



10-301/601 Introduction to Machine Learning

Machine Learning Department
School of Computer Science
Carnegie Mellon University

Backpropagation

Matt Gormley
Lecture 13
Oct. 9, 2022

Reminders

- **Post-Exam Followup:**
 - Exam Viewing
 - Exit Poll: Exam 1
 - Grade Summary 1
- **Homework 4: Logistic Regression**
 - Out: Tue, Oct 4
 - Due: Thu, Oct 13 at 11:59pm
- **Homework 5: Neural Networks**
 - Out: Thu, Oct 13
 - Due: Thu, Oct 27 at 11:59pm

THE CHAIN RULE OF CALCULUS

Training

Chain Rule

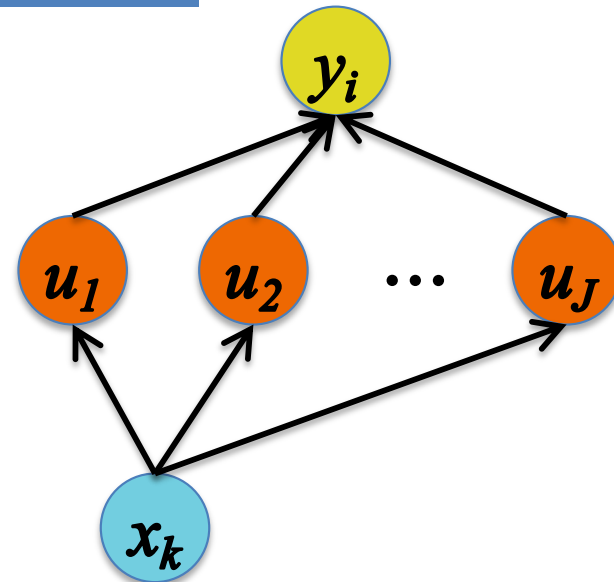
Whiteboard

– Chain Rule of Calculus

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

Chain Rule:

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

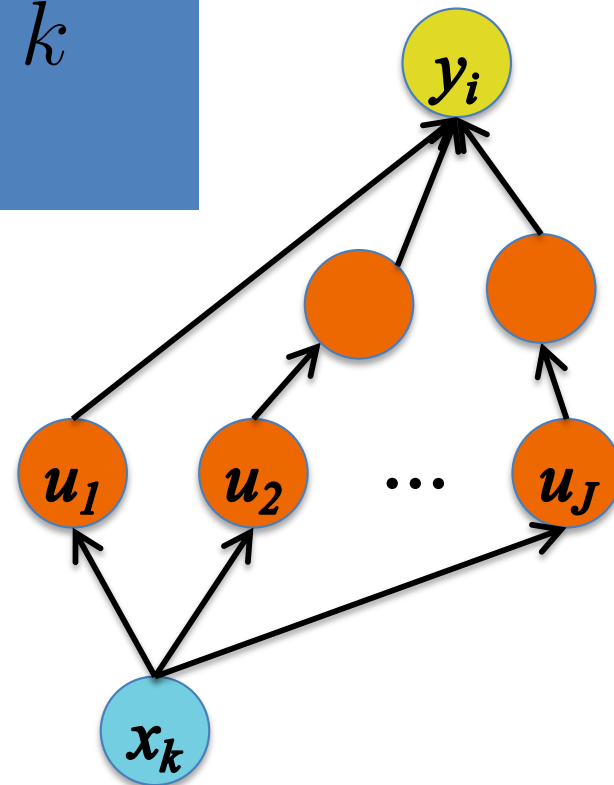


Given: $y = g(u)$ and $u = h(x)$.

Chain Rule:

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

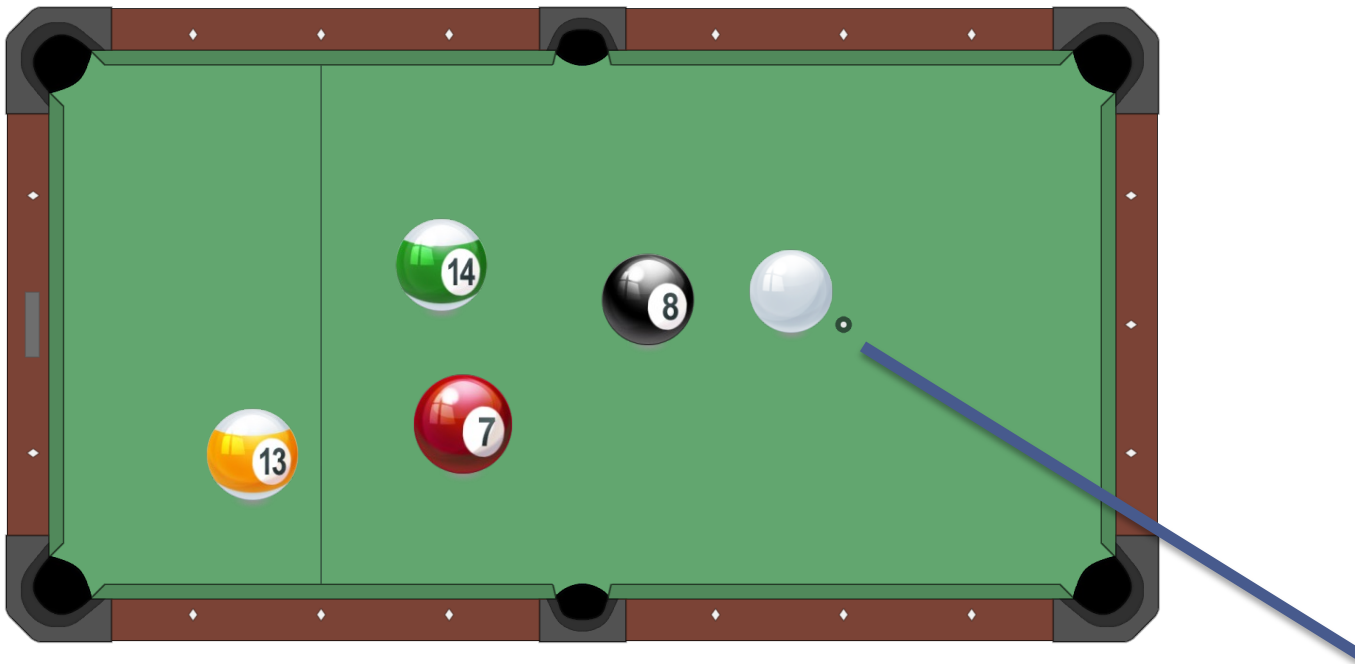
Backpropagation is just repeated application of the **chain rule** from Calculus 101.



