Machine Learning Department
School of Computer Science
Carnegie Mellon University

Neural Networks

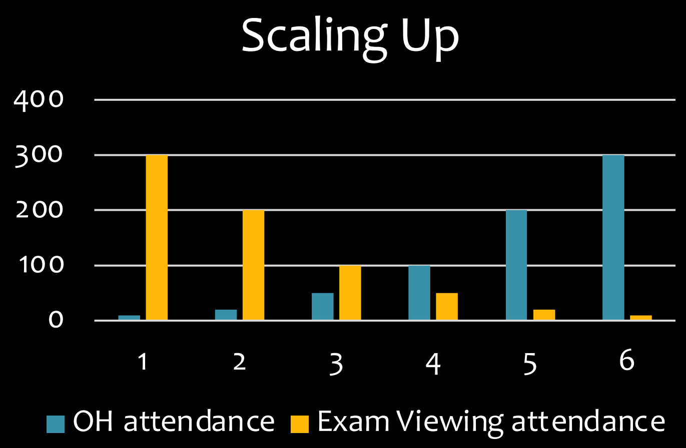
+

Backpropagation

Matt Gormley & Henry Chai
Lecture 12
Oct. 2, 2024

Reminders

- **Post-Exam Followup:**
  - Exam Viewing
  - Exit Poll: Exam 1
  - Grade Summary 1
- **Homework 4: Logistic Regression**
  - Out: Mon, Sep 30
  - Due: Wed, Oct 9 at 11:59pm



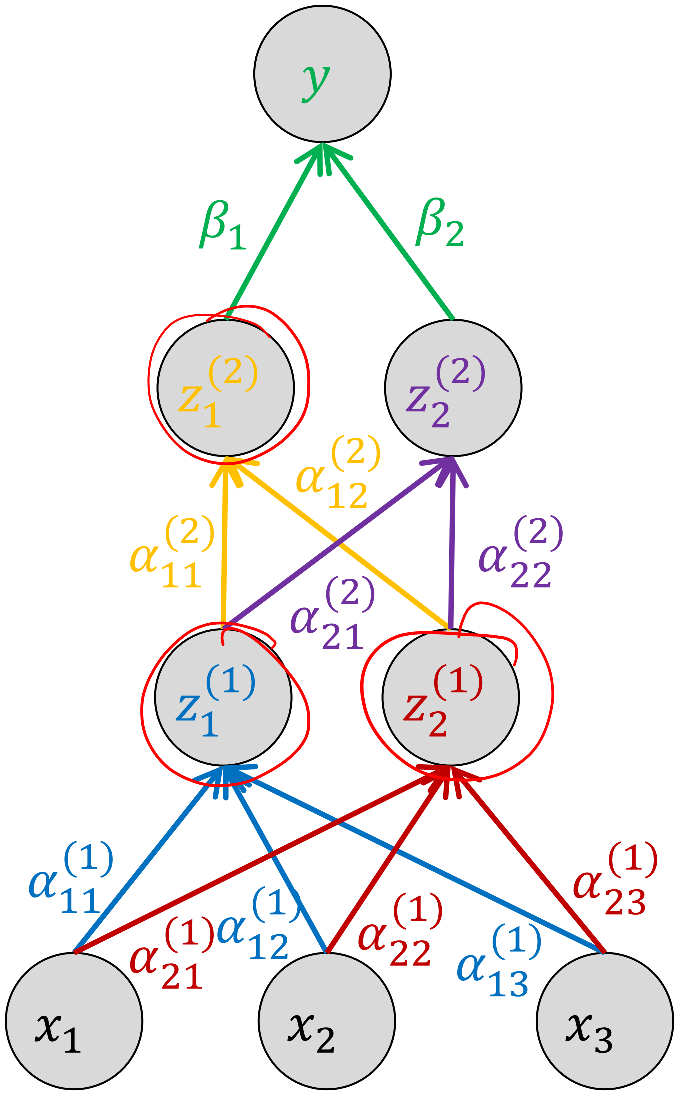
ARCHITECTURES

Neural Network Architectures

Even for a basic Neural Network, there are many design decisions to make:

1. # of hidden layers (depth)
2. # of units per hidden layer (width)
3. Type of activation function (nonlinearity)
4. Form of objective function
5. How to initialize the parameters

Neural Network



Example: Neural Network with 2 Hidden Layers and 2 Hidden Units

$$z_1^{(1)} = \sigma(\alpha_{11}^{(1)} x_1 + \alpha_{12}^{(1)} x_2 + \alpha_{13}^{(1)} x_3 + \overbrace{\alpha_{10}^{(1)}}^{\text{intercept term}})$$

$$z_2^{(1)} = \sigma(\alpha_{21}^{(1)} x_1 + \alpha_{22}^{(1)} x_2 + \alpha_{23}^{(1)} x_3 + \alpha_{20}^{(1)})$$

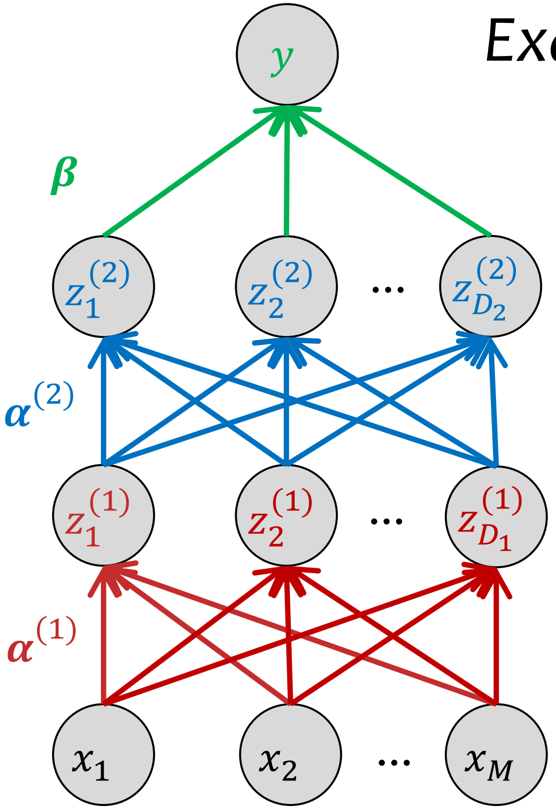
$$z_1^{(2)} = \sigma(\alpha_{11}^{(2)} z_1^{(1)} + \alpha_{12}^{(2)} z_2^{(1)} + \alpha_{10}^{(2)})$$

$$z_2^{(2)} = \sigma(\alpha_{21}^{(2)} z_1^{(1)} + \alpha_{22}^{(2)} z_2^{(1)} + \alpha_{20}^{(2)})$$

$$y = \sigma(\beta_1 z_1^{(2)} + \beta_2 z_2^{(2)} + \beta_0)$$

Neural Network (Matrix Form)

Example: Arbitrary Feed-forward Neural Network



$\boldsymbol{\beta} \in \mathbb{R}^{D_2}$

$\boxed{\beta_0 \in \mathbb{R}}$

$\boldsymbol{\alpha}^{(2)} \in \mathbb{R}^{M \times D_2}$

$\boxed{\boldsymbol{b}^{(2)} \in \mathbb{R}^{D_2}}$

$\boldsymbol{\alpha}^{(1)} \in \mathbb{R}^{M \times D_1}$

$\boxed{\boldsymbol{b}^{(1)} \in \mathbb{R}^{D_1}}$

$$y = \sigma((\boldsymbol{\beta})^T \boldsymbol{z}^{(2)} + \beta_0)$$

$$\boldsymbol{z}^{(2)} = \sigma((\boldsymbol{\alpha}^{(2)})^T \boldsymbol{z}^{(1)} + \boldsymbol{b}^{(2)})$$

$$\boldsymbol{z}^{(1)} = \sigma(\underbrace{\underbrace{\underbrace{(\boldsymbol{\alpha}^{(1)})^T}_{D_1 \times M} \underbrace{\boldsymbol{x}}_{M \times 1}}_{D_1 \times 1} + \underbrace{\boldsymbol{b}^{(1)}}_{D_1 \times 1}}_{D_1 \times 1})$$

$\sigma : \mathbb{R} \to \mathbb{R}$ apply "elementwise" to each entry in the vector

$\vec{v} \in \mathbb{R}^M$ $\quad \vec{z} = \sigma(\vec{v})$

s.t. $z_j = \sigma(v_j)$

Fold in the intercept term?

