



10-301/10-601 Introduction to Machine Learning

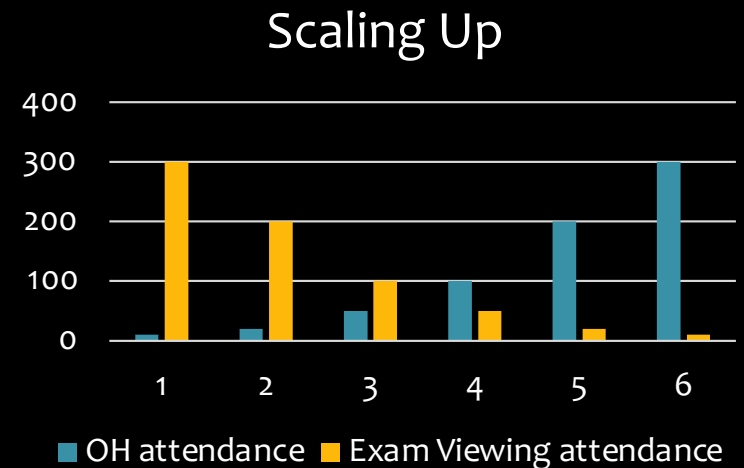
Machine Learning Department
School of Computer Science
Carnegie Mellon University

Neural Networks + Backpropagation

Matt Gormley
Lecture 12
Feb. 24, 2023

Reminders

- **Post-Exam Followup:**
 - Exam Viewing
 - Exit Poll: Exam 1
 - Grade Summary 1
- **Homework 4: Logistic Regression**
 - Out: Fri, Feb 17
 - Due: Sun, Feb 26 at 11:59pm
- **Homework 5: Neural Networks**
 - Out: Sun, Feb 26
 - Due: Fri, Mar 17 at 11:59pm



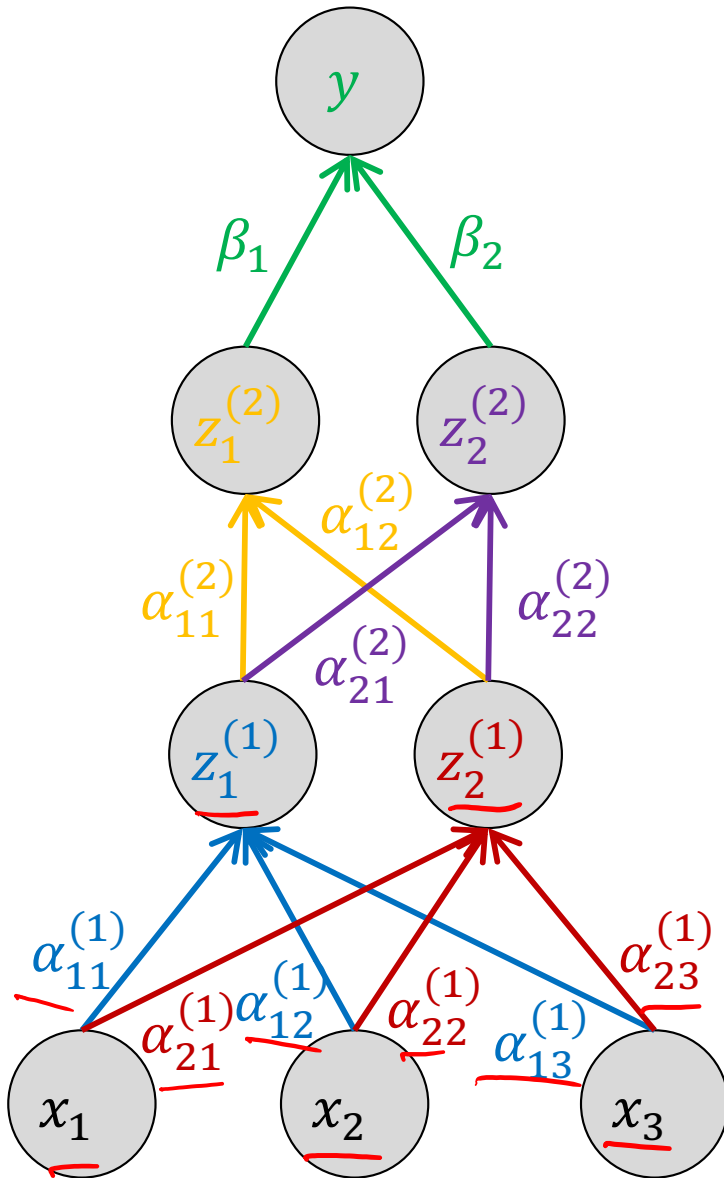
ARCHITECTURES

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

intercept

$$\alpha^{(1)} = \begin{bmatrix} \alpha_{11}^{(1)} & \alpha_{12}^{(1)} & \alpha_{13}^{(1)} \\ \alpha_{21}^{(1)} & \alpha_{22}^{(1)} & \alpha_{23}^{(1)} \end{bmatrix}$$

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

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

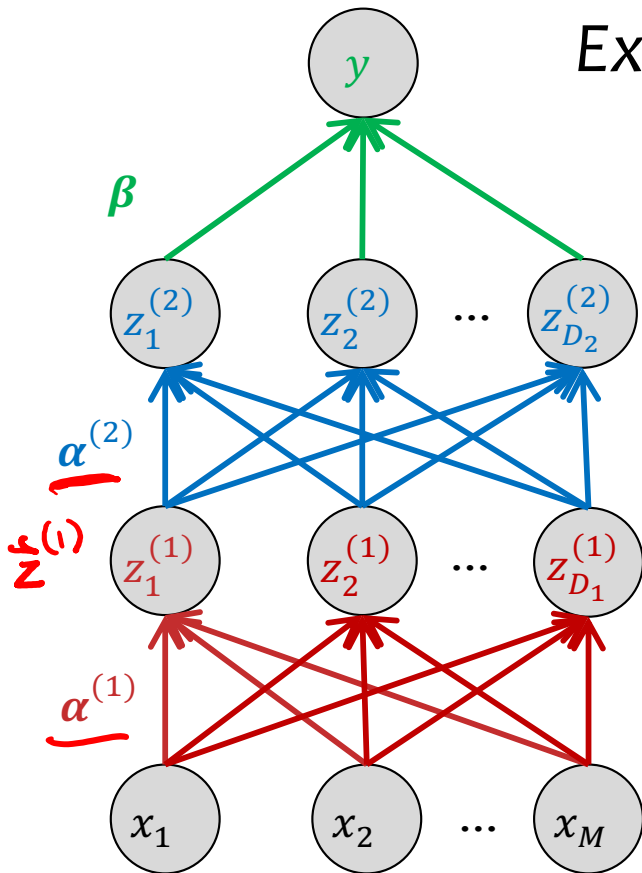
$$z_1^{(2)} = \sigma(\alpha_{11}^{(2)} z_1^{(1)} + \alpha_{12}^{(2)} z_2^{(1)} + \alpha_{10}^{(2)})$$

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

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

Neural Network (Matrix Form)

Example: Arbitrary Feed-forward Neural Network



$$\alpha^{(3)} \equiv \beta \in \mathbb{R}^{D_2}$$

$$b^{(3)} \equiv \beta_0 \in \mathbb{R}$$

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

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

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

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

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

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

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

$D_1 \times 1$

Apply σ "elementwise" to each entry in the vector $\vec{u} \in \mathbb{R}^M \quad \vec{z} = \sigma(\vec{u}) \in \mathbb{R}^M$

$$z_j = \sigma(u_j)$$

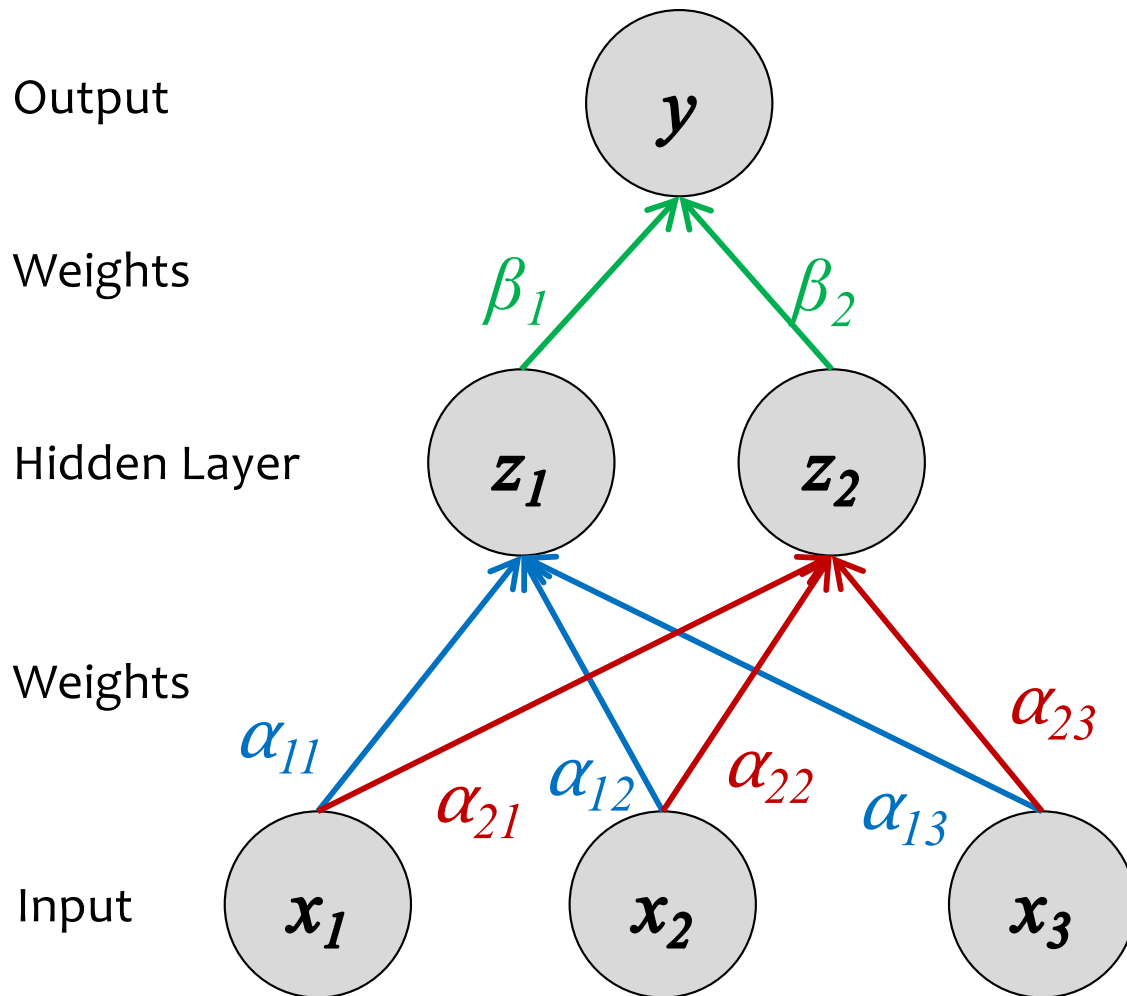
Fold in the intercept?

Set $x_1 = 1, z_1^{(1)} = 1, z_1^{(2)} = 1$.
Drop $b^{(1)}, b^{(2)}, \beta_0$

Caution: tricky to implement

Neural Network (Vector Form)