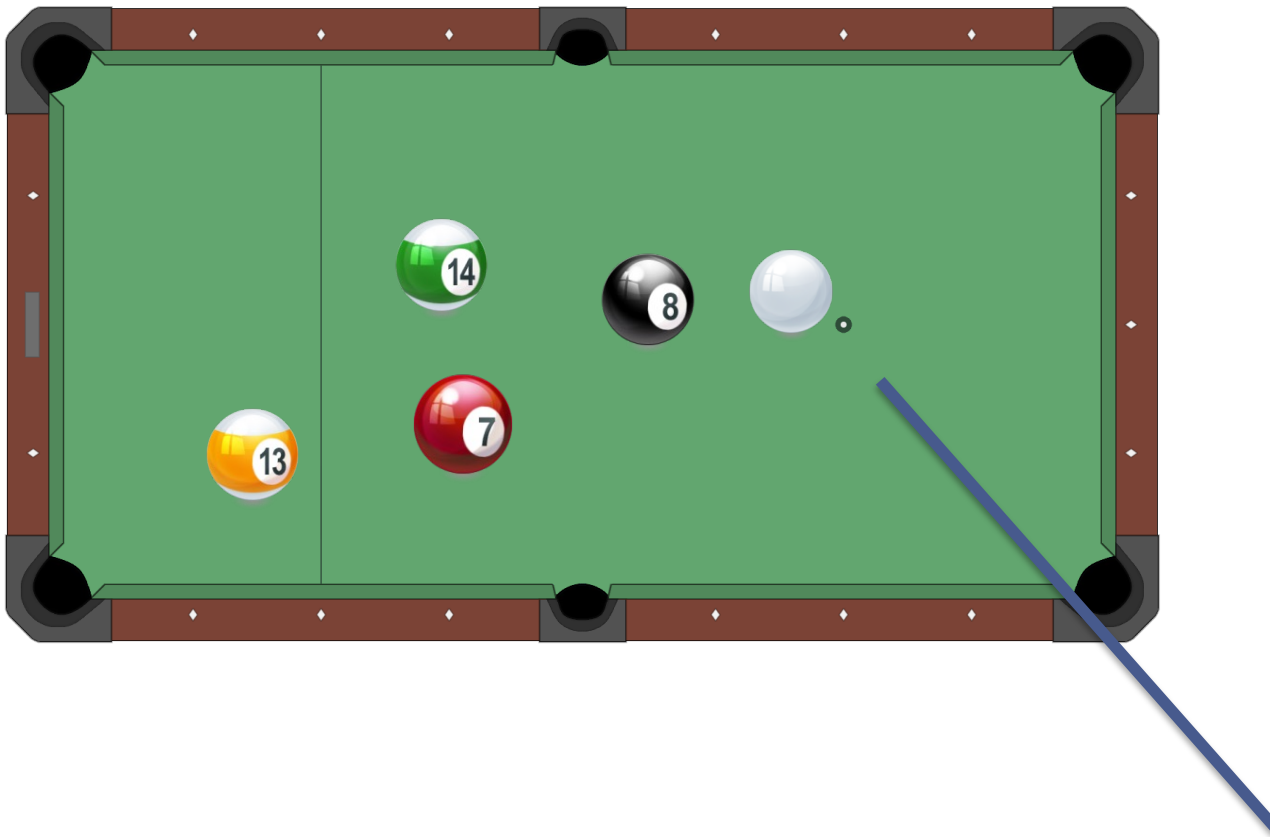
Intuitions

BACKPROPAGATION OF ERRORS

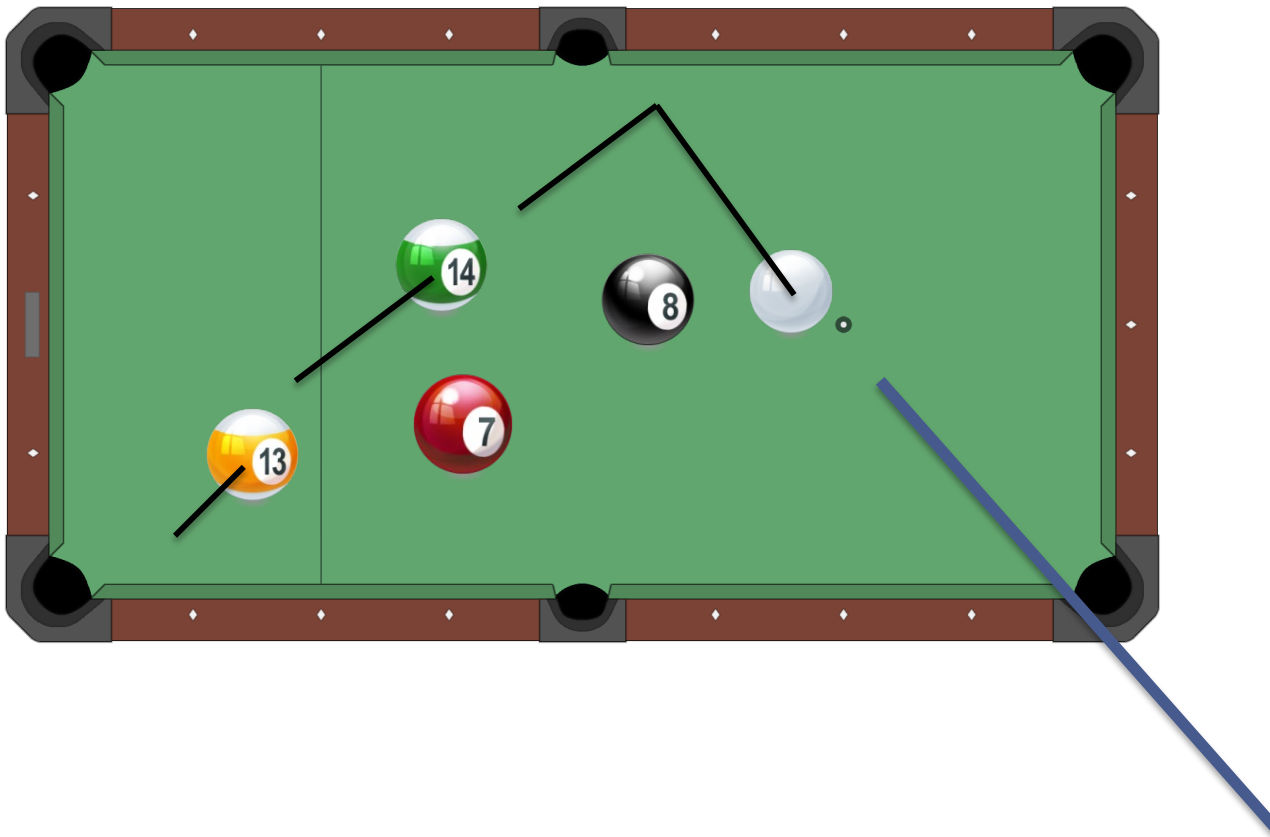
Error Back-Propagation



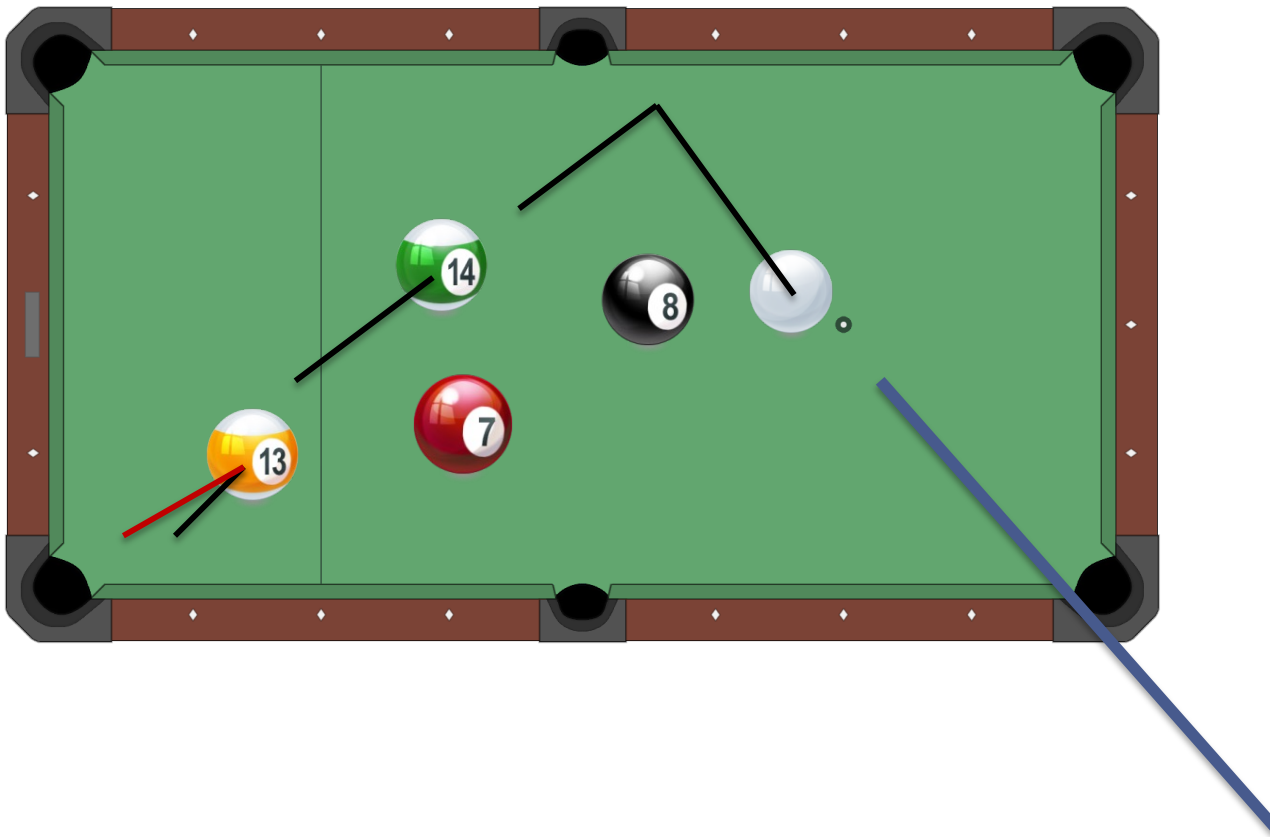
Error Back-Propagation



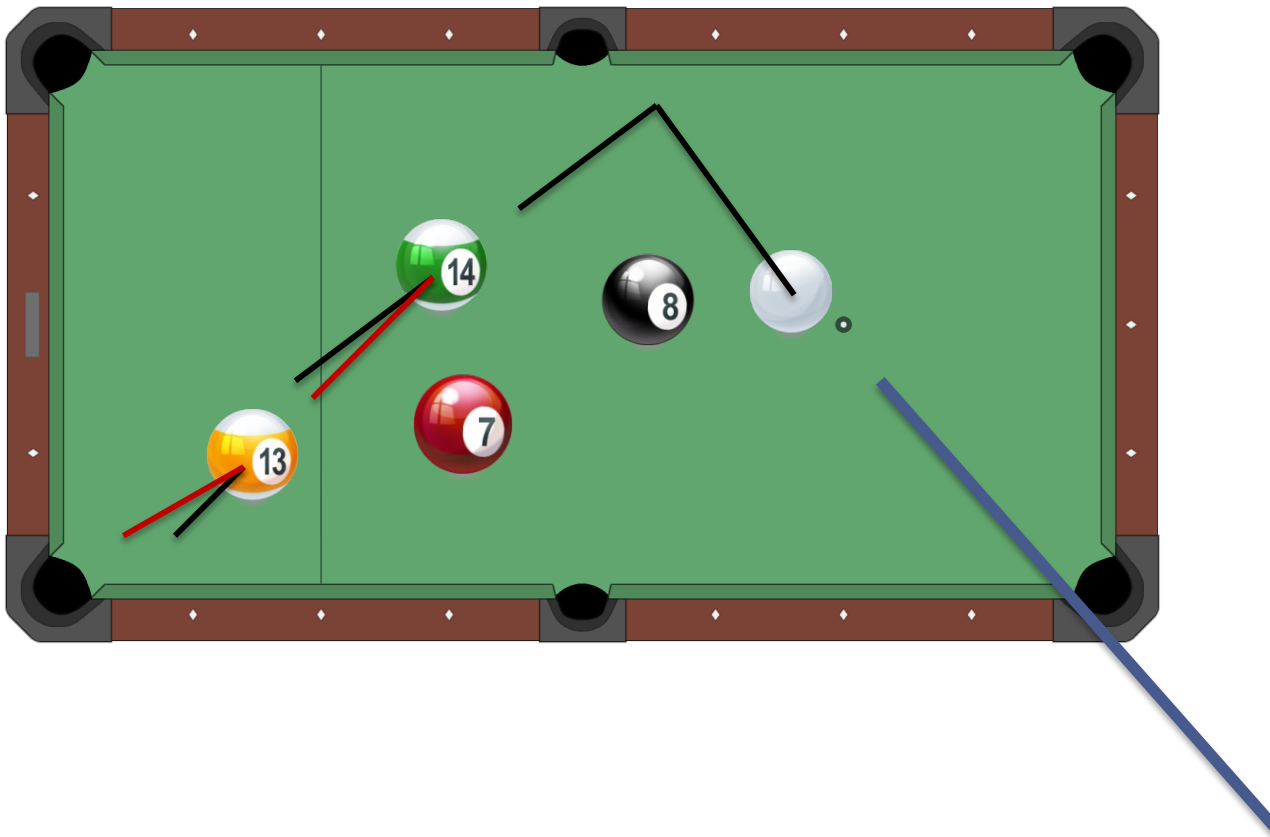
Error Back-Propagation



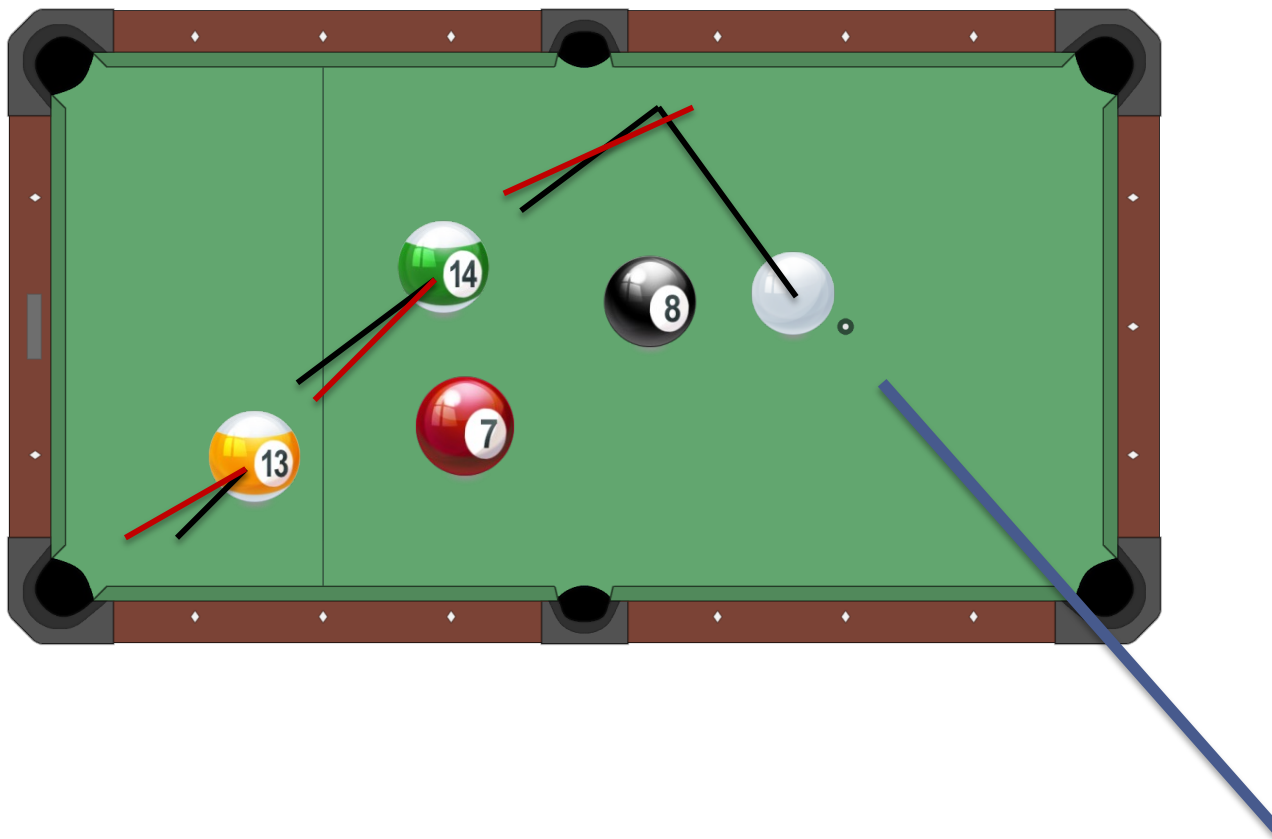
