_{j0}\right) + w^{(2)}_{k0}\right)$$

$$f(\boldsymbol{x}) = f^{(2)}[\, f^{(1)}(\boldsymbol{x})]$$

$f^{(1)}$ is a vector of M dimensions and
$f^{(2)}$ is a vector of K dimensions

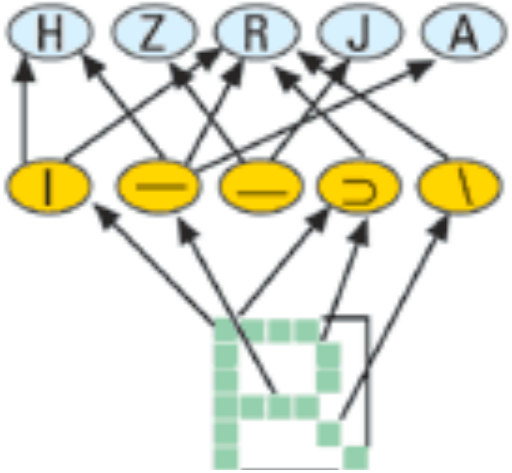
$$f_m{}^{(1)} = z_m = h(\boldsymbol{x}^T \boldsymbol{w}^{(1)}),\ m=1,..M$$
$$f_k{}^{(2)} = \sigma\,(\boldsymbol{z}^T \boldsymbol{w}^{(2)}),\ k=1,..K$$

Feedforward net with depth 2

- Recognition of printed characters (OCR)

$$f(\boldsymbol{x}) = f^{(2)}[f^{(1)}(\boldsymbol{x})]$$

  - Hidden layer $f^{(1)}$ compares raw pixel inputs to component patterns



Width of Model

- Each hidden layer is typically vector-valued
- Dimensionality of hidden layer vector is *width* of the model

Units of a model

- Each element of vector viewed as a neuron
  - Instead of thinking of it as a vector-vector function, they are regarded as units in parallel
- Each unit receives inputs from many other units and computes its own activation value

Depth versus Width

- Going deeper makes network more expressive
  - It can capture variations of the data better.
  - Yields expressiveness more efficiently than width
- Tradeoff for more expressiveness is increased tendency to overfit
  - You will need more data or additional regularization
    - network should be as deep as training data allows.
  - But you can only determine a suitable depth by experiment.
    - Also computation increases with no. of layers.

Very Deep CNNs

CNNs with depth 11 to 19

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Depth increases from left (A) to right (E)
as more layers are added
(the added layers are shown in bold)

Convolutional layer parameters denoted
"conv (receptive field size) –(no. of channels)"

ReLU activation not shown for brevity

Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

Why are they neural networks?

- These networks are loosely inspired by neuroscience
- Each unit resembles a neuron
  - Receives input from many other units
  - Computes its own activation value
- Choice of functions $f^{(i)}(\boldsymbol{x})$:
  - Loosely guided by neuroscientific observations about biological neurons
    - Modern neural networks are guided by many mathematical and engineering disciplines
    - Not perfectly model the brain

Function Approximation is goal

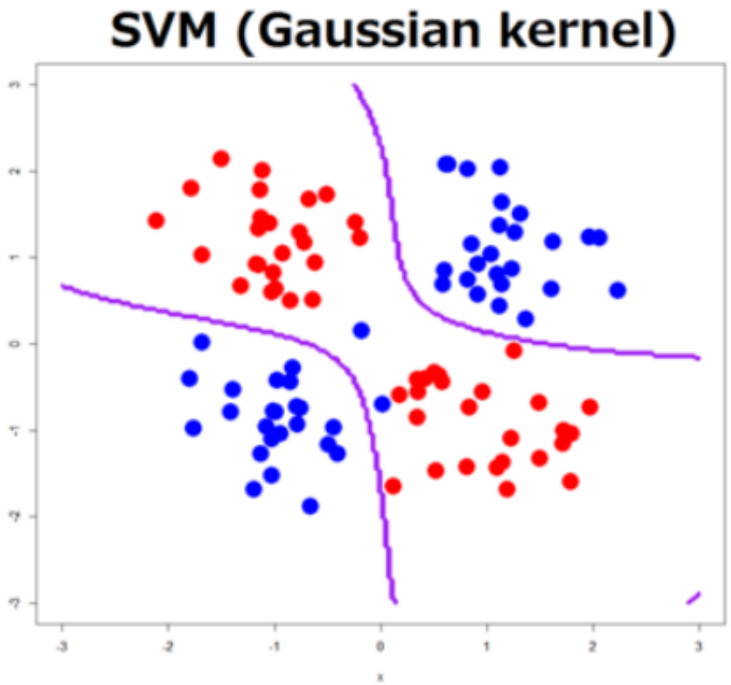
- Think of feedforward networks as function approximation machines
  - Designed to achieve statistical generalization
- Occasionally draw insights from what we know about the brain
  - Rather than as models of brain function