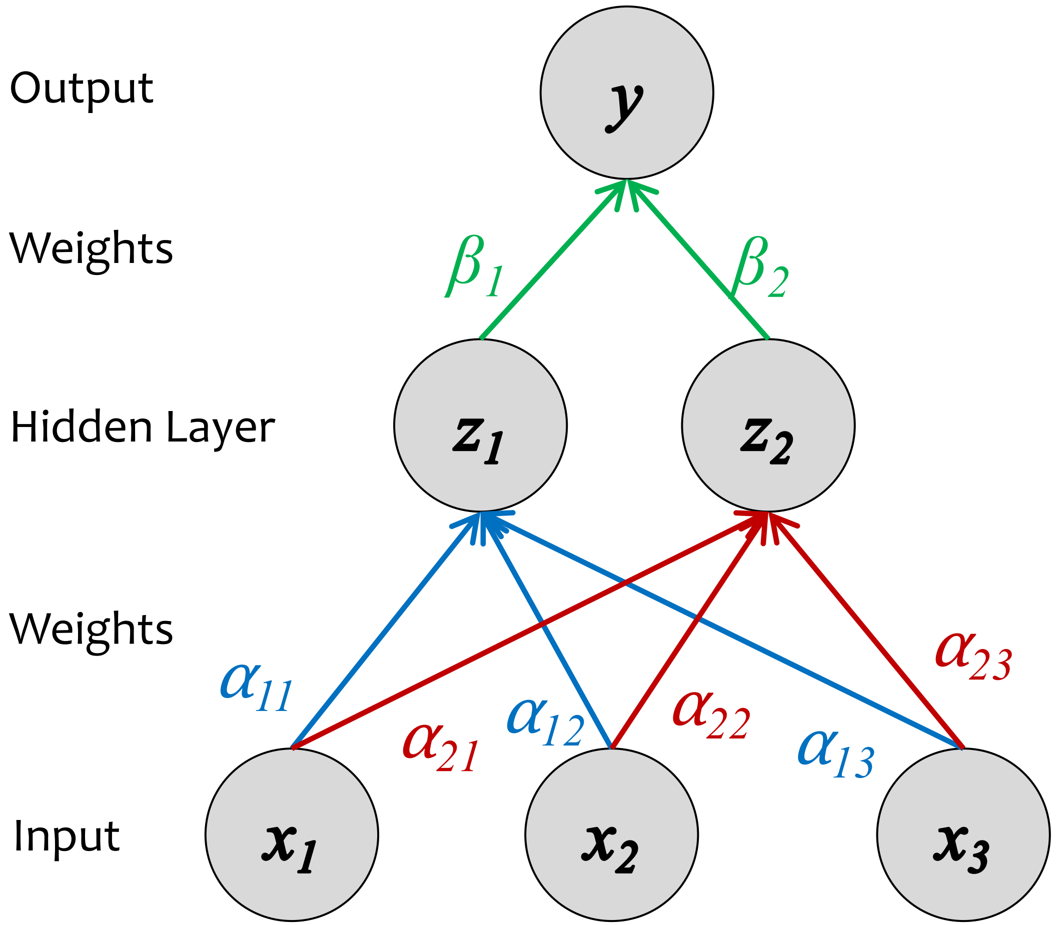
Set $x_1 = 1$, $z_1^{(1)} = 1$, $z_1^{(2)} = 1$

Remove $b^{(1)}, b^{(2)}, \beta_0$

Caution: tricky to implement

Neural Network (Vector Form)

Neural Network with 1 Hidden Layers and 2 Hidden Units (Matrix Form)



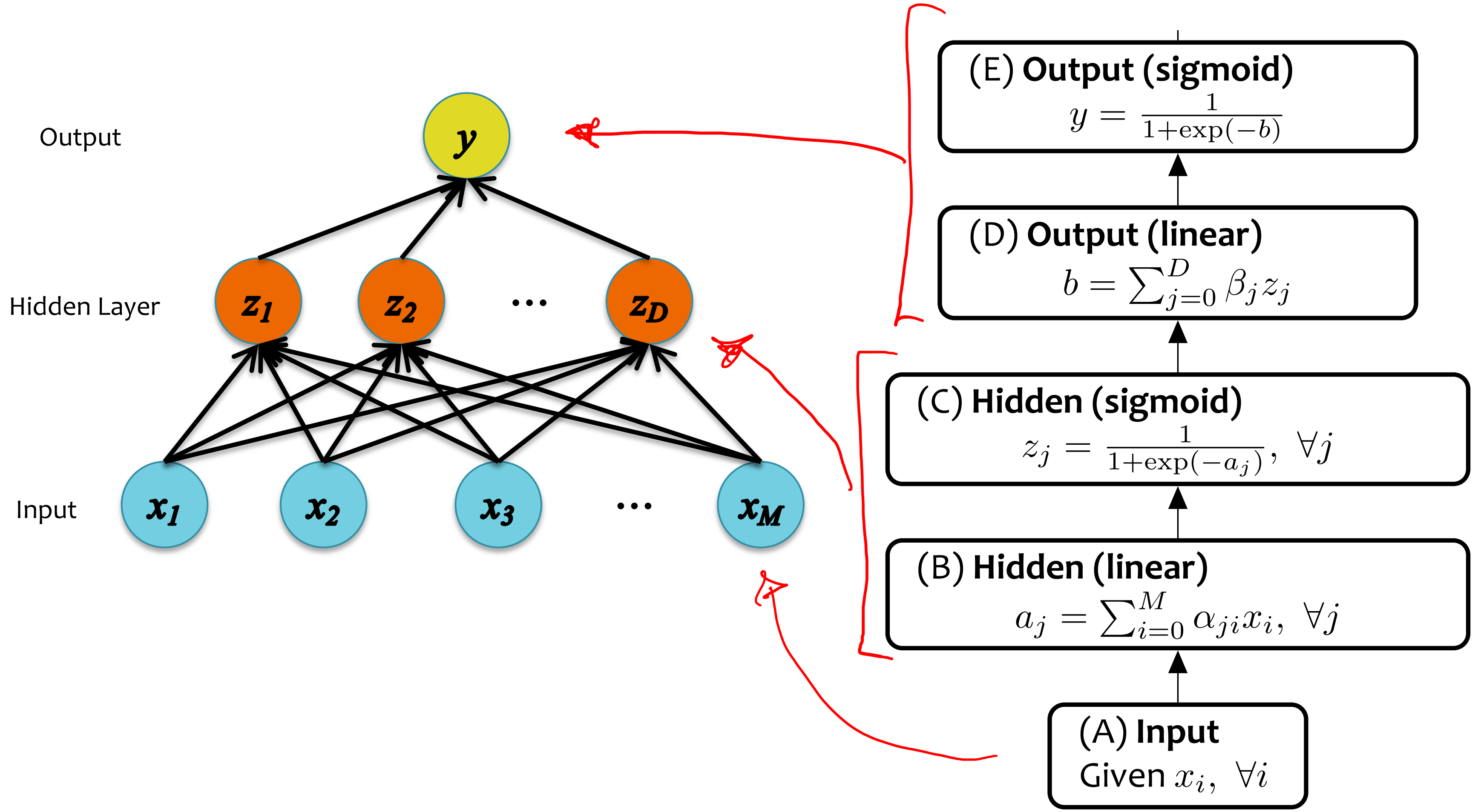
$$y = \sigma(\boldsymbol{\beta}^T \mathbf{z})$$

$$z_2 = \sigma(\boldsymbol{\alpha}_{2,\cdot}^T \mathbf{x})$$

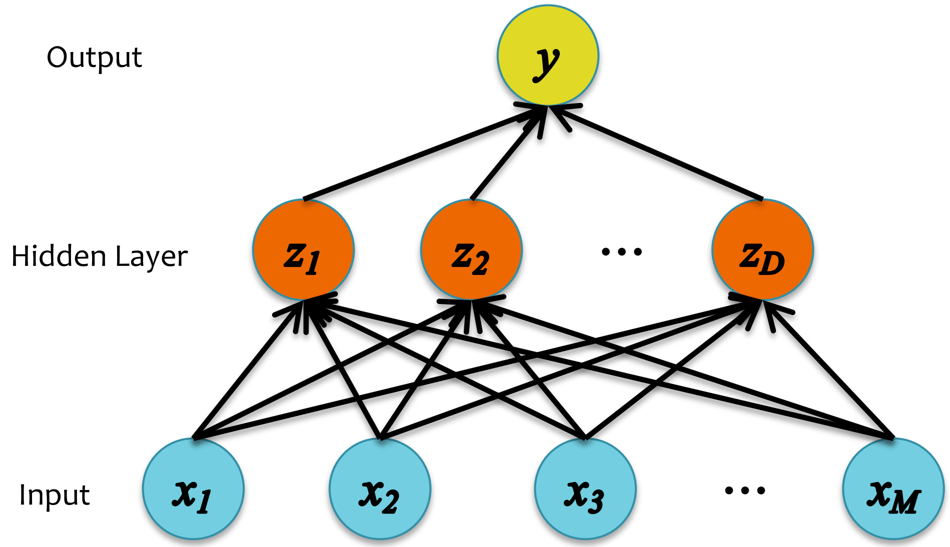
$$z_1 = \sigma(\boldsymbol{\alpha}_{1,\cdot}^T \mathbf{x})$$