Error Back-Propagation



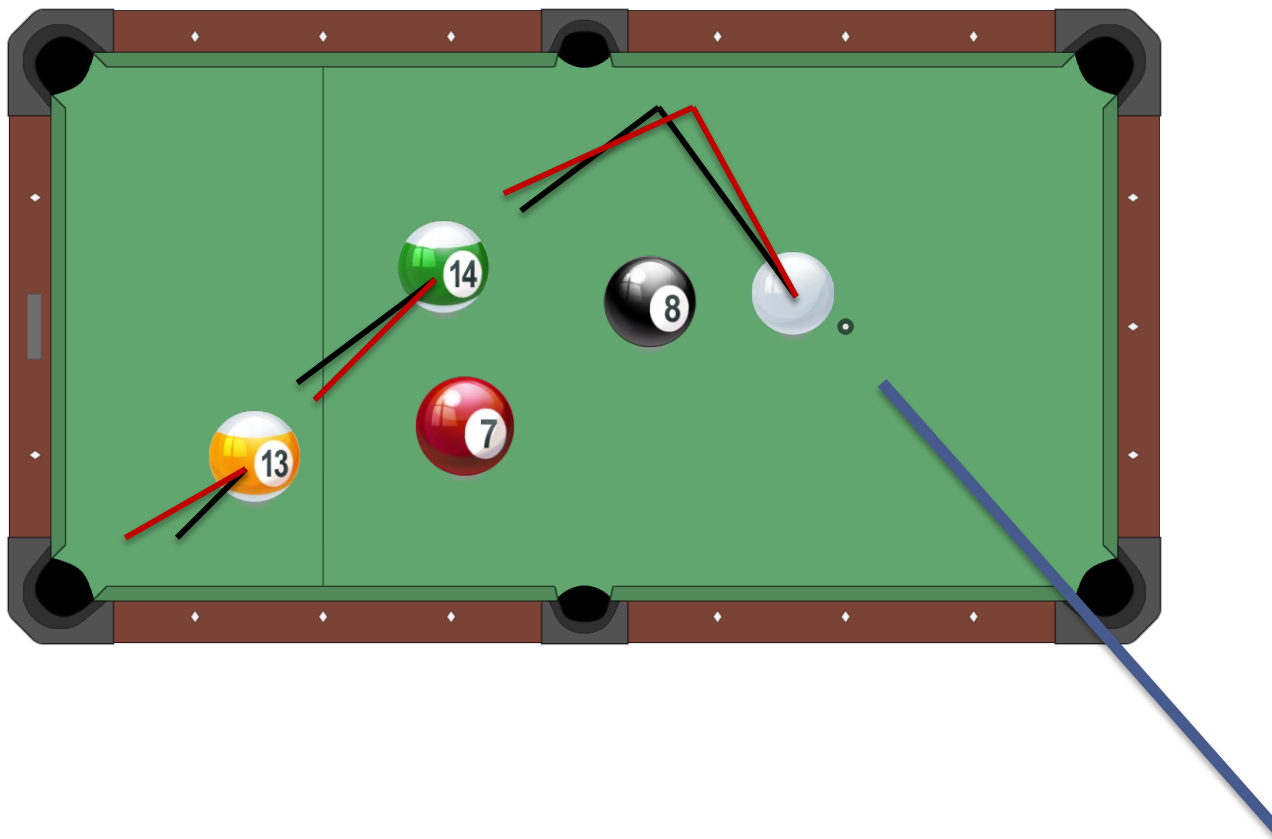
Error Back-Propagation



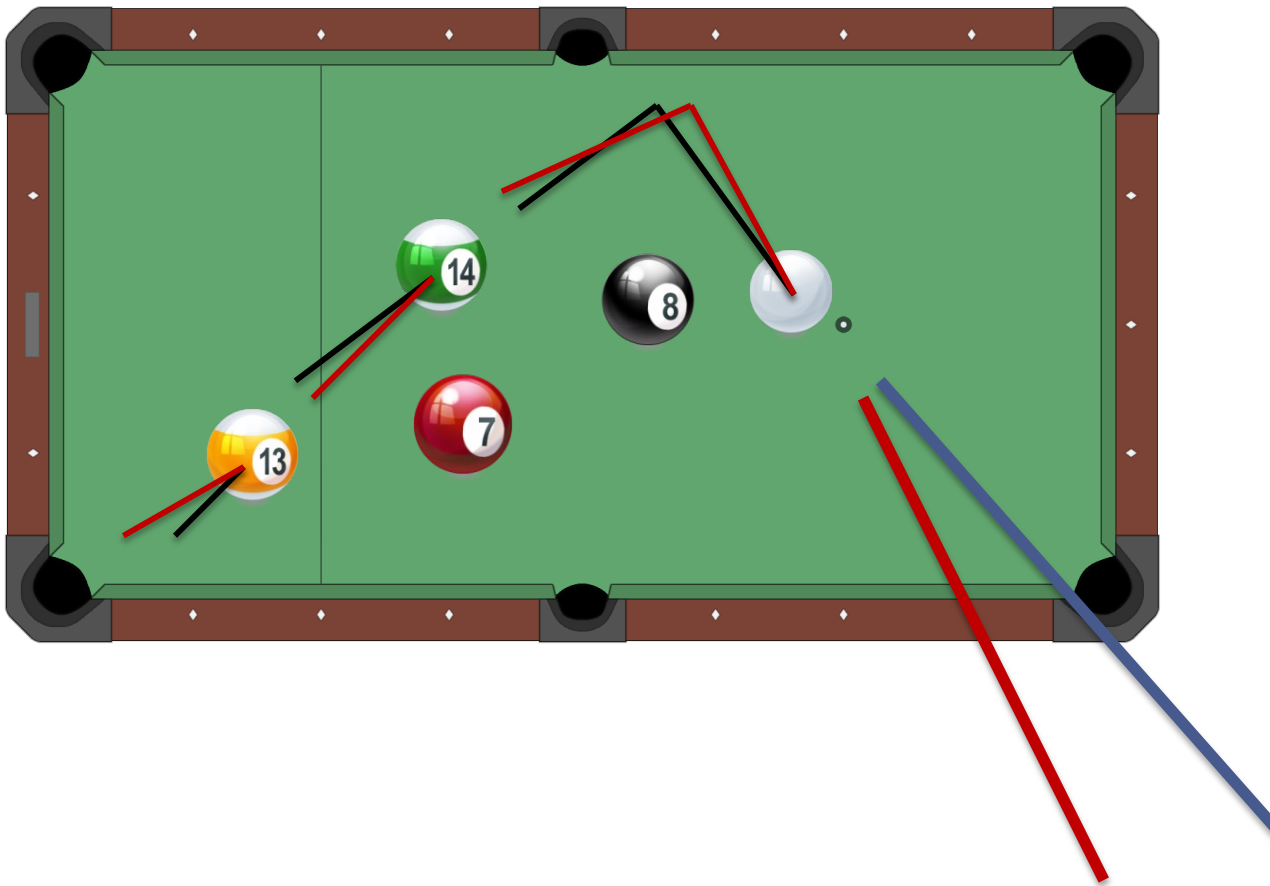
Error Back-Propagation



Error Back-Propagation



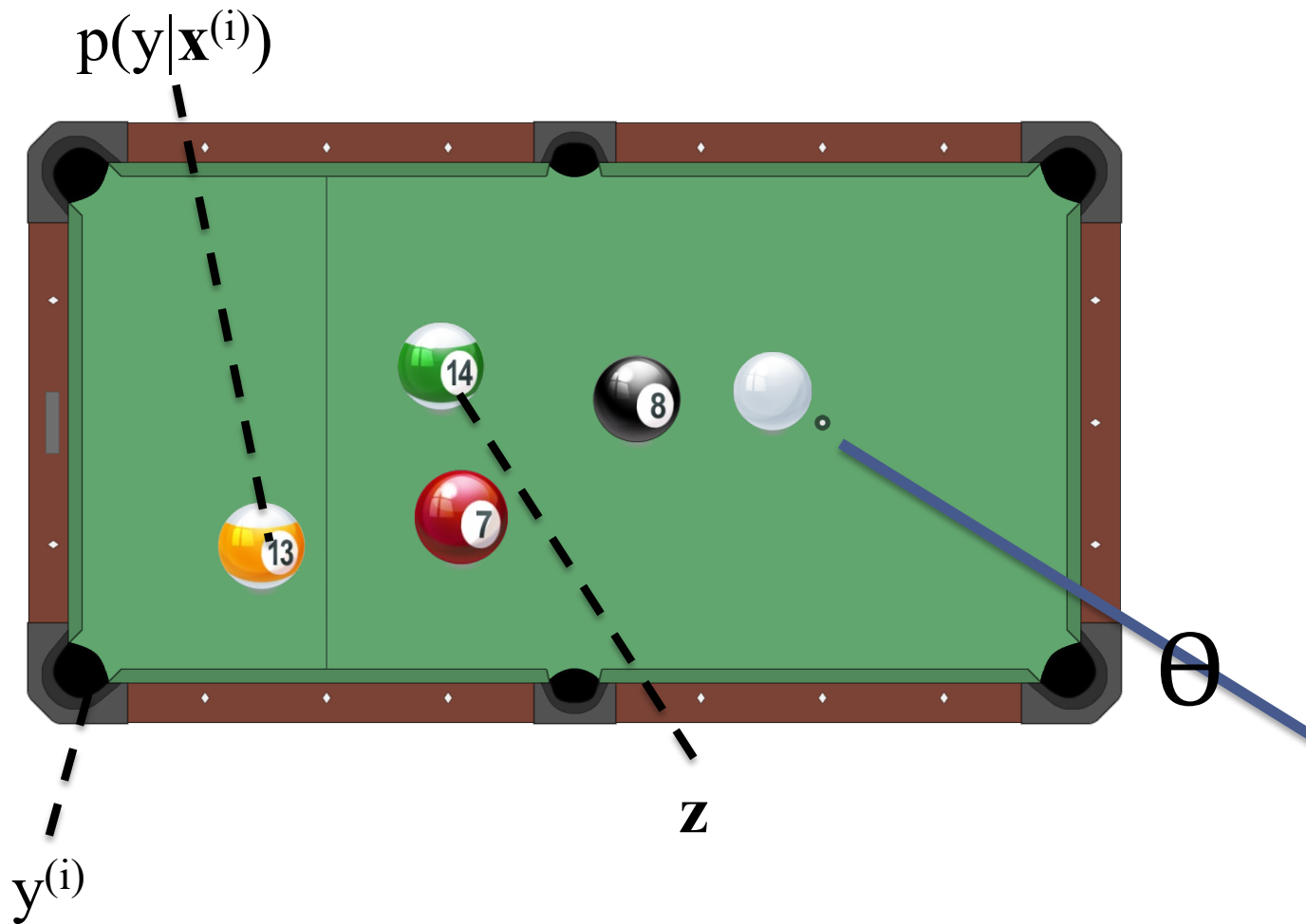
Error Back-Propagation



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.**

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

Algorithm

BACKPROPAGATION FOR A COMPUTATION GRAPH

Whiteboard

- Backpropagation on a simple 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.**

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

Simple Example: The goal is to compute $J = \cos(\sin(x^2) + 3x^2)$ on the forward pass and the derivative $\frac{dJ}{dx}$ on the backward pass.

Forward

$$J = \cos(u)$$

$$u = u_1 + u_2$$

$$u_1 = \sin(t)$$

$$u_2 = 3t$$

$$t = x^2$$

Training

Backpropagation

Simple Example: The goal is to compute $J = \cos(\sin(x^2) + 3x^2)$ on the forward pass and the derivative $\frac{dJ}{dx}$ on the backward pass.