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



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

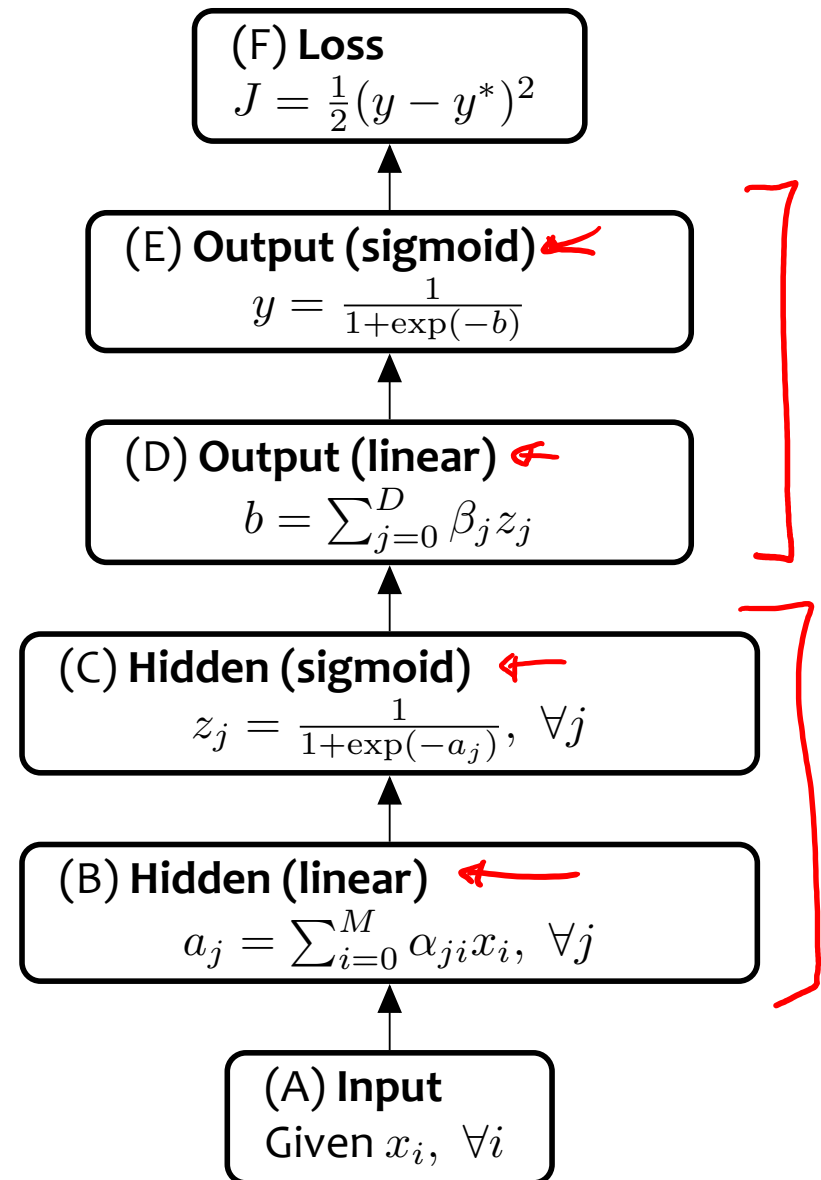
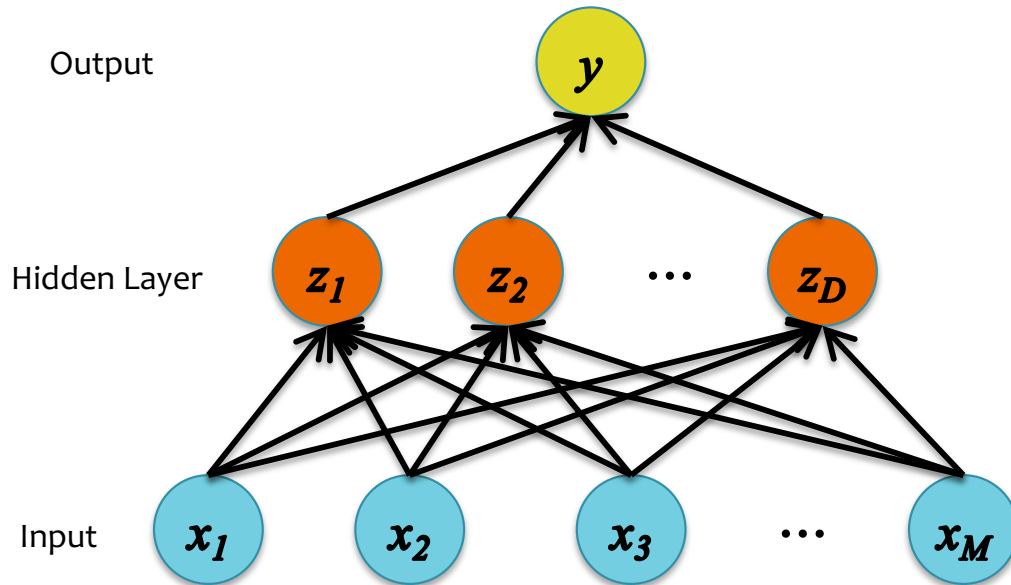
$$z_2 = \sigma(\alpha_{2, \cdot}^T \mathbf{x})$$

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

ACTIVATION FUNCTIONS

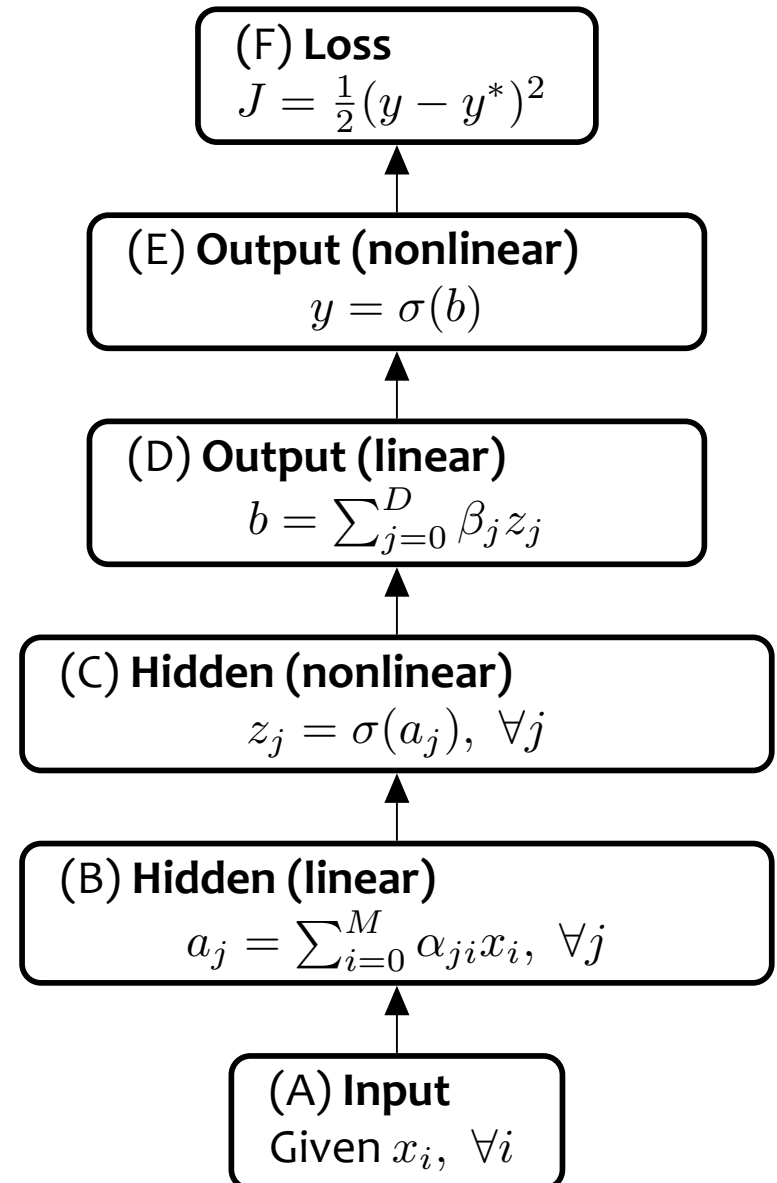
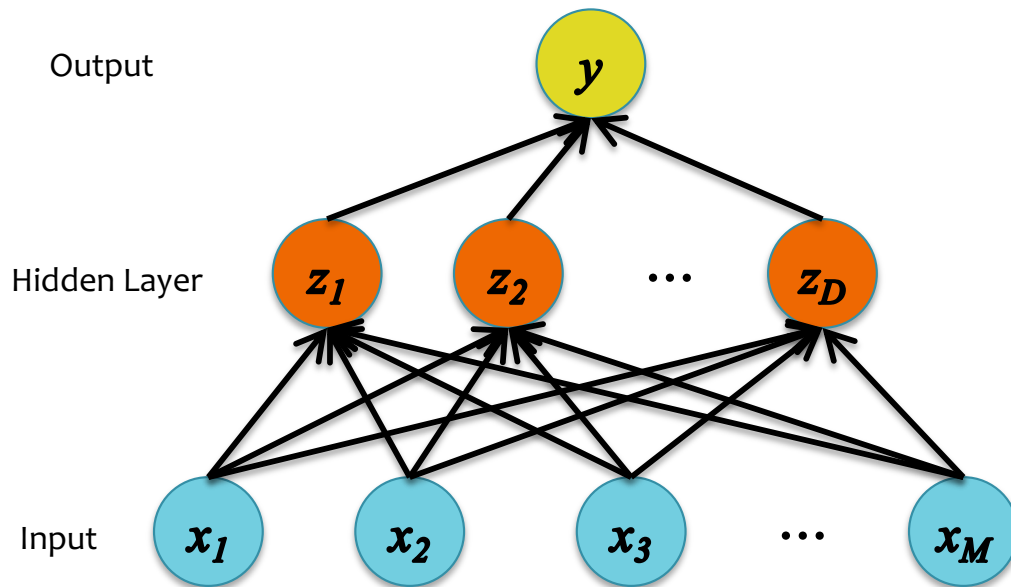
Activation Functions

Neural Network with sigmoid activation functions



Activation Functions

Neural Network with arbitrary nonlinear activation functions

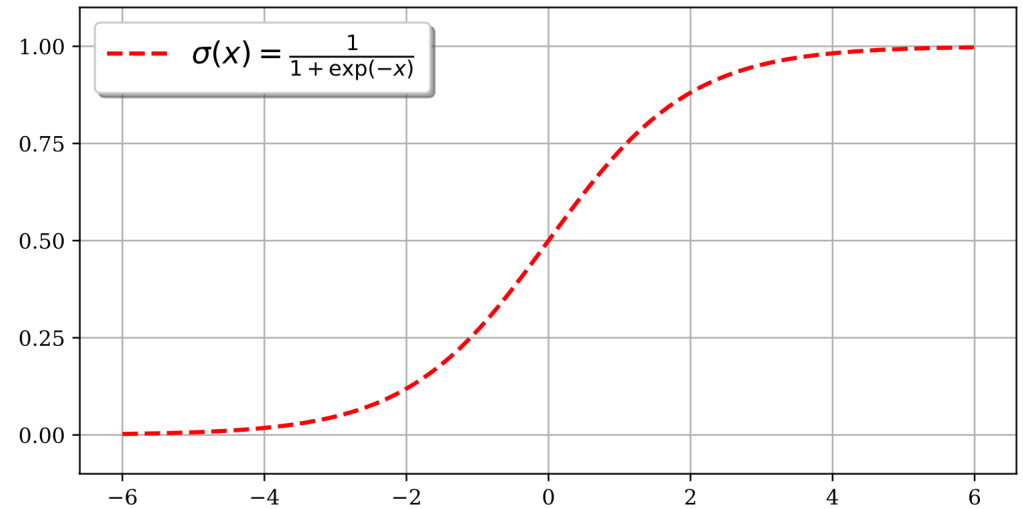


Activation Functions

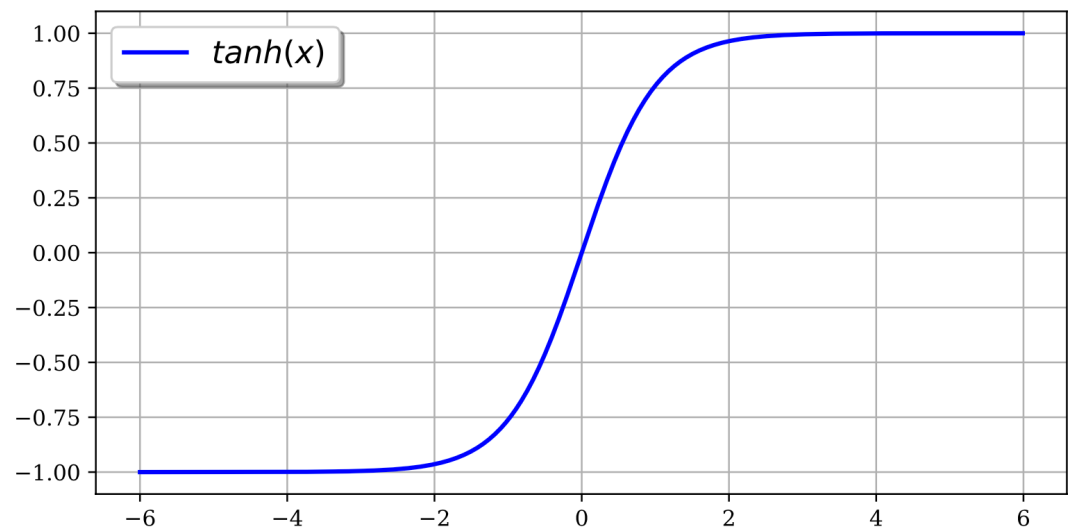
So far, we've assumed that the activation function (nonlinearity) is always the sigmoid function...

