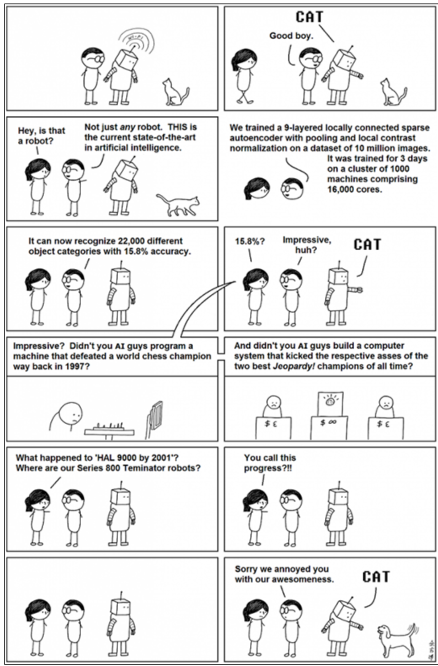


Artificial Neural Networks

Mark Dredze

Machine Learning
CS 600.475

Handling Non-Linear Data

- Option 1: Add features by hand that make the data separable
  - Requires feature engineering
- Option 2: Learn a small number of additional features that will suffice
  - Today
- Option 3: Kernel trick

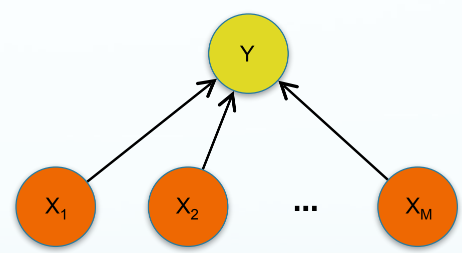
Motivation

- Where do features come from?
  - We build them by hand
- What if we wanted to *learn* features?
  - Goal: learn features that give linearly separable data
- After learning features apply usual linear classifier