LOSS FUNCTIONS & OUTPUT LAYERS

Neural Network for Classification



Neural Network for Regression



(D) Output (linear)
$y = \sum_{j=0}^{D} \beta_j z_j$

(C) Hidden (sigmoid)
$z_j = \frac{1}{1+\exp(-a_j)}, \ \forall j$

(B) Hidden (linear)
$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i, \ \forall j$

(A) Input
Given $x_i, \ \forall i$

Objective Functions for NNs

1. Quadratic Loss:
   - the same objective as Linear Regression
   - i.e. mean squared error

$$J = \ell_Q(y, y^{(i)}) = \frac{1}{2}(y - y^{(i)})^2$$

$$\frac{dJ}{dy} = y - y^{(i)}$$

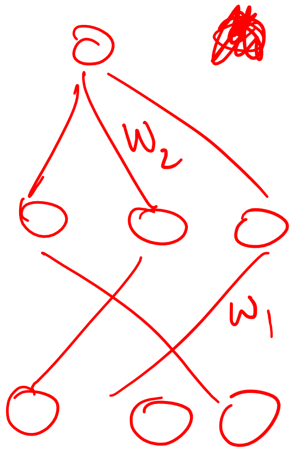
2. Binary Cross-Entropy:
   - the same objective as Binary Logistic Regression
   - i.e. negative log likelihood
   - This requires our output y to be a probability in [0,1]

$$J = \ell_{CE}(y, y^{(i)}) = -(y^{(i)} \log(y) + (1 - y^{(i)}) \log(1 - y))$$

$$\frac{dJ}{dy} = -\left(y^{(i)} \frac{1}{y} + (1 - y^{(i)}) \frac{1}{y - 1}\right)$$

Objective Functions for NNs

Cross-entropy vs. Quadratic loss



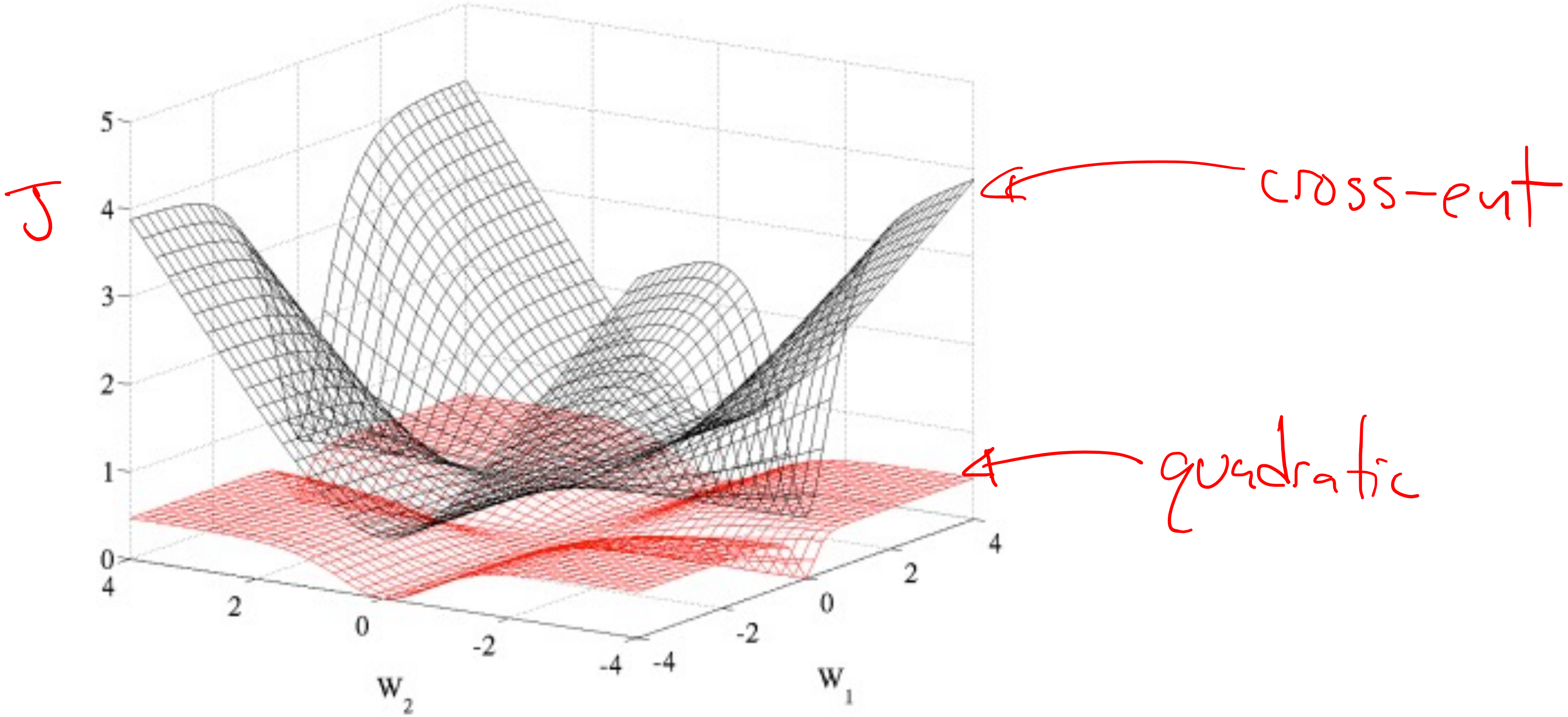
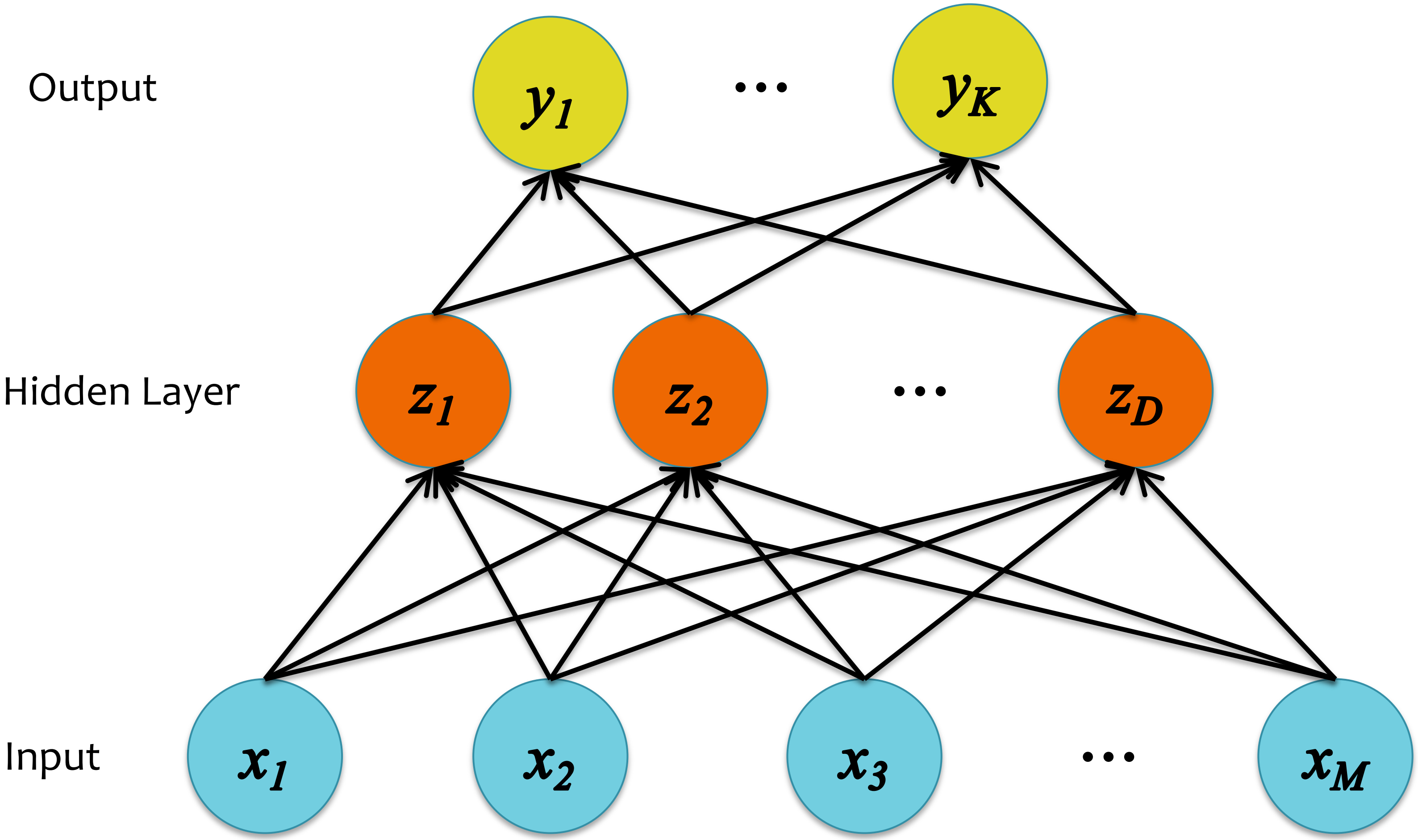


Figure 5: Cross entropy (black, surface on top) and quadratic (red, bottom surface) cost as a function of two weights (one at each layer) of a network with two layers, W_1 respectively on the first layer and W_2 on the second, output layer.