Forward

Backward

$$J = \cos(u)$$

$$\frac{dJ}{du} += -\sin(u)$$

$$u = u_1 + u_2$$

$$\frac{dJ}{du_1} += \frac{dJ}{du} \frac{du}{du_1}, \quad \frac{du}{du_1} = 1 \qquad \frac{dJ}{du_2} += \frac{dJ}{du} \frac{du}{du_2}, \quad \frac{du}{du_2} = 1$$

$$u_1 = \sin(t)$$

$$\frac{dJ}{dt} += \frac{dJ}{du_1} \frac{du_1}{dt}, \quad \frac{du_1}{dt} = \cos(t)$$

$$u_2 = 3t$$

$$\frac{dJ}{dt} += \frac{dJ}{du_2} \frac{du_2}{dt}, \quad \frac{du_2}{dt} = 3$$

$$t = x^2$$

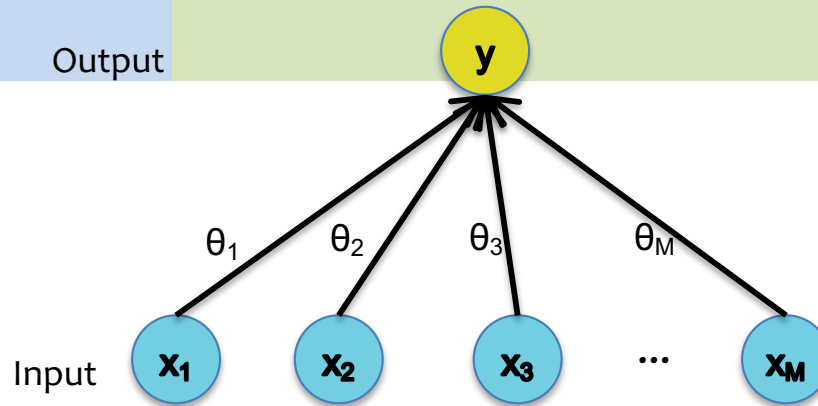
$$\frac{dJ}{dx} += \frac{dJ}{dt} \frac{dt}{dx}, \quad \frac{dt}{dx} = 2x$$

Training

Backpropagation

Output

Case 1:
Logistic
Regression



Forward

$$J = y^* \log y + (1 - y^*) \log(1 - y)$$

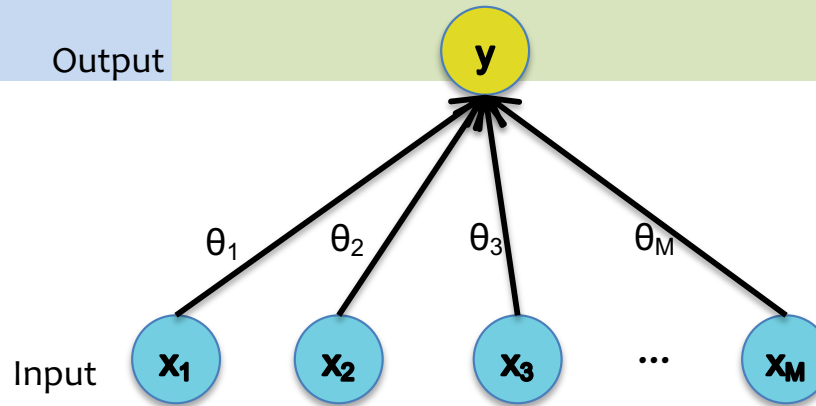
$$y = \frac{1}{1 + \exp(-a)}$$

$$a = \sum_{j=0}^D \theta_j x_j$$

Training

Backpropagation

Case 1: Logistic Regression



Forward

$$J = y^* \log y + (1 - y^*) \log(1 - y)$$

$$y = \frac{1}{1 + \exp(-a)}$$

$$a = \sum_{j=0}^D \theta_j x_j$$

Backward

$$\frac{dJ}{dy} = \frac{y^*}{y} + \frac{(1 - y^*)}{y - 1}$$

$$\frac{dJ}{da} = \frac{dJ}{dy} \frac{dy}{da}, \quad \frac{dy}{da} = \frac{\exp(-a)}{(\exp(-a) + 1)^2}$$

$$\frac{dJ}{d\theta_j} = \frac{dJ}{da} \frac{da}{d\theta_j}, \quad \frac{da}{d\theta_j} = x_j$$

$$\frac{dJ}{dx_j} = \frac{dJ}{da} \frac{da}{dx_j}, \quad \frac{da}{dx_j} = \theta_j$$

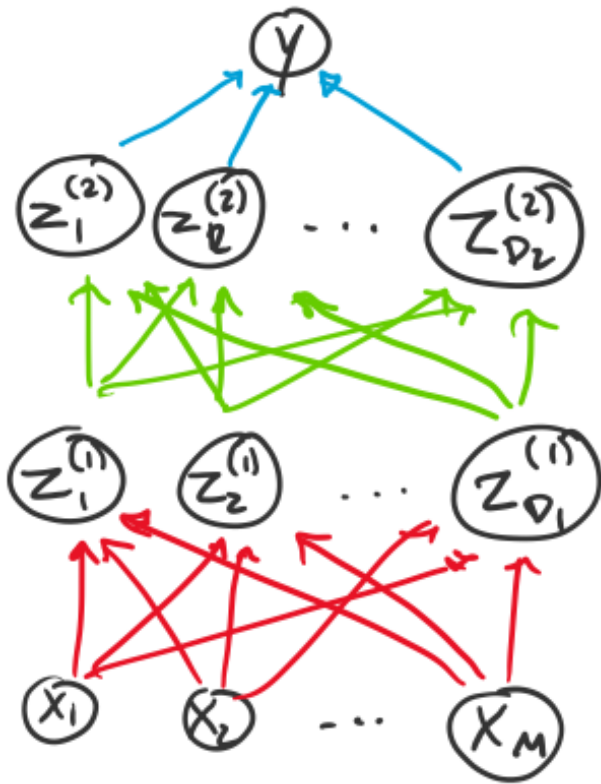
A 2-Hidden Layer Neural Network

TRAINING / FORWARD COMPUTATION / BACKWARD COMPUTATION

Training Backpropagation

Recall: Our 2-Hidden Layer Neural Network

Question: How do we train this model?



$$\begin{aligned}\beta &\in \mathbb{R}^{D_2} \\ \beta_0 &\in \mathbb{R} \\ \alpha^{(2)} &\in \mathbb{R}^{D_1 \times D_2} \\ \vec{b}^{(2)} &\in \mathbb{R}^{D_2}\end{aligned}$$

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

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

$$\begin{aligned}\alpha^{(1)} &\in \mathbb{R}^{M \times D_1} \\ \vec{b}^{(1)} &\in \mathbb{R}^{D_1}\end{aligned}$$