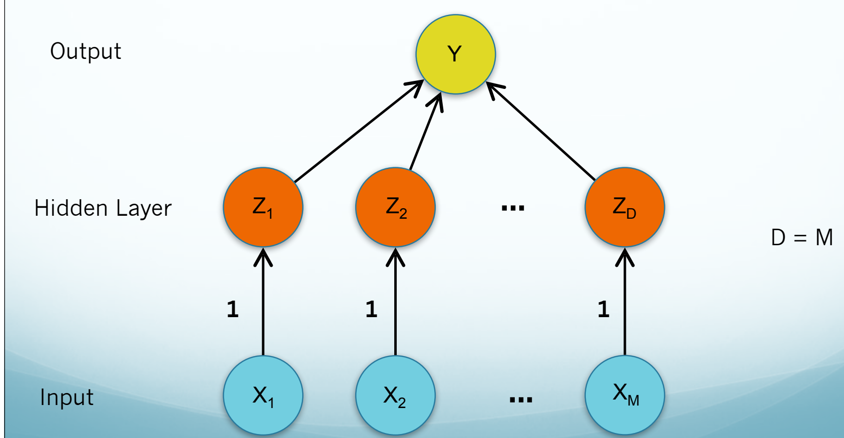
Perceptron: Graphical Representation

Output

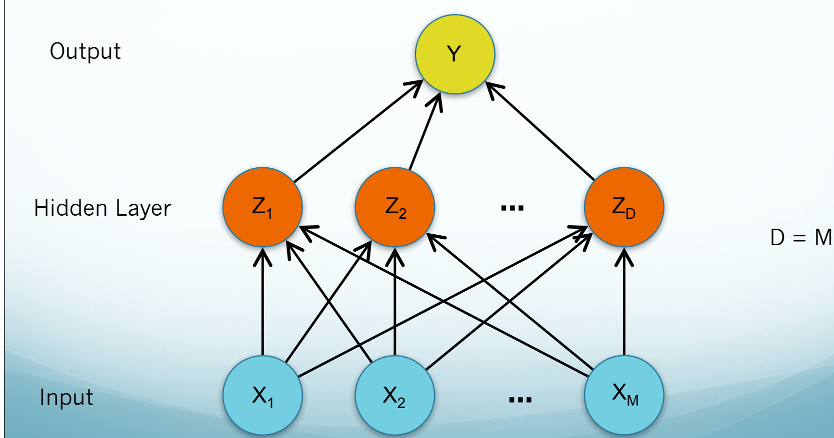
Features



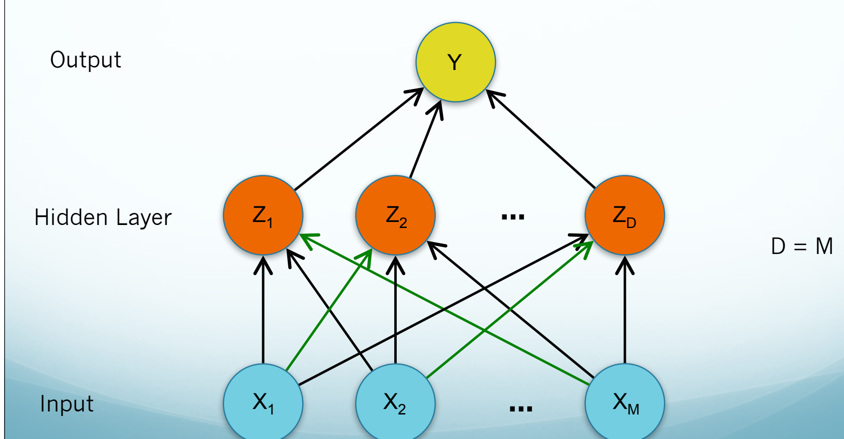
Perceptron: Graphical Representation



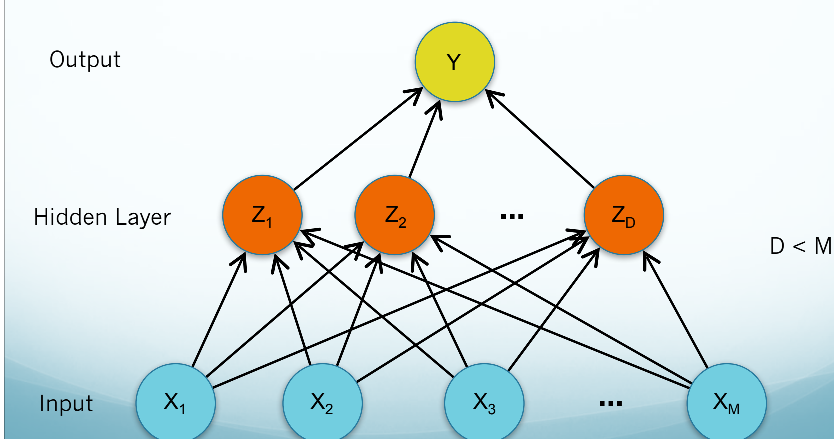
Perceptron: Graphical Representation



Perceptron: Graphical Representation



Perceptron: Graphical Representation



Why?

- Constraints from X
  - When D = M, likely to copy the features from X to Z
  - When D < M, cannot make an exact copy of X
    - Must come up with a representation that is more efficient
- Constraints from Y
  - Z should be a representation that helps learn Y
  - Forces the low-dimensional representation to capture properties of X useful in predicting Y

Why Non-Linear

- Generalized linear classifiers!
  - Start with linear function
    - $w \cdot x$
  - Pass the output through a non-linear function
    - $\hat{y} = \mathrm{h}(w \cdot x)$
  - What is h?
    - Non-linear function
      - Logistic function
      - Sign function
- Each Z is the output of a non-linear function
  - Combinations of Z are now non-linear in X

Multi-Layer Perceptrons

Fitting a function to data

- **Fitting: what type of optimization algorithm?**
- Function: non-linear: linear combination of generalized linear functions
- Data: Data/model assumptions? How we use data?

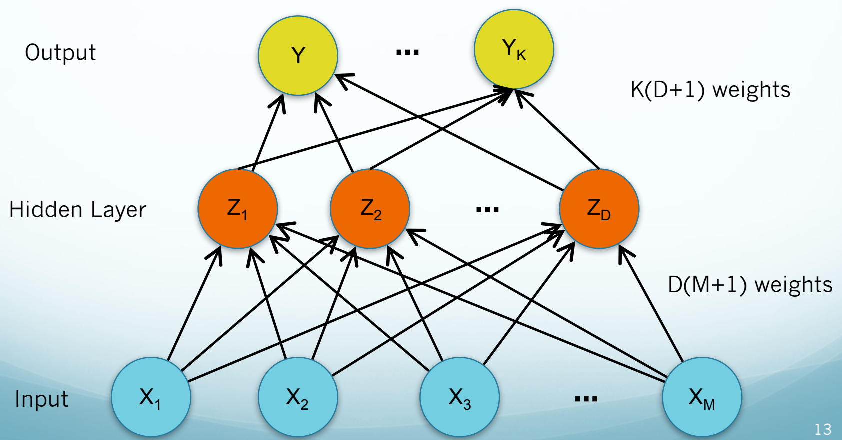
How Will We Learn?

