



10-301/10-601 Introduction to Machine Learning

Machine Learning Department
School of Computer Science
Carnegie Mellon University

Neural Networks + Backpropagation

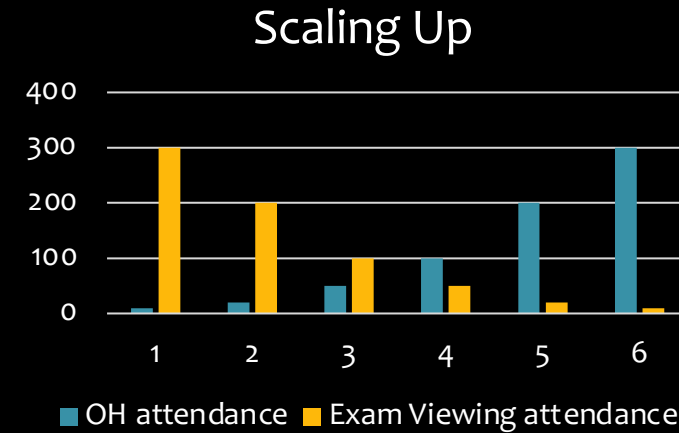
Matt Gormley & Henry Chai

Lecture 12

Feb. 19, 2025

Reminders

- **Post-Exam Followup:**
 - Exam Viewing
 - Exit Poll: Exam 1
 - Grade Summary 1
- **Homework 4: Logistic Regression**
 - Out: Mon, Feb 17
 - Due: Wed, Feb 26 at 11:59pm



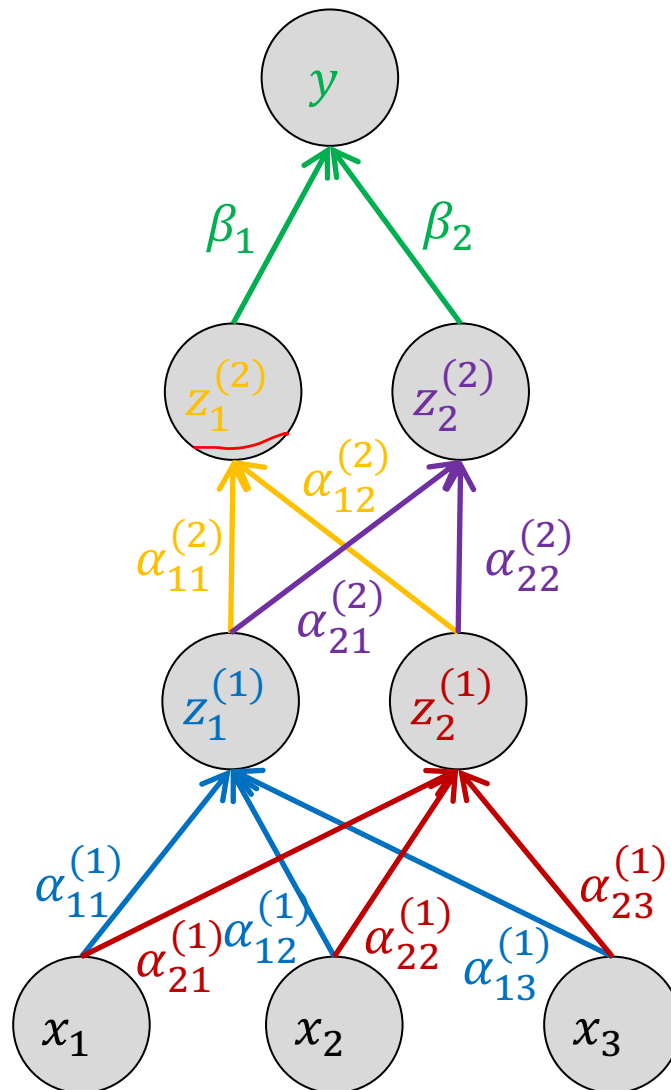
Neural Network Architectures

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



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

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

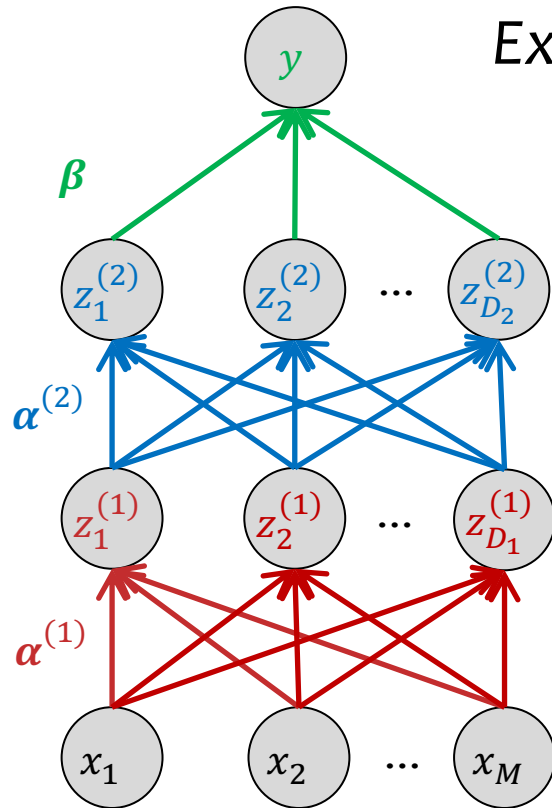
$$z_1^{(2)} = \sigma(\alpha_{11}^{(2)}z_1^{(1)} + \alpha_{12}^{(2)}z_2^{(1)} + \alpha_{10}^{(2)})$$

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

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

Neural Network (Matrix Form)