... but the sigmoid is not widely used in modern neural networks

Sigmoid (aka. logistic) function



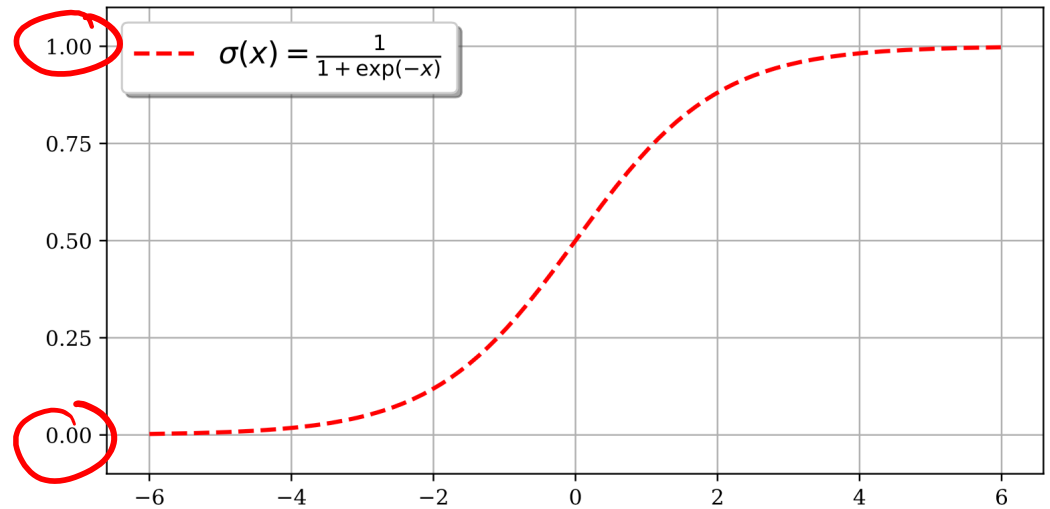
Hyperbolic tangent function



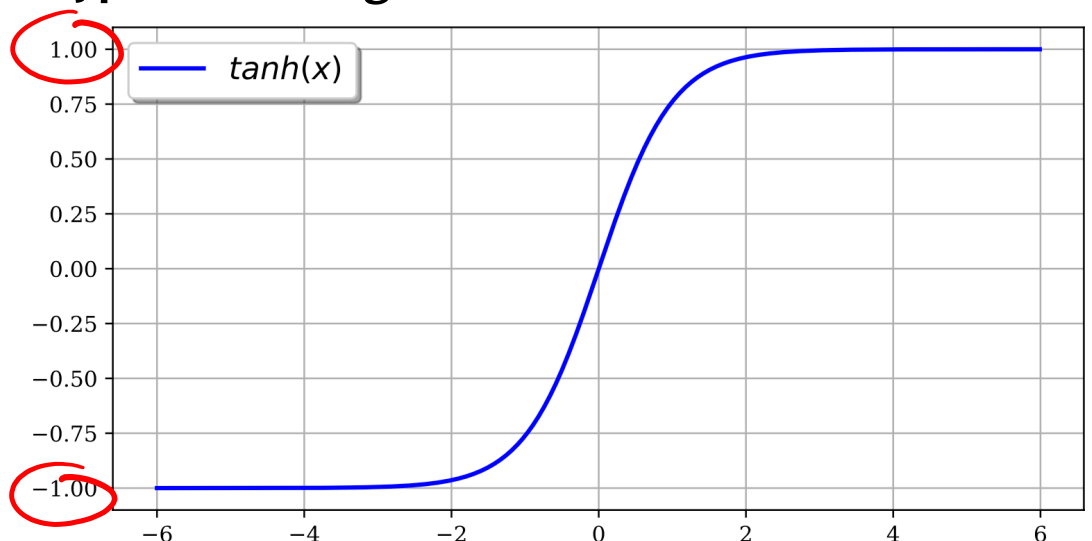
Activation Functions

- sigmoid, $\sigma(x)$
 - output in range (0,1)
 - good for probabilistic outputs
- hyperbolic tangent, $\tanh(x)$
 - similar shape to sigmoid, but output in range (-1,+1)

Sigmoid (aka. logistic) function



Hyperbolic tangent function



Understanding the difficulty of training deep feedforward neural networks

AI Stats 2010

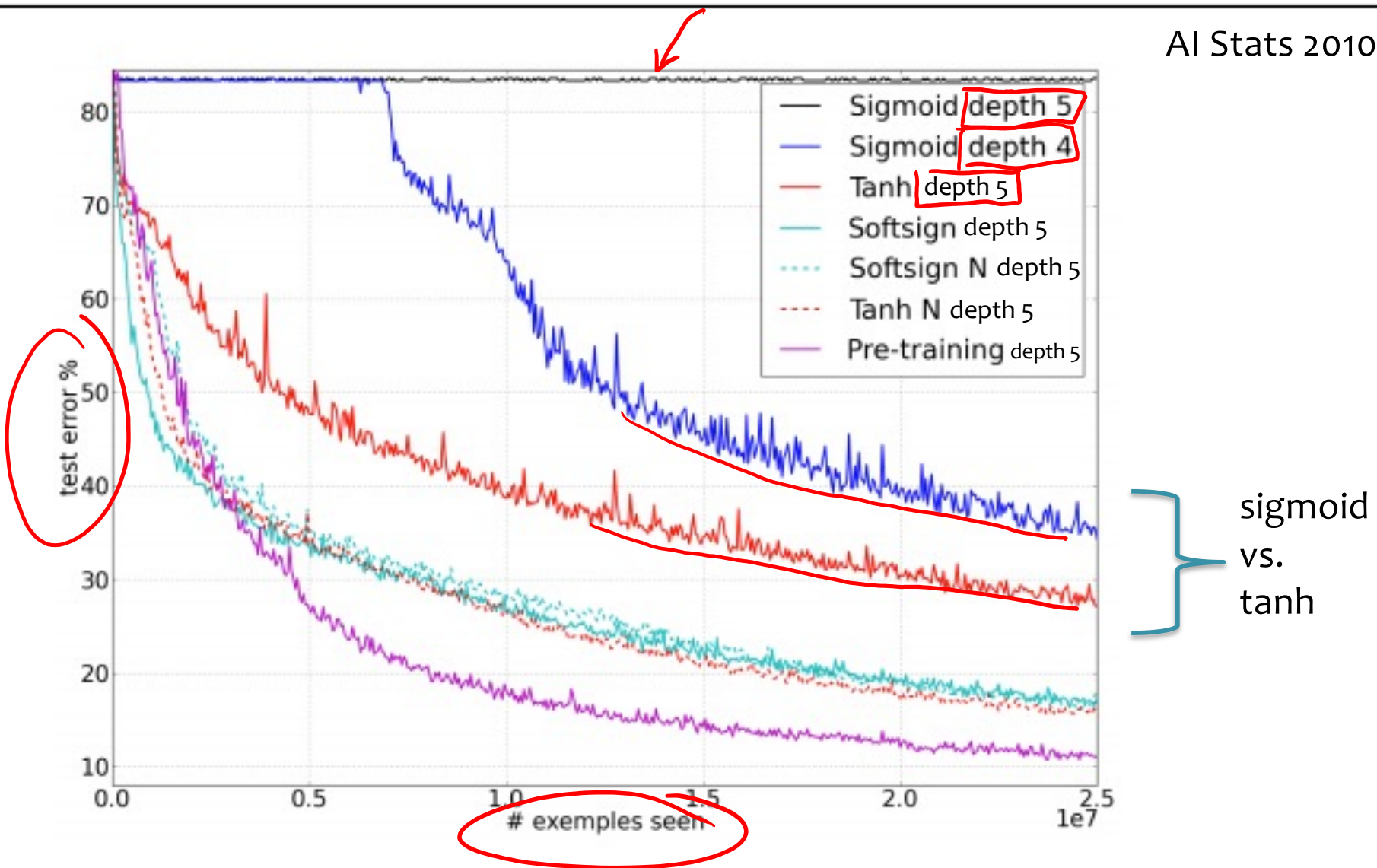
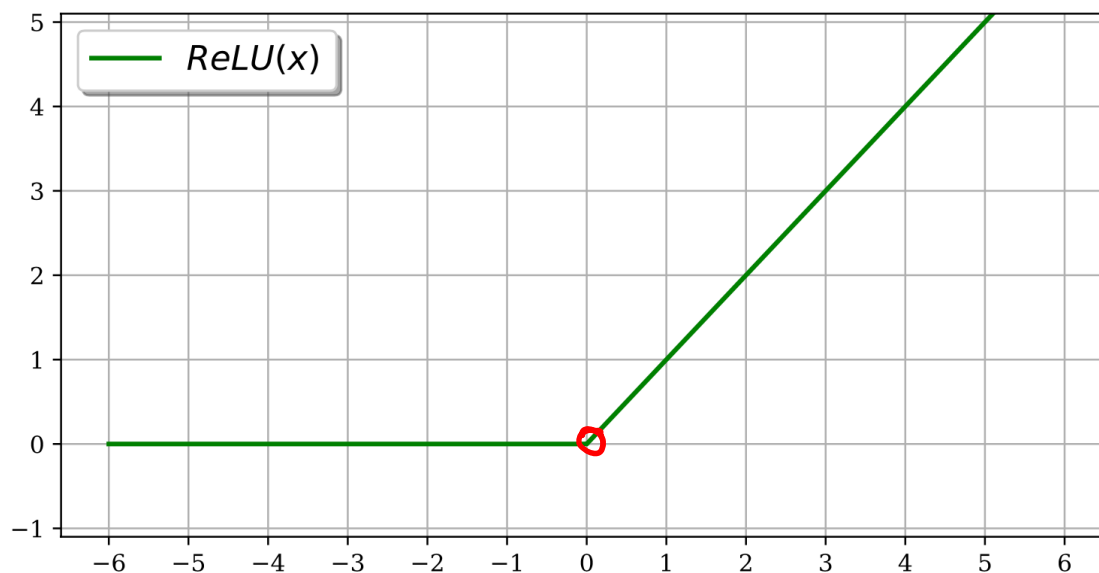


Figure from Glorot & Benthio (2010)

Activation Functions

- Rectified Linear Unit (ReLU)
 - avoids the vanishing gradient problem
 - derivative is fast to compute

$$\text{ReLU}(x) = \max(0, x)$$

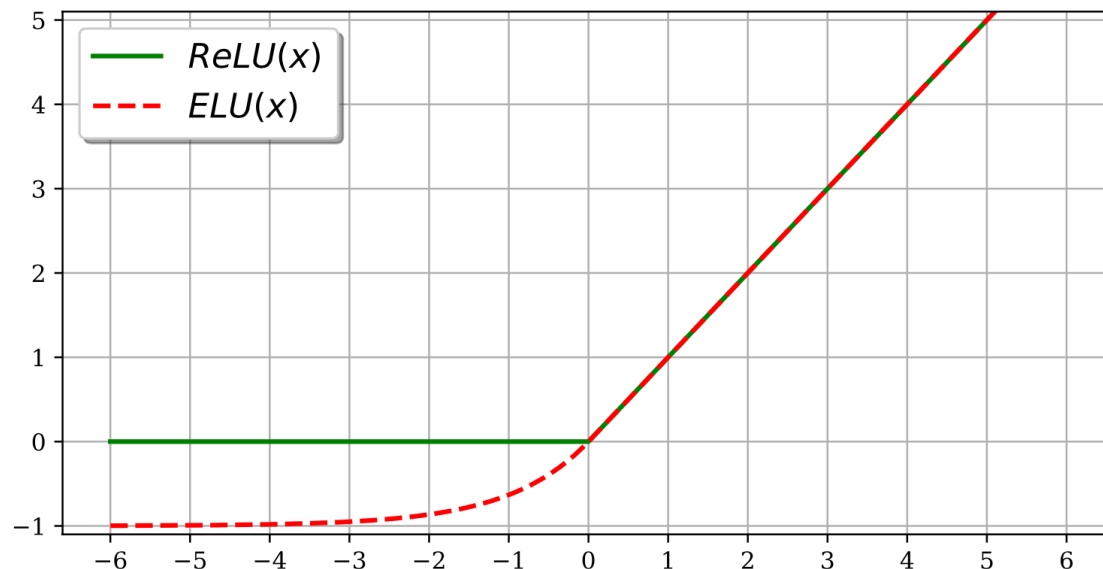


Activation Functions

- Rectified Linear Unit (ReLU)

- avoids the vanishing gradient problem
- derivative is fast to compute

$$\text{ReLU}(x) = \max(0, x)$$



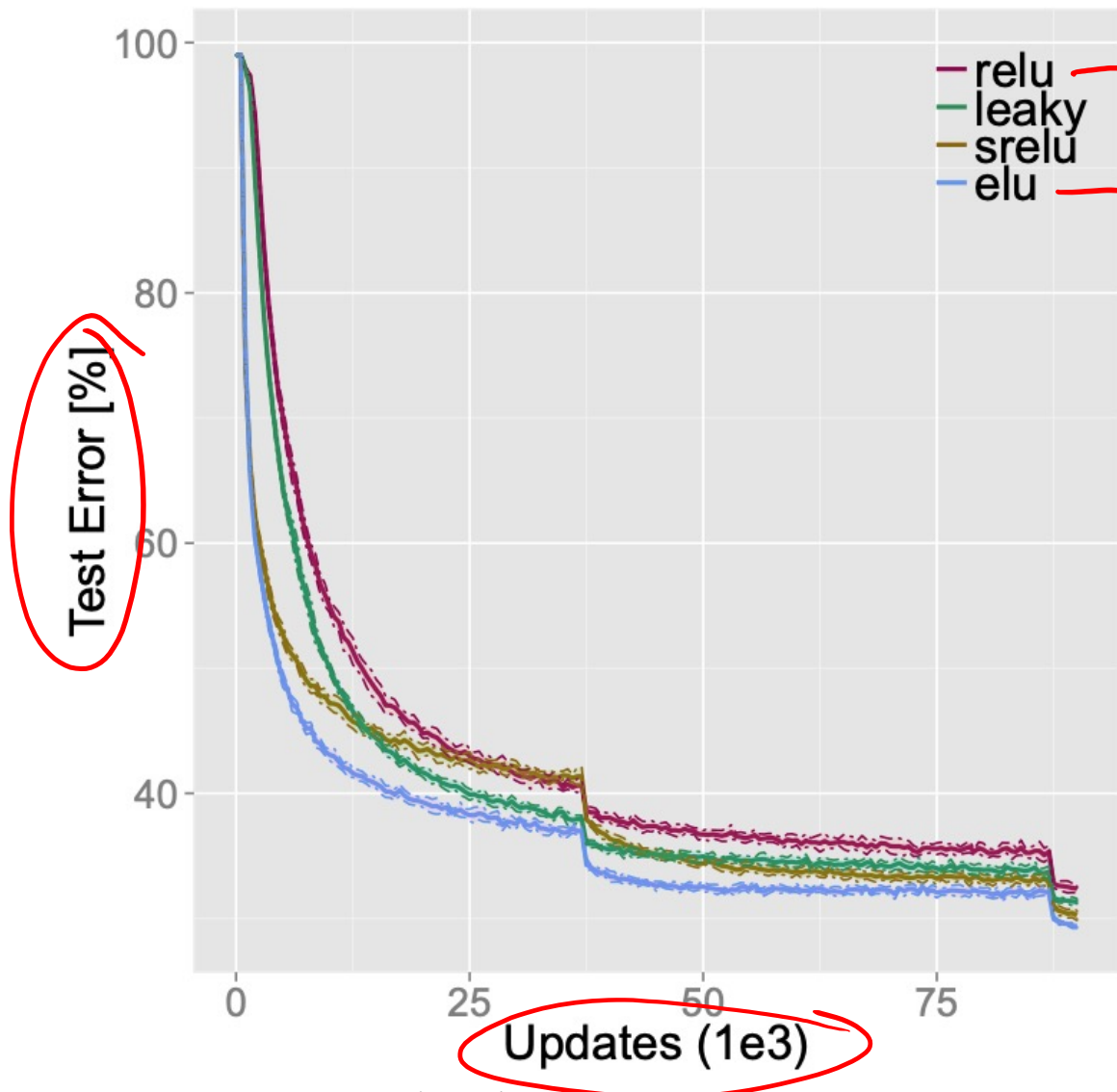
- Exponential Linear Unit (ELU)