Understanding Feedforward Nets

- Begin with linear networks and understand their limitations
- Linear models such as logistic regression and linear regression can be fit reliably and efficiently using either
  - Closed-form solution
  - Convex optimization
- Limitation

Extending Linear Models

- To represent non-linear functions of $\boldsymbol{x}$
  - apply linear model to transformed input $\phi(\boldsymbol{x})$
    - where ϕ is non-linear
  - Equivalently kernel trick of SVM obtains nonlinearity



SVM Kernel trick

- Many ML algos can be rewritten as dot products between examples:

  $f(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + b$ written as $b + \sum_i \alpha_i \boldsymbol{x}^T\boldsymbol{x}^{(i)}$

  where $\boldsymbol{x}^{(i)}$ is a training example and $\boldsymbol{\alpha}$ is a vector of coeffts

  - This allows us to replace $\boldsymbol{x}$ with a feature function $\phi(\boldsymbol{x})$ and dot product with function $k(\boldsymbol{x},\boldsymbol{x}^{(i)}) = \phi(\boldsymbol{x}) \bullet \phi(\boldsymbol{x}^{(i)})$ called a kernel
    - The • operator represents an inner product analogous to $\phi(\boldsymbol{x})^T\phi(\boldsymbol{x}^{(i)})$
    - For some feature spaces we may not literally use an inner product
      - In continuous spaces an inner product based on integration
  - Gaussian kernel
    - Consider $k(\boldsymbol{u},\boldsymbol{v}) = \exp(-||\boldsymbol{u}-\boldsymbol{v}||^2/2\sigma^2)$
      - By expanding the square $||\boldsymbol{u}-\boldsymbol{v}||^2 = \boldsymbol{u}^T\boldsymbol{u} + \boldsymbol{v}^T\boldsymbol{v} - 2\boldsymbol{u}^T\boldsymbol{v}$
      - we get $k(\boldsymbol{u},\boldsymbol{v}) = \exp(-\boldsymbol{u}^T\boldsymbol{u}/2\sigma^2)\exp(-\boldsymbol{u}^T\boldsymbol{v}/\sigma^2)\exp(-\boldsymbol{v}^T\boldsymbol{v}/2\sigma^2)$
    - Validity follows from kernel construction rules

SVM Prediction

- Use linear regression on Lagrangian for determining the weights α_i
- We can make predictions using
  - $f(\boldsymbol{x}) = b + \Sigma_i \alpha_i k(\boldsymbol{x}, \boldsymbol{x}^{(i)})$
  - Function is nonlinear wrt $\boldsymbol{x}$ but relationship between $\phi(\boldsymbol{x})$ and $f(\boldsymbol{x})$ is linear
  - Also the relationship between α and $f(\boldsymbol{x})$ is linear
  - We can think of ϕ as providing a set of features
    - describing $\boldsymbol{x}$ or providing a new representation for $\boldsymbol{x}$

Disadvantages of Kernel Methods

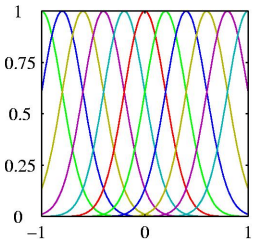
- Cost of decision function evaluation: linear in m
  - Because the i^{th} example contributes a term $\alpha_i k(\boldsymbol{x}, \boldsymbol{x}^{(i)})$ to the decision function
  - Can mitigate this by learning an $\boldsymbol{\alpha}$ with mostly zeros
    - Classification requires evaluating the kernel function only for training examples that have a nonzero α_i
    - These are known as *support vectors*
- Cost of training: high with large data sets
- Generic kernels struggle to generalize well
  - Neural net outperformed RBF-SVM on MNIST
- Also, how to choose the mapping ϕ?