Example: Arbitrary Feed-forward Neural Network



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

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

$$\alpha^{(2)} \in \mathbb{R}^{D_1 \times D_2}$$

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

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

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

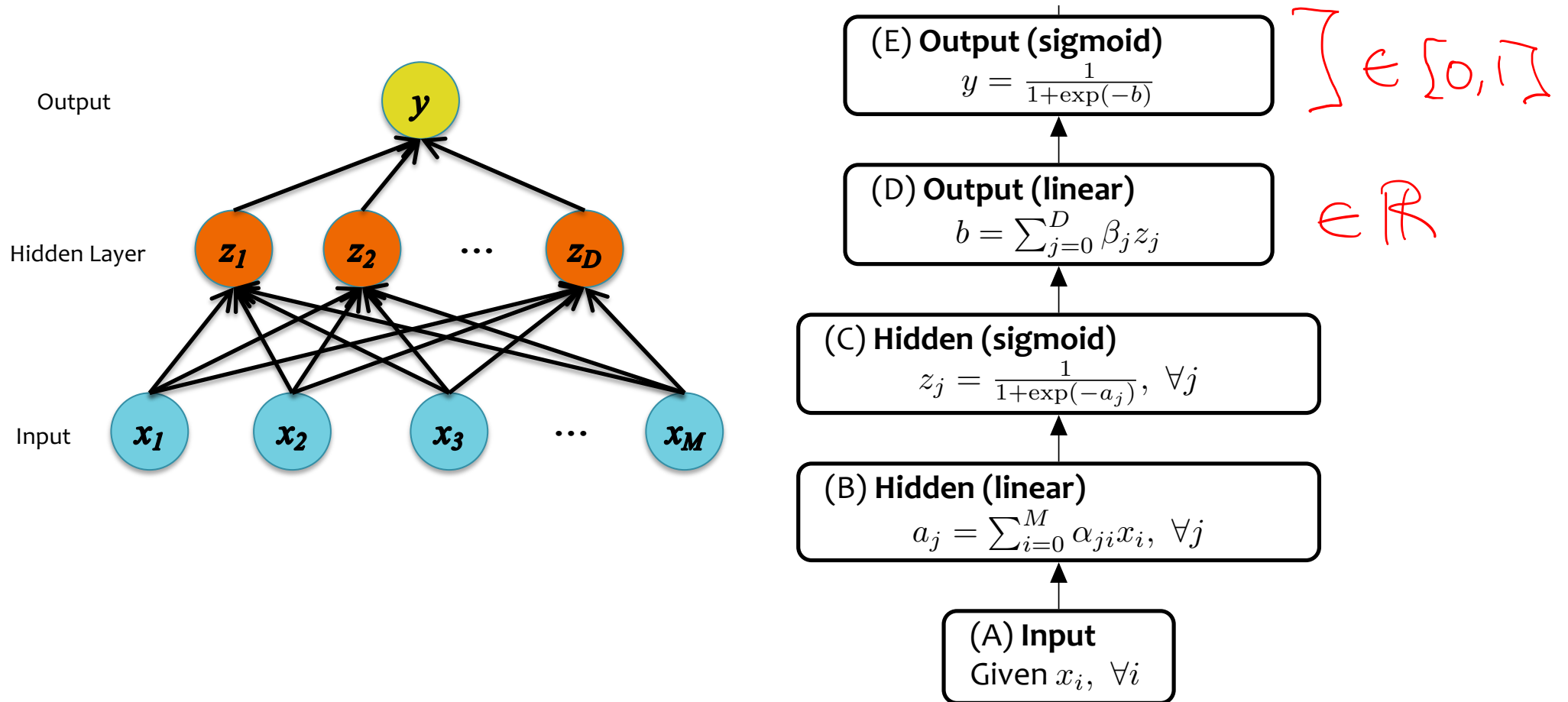
$$y = \sigma((\beta)^T \mathbf{z}^{(2)} + \beta_0)$$

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

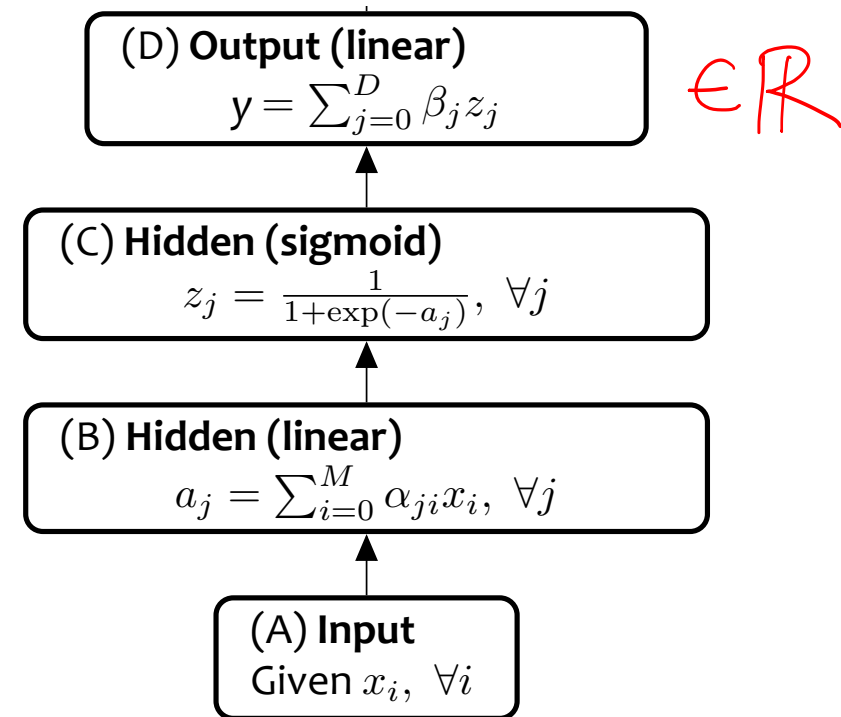
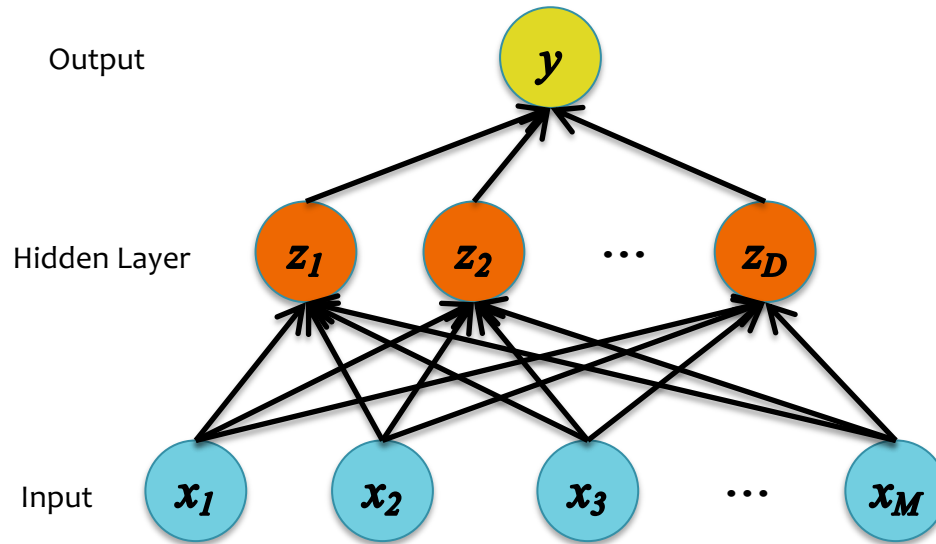
$$\mathbf{z}^{(1)} = \sigma((\alpha^{(1)})^T \mathbf{x} + \mathbf{b}^{(1)})$$