- same as ReLU on positive inputs
- unlike ReLU, allows negative outputs and smoothly transitions for $x < 0$

$$\text{ELU}(x) = \begin{cases} x, & \text{if } x > 0 \\ \alpha(\exp(x) - 1), & \text{if } x \leq 0 \end{cases}$$

Activation Functions

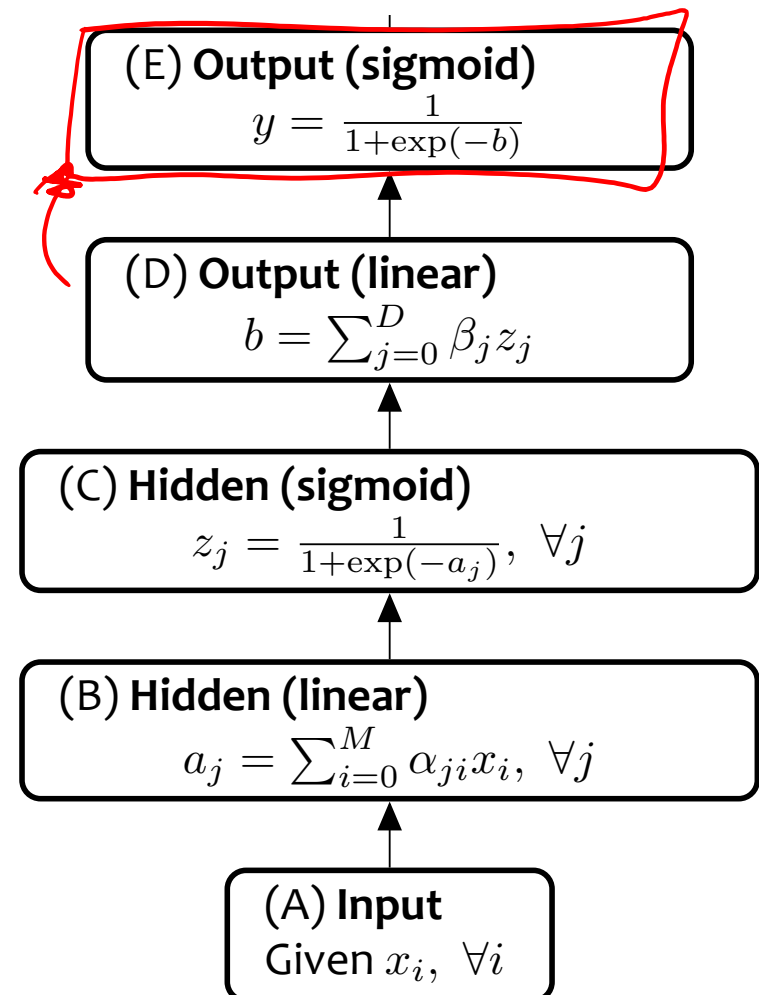
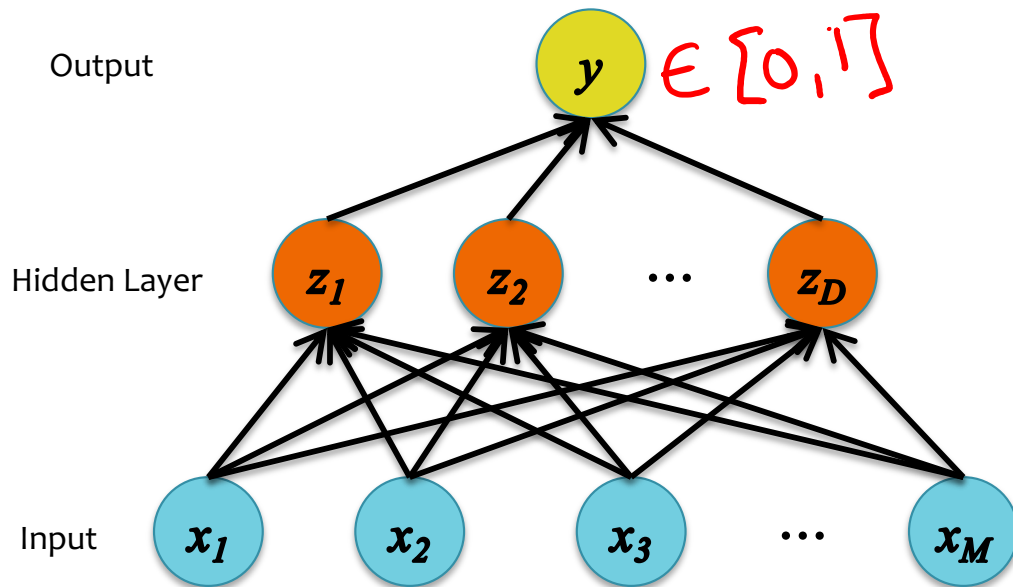
Image Classification Benchmark (CIFAR-10)



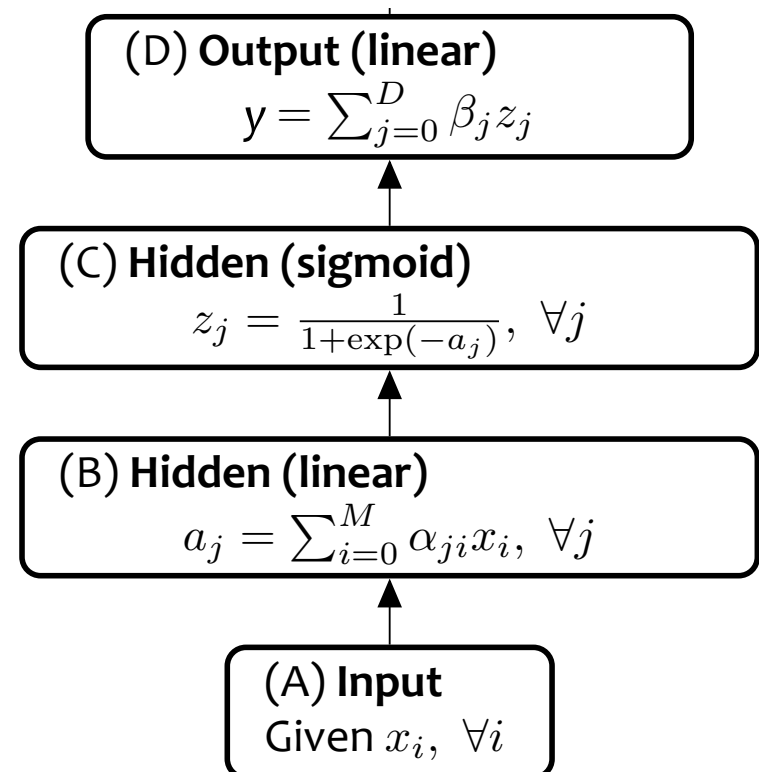
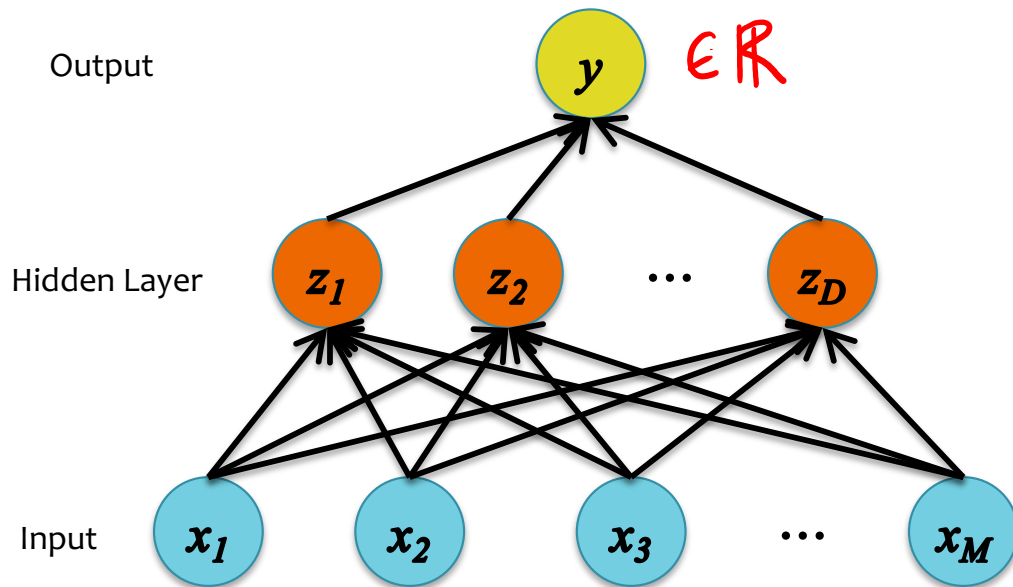
1. Training loss converges fastest with ELU
2. ELU(x) yields lower test error than ReLU(x) on CIFAR-10