- Perceptron: a training method for generalized linear classifiers
  - Training method for linear classifiers
  - Minimize the error of the training data
  - Chain multiple Perceptrons together
  - Update rule:

$$w^{i+1} = w^{i} + \nabla f(x, y)$$

  - The real work will be in computing the gradient

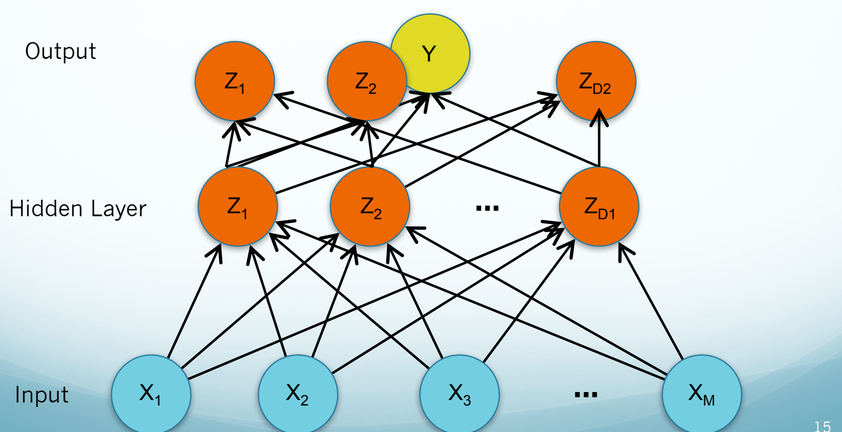
Example: Multi-Class



Network Terminology

- Input nodes: x
- Output node: y
- Hidden nodes: z
  - This network has 1 hidden layer
  - 2 layer network (two layers to learn)
- h for hidden nodes are called activation functions
- h for output depends on task
  - Identity for regression
  - Logistic for classification

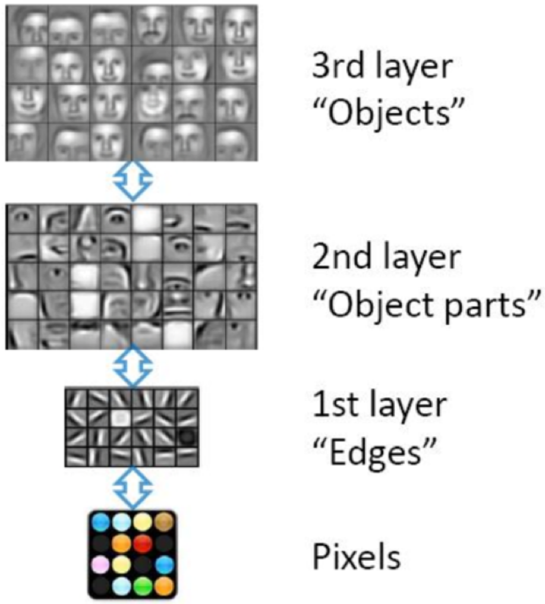
Deep Networks



Deep Networks

- Learn multiple levels of features at higher and higher abstractions
- Same learning techniques
  - Just more complex gradients

Feature representation



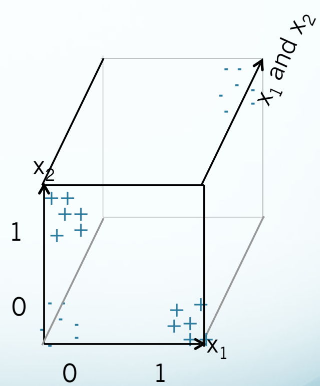
Example: Image Processing

Outline

- Lecture 1: Neural Networks
  - Nonlinearities
  - Objective Functions
  - Training
  - Gradient Computations
- Lecture 2: Deep Learning
  - Supervised and unsupervised training
  - Pre-training
  - Auto-encoders

An Non-linear Example

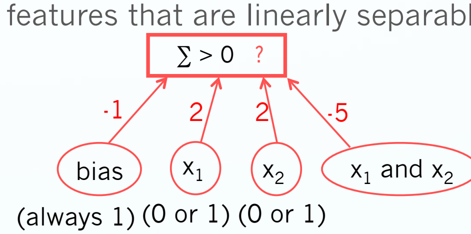
- Consider the xor function
  - $y(x) = 1$ iff x_1 xor x_2
- Clearly non-linear
  - No values for w will produce desired output
- We could solve this by adding a new feature
  - $x_3 = x_1$ xor x_2



Example from Jason Eisner

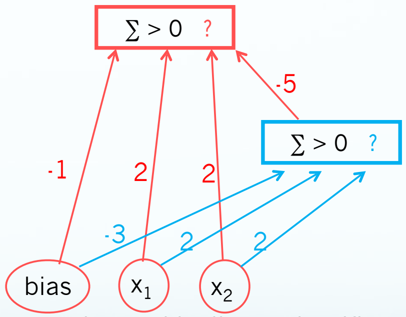
The Neural Network Solution

- Learn new features that are linearly separable



- We now have a linear classifier for XOR
- How do we learn these features?