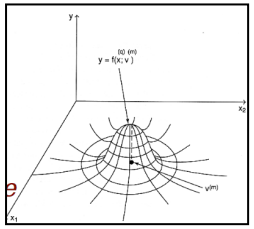
Options for choosing mapping ϕ

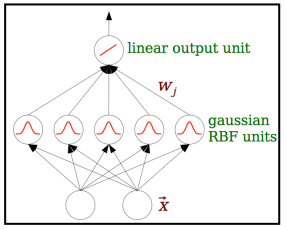
1. Generic feature function $\phi(\boldsymbol{x})$
   - Radial basis function
2. Manually engineer ϕ
   - Feature engineering
3. Principle of Deep Learning: Learn ϕ

Option 1 to choose the mapping ϕ

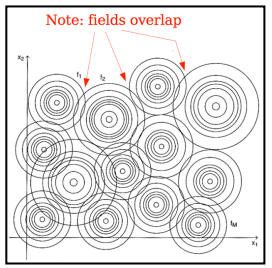
- Generic feature function $\phi(\boldsymbol{x})$
  - Infinite-dimensional ϕ that is implicitly used by kernel machines based on RBF
    - RBF: $N(\boldsymbol{x}\ ;\ \boldsymbol{x}^{(i)}, \sigma^2 \mathrm{I})$ centered at $\boldsymbol{x}^{(i)}$







$x^{(i)}$: From k-means clustering



σ =mean distance between each unit j and its closest neighbor

  - If $\phi(\boldsymbol{x})$ is of high enough dimension we can have enough capacity to fit the training set
    - Generalization to test set remains poor
    - Generic feature mappings are based on smoothness
      - Do not include prior information to solve advanced problems

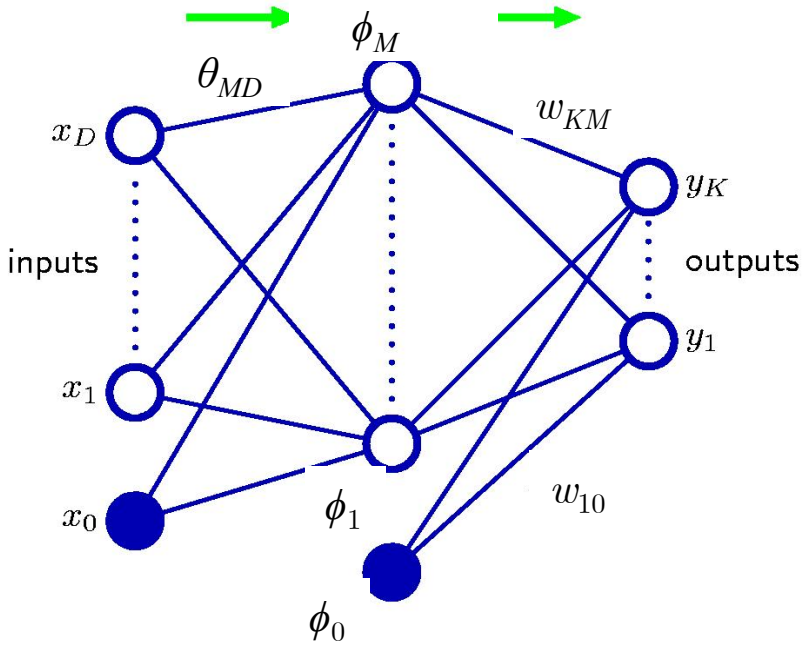
Option 2 to choose the mapping ϕ

- Manually engineer ϕ
- This was the dominant approach until arrival of deep learning
- Requires decades of effort
  - e.g., speech recognition, computer vision
- Little transfer between domains

Option 3 to choose the mapping ϕ

- Strategy of Deep learning: Learn ϕ
- Model is $y = f(\boldsymbol{x}; \boldsymbol{\theta}, \boldsymbol{w}) = \phi(\boldsymbol{x}; \boldsymbol{\theta})^{\mathrm{T}} \boldsymbol{w}$
  - $\boldsymbol{\theta}$ used to learn ϕ from broad class of functions
  - Parameters $\boldsymbol{w}$ map from $\phi(\boldsymbol{x})$ to output
  - Defines FFN where ϕ define a hidden layer
- Unlike other two (basis functions, manual engineering), this approach gives-up on convexity of training
  - But its benefits outweigh harms