LOSS FUNCTIONS & OUTPUT LAYERS

Neural Network for Classification



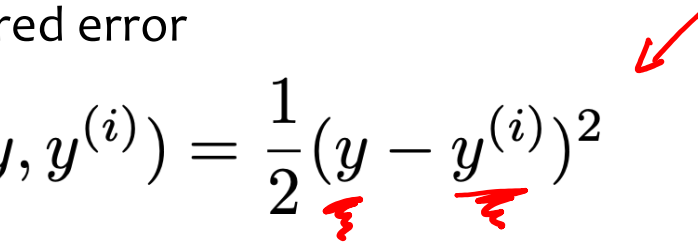
Neural Network for Regression



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

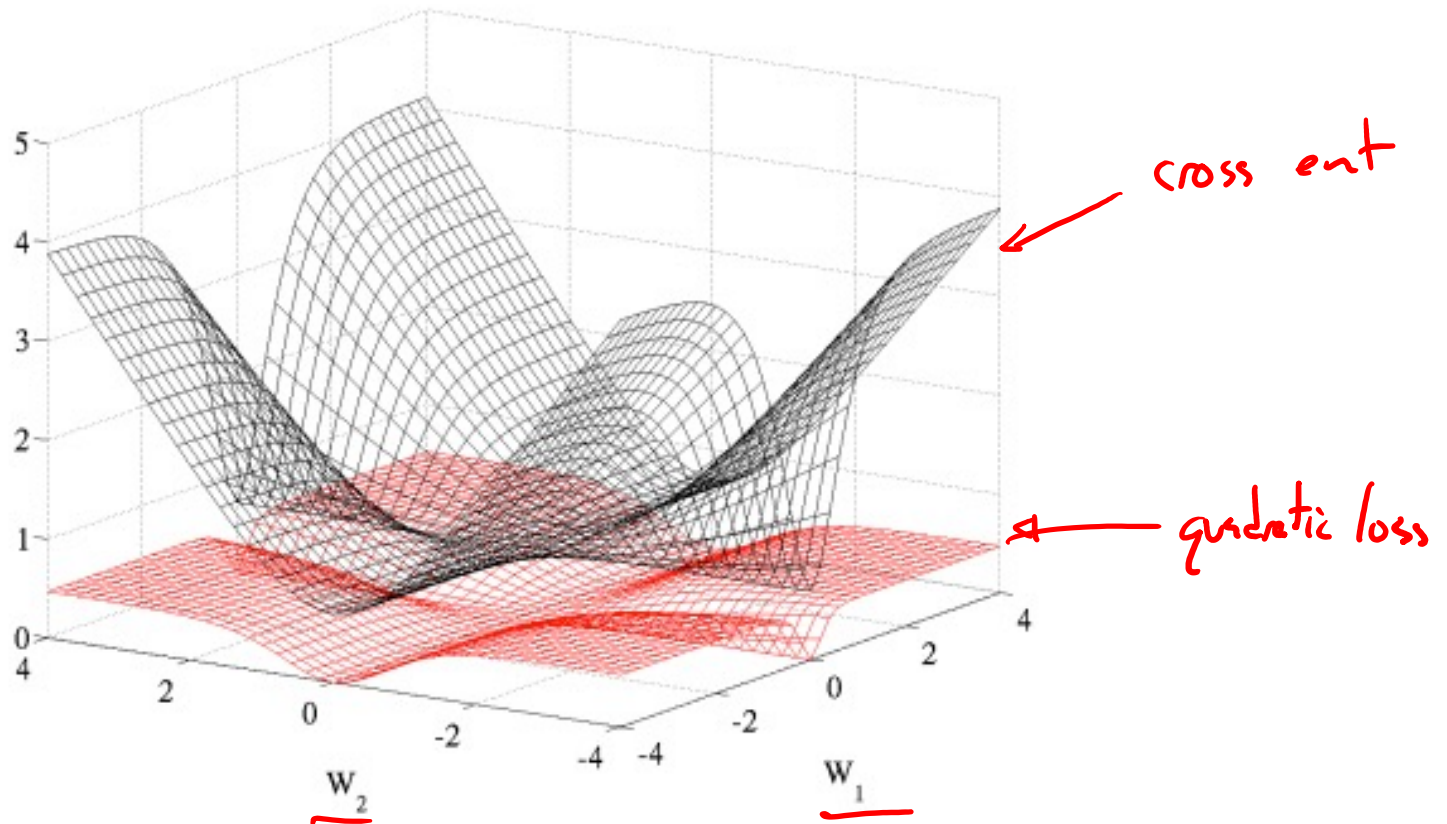
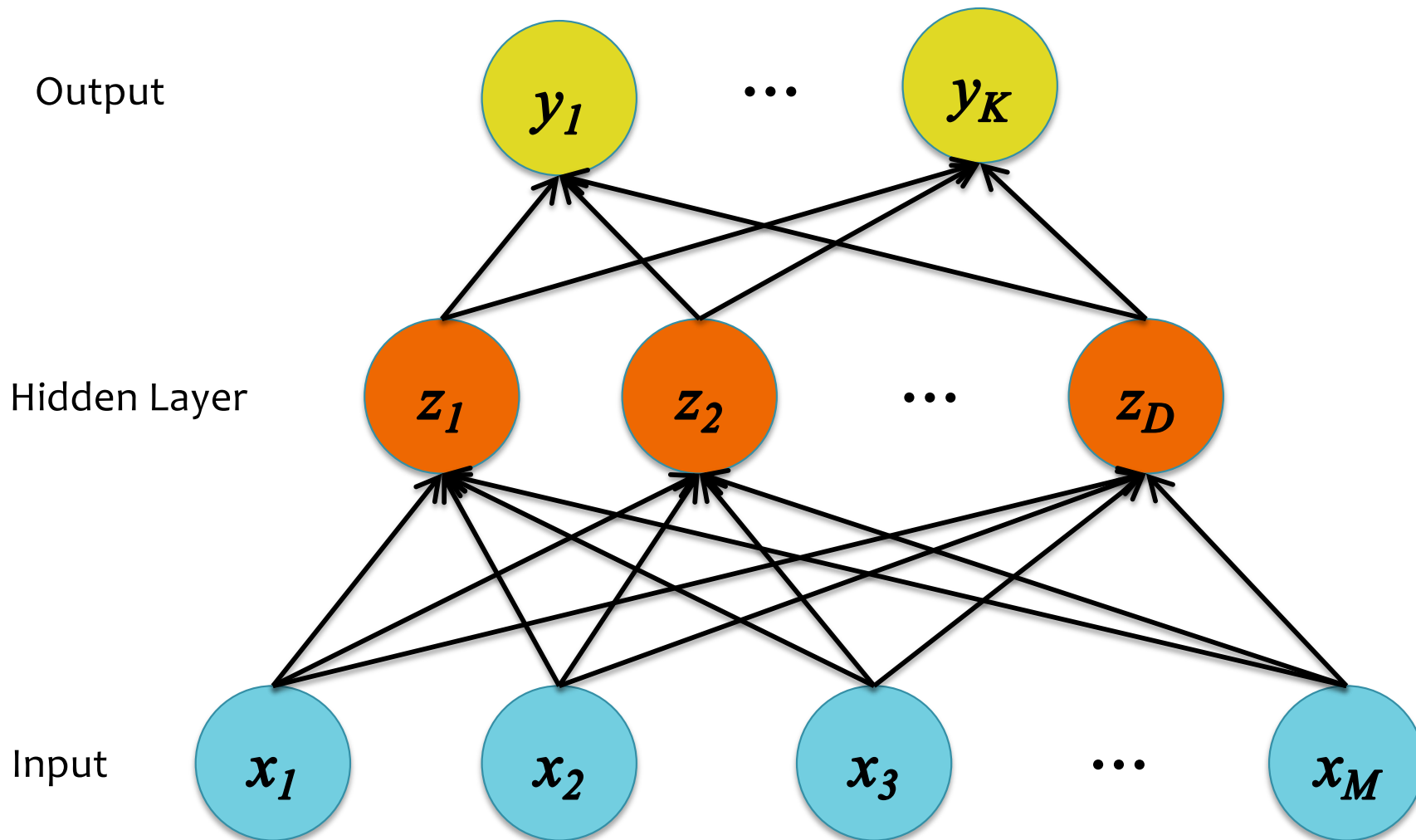


Figure 5: Cross entropy (black, surface on top) and quadratic (red, bottom surface) cost as a function of two weights (one at each layer) of a network with two layers, W_1 respectively on the first layer and W_2 on the second, output layer.

Multiclass Output

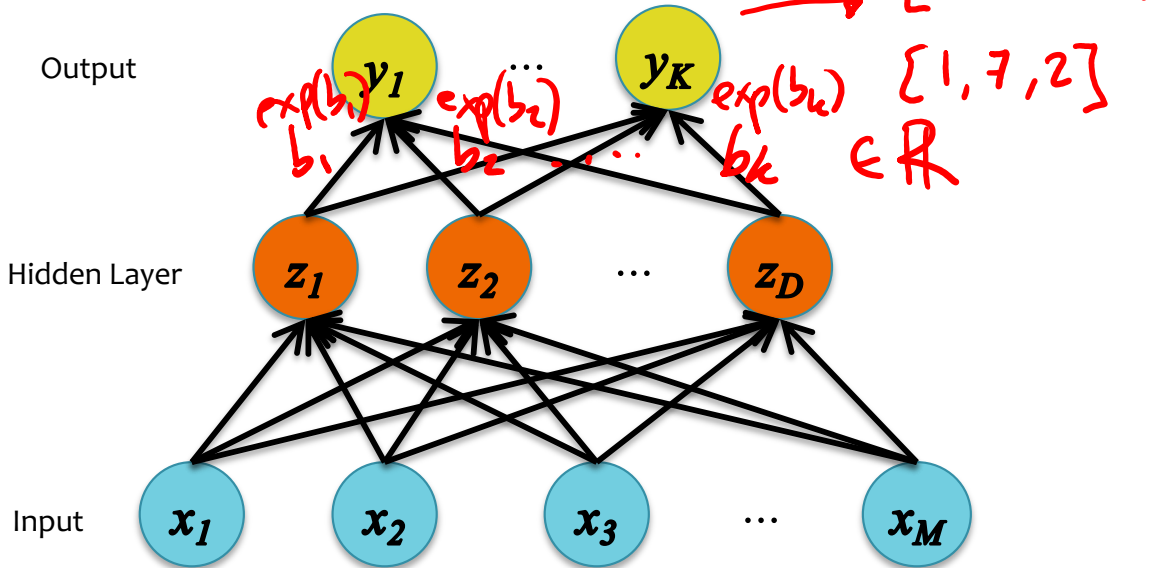
$$y^{(i)} \in \{1, \dots, K\}$$



Multiclass Output

Softmax:

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



(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)

$$b_k = \sum_{j=0}^D \beta_{kj} z_j \quad \forall k$$

(C) Hidden (nonlinear)

$$z_j = \sigma(a_j), \quad \forall j$$

(B) Hidden (linear)

$$a_j = \sum_{i=0}^M \alpha_{ji} x_i, \quad \forall j$$

(A) Input

Given $x_i, \forall i$

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:

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

$$Y^{(i)} = 4$$

- 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:

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

$$= - \log(y_4)$$

$$= - \log(\text{Pr}[Y=4 | \vec{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}])$$

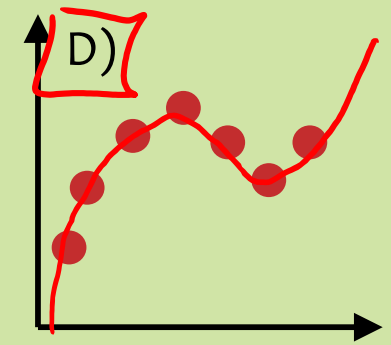
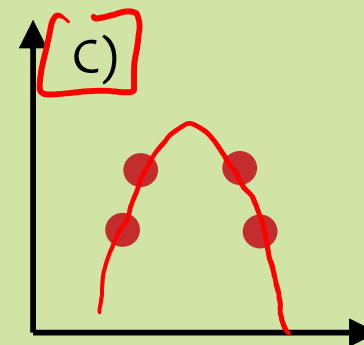
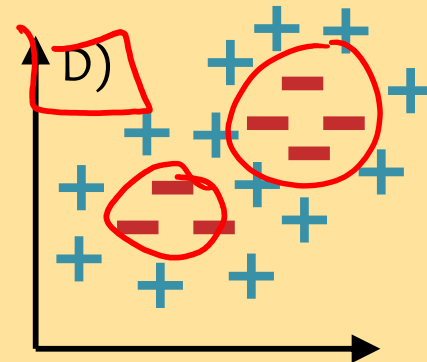
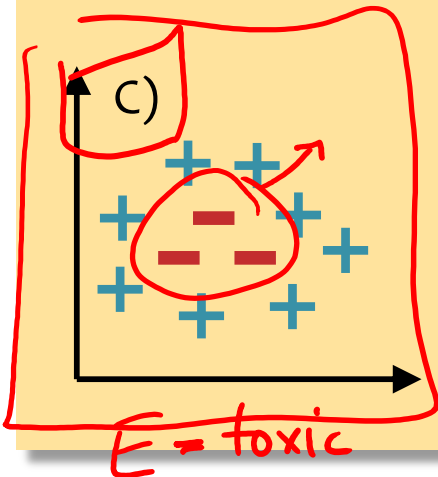
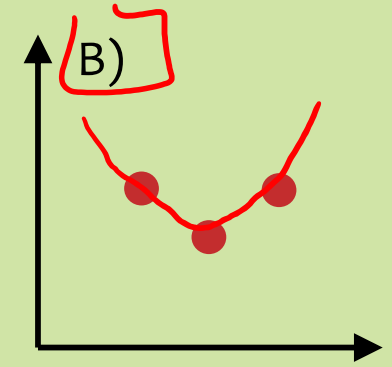
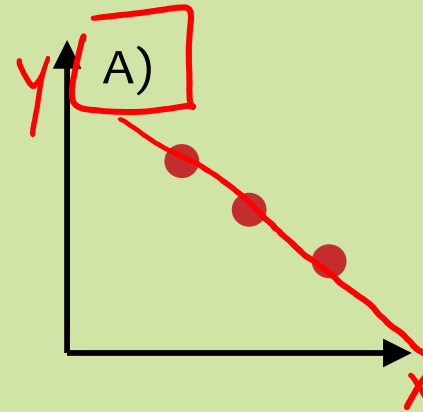
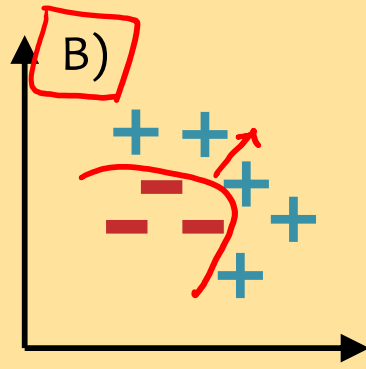
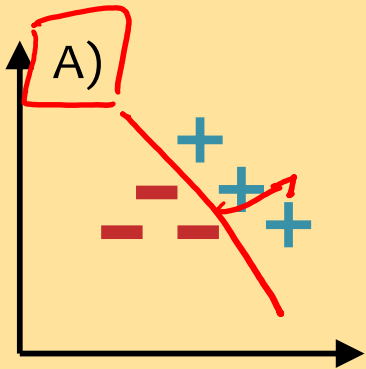
Neural Network Errors

Q1

Q2

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.**

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.**



Neural Networks Objectives

You should be able to... *Q3: What Q's do you have?*

- 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:

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

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)

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

Background

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

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


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!

(opposite the gradient)


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

- **Question 1:**
When can we compute the gradients for an arbitrary neural network?
- **Question 2:**
When can we make the gradient computation efficient?