LOSS FUNCTIONS & OUTPUT LAYERS

Neural Network for Classification



Neural Network for Regression



Objective Functions for NNs nonconvex

1. Quadratic Loss:

- the same objective as Linear Regression
- i.e. mean squared error

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

true label predicted

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 $[0,1]$, $y^{(i)} \in \{0,1\}$

$$J = \ell_{CE}(y, y^{(i)}) = -(y^{(i)} \log(y) + (1 - y^{(i)}) \log(1 - y))$$
$$\frac{dJ}{dy} = -\left(y^{(i)} \frac{1}{y} + (1 - y^{(i)}) \frac{1}{y - 1}\right)$$

Cross-entropy vs. Quadratic loss

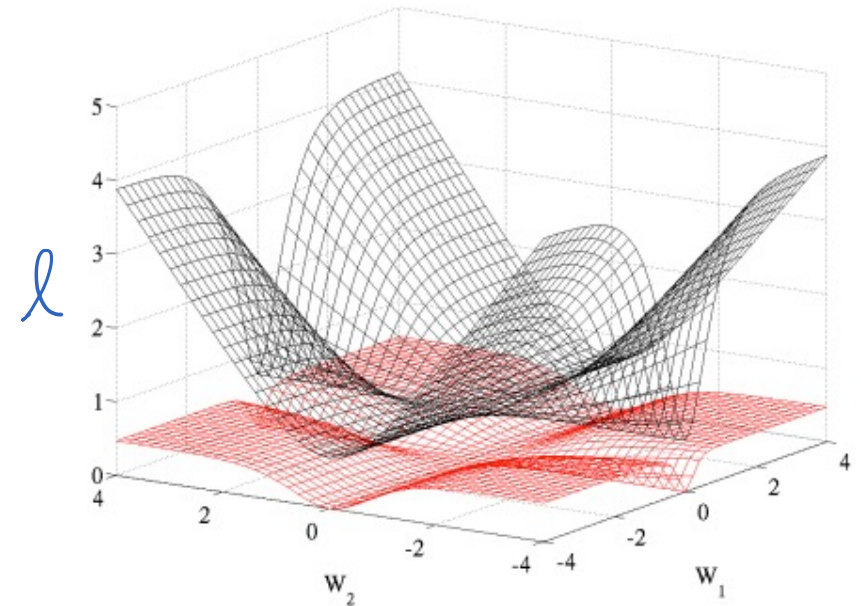
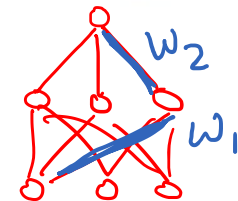


Figure 5: Cross entropy (black, surface on top) and quadratic (red, bottom surface) cost as a function of two weights (one at each layer) of a network with two layers, W_1 respectively on the first layer and W_2 on the second, output layer.



Handwritten calculation for Softmax:

$$\vec{b} = [-3 \quad 2.7 \quad 0.5]$$

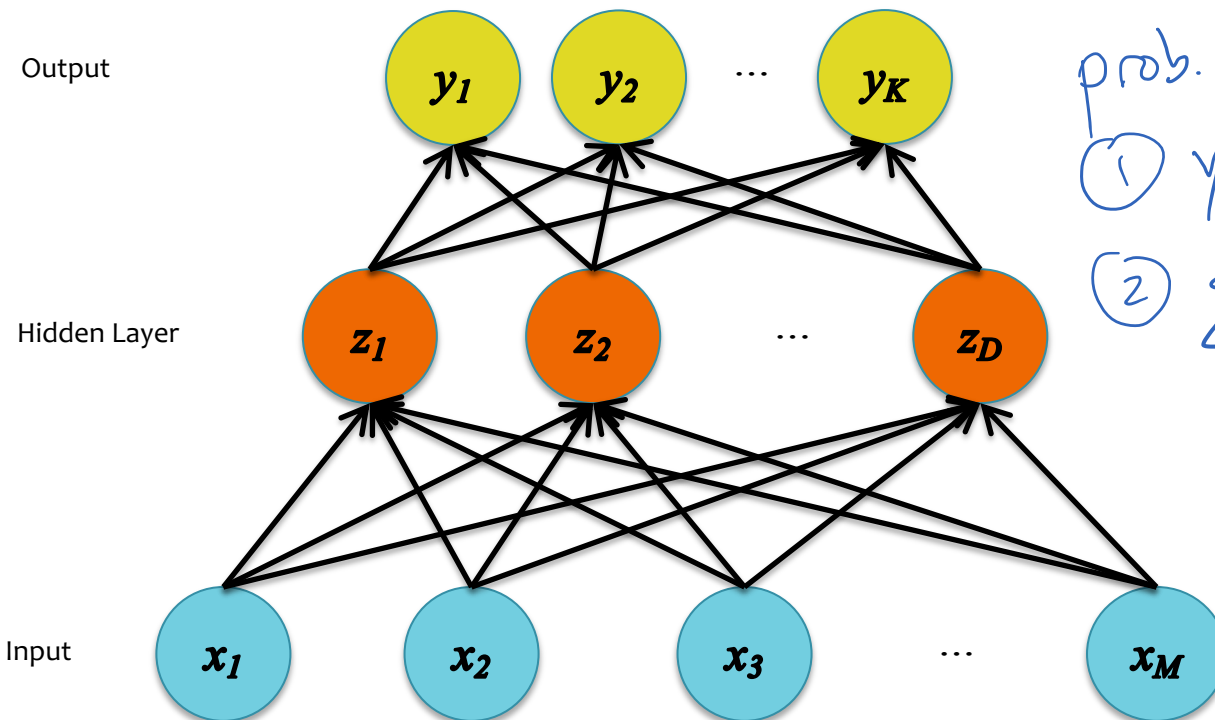
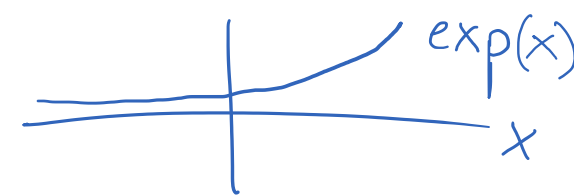
$$\exp(\vec{b}) = [0.0001 \quad 2.4 \quad 0.2]$$

$$\vec{y} = [0.0001 \quad 0.97 \quad 0.0001]$$

Softmax:

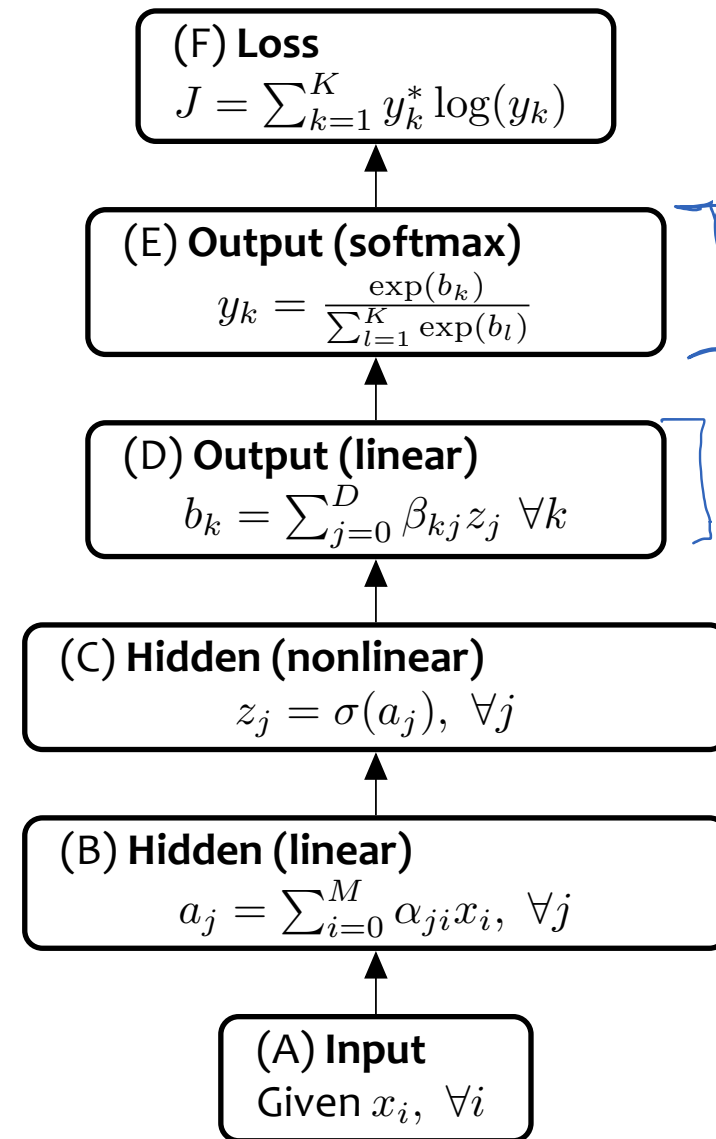
$$y_k = \frac{\exp(b_k)}{\sum_{l=1}^K \exp(b_l)}$$

Multiclass Output



prob. dist.

- $y_k \in [0, 1] \forall k$
- $\sum_{k=1}^K y_k = 1$



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:

$$\mathbf{y}^{(i)} = \begin{array}{|c|c|c|c|c|c|c|c|} \hline 0 & 0 & 0 & 1 & 0 & 0 & \dots & 0 \\ \hline 1 & 2 & 3 & 4 & 5 & 6 & \dots & K \\ \hline \end{array}$$

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

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

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

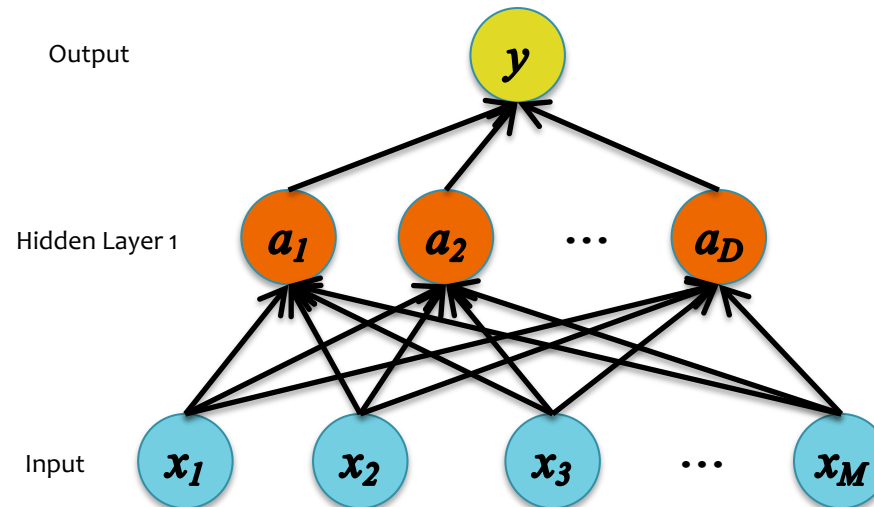
- Then we can write the log-likelihood of a single training example $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$ as:

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

$$\begin{aligned} &= - (0 \log(y_1) + 0 \log(y_2) + 0 \log(y_3) + 1 \log(y_4) + \dots + 0 \log(y_K)) \\ &= -\log(y_4) \end{aligned}$$

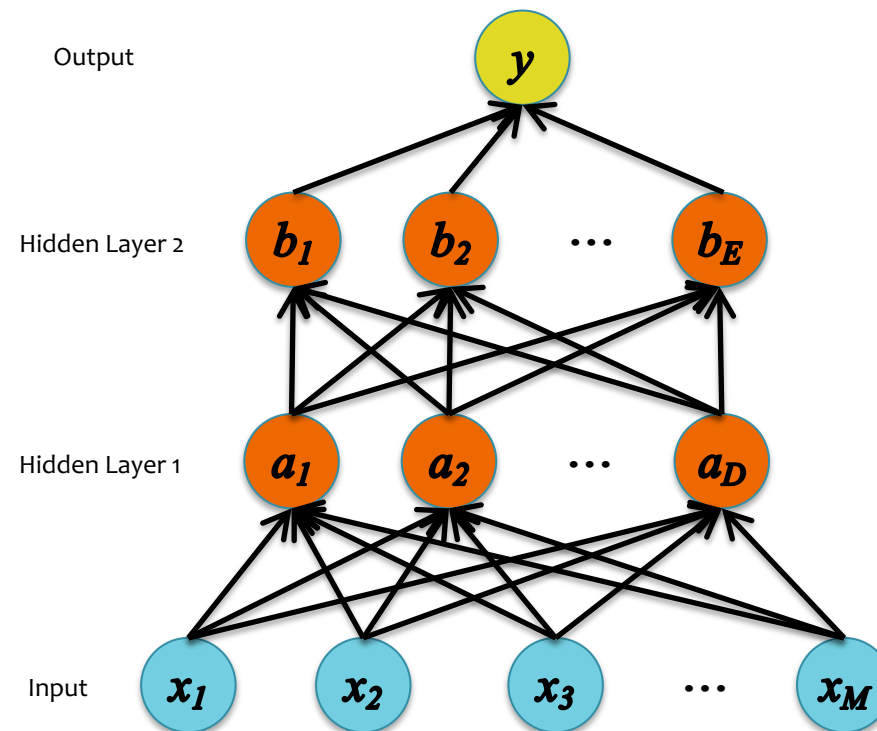
Deeper Networks

Q: How many layers should we use?