Extend Linear Methods to Learn ϕ



K outputs $y_1,..y_K$ for a given input $\boldsymbol{x}$
Hidden layer consists of M units

$$y_k(\boldsymbol{x};\boldsymbol{\theta},\mathbf{w}) = \sum_{j=1}^{M} w_{kj}\phi_j\left(\sum_{i=1}^{D}\theta_{ji}x_i + \theta_{j0}\right) + w_{k0}$$

$$y_k = f_k(\boldsymbol{x};\boldsymbol{\theta},\mathrm{w}) = \boldsymbol{\phi}(\boldsymbol{x};\boldsymbol{\theta})^T \mathrm{w}$$

Can be viewed as a generalization of linear models

- Nonlinear function f_k with $M+1$ parameters $\mathbf{w}_k = (w_{k0},..w_{kM})$ with
- M basis functions, ϕ_j $j=1,..M$ each with D parameters $\boldsymbol{\theta}_j = (\theta_{j1},..\theta_{jD})$
- Both w_k and $\boldsymbol{\theta}_j$ are learnt from data

Approaches to Learning ϕ

- Parameterize the basis functions as $\phi(\boldsymbol{x};\boldsymbol{\theta})$
  - Use optimization to find $\boldsymbol{\theta}$ that corresponds to a good representation
- Approach can capture benefit of first approach (fixed basis functions) by being highly generic
  - By using a broad family for $\phi(\boldsymbol{x};\boldsymbol{\theta})$
- Can also capture benefits of second approach
  - Human practitioners design families of $\phi(\boldsymbol{x};\boldsymbol{\theta})$ that will perform well
  - Need only find right function family rather than precise right function

Importance of Learning ϕ

- Learning ϕ is discussed beyond this first introduction to feed-forward networks
  - It is a recurring theme throughout deep learning applicable to all kinds of models
- Feedforward networks are application of this principle to learning deterministic mappings form x to y without feedback
- Applicable to
  - learning stochastic mappings
  - functions with feedback
  - learning probability distributions over a single vector

Plan of Discussion: Feedforward Networks

1. A simple example: learning XOR
2. Design decisions for a feedforward network
   - Many are same as for designing a linear model
     - Basics of gradient descent
       - Choosing the optimizer, Cost function, Form of output units
   - Some are unique
     - Concept of hidden layer
       - Makes it necessary to have activation functions
     - Architecture of network
       - How many layers , How are they connected to each other, How many units in each later
     - Learning requires gradients of complicated functions
       - Backpropagation and modern generalizations

1. Ex: XOR problem

- XOR: an operation on binary variables x_1 and x_2
  - When exactly one value equals 1 it returns 1 otherwise it returns 0
  - Target function is $y = f^*(\boldsymbol{x})$ that we want to learn
    - Our model is $y = f([x_1, x_2]; \boldsymbol{\theta})$ which we learn, i.e., adapt parameters $\boldsymbol{\theta}$ to make it similar to f^*
- Not concerned with statistical generalization
  - Perform correctly on four training points:
    - $X = \{[0,0]^T, [0,1]^T, [1,0]^T, [1,1]^T\}$
  - Challenge is to fit the training set
    - We want $f([0,0]^T; \boldsymbol{\theta}) = f([1,1]^T; \boldsymbol{\theta}) = 0$
    - $f([0,1]^T; \boldsymbol{\theta}) = f([1,0]^T; \boldsymbol{\theta}) = 1$

ML for XOR: linear model doesn't fit

- Treat it as regression with MSE loss function

$$J(\boldsymbol{\theta}) = \frac{1}{4}\sum_{\boldsymbol{x}\in X}\left(f^*(\boldsymbol{x}) - f(\boldsymbol{x};\boldsymbol{\theta})\right)^2 = \frac{1}{4}\sum_{n=1}^{4}\left(f^*(\boldsymbol{x}_n) - f(\boldsymbol{x}_n;\boldsymbol{\theta})\right)^2$$