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

$$J: \mathbb{R}^M \rightarrow \mathbb{R}$$

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 $f(\mathbf{x})$
 - 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 $f(\mathbf{x})$

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

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

THE FINITE DIFFERENCE METHOD

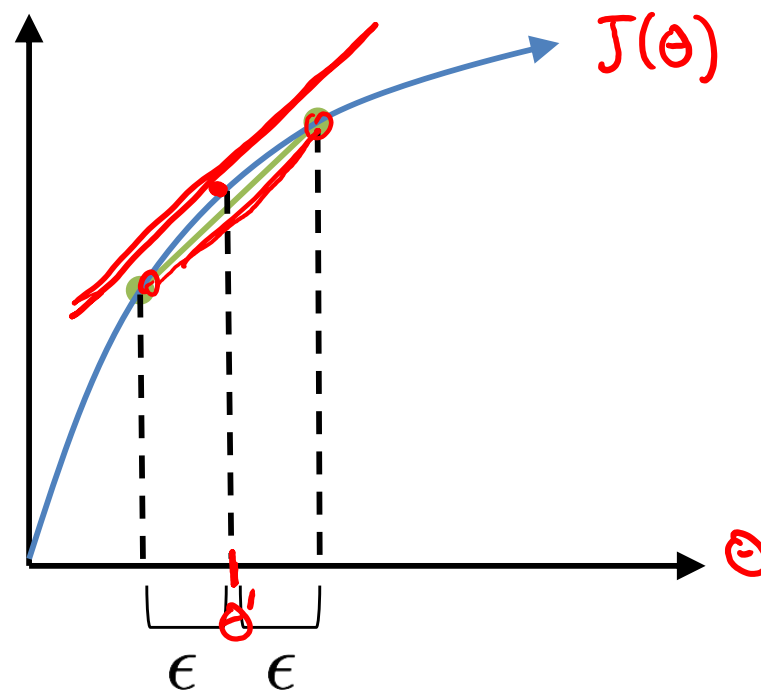
The *centered* finite difference approximation is:

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

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



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.**

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

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

- | | |
|---------------|--|
| A. [42, -72] | <input checked="" type="checkbox"/> E. [1208, 810] |
| B. [72, -42] | F. [810, 1208] |
| C. [100, 127] | G. [1505, 94] |
| D. [127, 100] | H. [94, 1505] |

Differentiation Quiz #2:

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

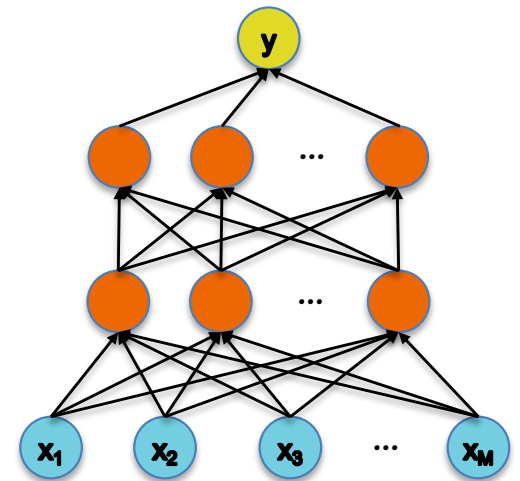
$$y = \sigma(\boldsymbol{\beta}^T \sigma((\boldsymbol{\alpha}^{(2)})^T \sigma((\boldsymbol{\alpha}^{(1)})^T \mathbf{x})))$$

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:

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

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

Training

Chain Rule

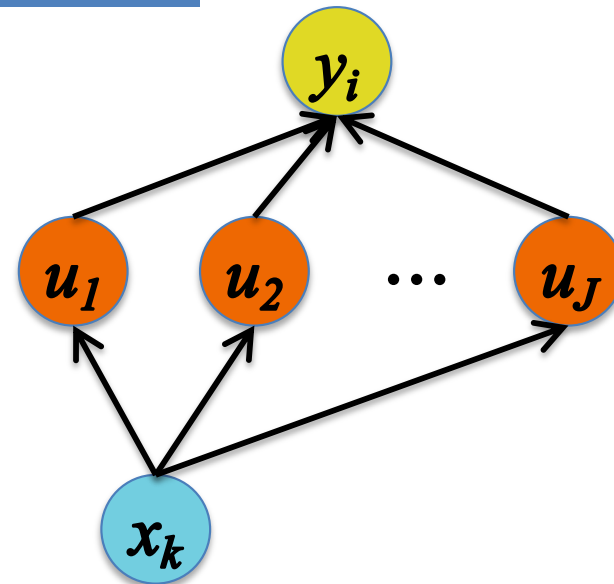
Whiteboard

– Chain Rule of Calculus

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

Chain Rule:

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

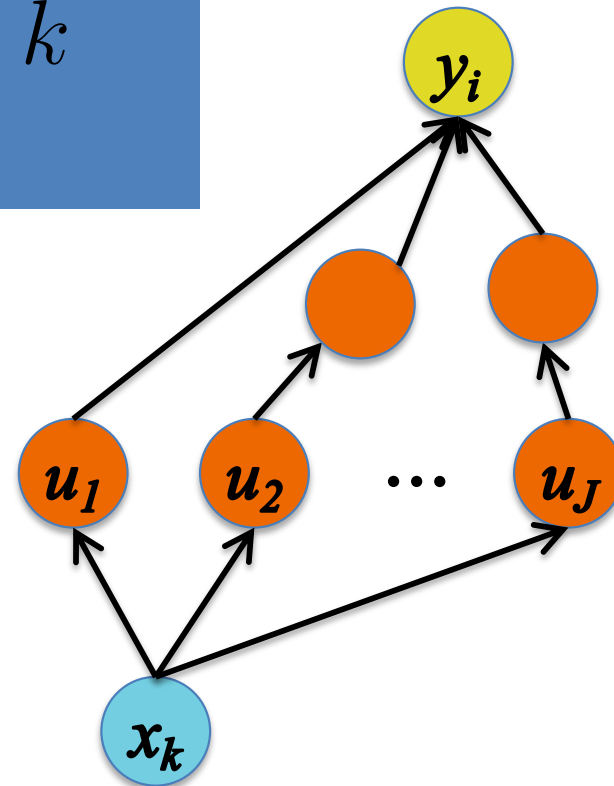


Given: $y = g(u)$ and $u = h(x)$.

Chain Rule:

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

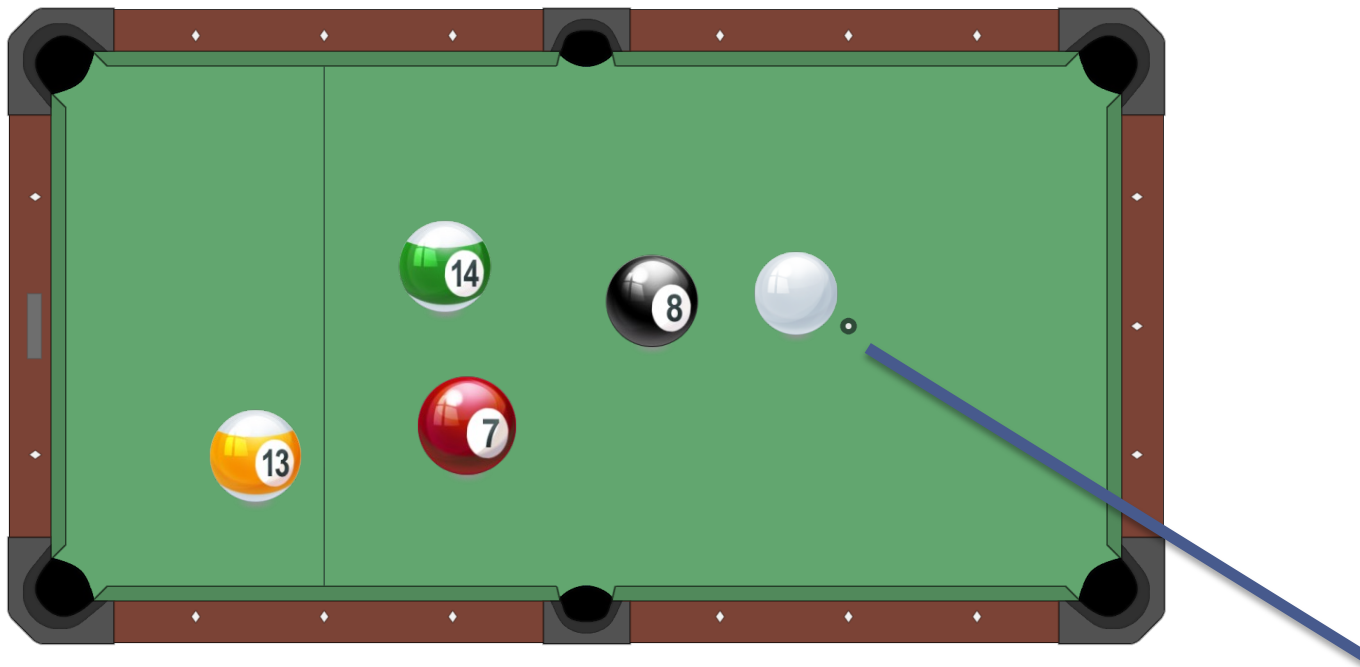
Backpropagation is just repeated application of the **chain rule** from Calculus 101.