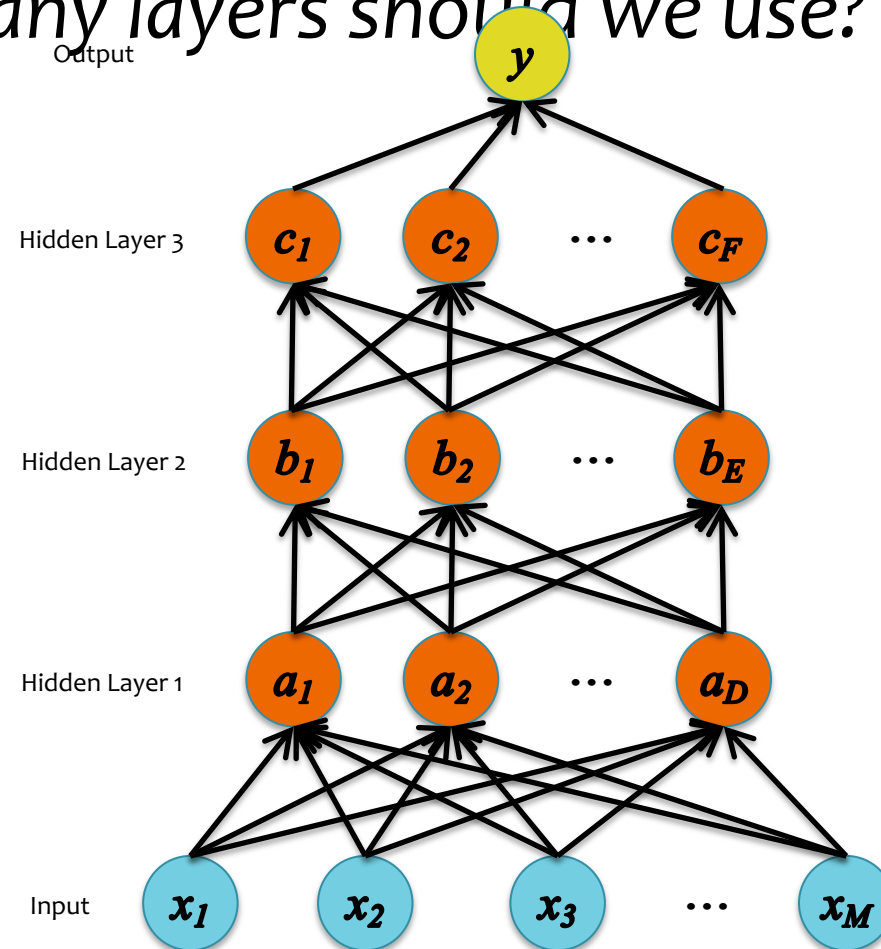
Deeper Networks

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

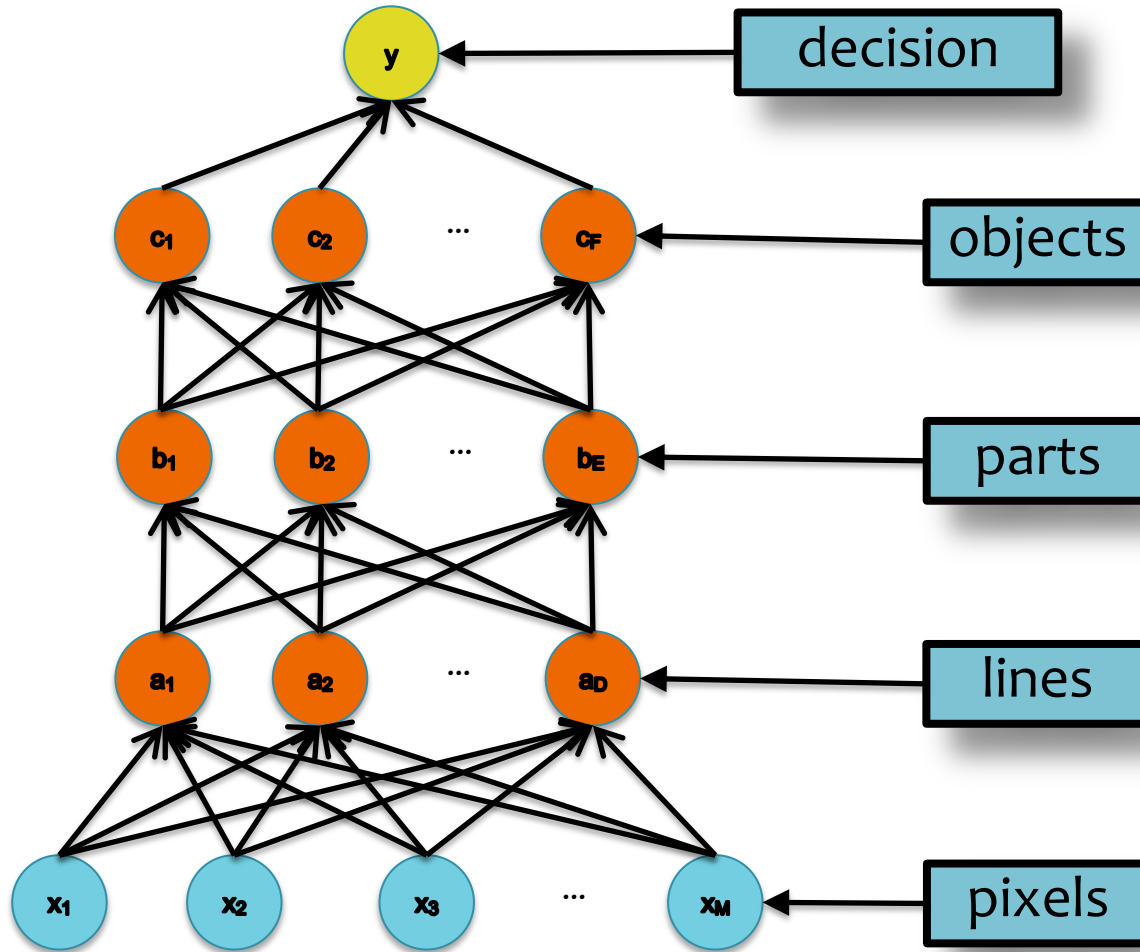
↑ for any ϵ

- **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”