  - Usually not used for binary data
  - But math is simple

Alternative is Cross-entropy $J(\boldsymbol{\theta})$

$$J(\boldsymbol{\theta}) = -\ln p(\boldsymbol{t}\,|\,\boldsymbol{\theta}) = -\sum_{n=1}^{N}\left\{t_n \ln y_n + (1-t_n)\ln(1-y_n)\right\}$$

$$y_n = \sigma(\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{x}_n)$$

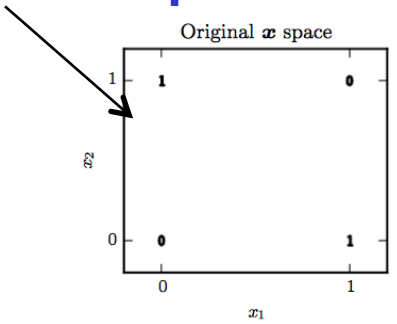
- We must choose the form of the model
- Consider a linear model with $\boldsymbol{\theta} = \{\boldsymbol{w}, b\}$ where

$$f(\boldsymbol{x};\boldsymbol{w},b) = \boldsymbol{x}^T\boldsymbol{w} + b$$

  - Minimize $J(\boldsymbol{\theta}) = \frac{1}{4}\sum_{n=1}^{4}\left(t_n - \boldsymbol{x}_n^T\boldsymbol{w} - b\right)^2$ to get closed-form solution
    - Differentiate wrt $\boldsymbol{w}$ and b to obtain $\boldsymbol{w} = \mathbf{0}$ and $b = ½$
      - Then the linear model $f(\boldsymbol{x};\boldsymbol{w},b) = ½$ simply outputs 0.5 everywhere
  - Why does this happen?

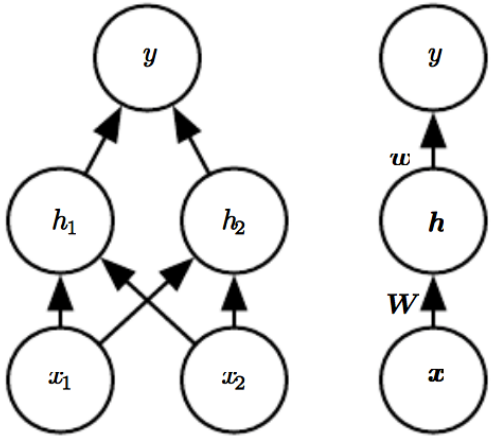
Linear model cannot solve XOR

- Bold numbers are values system must output
  - When $x_1=0$, output has to increase with x_2
  - When $x_1=1$, output has to decrease with x_2



  - Linear model $f(\boldsymbol{x};\boldsymbol{w},b)= x_1w_1+x_2w_2+b$ has to assign a single weight to x_2, so it cannot solve this problem
- A better solution:
  - use a model to learn a different representation
    - in which a linear model is able to represent the solution
  - We use a simple feedforward network
    - one hidden layer containing two hidden units

Feedforward Network for XOR



- Introduce a simple feedforward network
  - with one hidden layer containing two units
- Same network drawn in two different styles
  - Matrix W describes mapping from $\boldsymbol{x}$ to $\boldsymbol{h}$
  - Vector $\boldsymbol{w}$ describes mapping from $\boldsymbol{h}$ to y
  - Intercept parameters b are omitted

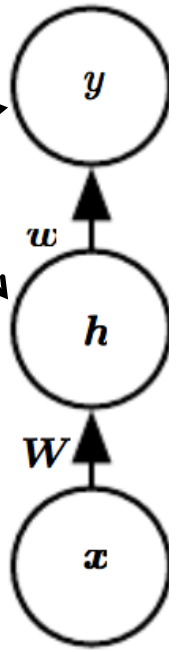
Functions computed by Network

- Layer 1 (hidden layer): vector of hidden units h computed by function $f^{(1)}(\boldsymbol{x};\ W, \boldsymbol{c})$
  - $\boldsymbol{c}$ are bias variables
- Layer 2 (output layer) computes

  $$f^{(2)}(\boldsymbol{h};\ \boldsymbol{w}, b)$$

  - w are linear regression weights
  - Output is linear regression applied to $\boldsymbol{h}$ rather than to $\boldsymbol{x}$
- Complete model is

  $$f(\boldsymbol{x};\ W, \boldsymbol{c}, \boldsymbol{w}, \boldsymbol{b}) = f^{(2)}(f^{(1)}(x))$$