The Neural Network Solution

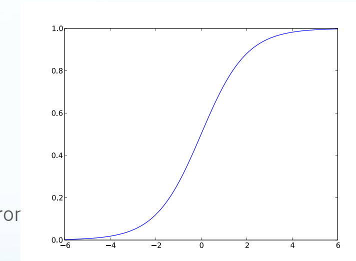


- The new features are learned by linear classifiers
  - All other hidden nodes (not shown) just replicate input
- The activation function makes the feature 1 or 0

Non-linear Activation Functions

- What non-linear function should we use for activation function h?
  - Typically use sigmoid functions
    - Logistic function

$$g_\alpha(x) = \frac{1}{1 + e^{-\alpha x}}$$



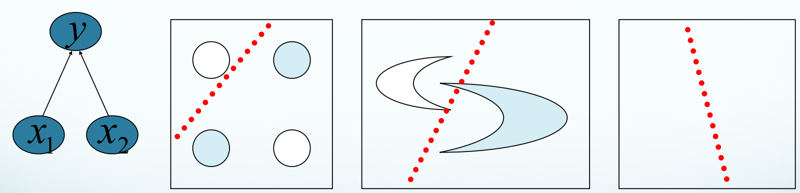
- Each hidden node has a threshold for activation
  - Will be 0 and then quickly transition to 1
- This is what we use when we stack Perceptrons
- This is why we think of hidden nodes as features
  - They are off and then when enough input they turn on
  - Learning input weights turns on the feature!

Hypothesis Class

- What can a neural network learn?
  - Obviously highly non-linear outputs
- Universal approximators
  - With enough hidden layers and hidden nodes a neural network can model any continuous function on compact input domain (some number of inputs)
  - The power of the networks depends on its structure
    - General result independent of activation functions

Decision Boundary

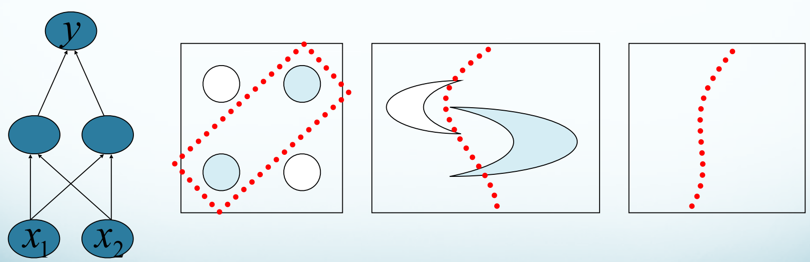
- 0 hidden layers: linear classifier
  - Hyperplanes



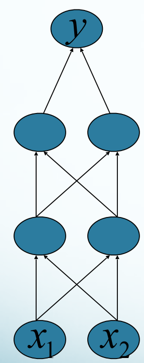
Example from to Eric Postma via Jason Eisner

Decision Boundary

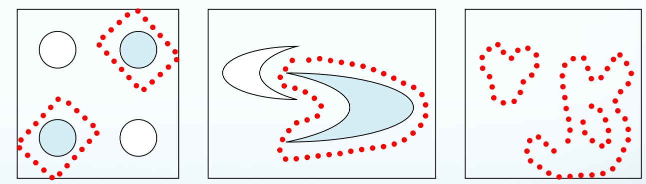
- 1 hidden layer
  - Boundary of convex region (open or closed)