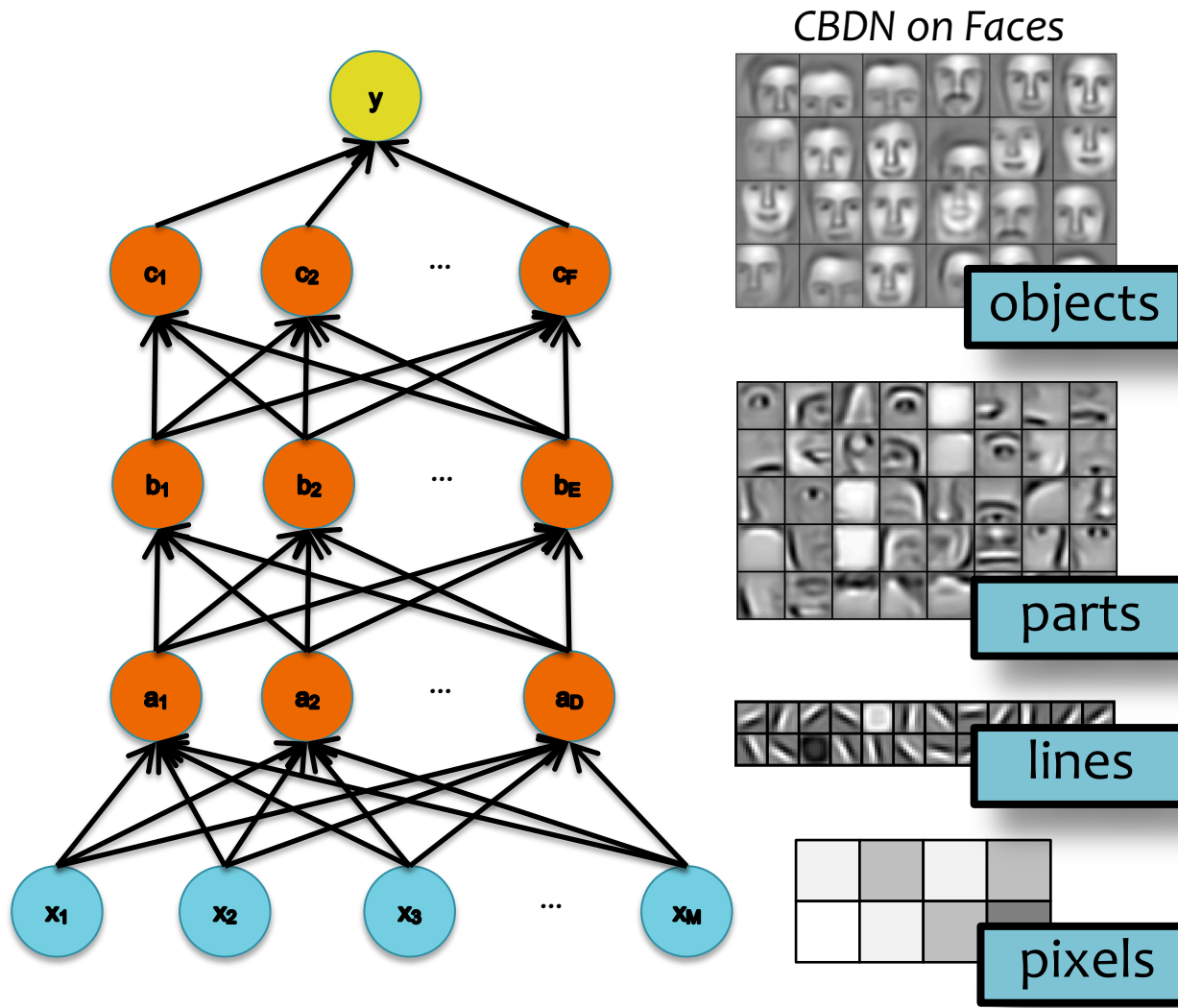
Big caveat: You need to know and use the right tricks.