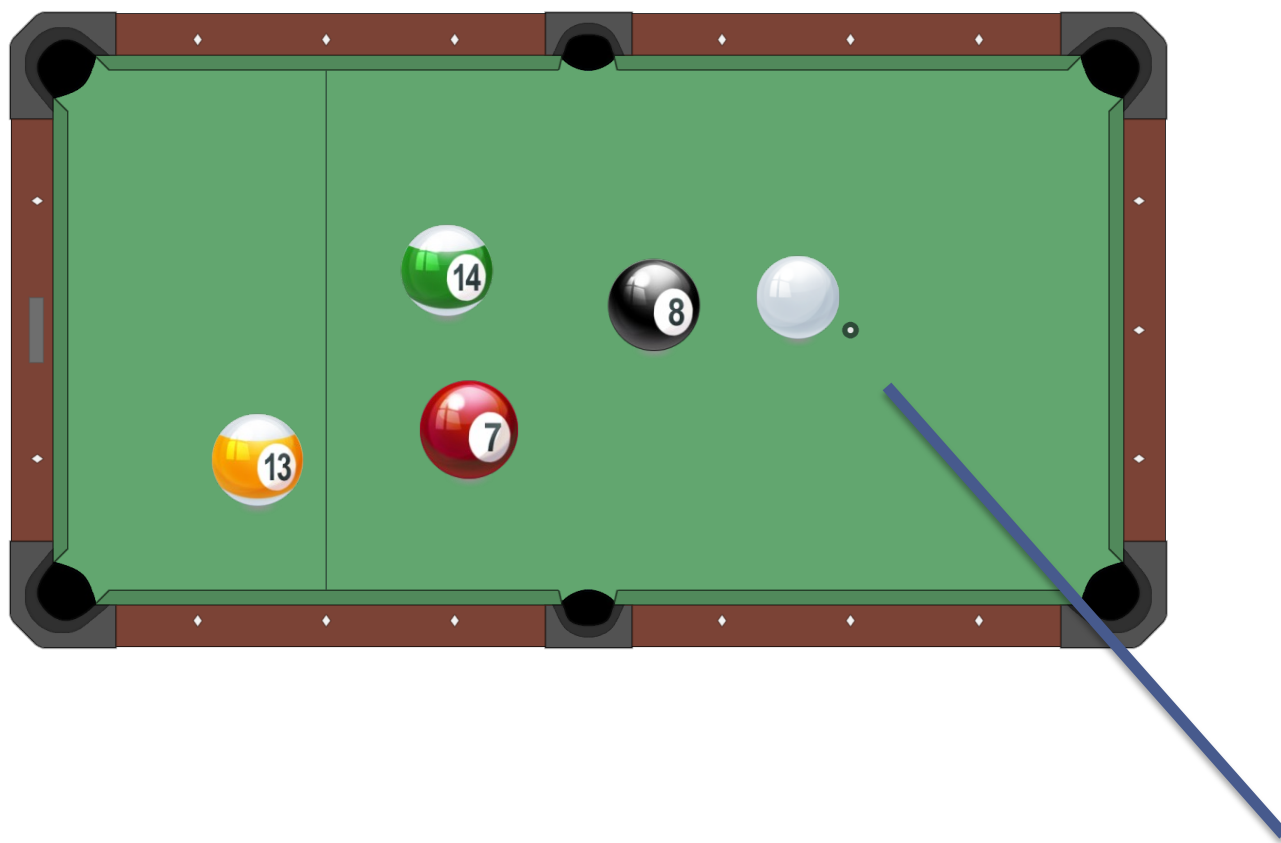
Intuitions

BACKPROPAGATION OF ERRORS

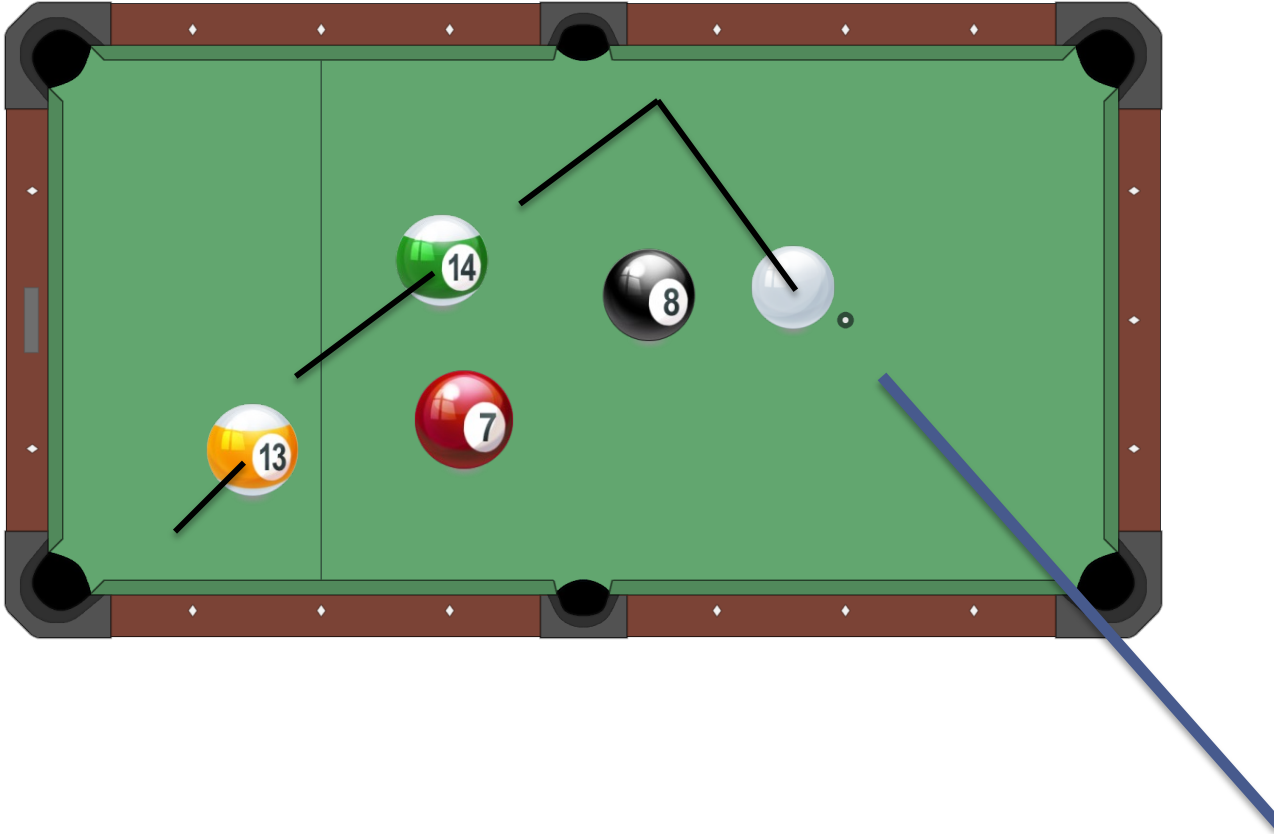
Error Back-Propagation



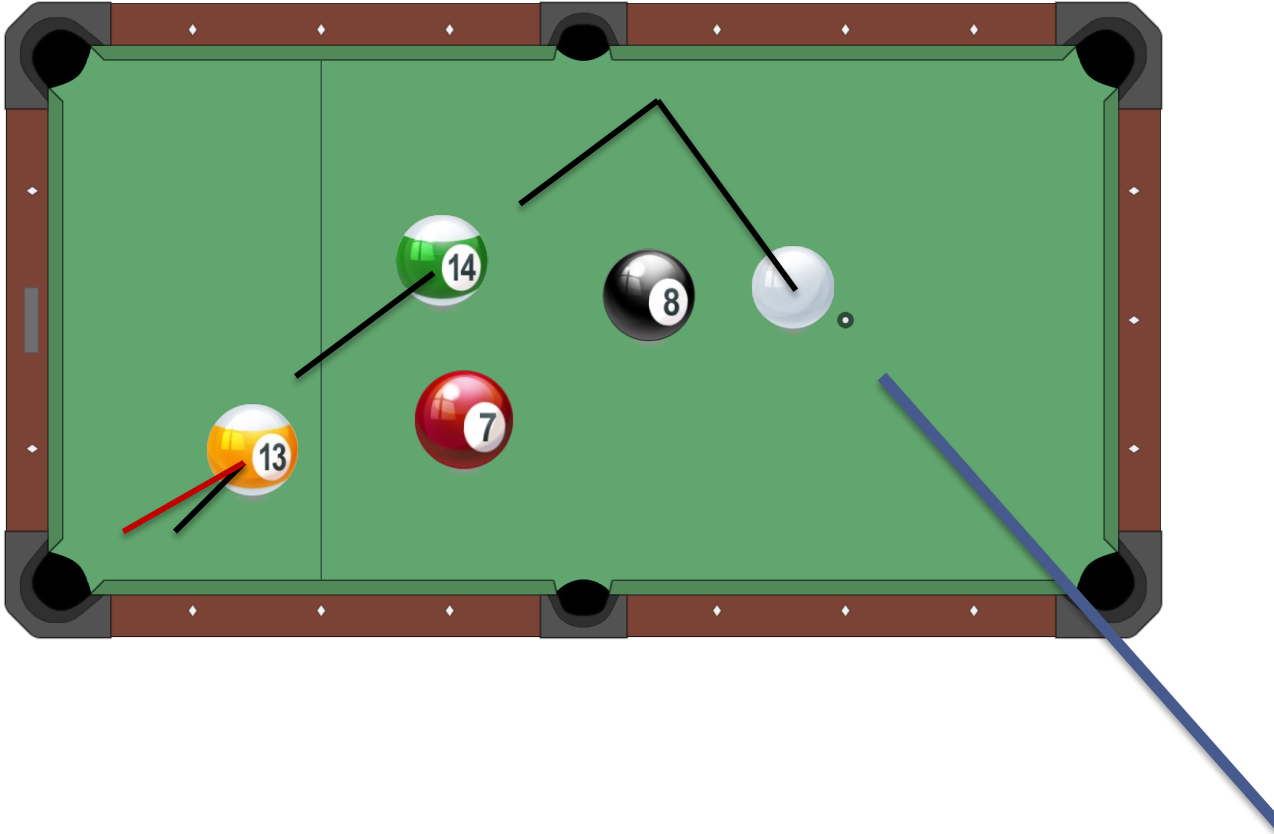
Error Back-Propagation



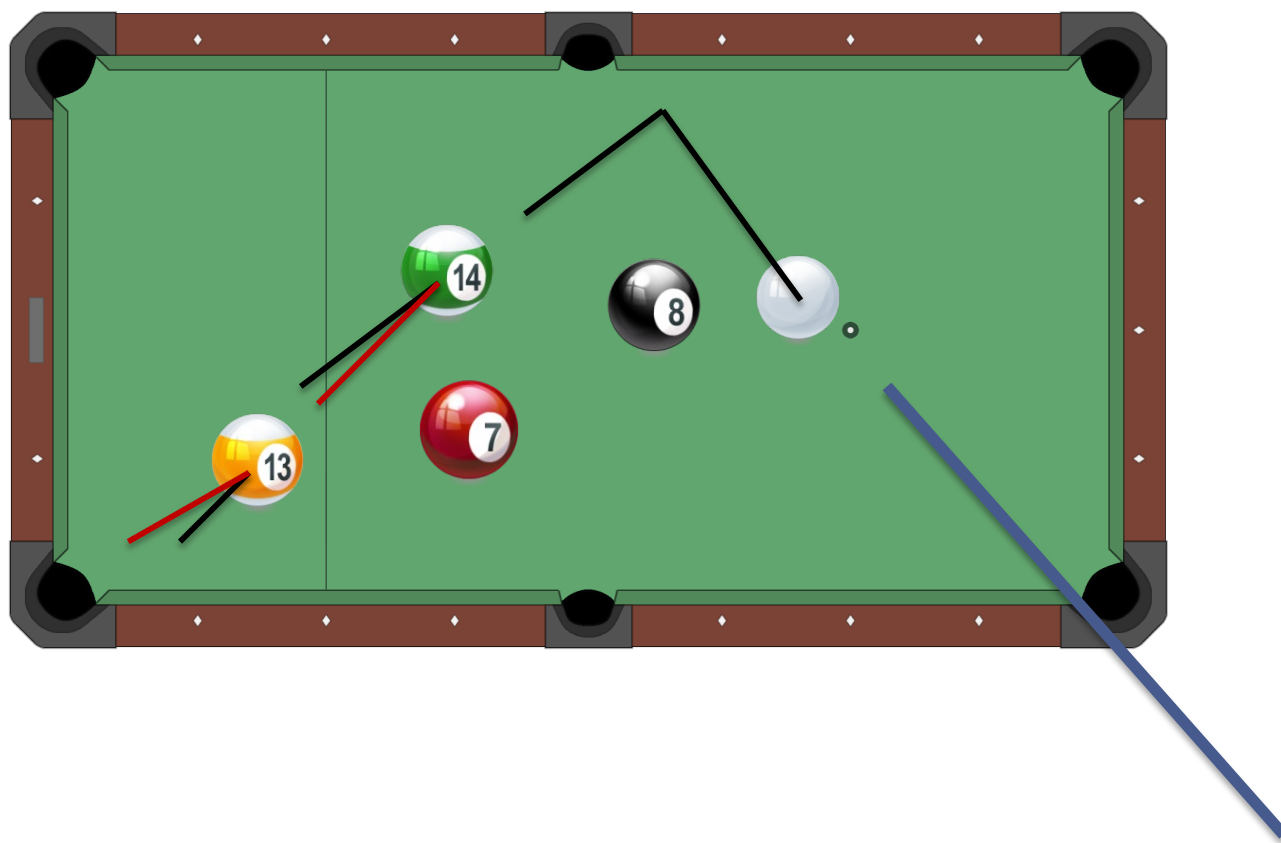
Error Back-Propagation



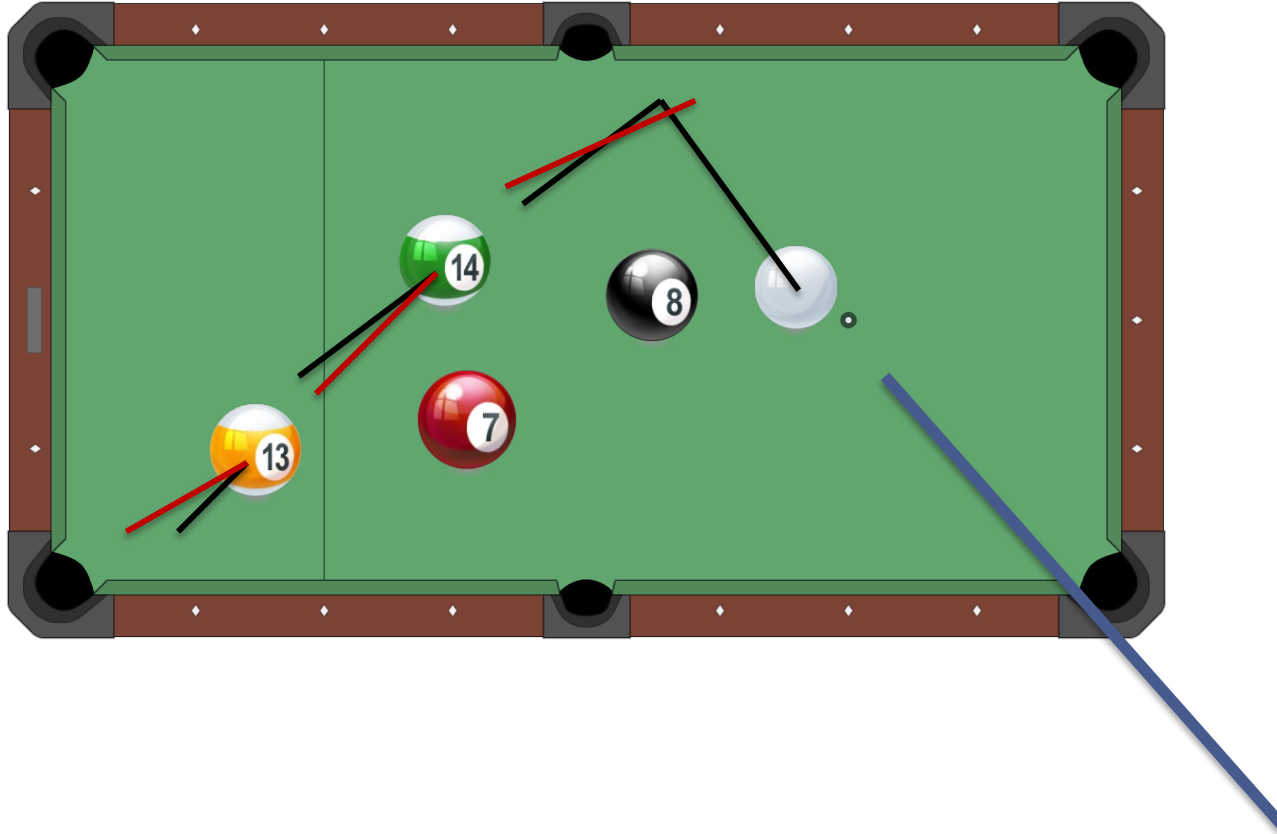
Error Back-Propagation



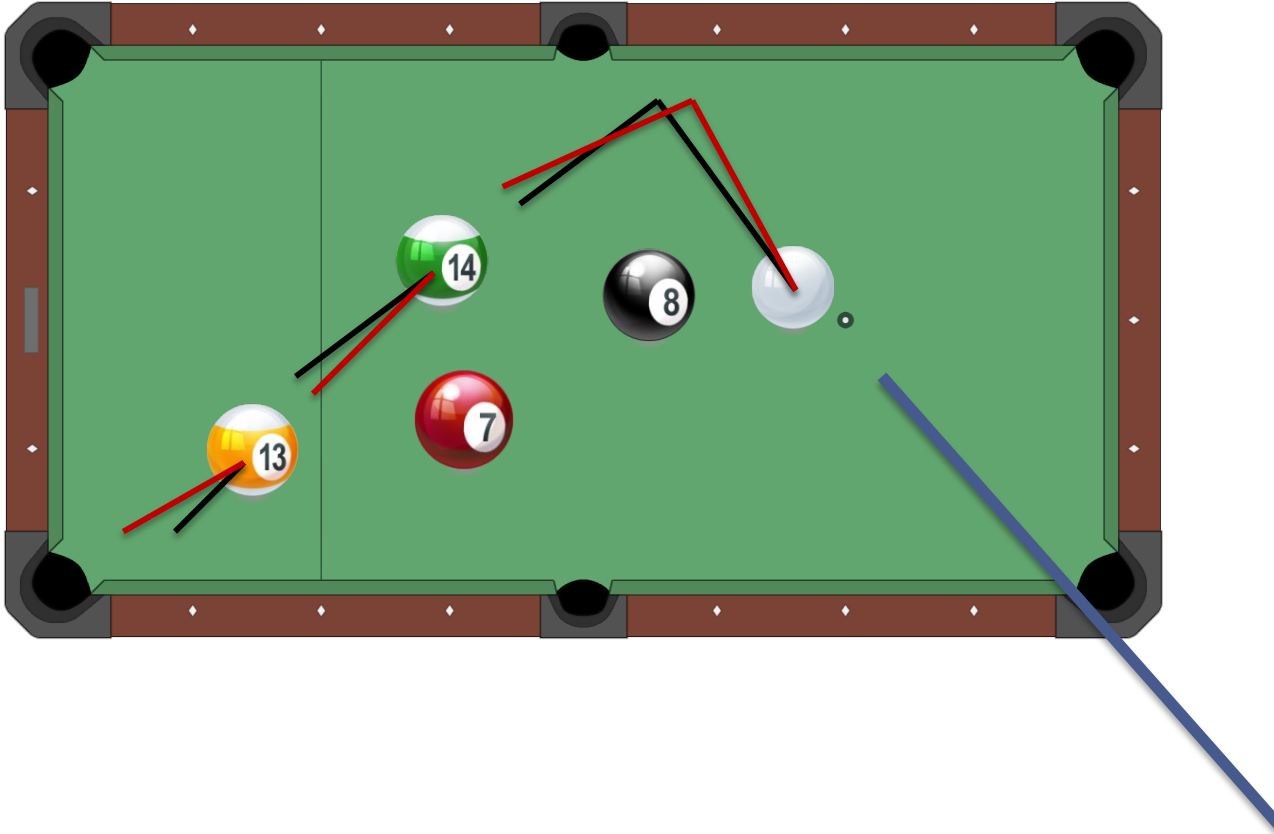