Decision Boundary



- 2 hidden layers
  - Combinations of convex regions

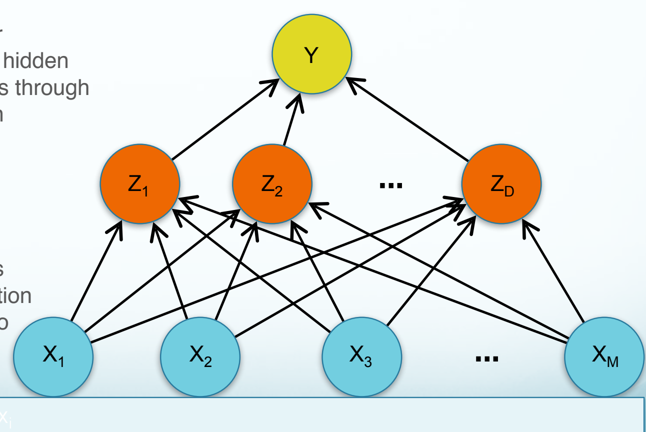


Prediction

Compute linear combination of hidden nodes and pass through logistic function

Hidden nodes are now new features

Compute each hidden node as linear combination of x and pass to logistic



- Forward propagation through the network

Classification Objective

- Define an error function and minimize
- Cross entropy error function

$$E(w) = -\sum_{i=1}^{N} \{ y_i \ln \hat{y}_i + (1 - y_i) \ln(1 - \hat{y}_i) \}$$

- This arises naturally when we consider a logistic probability model and take the negative log likelihood
  - Details in the book

Regression Objective

- For regression we use the sum of squares error

$$E(w) = \frac{1}{2}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2$$

- If we assume a Gaussian model for y, the error function arises from maximizing the likelihood function
  - We saw the same thing for linear regression

Combined Model

(See lecture notes for derivation)

Combined Model

- Writing the generalized linear classifier with generalized non-linear basis functions

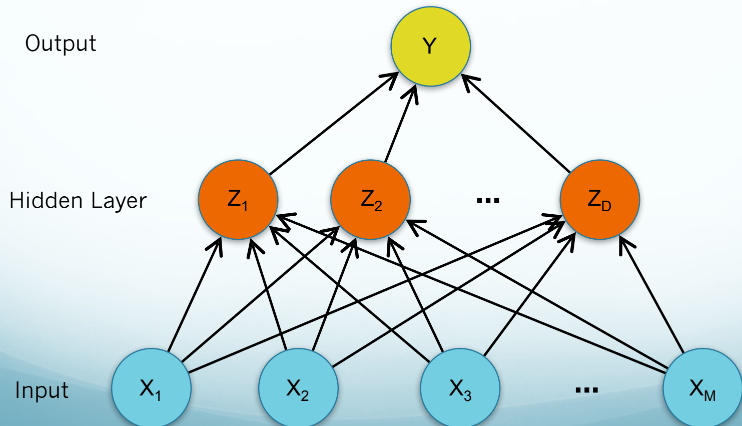
$$y(x; w) = h^{(2)}\left(\sum_{j=1}^{D} w_j^{(2)} h^{(1)}\left(\sum_{i=1}^{M} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_0^{(2)}\right)$$

  - $h^{(1)}$ is the non-linear function for the basis function
  - $h^{(2)}$ is the non-linear function for the output
  - $w^{(1)}$ are the parameters for the basis function
  - $w^{(2)}$ are the parameters for the linear model
  - w_0 are the bias parameters (shown here for clarity)

Training

- Prediction is relatively easy
- Learning is where the magic happens
- Strategy: compute the gradient of the objective function
  - Similar to perceptron
  - Gradient based update

Graphical Representation

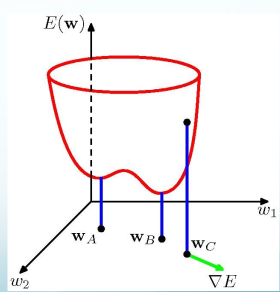


Training

- Prediction is relatively easy
- Learning is where the magic happens
- Strategy: compute the gradient of the objective function
  - Similar to perceptron
  - Gradient based update
- For the moment: assume black box computes gradient

Gradient Based Optimization

- The objective function is now non-convex
- Gradient based optimization NOT guaranteed to find global optimum
- For now: use gradient stochastic gradient and hope for the best
- Next time: tricks for non-convexity key to learning good networks



Computing the Gradient

- For arbitrary Neural Network architectures we can use Backpropagation!

(See lecture notes for derivation)

Algorithm: Neural Network

- Train: Given examples X and Y
  - Y can be multiple outputs
  - Define a network structure
  - ex. 2 layer feed forward, D nodes in hidden layer
  - Learn parameters w
- Predict: given example x
  - For 2 layer feed forward, compute output as

$$\hat{y} = h^{(2)}\left(\sum_{j=1}^{D} w_j^{(2)} h^{(1)}\left(\sum_{i=1}^{M} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_0^{(2)}\right)$$

Multi-Layer Perceptrons

Fitting a function to data

- Fitting: gradient based optimization with back-propogation
- Function: non-linear: linear combination of generalized linear functions
  - Universal approximations
  - can model any continuous function on compact input domain (some number of inputs)
- Data: Batch training using stochastic methods

Next Time

Deep(er) Networks