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

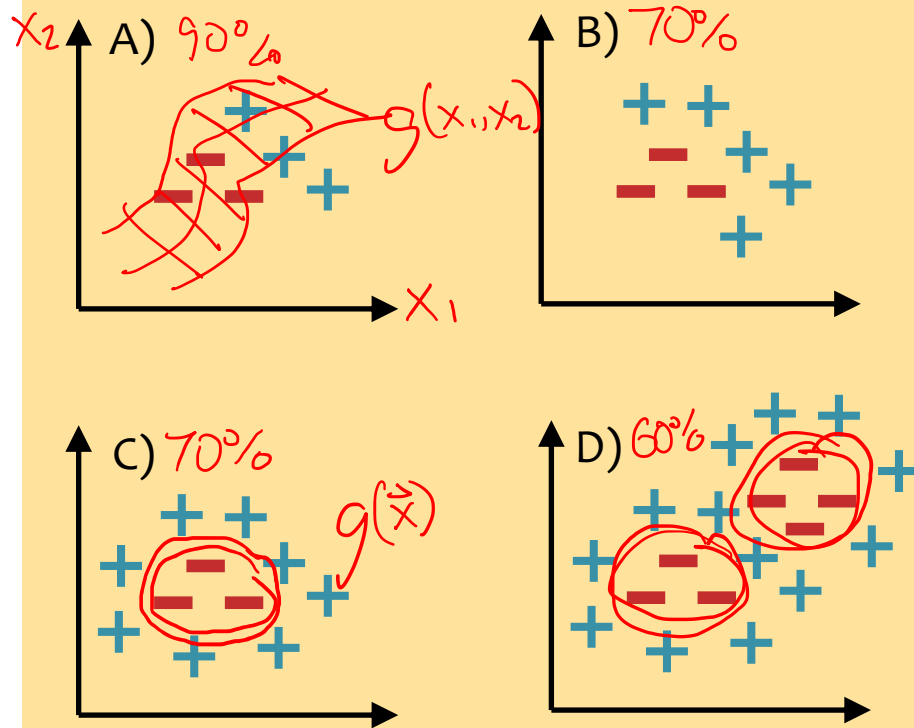
Feature Learning



- **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

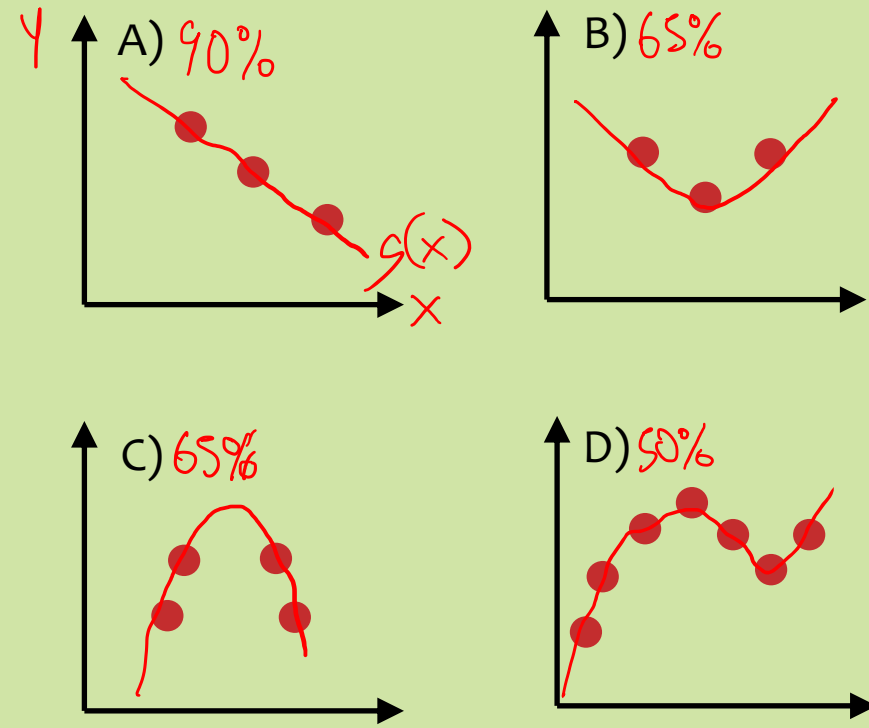
Neural Network Errors

Poll Question 1: For which of the datasets below does there exist a one-hidden layer neural network that achieves zero classification error? **Select all that apply.**



$E = \text{toxic}$

Poll Question 2: 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.**



$E = \text{toxic}$

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

A Recipe for Machine Learning

1. Given training data:

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

2. Choose each of these:

- Decision function

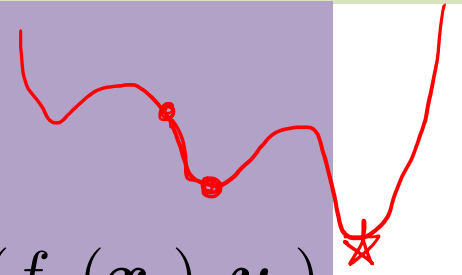
$$\hat{\mathbf{y}} = f_{\boldsymbol{\theta}}(\mathbf{x}_i)$$

- Loss function

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

3. Define goal:

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



4. Train with SGD:

(take small steps
opposite the gradient)

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

Gradients

1. Given training data

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

2. Choose each of the

- Decision function

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

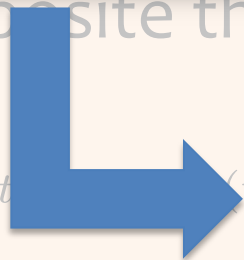
- Loss function

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

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!

opposite the gradient)


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

- **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 $\frac{\partial f(\mathbf{x})_i}{\partial x_j} \forall i, j$
 $A \times B$

Approaches to Differentiation

For ML: $f \rightarrow \mathcal{J}(\vec{\theta})$ $A=1$, $B = \# \text{ parameters}$

1. Finite Difference Method

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

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 $f(\mathbf{x})$

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

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 polynomial in the computation time of $f(\mathbf{x})$
- 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 polynomial in the computation time 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

The centered finite difference approximation is:

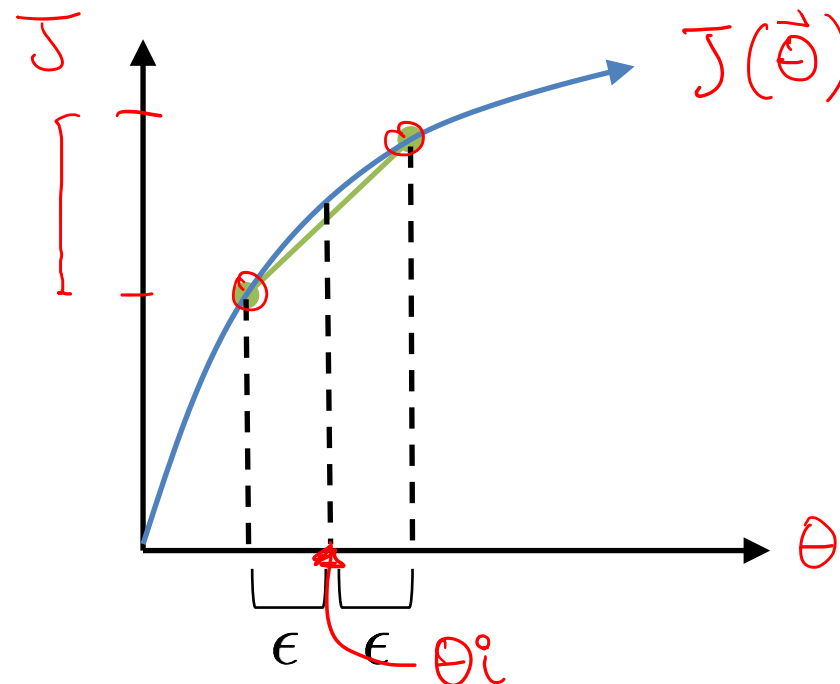
for ϵ small

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

where $\mathbf{d}_i$ is a 1-hot vector consisting of all zeros except for the i th entry of $\mathbf{d}_i$, 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



Differentiation Quiz

Speed Quiz:
2 minute time limit.

Poll Question 3: 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.**

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

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 #2:

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

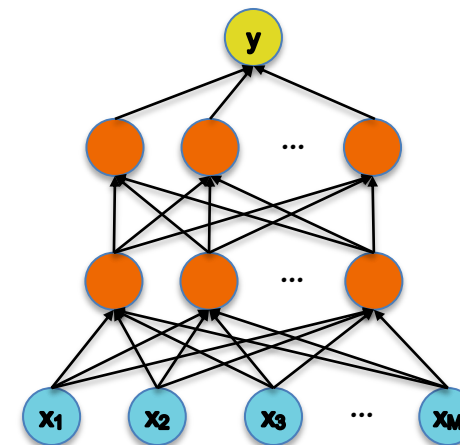
$$y = \sigma(\beta^T \sigma((\alpha^{(2)})^T \sigma((\alpha^{(1)})^T \mathbf{x})))$$

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

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

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

Definition 1:

$$y = f(u)$$

$$u = g(x)$$



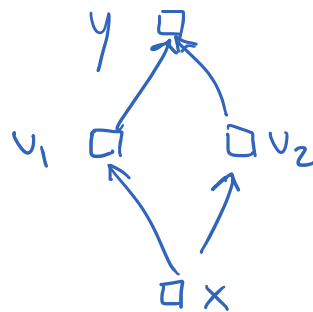
$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial u} \frac{\partial u}{\partial x}$$

Definition 2:

$$y = f(u_1, u_2)$$

$$u_2 = g_2(x)$$

$$u_1 = g_1(x)$$



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

Definition 3:

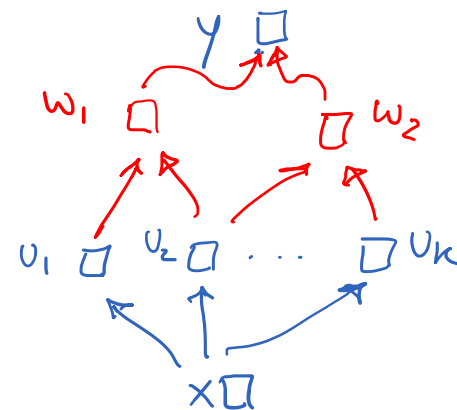
$$y = f(\vec{u})$$

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

$$y \in \mathbb{R}$$

$$\vec{u} \in \mathbb{R}^k$$

$$x \in \mathbb{R}$$

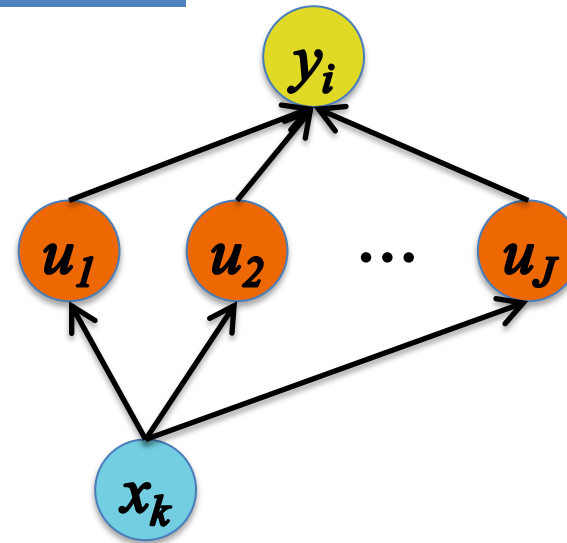


$$\frac{\partial y}{\partial x} = \sum_{k=1}^K \frac{\partial y}{\partial u_k} \frac{\partial u_k}{\partial x}$$

Given: $y = g(u)$ and $u = h(x)$.

Chain Rule:

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

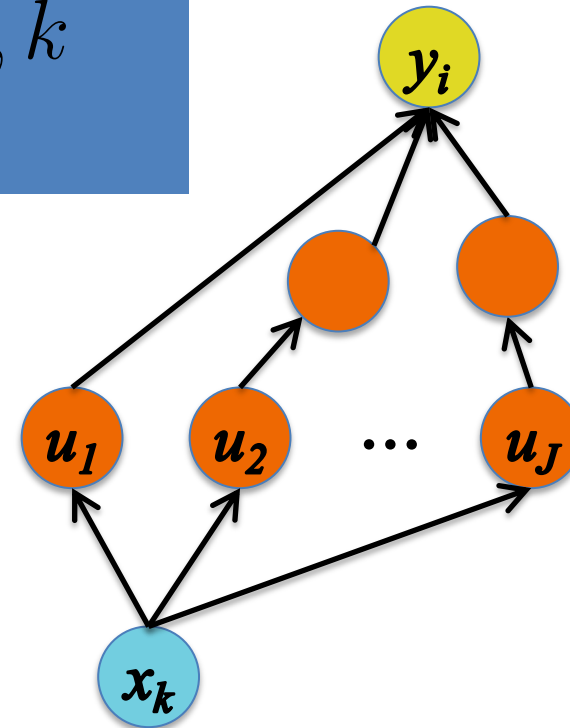


Given: $y = g(u)$ and $u = h(x)$.

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

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.**

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

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!

Speed Quiz:
2 minute time limit.

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

Forward Computation:

Computation Graph:

Backward Computation

Given $x = 2, z = 3$

$a = xz$

$b = \log(x)$

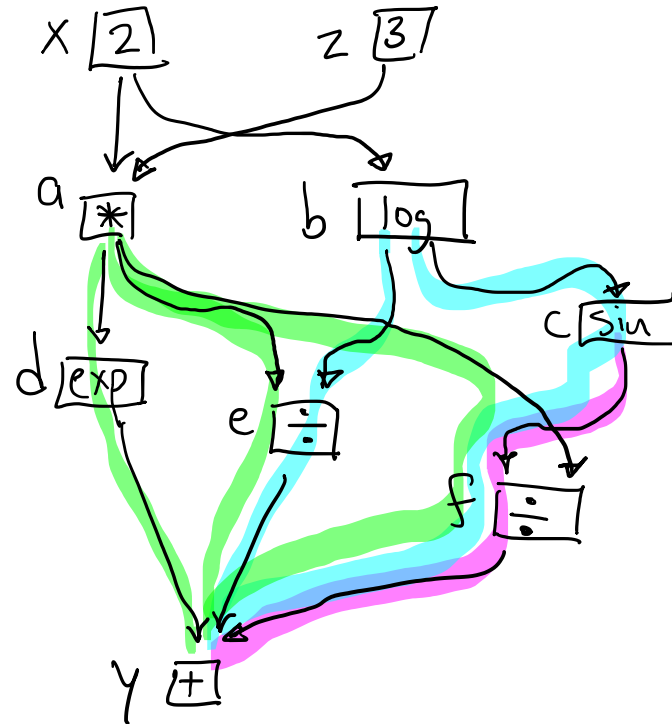
$c = \sin(b)$

$d = \exp(a)$

$e = a/b$

$f = c/a$

$y = d + e + f$



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

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

$g_c = \frac{\partial y}{\partial c} = \frac{\partial y}{\partial f} \frac{\partial f}{\partial c} = (g_f) \left(\frac{1}{a} \right) = -\frac{a}{b^2}$

$g_b = \frac{\partial y}{\partial b} = \frac{\partial y}{\partial c} \frac{\partial c}{\partial b} + \frac{\partial y}{\partial e} \frac{\partial e}{\partial b} = (g_c) (\cos(b)) + (g_e) \left(\frac{1}{b} \right)$

$g_a = \frac{\partial y}{\partial d} \frac{\partial d}{\partial a} + \frac{\partial y}{\partial e} \frac{\partial e}{\partial a} + \frac{\partial y}{\partial f} \frac{\partial f}{\partial a}$
 $= (g_d) (\exp(a)) + (g_e) \left(\frac{1}{b} \right) + (g_f) \left(-\frac{c}{a^2} \right)$

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

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

Updates for
Backpropagation:

$$\begin{aligned} g_x = \frac{\partial y}{\partial x} &= \sum_{k=1}^K \frac{\partial y}{\partial u_k} \frac{\partial u_k}{\partial x} \\ &= \sum_{k=1}^K g_{u_k} \frac{\partial u_k}{\partial x} \end{aligned}$$

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