Figure from Glorot & Bentio (2010)

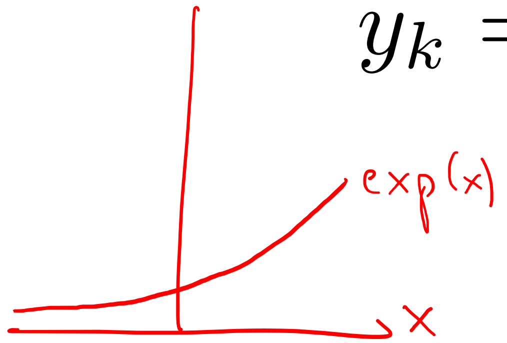
Multiclass Output

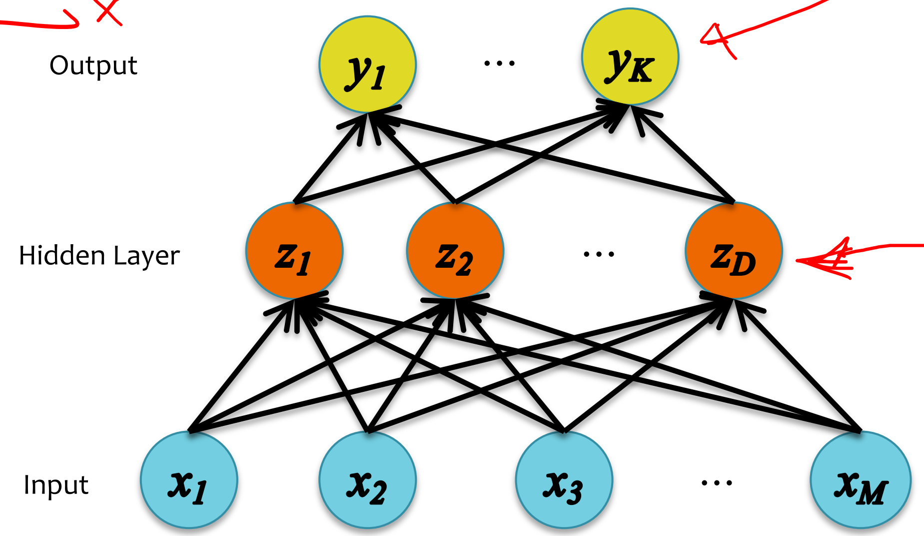


Multiclass Output

Softmax:

$$y_k = \frac{\exp(b_k)}{\sum_{l=1}^{K} \exp(b_l)} = P(Y=k)$$





(F) **Loss**
$J = \sum_{k=1}^{K} y_k^* \log(y_k)$

(E) **Output (softmax)**
$y_k = \frac{\exp(b_k)}{\sum_{l=1}^{K} \exp(b_l)}$

(D) **Output (linear)**
$b_k = \sum_{j=0}^{D} \beta_{kj} z_j \;\; \forall k$

(C) **Hidden (nonlinear)**
$z_j = \sigma(a_j), \;\; \forall j$

(B) **Hidden (linear)**
$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i, \;\; \forall j$

(A) **Input**
Given $x_i, \;\; \forall i$

$\vec{y} = [4/14, 3/14, 7/14]$

$y_1, y_2, \ldots, y_K$

$\exp(\vec{b}) = [4, 3, 7]$

$b_1, b_2, \ldots, b_K$

Objective Functions for NNs

3. Cross-Entropy for Multiclass Outputs:
   - i.e. negative log likelihood for multiclass outputs
   - Suppose output is a random variable Y that takes one of K values
   - Let $\mathbf{y}^{(i)}$ represent our true label as a one-hot vector:

$\tilde{y}^{(i)} = 4$

$\mathbf{y}^{(i)}$ =

0	0	0	1	0	0	…	0
1	2	3	4	5	6	…	K

   - Assume our model outputs a length K vector of probabilities:

$$\mathbf{y} = \text{softmax}(f_{\text{scores}}(\mathbf{x}, \boldsymbol{\theta}))$$

   - Then we can write the log-likelihood of a single training example $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$ as:

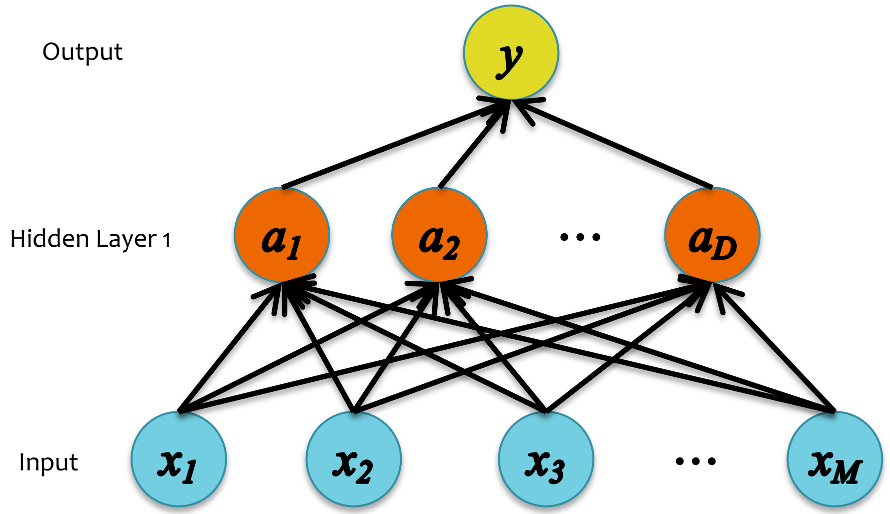
$$J = \ell_{CE}(\mathbf{y}, \mathbf{y}^{(i)}) = -\sum_{k=1}^{K} y_k^{(i)} \log(y_k) = -\log(y_{\tilde{y}_i})$$

$$= -(0\log(y_1) + 0\log(y_2) + 0\log(y_3) + 1\log(y_4) + 0\log(y_5) + \cdots 0\log(y_K))$$

$$= -\log(y_4) = -\log P(Y=4)$$

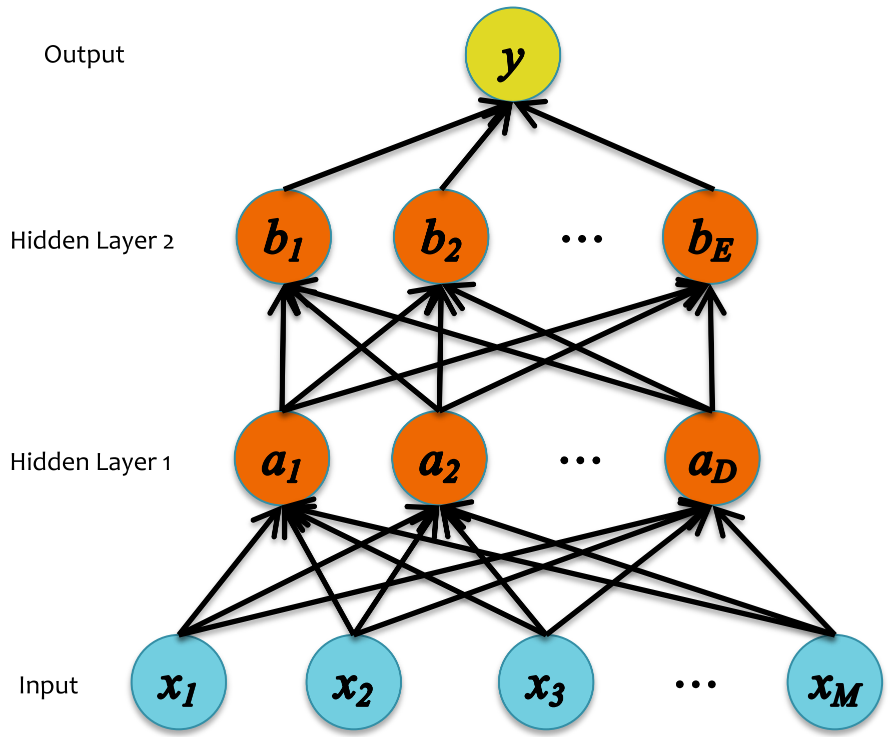
Deeper Networks

Q: How many layers should we use?