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

Training Backpropagation

Whiteboard

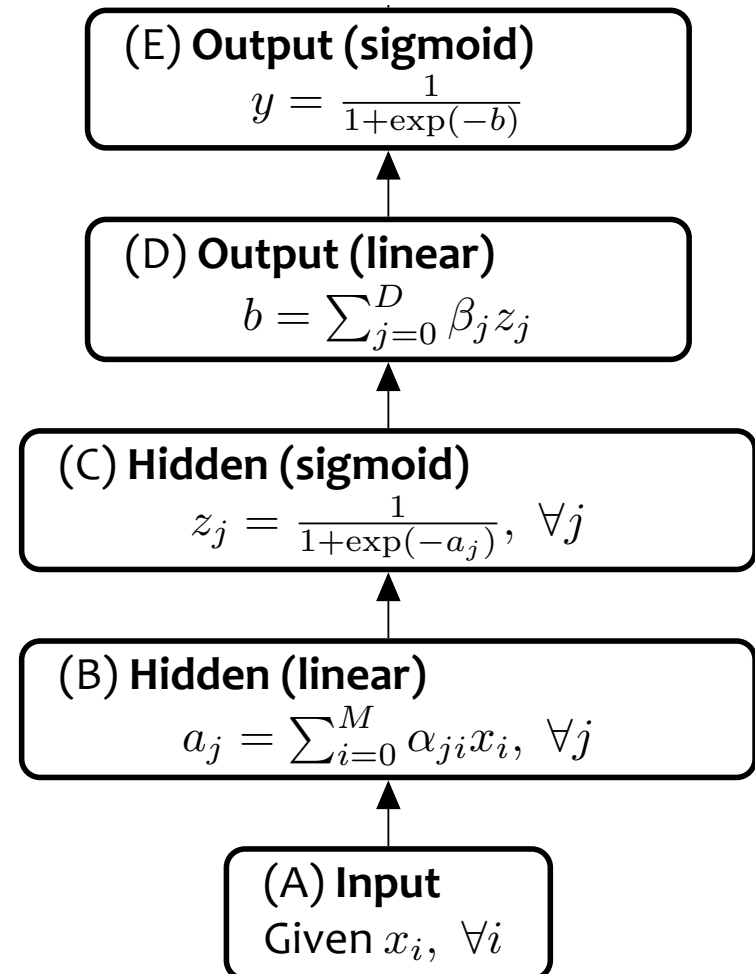
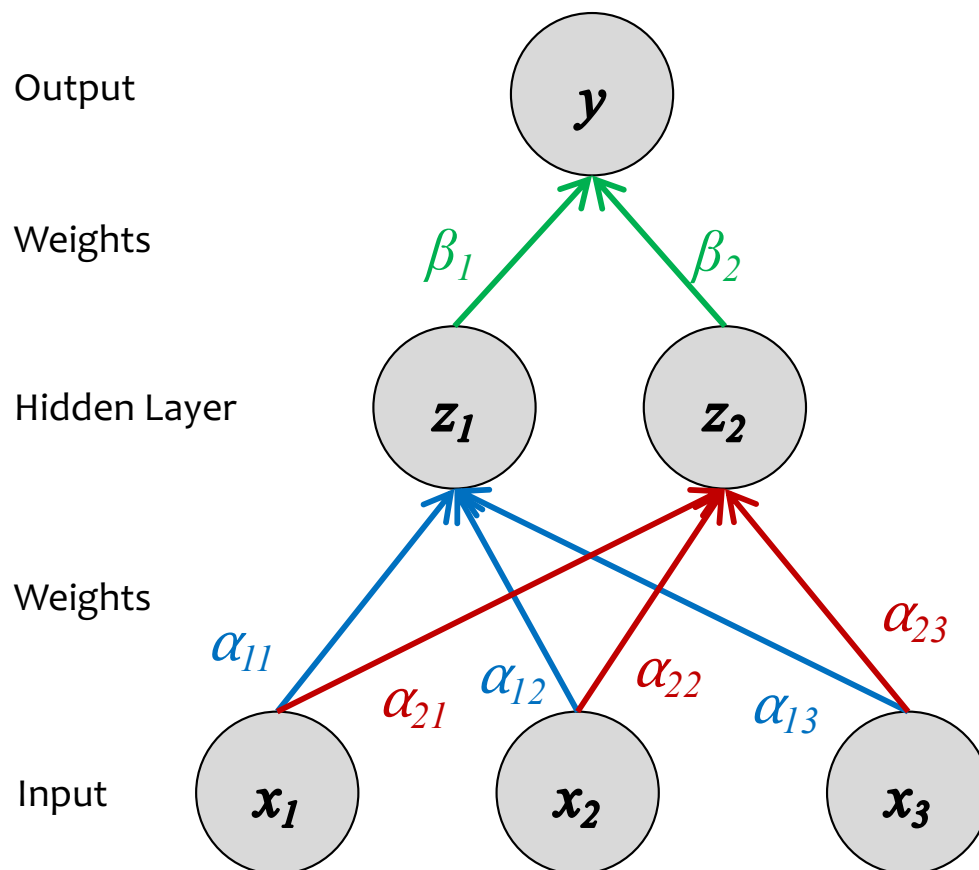
- Example: Backpropagation for Neural Network with 2-Hidden Layers
 - SGD Training
 - Forward Computation
 - Computation Graph
 - Backward Computation