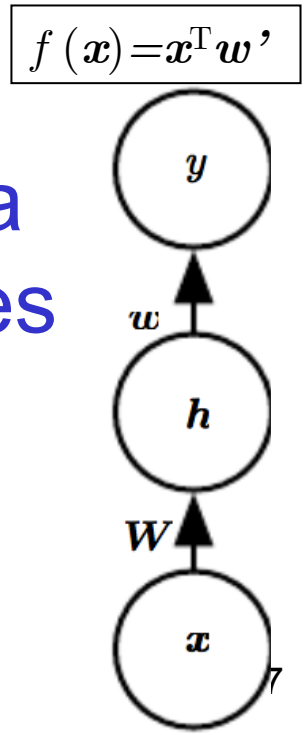
Linear vs Nonlinear functions

- If we choose both $f^{(1)}$ and $f^{(2)}$ to be linear, the total function will still be linear $f(\boldsymbol{x})=\boldsymbol{x}^{\mathrm{T}}\boldsymbol{w}'$
  - Suppose $f^{(1)}(\boldsymbol{x})= W^{\mathrm{T}}x$ and $f^{(2)}(\boldsymbol{h})=\boldsymbol{h}^{\mathrm{T}}\boldsymbol{w}$
  - Then we could represent this function as

$$f(\boldsymbol{x})=\boldsymbol{x}^{\mathrm{T}}\boldsymbol{w}' \text{ where } \boldsymbol{w}'=W\boldsymbol{w}$$

- Since linear is insufficient, we must use a nonlinear function to describe the features
  - We use the strategy of neural networks
  - by using a nonlinear activation function

$$\boldsymbol{h}=g(W^{\mathrm{T}}\boldsymbol{x}+\boldsymbol{c})$$

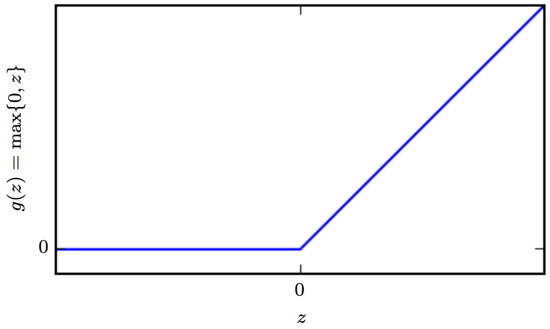


Activation Function

- In linear regression we used a vector of weights $\boldsymbol{w}$ and scalar bias b $\boxed{f(\boldsymbol{x};\boldsymbol{w},b)=\boldsymbol{x}^T\boldsymbol{w}+b}$
  - to describe an affine transformation from an input vector to an output scalar
- Now we describe an affine transformation from a vector $\boldsymbol{x}$ to a vector $\boldsymbol{h}$, so an entire vector of bias parameters is needed
- Activation function g is typically chosen to be applied element-wise $\boldsymbol{h}_i=g(x^TW_{:,i}+c_i)$

Default Activation Function

- Activation: $g(z) = \max\{0, z\}$
  - Applying this to the output of a linear transformation yields a nonlinear transformation
  - However function remains close to linear
    - Piecewise linear with two pieces
    - Therefore preserve properties that make linear models easy to optimize with gradient-based methods
    - Preserve many properties that make linear models generalize well



A principle of CS:
Build complicated systems from minimal components.

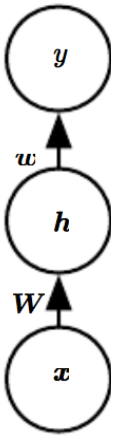
A Turing Machine Memory needs only 0 and 1 states.