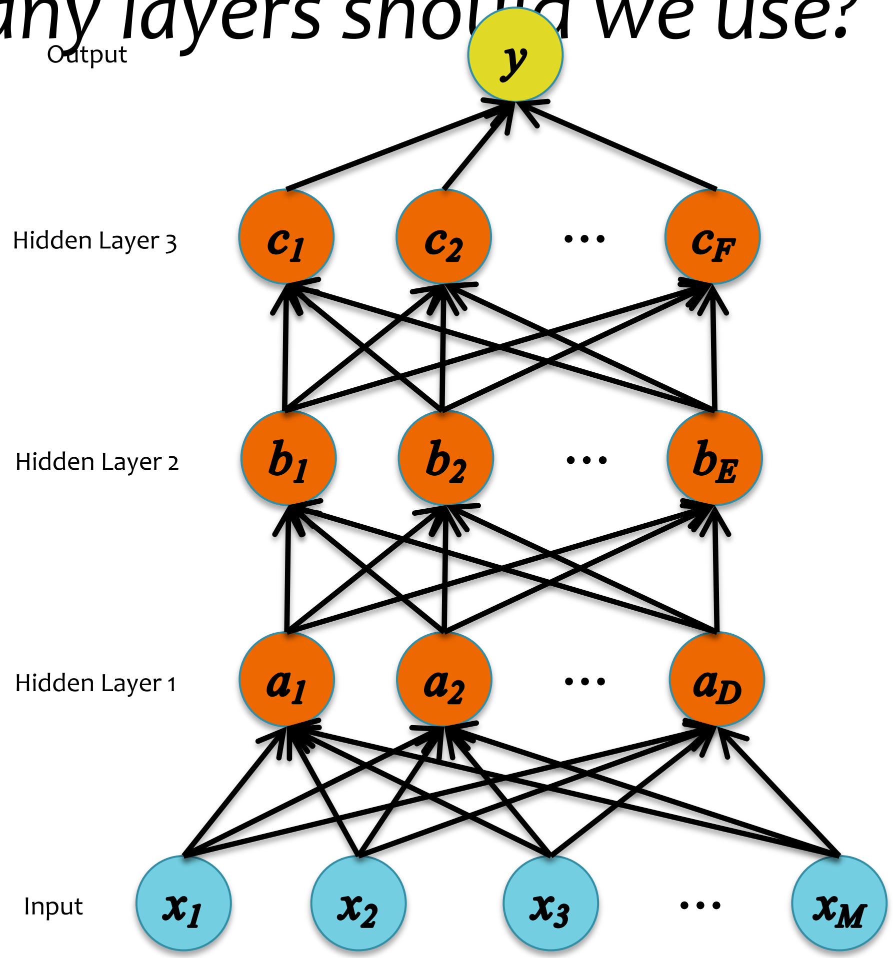
Deeper Networks

Q: How many layers should we use?



Deeper Networks

Q: How many layers should we use?



Deeper Networks

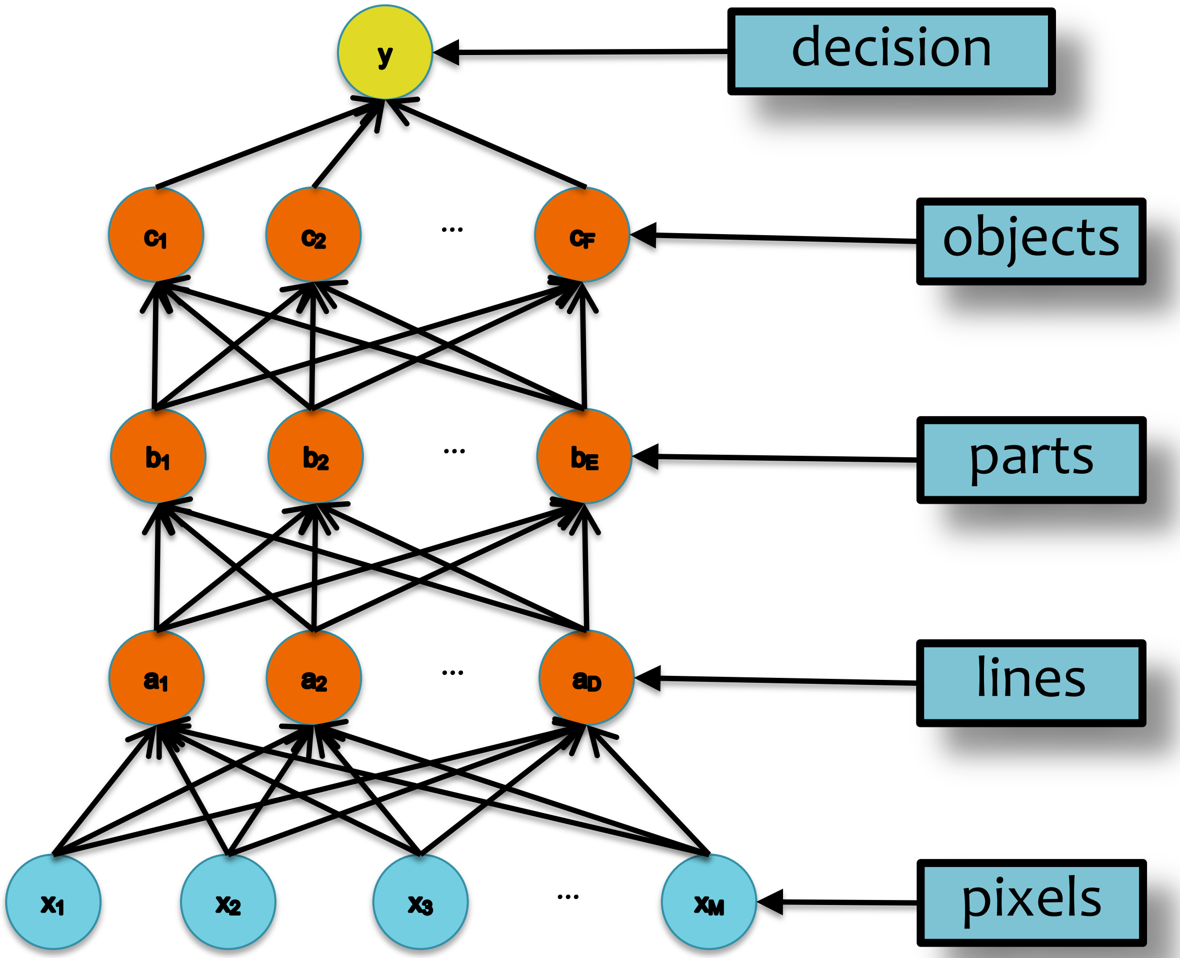
Q: How many layers should we use?

- **Theoretical answer:**
  - A neural network with 1 hidden layer is a **universal function approximator**
  - Cybenko (1989): For any continuous function $g(\mathbf{x})$, there exists a 1-hidden-layer neural net $h_\theta(\mathbf{x})$ s.t. $|\, h_\theta(\mathbf{x}) - g(\mathbf{x}) \,| < \epsilon$ for all $\mathbf{x}$, assuming sigmoid activation functions

    for any epsilon
- **Empirical answer:**
  - Before 2006: "Deep networks (e.g. 3 or more hidden layers) are too hard to train"
  - After 2006: "Deep networks are easier to train than shallow networks (e.g. 2 or fewer layers) for many problems"

    Big caveat: You need to know and use the right tricks.