Error Back-Propagation



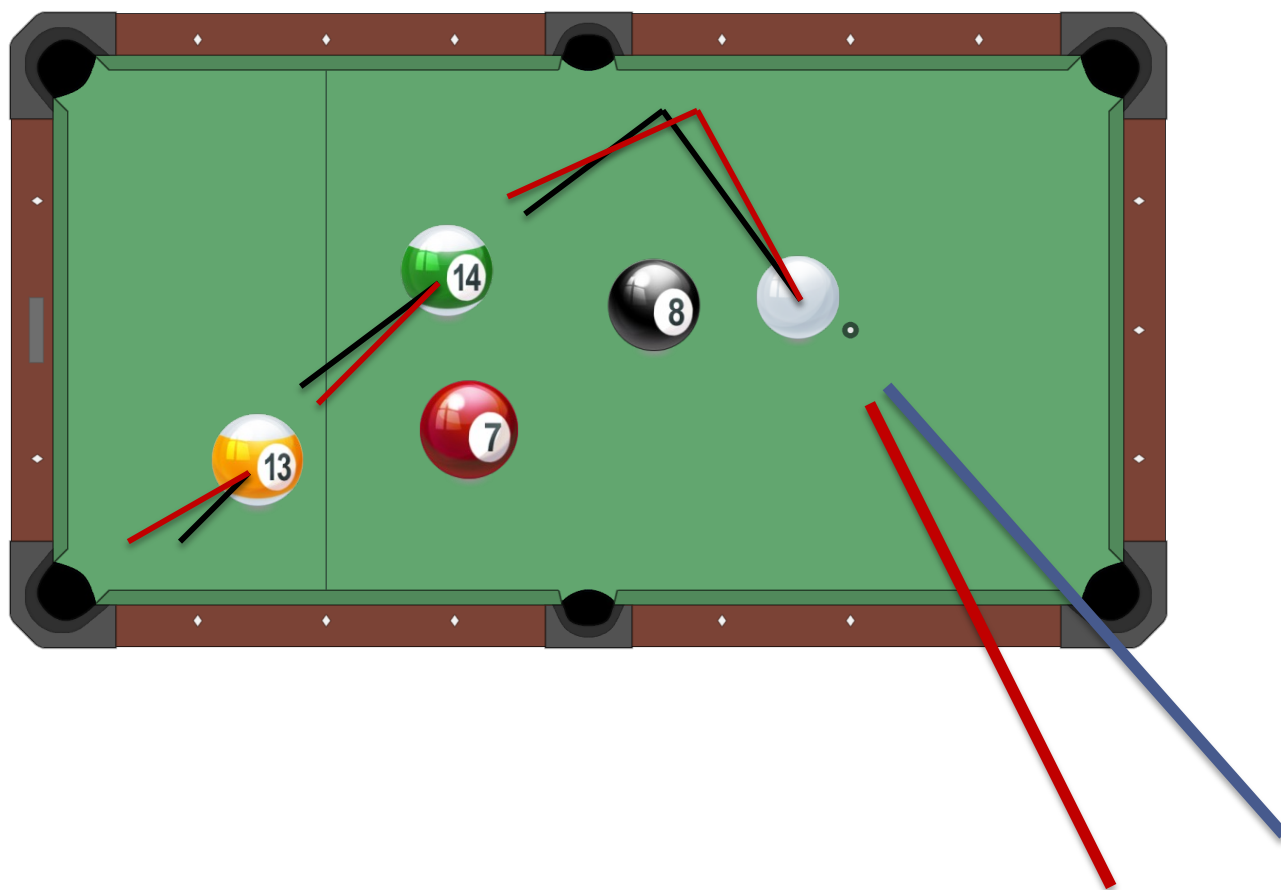
Error Back-Propagation



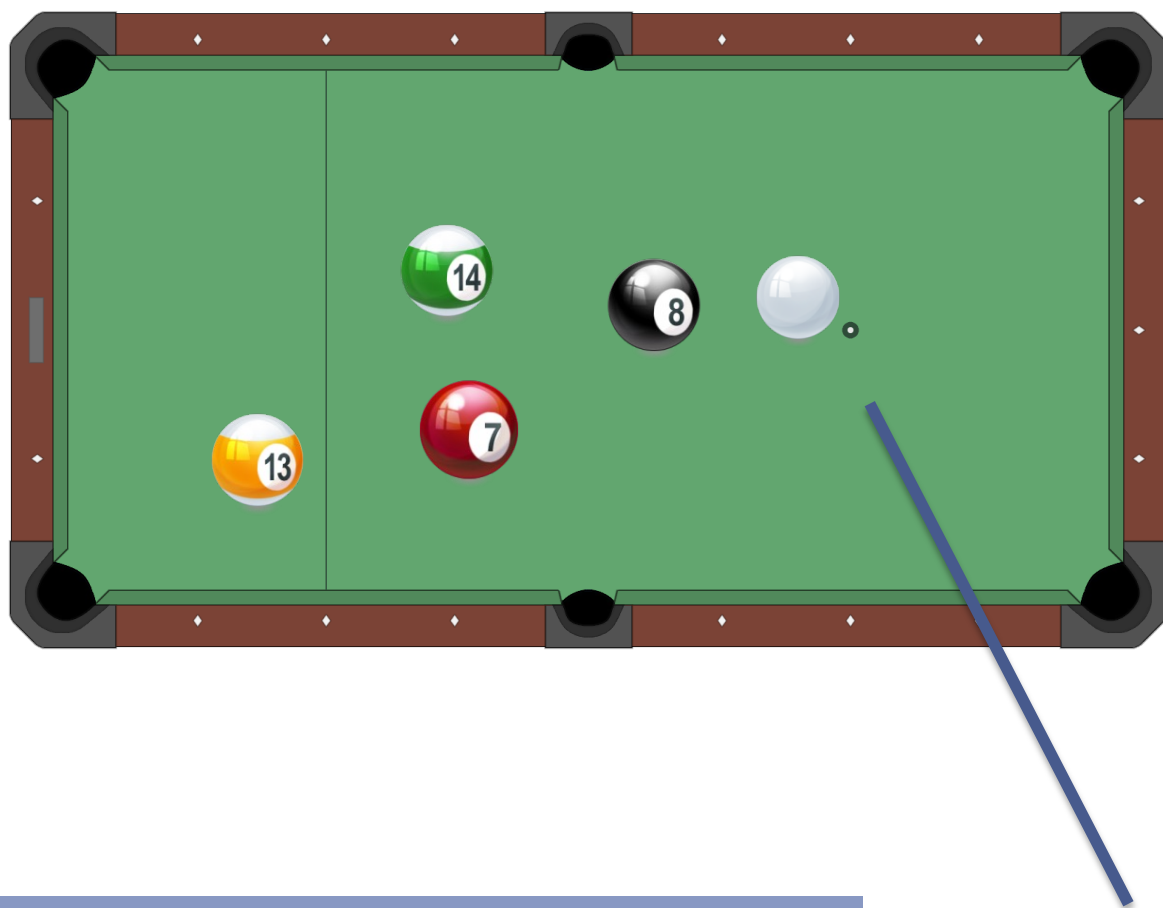
Error Back-Propagation



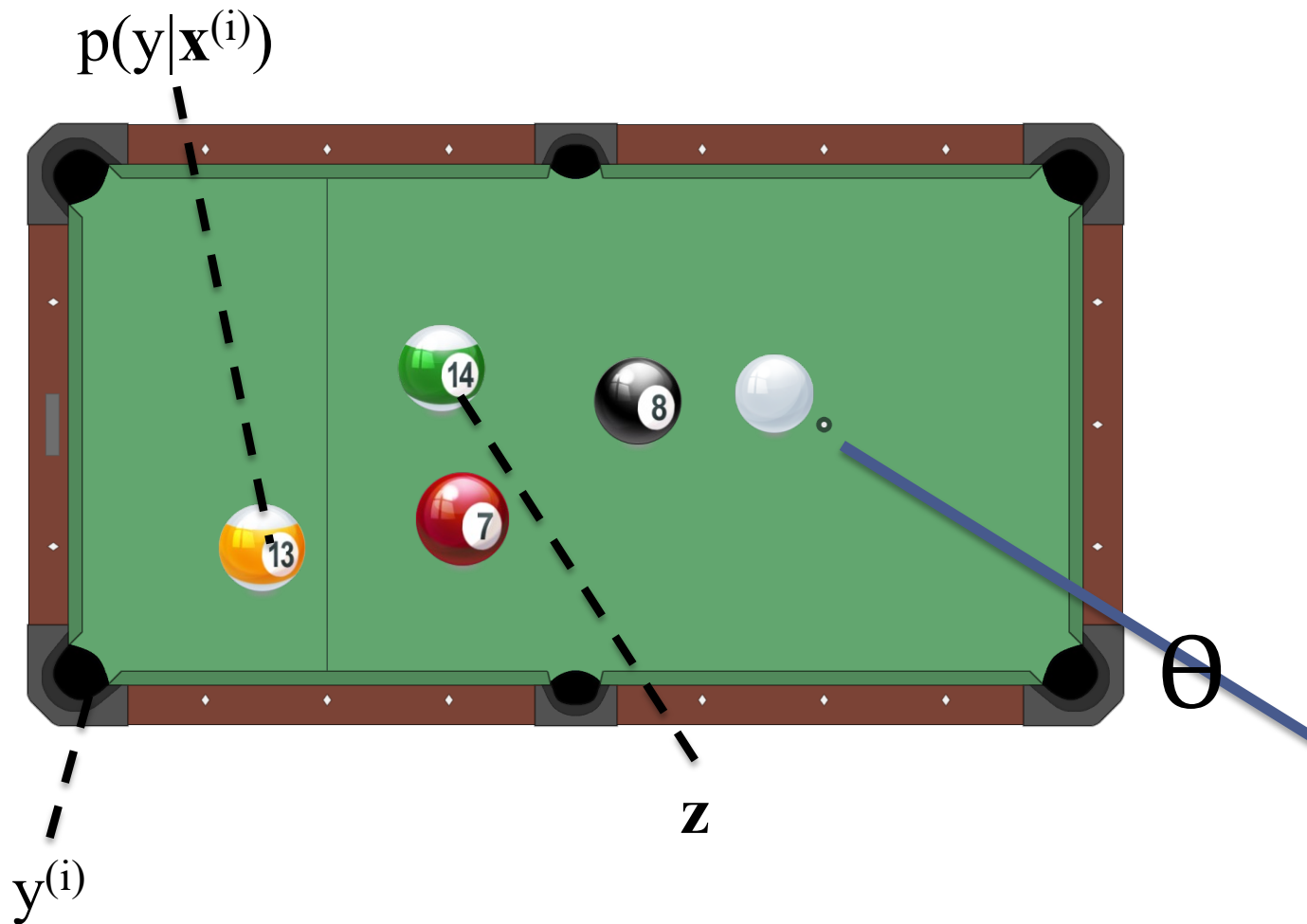
Error Back-Propagation



Error Back-Propagation



Error Back-Propagation



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.**

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