We can build Universal Function approximator from ReLUs

Specifying the Network using ReLU

- Activation: $g(z)=\max\{0,z\}$
- We can now specify the complete network as

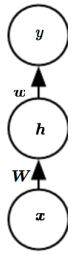
$$f(\boldsymbol{x};\ W,\boldsymbol{c},\boldsymbol{w},\boldsymbol{b})=f^{(2)}(f^{(1)}(x))=\boldsymbol{w}^T\max\{0,W^T\boldsymbol{x}+\boldsymbol{c}\}+b$$



We can now specify XOR Solution

- Let $W = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \ \boldsymbol{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \ \boldsymbol{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}, \ b = 0$

$$f(\boldsymbol{x}; W, \boldsymbol{c}, \boldsymbol{w}, \boldsymbol{b}) = \boldsymbol{w}^T max\{0, W^T\boldsymbol{x} + \boldsymbol{c}\} + b$$

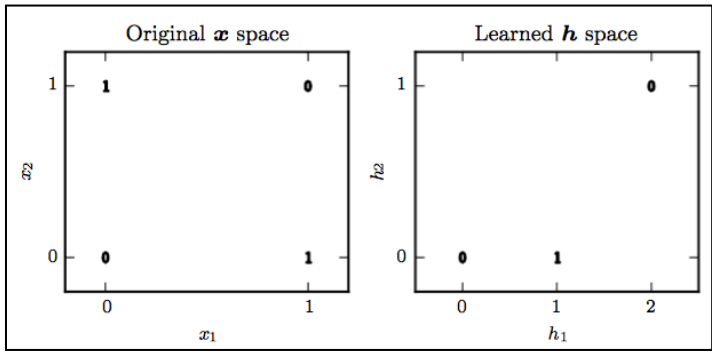


- Now walk through how model processes a batch of inputs
  - Design matrix X of all four points: $X = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}$
  - First step is XW: $XW = \begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 2 & 2 \end{bmatrix}$
  - Adding $\boldsymbol{c}$: $XW + c = \begin{bmatrix} 0 & -1 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$

    In this space all points lie along a line with slope 1. Cannot be implemented by a linear model
  - Compute $\boldsymbol{h}$ Using ReLU $\max\{0, XW + c\} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$

    Has changed relationship among examples. They no longer lie on a single line. A linear model suffices
  - Finish by multiplying by $\boldsymbol{w}$: $f(\boldsymbol{x}) = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$
  - Network has obtained

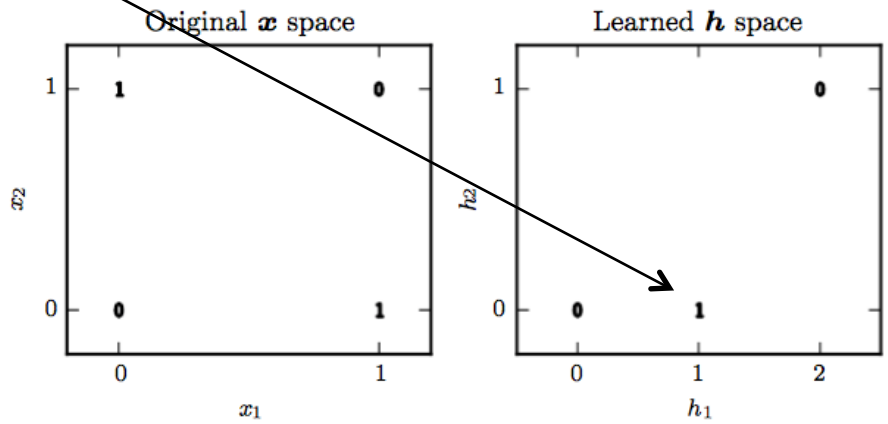
    correct answer for all 4 examples



Learned representation for XOR

- Two points that must have output 1 have been collapsed into one
- Points $\boldsymbol{x}=[0,1]^{\mathrm{T}}$ and $\boldsymbol{x}=[1,0]^{\mathrm{T}}$ have been mapped into $\boldsymbol{h}=[0,1]^{\mathrm{T}}$

When $x_1=0$, output has to increase with x_2
When $x_1=1$, output has to decrease with x_2



- Described in linear model
  - For fixed h_2, output increases in h_1

When $h_1=0$, output is constant 0 with h_2
When $h_1=1$, output is constant 1 with h_2
When $h_1=2$, output is constant 0 with h_2

About the XOR example

- We simply specified the solution
  - Then showed that it achieves zero error
- In real situations there might be billions of parameters and billions of training examples
  - So one cannot simply guess the solution
- Instead gradient descent optimization can find parameters that produce very little error
  - The solution described is at the global minimum
    - Gradient descent could converge to this solution
    - Convergence depends on initial values
    - Would not always find easily understood integer solutions