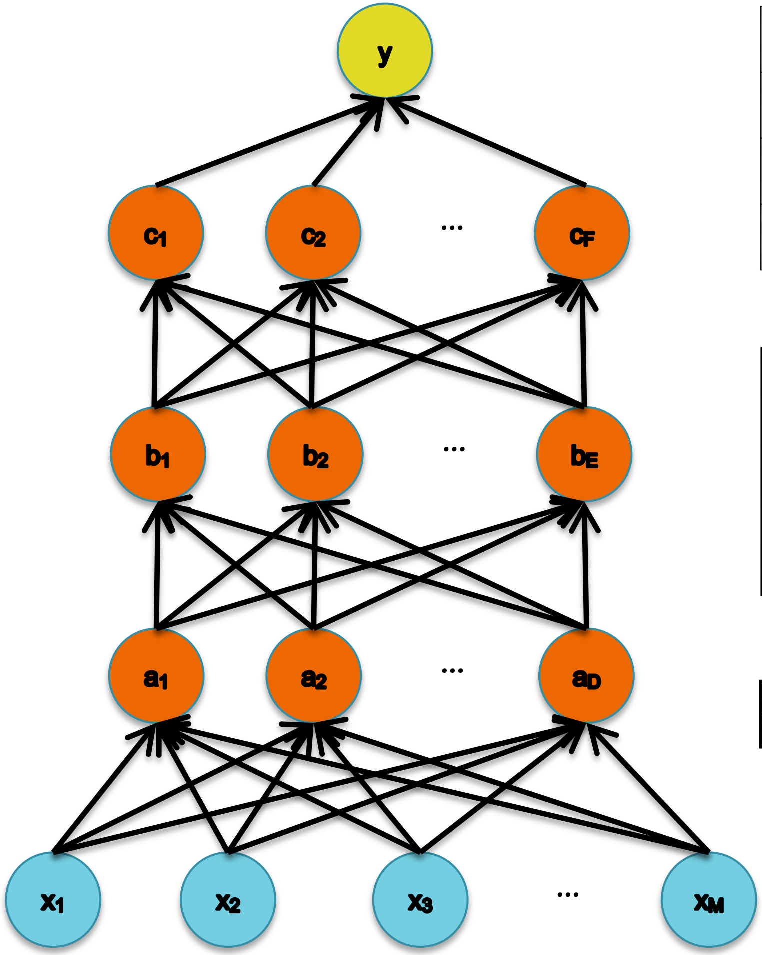
Feature Learning

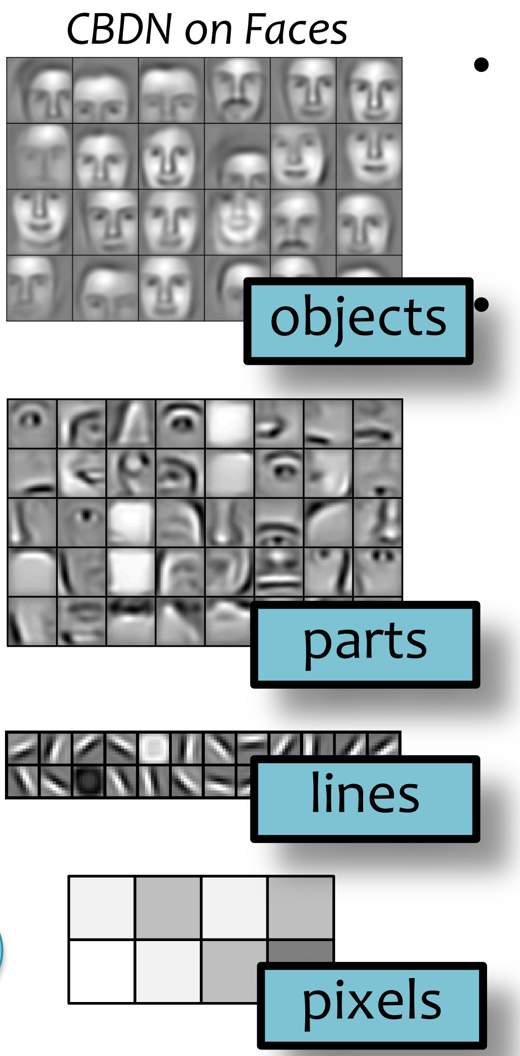


- **Traditional feature engineering:** build up levels of abstraction by hand
- **Deep networks** (e.g. convolution networks): learn the increasingly higher levels of abstraction from data
  - each layer is a learned feature representation
  - sophistication increases in higher layers

Figures from Lee et al. (ICML 2009)

Feature Learning





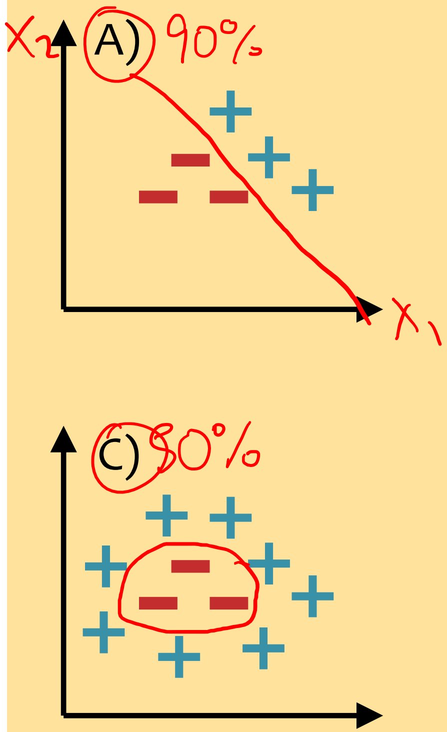
- **Traditional feature engineering:** build up levels of abstraction by hand
- **Deep networks** (e.g. convolution networks): learn the increasingly higher levels of abstraction from data
  – each layer is a learned feature representation
  – sophistication increases in higher layers

Figures from Lee et al. (ICML 2009)

Neural Network Errors

1

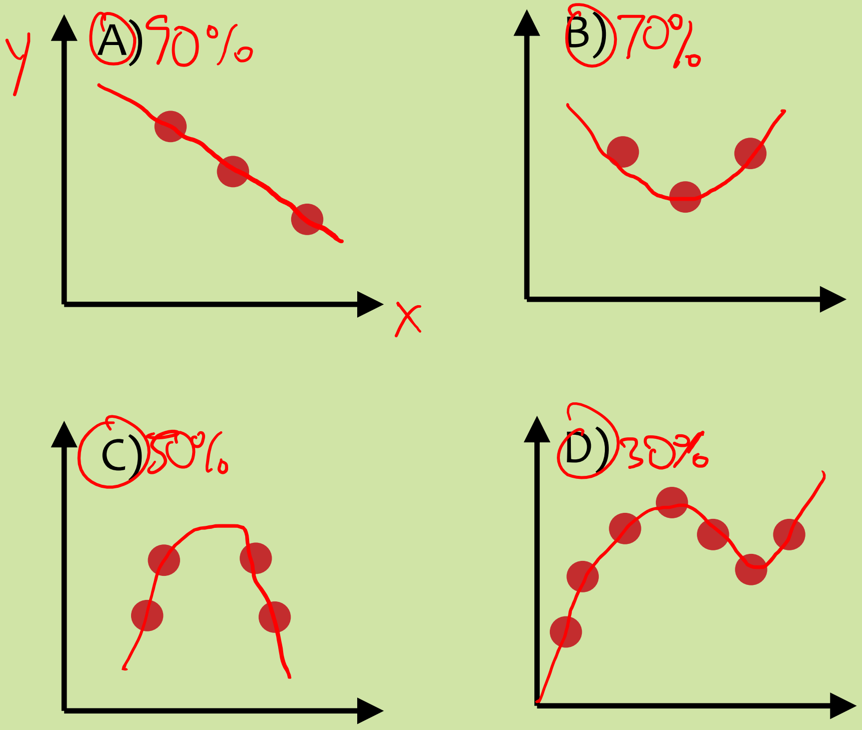
Question X: For which of the datasets below does there exist a one-hidden layer neural network that achieves zero *classification* error? **Select all that apply.**



E=toxic

2

Question Y: For which of the datasets below does there exist a one-hidden layer neural network for *regression* that achieves *nearly* zero MSE? **Select all that apply.**



E=toxic

Neural Networks Objectives

You should be able to…

- Explain the biological motivations for a neural network
- Combine simpler models (e.g. linear regression, binary logistic regression, multinomial logistic regression) as components to build up feed-forward neural network architectures
- Explain the reasons why a neural network can model nonlinear decision boundaries for classification
- Compare and contrast feature engineering with learning features
- Identify (some of) the options available when designing the architecture of a neural network
- Implement a feed-forward neural network

Computing Gradients

APPROACHES TO DIFFERENTIATION

Background

A Recipe for Machine Learning

1. Given training data:

$$\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^N$$

2. Choose each of these:

– Decision function

$$\hat{\boldsymbol{y}} = f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$$

– Loss function

$$\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}_i) \in \mathbb{R}$$

3. Define goal:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^N \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

4. Train with SGD:

(take small steps opposite the gradient)

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta_t \nabla \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

Background

A Recipe for

Gradients

Backpropagation can compute this gradient!

And it's a **special case of a more general algorithm** called reverse-mode automatic differentiation that can compute the gradient of any differentiable function efficiently!

1. Given training dat

$$\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^N$$

2. Choose each of t

– Decision functio

$$\hat{y} = f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$$

– Loss function

$$\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}_i) \in \mathbb{R}$$

$\boldsymbol{y}_i)$

opp site the gradient)

$$\boldsymbol{\theta}^{(t} \quad {}^{(t)} - \eta_t \nabla \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

Approaches to Differentiation

- **Question 1:**
  When can we compute the gradients for an arbitrary neural network?

- **Question 2:**
  When can we make the gradient computation efficient?

Given $f : \mathbb{R}^A \rightarrow \mathbb{R}^B, f(\mathbf{x})$

Compute $\frac{\partial f(\mathbf{x})_i}{\partial x_j} \forall i, j$

Approaches to Differentiation

1. Finite Difference Method

- Pro: Great for testing implementations of backpropagation
- Con: Slow for high dimensional inputs / outputs
- Required: Ability to call the function f(**x**) on any input **x**

2. Symbolic Differentiation

- Note: The method you learned in high-school
- Note: Used by Mathematica / Wolfram Alpha / Maple
- Pro: Yields easily interpretable derivatives
- Con: Leads to exponential computation time if not carefully implemented
- Required: Mathematical expression that defines f(**x**)

Given $f : \mathbb{R}^A \rightarrow \mathbb{R}^B, f(\mathbf{x})$

Compute $\frac{\partial f(\mathbf{x})_i}{\partial x_j} \forall i, j$

Approaches to Differentiation

For ML: B = 1, objective

A = # parameters

3. Automatic Differentiation – Reverse Mode

- Note: Called *Backpropagation* when applied to Neural Nets
- Pro: Computes partial derivatives of one output $f(\mathbf{x})_i$ with respect to all inputs x_j in time ~~proportional~~ to computation of $f(\mathbf{x})$ polynomial in
- Con: Slow for high dimensional outputs (e.g. vector-valued functions)
- Required: Algorithm for computing $f(\mathbf{x})$

4. Automatic Differentiation - Forward Mode

- Note: Easy to implement. Uses dual numbers.
- Pro: Computes partial derivatives of all outputs $f(\mathbf{x})_i$ with respect to one input x_j in time proportional to computation of $f(\mathbf{x})$
- Con: Slow for high dimensional inputs (e.g. vector-valued $\mathbf{x}$)
- Required: Algorithm for computing $f(\mathbf{x})$

THE FINITE DIFFERENCE METHOD

Finite Difference Method

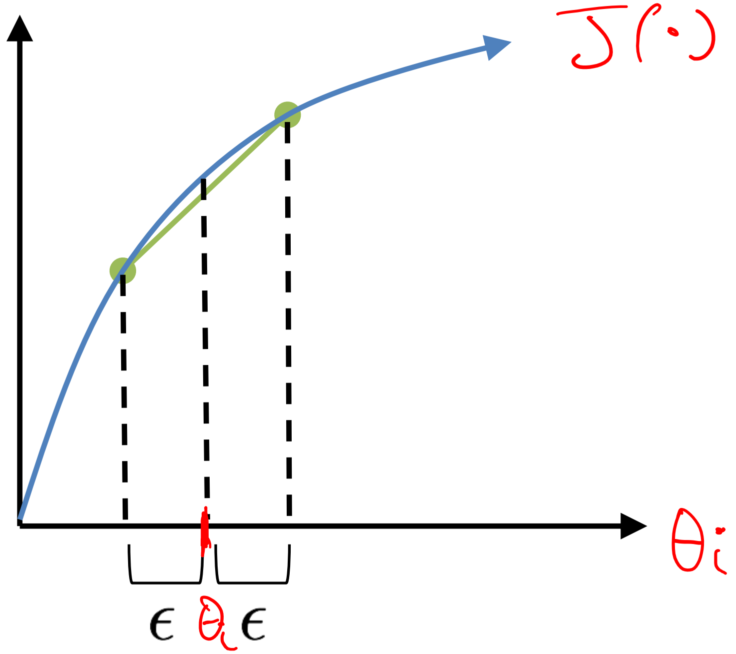
The *centered* finite difference approximation is:

$$\frac{\partial}{\partial \theta_i} J(\boldsymbol{\theta}) \approx \frac{(J(\boldsymbol{\theta} + \epsilon \cdot \boldsymbol{d}_i) - J(\boldsymbol{\theta} - \epsilon \cdot \boldsymbol{d}_i))}{2\epsilon} \tag{1}$$

where $\boldsymbol{d}_i$ is a 1-hot vector consisting of all zeros except for the ith entry of $\boldsymbol{d}_i$, which has value 1.

Notes:

- Suffers from issues of floating point precision, in practice
- Typically only appropriate to use on small examples with an appropriately chosen epsilon



Training

Differentiation Quiz

Speed Quiz:
2 minute time limit.

Differentiation Quiz #1:

Suppose x = 2 and z = 3, what are dy/dx and dy/dz for the function below? **Round your answer to the nearest integer**.

$$y = \exp(xz) + \frac{xz}{\log(x)} + \frac{\sin(\log(x))}{xz}$$

Q3

Answer: *Answers below are in the form [dy/dx, dy/dz]*

A. [42, -72]
B. [72, -42]
C. [100, 127]
D. [127, 100]
E. [1208, 810]
F. [810, 1208]
G. [1505, 94]
H. [94, 1505]

I= toxic

Differentiation Quiz

Differentiation Quiz #2:

A neural network with 2 hidden layers can be written as:

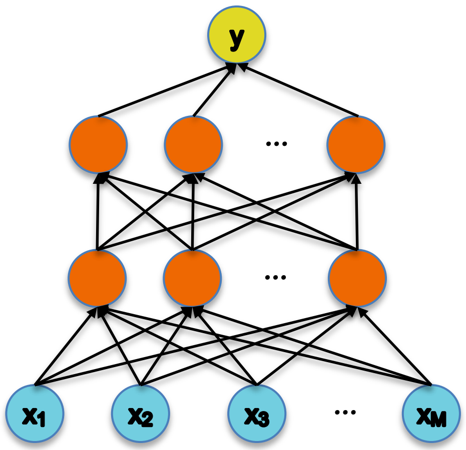
$$y = \sigma(\boldsymbol{\beta}^T \sigma((\boldsymbol{\alpha}^{(2)})^T \sigma((\boldsymbol{\alpha}^{(1)})^T \mathbf{x}))$$

where $y \in \mathbb{R}$, $\mathbf{x} \in \mathbb{R}^{D^{(0)}}$, $\boldsymbol{\beta} \in \mathbb{R}^{D^{(2)}}$ and $\boldsymbol{\alpha}^{(i)}$ is a $D^{(i)} \times D^{(i-1)}$ matrix. Nonlinear functions are applied elementwise:

$$\sigma(\mathbf{a}) = [\sigma(a_1), \ldots, \sigma(a_K)]^T$$

Let σ be sigmoid: $\sigma(a) = \frac{1}{1+exp-a}$

What is $\frac{\partial y}{\partial \beta_j}$ and $\frac{\partial y}{\partial \alpha_j^{(i)}}$ for all i, j.



THE CHAIN RULE OF CALCULUS

Training

Chain Rule

Definition 1:

Given

$$y = f(u)$$
$$u = g(x)$$

Computation Graph

Chain Rule

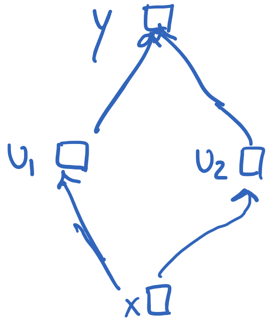
$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial u}\frac{\partial u}{\partial x}$$

Definition 2:

Given

$$y = f(u_1, u_2)$$
$$u_2 = g_2(x)$$
$$u_1 = g_1(x)$$

Computation Graph



Chain Rule

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial u_1}\frac{\partial u_1}{\partial x} + \frac{\partial y}{\partial u_2}\frac{\partial u_2}{\partial x}$$

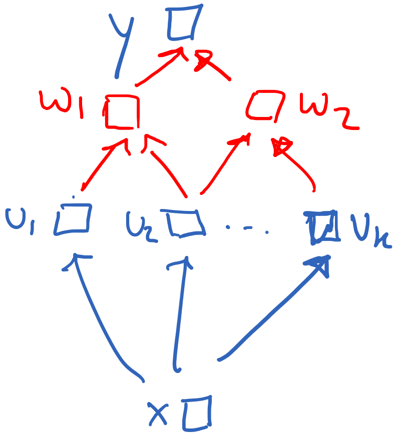
Definition 3:

Given

$$y = f(\vec{u})$$
$$\vec{u} = g(x)$$

$y \in \mathbb{R}$
$\vec{u} \in \mathbb{R}^K$
$x \in \mathbb{R}$

Computation Graph



Chain Rule

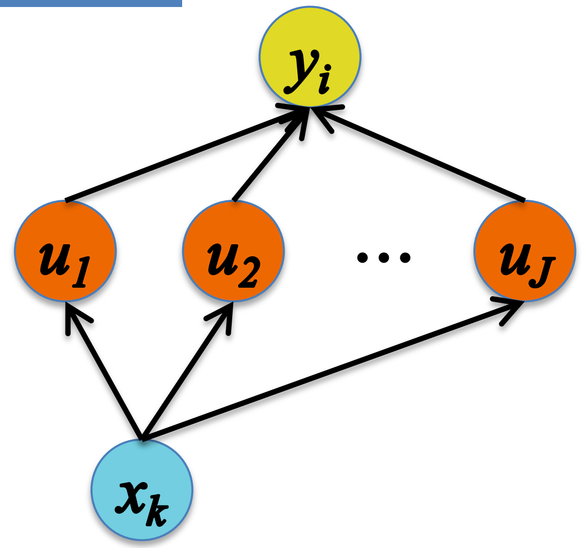
$$\frac{\partial y}{\partial x} = \sum_{k=1}^{K} \frac{\partial y}{\partial u_k}\frac{\partial u_k}{\partial x}$$

Chain Rule

Given: $\boldsymbol{y} = g(\boldsymbol{u})$ and $\boldsymbol{u} = h(\boldsymbol{x})$.

Chain Rule:

$$\frac{dy_i}{dx_k} = \sum_{j=1}^{J} \frac{dy_i}{du_j} \frac{du_j}{dx_k}, \quad \forall i, k$$



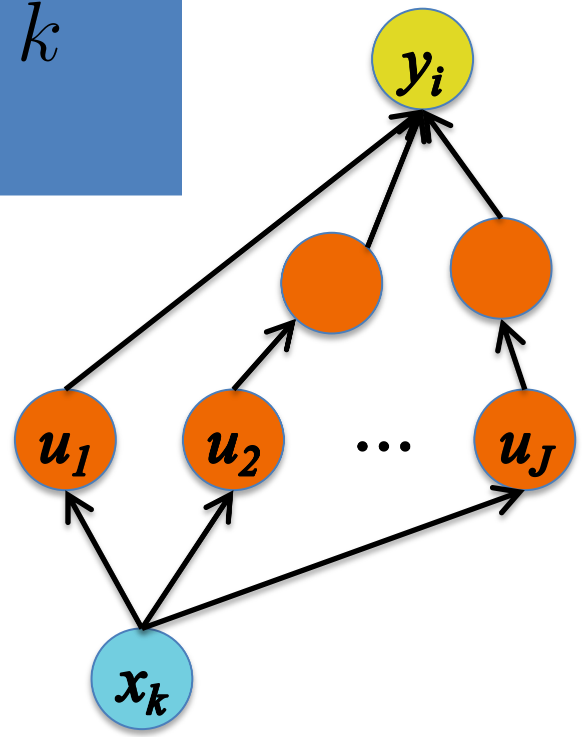
Chain Rule

Given: $\boldsymbol{y} = g(\boldsymbol{u})$ and $\boldsymbol{u} = h(\boldsymbol{x})$.

Chain Rule:

$$\frac{dy_i}{dx_k} = \sum_{j=1}^{J} \frac{dy_i}{du_j} \frac{du_j}{dx_k}, \quad \forall i, k$$

Backpropagation is just repeated application of the **chain rule** from Calculus 101.



Algorithm

FORWARD COMPUTATION FOR A COMPUTATION GRAPH

Backpropagation

Whiteboard

– From equation to forward computation
– Representing a simple function as a computation graph

Differentiation Quiz #1:

Suppose x = 2 and z = 3, what are dy/dx and dy/dz for the function below? **Round your answer to the nearest integer**.

$$y = \exp(xz) + \frac{xz}{\log(x)} + \frac{\sin(\log(x))}{xz}$$

Backpropagation

Speed Quiz:
2 minute time limit.

Differentiation Quiz #1:

Suppose x = 2 and z = 3, what are dy/dx and dy/dz for the function below? **Round your answer to the nearest integer**.

$$y = \exp(xz) + \frac{xz}{\log(x)} + \frac{\sin(\log(x))}{xz}$$

Now let's solve this in a different way!

Given: $$y = \exp(xz) + \frac{xz}{\log(x)} + \frac{\sin(\log(x))}{xz}$$

Forward Computation:

Given $x=2$, $z=3$

$a = xz$

$b = \log(x)$

$c = \sin(b)$

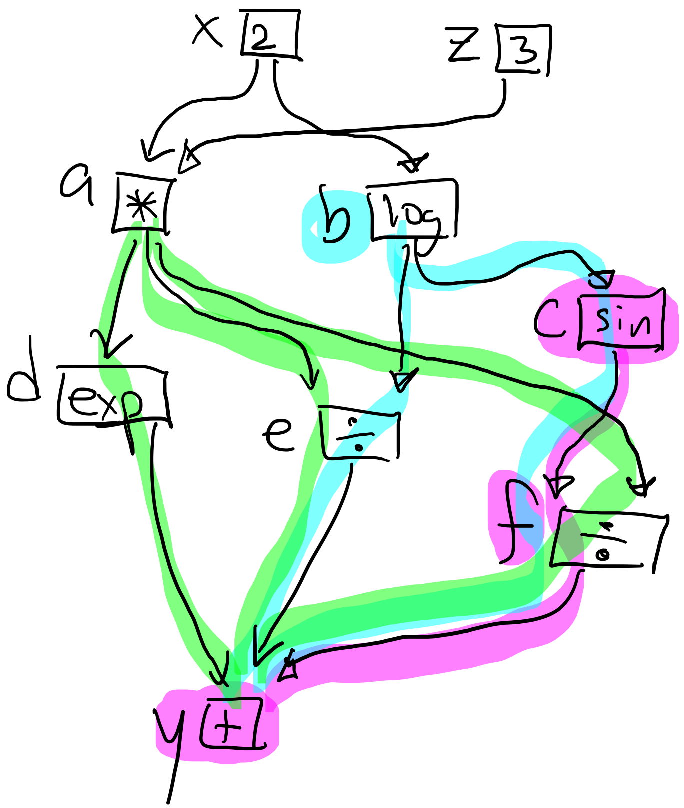
$d = \exp(a)$

$e = a/b$

$f = c/a$

$y = d+e+f$

Computation Graph:



Backward Computation

$g_y = \frac{\partial y}{\partial y} = 1$

$g_f = \frac{\partial y}{\partial f} = 1$, $g_e = \frac{\partial y}{\partial e} = 1$, $g_d = \frac{\partial y}{\partial d} = 1$

$g_c = \frac{\partial y}{\partial c} = \frac{\partial y}{\partial f}\frac{\partial f}{\partial c} = (g_f)\left(\frac{1}{a}\right)$

$g_b = \frac{\partial y}{\partial b} = \frac{\partial y}{\partial c}\frac{\partial c}{\partial b} + \frac{\partial y}{\partial e}\frac{\partial e}{\partial b} = (g_c)(\cos(b)) + (g_e)\left(-\frac{a}{b^2}\right)$

$g_a = \frac{\partial y}{\partial a} = \frac{\partial y}{\partial d}\frac{\partial d}{\partial a} + \frac{\partial y}{\partial e}\frac{\partial e}{\partial a} + \frac{\partial y}{\partial f}\frac{\partial f}{\partial a}$

$= (g_d)(\exp(a)) + (g_e)\left(\frac{1}{b}\right) + (g_f)\left(-\frac{c}{a^2}\right)$

$g_x = \frac{\partial y}{\partial x} = (g_a)(z) + (g_b)\left(\frac{1}{x}\right)$

$g_z = \frac{\partial y}{\partial z} = (g_x)(x)$

Backpropagation

Updates for Backpropagation:

$$g_x = \frac{\partial y}{\partial x} = \sum_{k=1}^{K} \frac{\partial y}{\partial u_k} \frac{\partial u_k}{\partial x}$$

$$= \sum_{k=1}^{K} g_{u_k} \frac{\partial u_k}{\partial x}$$

Backprop is efficient b/c of reuse in the forward pass and the backward pass.