A 1-Hidden Layer Neural Network

TRAINING A NEURAL NETWORK

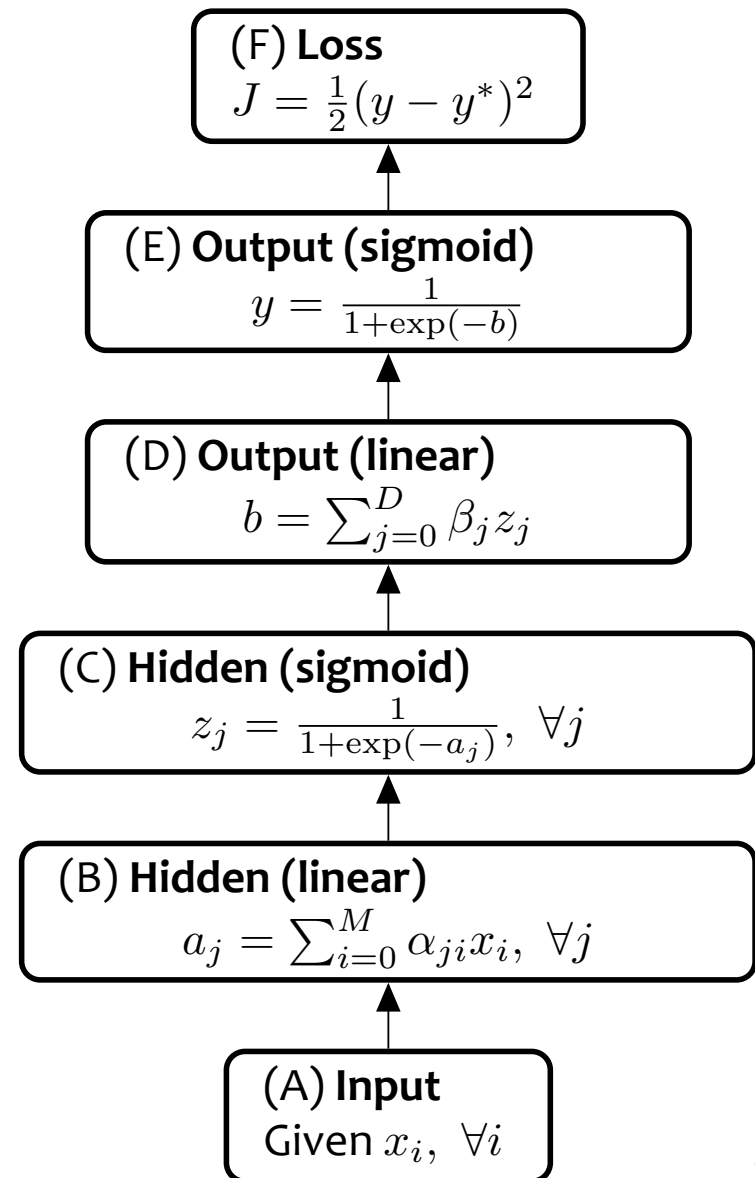
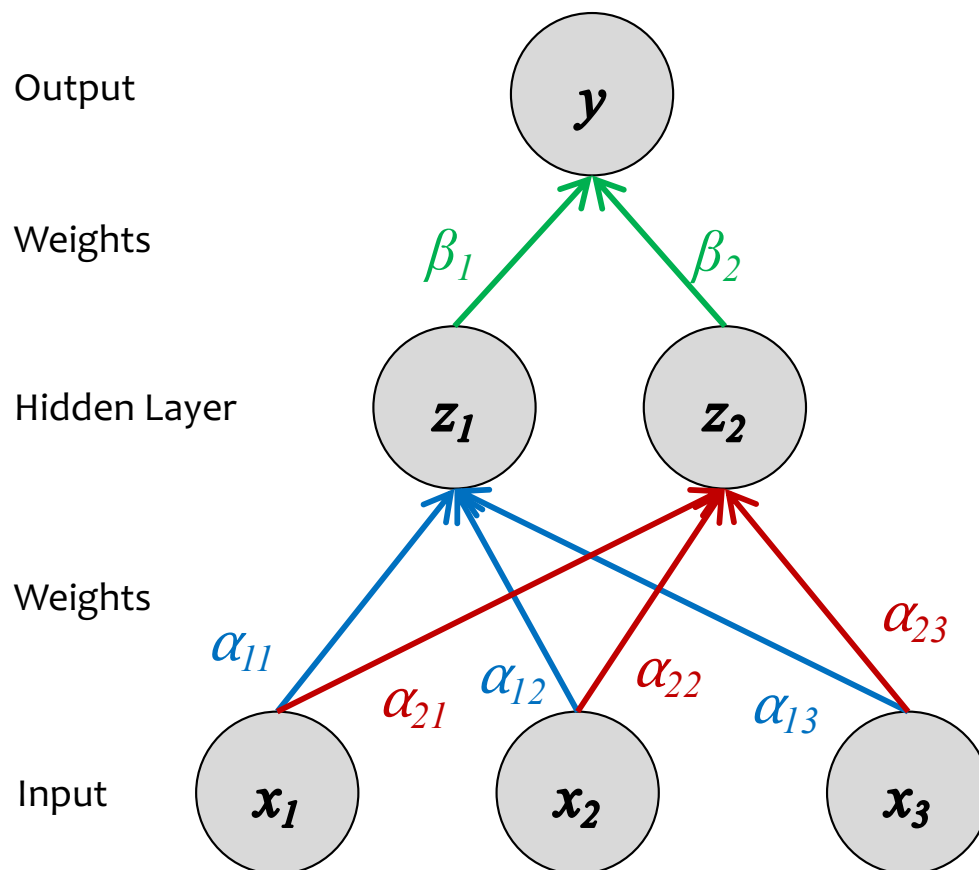
Training

Backpropagation



Training

Backpropagation



Example: 1-Hidden Layer Neural Network

Algorithm 1 Stochastic Gradient Descent (SGD)

```

1: procedure SGD(Training data  $\mathcal{D}$ , test data  $\mathcal{D}_t$ )
2:   Initialize parameters  $\alpha, \beta$ 
3:   for  $e \in \{1, 2, \dots, E\}$  do
4:     for  $(\mathbf{x}, \mathbf{y}) \in \mathcal{D}$  do
5:       Compute neural network layers:
6:        $\mathbf{o} = \text{object}(\mathbf{x}, \mathbf{a}, \mathbf{b}, \mathbf{z}, \hat{\mathbf{y}}, J) = \text{NNFORWARD}(\mathbf{x}, \mathbf{y}, \alpha, \beta)$ 
7:       Compute gradients via backprop:
8:        $\left. \begin{array}{l} \mathbf{g}_\alpha = \nabla_\alpha J \\ \mathbf{g}_\beta = \nabla_\beta J \end{array} \right\} = \text{NNBACKWARD}(\mathbf{x}, \mathbf{y}, \alpha, \beta, \mathbf{o})$ 
9:       Update parameters:
10:       $\alpha \leftarrow \alpha - \gamma \mathbf{g}_\alpha$ 
11:       $\beta \leftarrow \beta - \gamma \mathbf{g}_\beta$ 
12:      Evaluate training mean cross-entropy  $J_{\mathcal{D}}(\alpha, \beta)$ 
13:      Evaluate test mean cross-entropy  $J_{\mathcal{D}_t}(\alpha, \beta)$ 
14:   return parameters  $\alpha, \beta$ 

```

A 1-Hidden Layer Neural Network

FORWARD COMPUTATION FOR A NEURAL NETWORK

Example: 1-Hidden Layer Neural Network

Algorithm 1 Stochastic Gradient Descent (SGD)

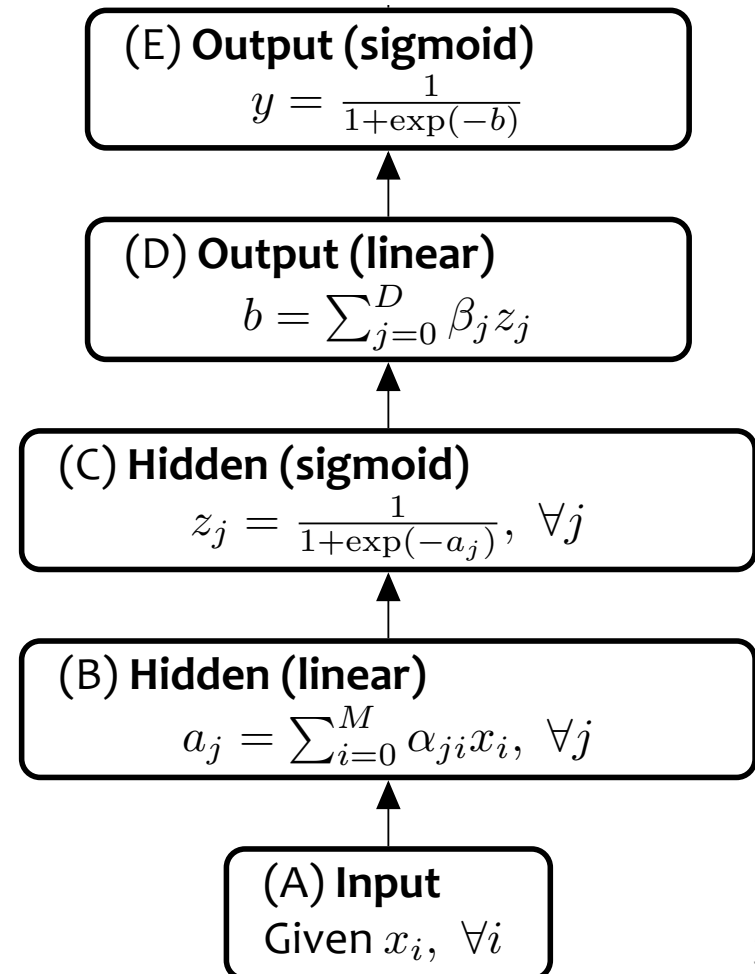
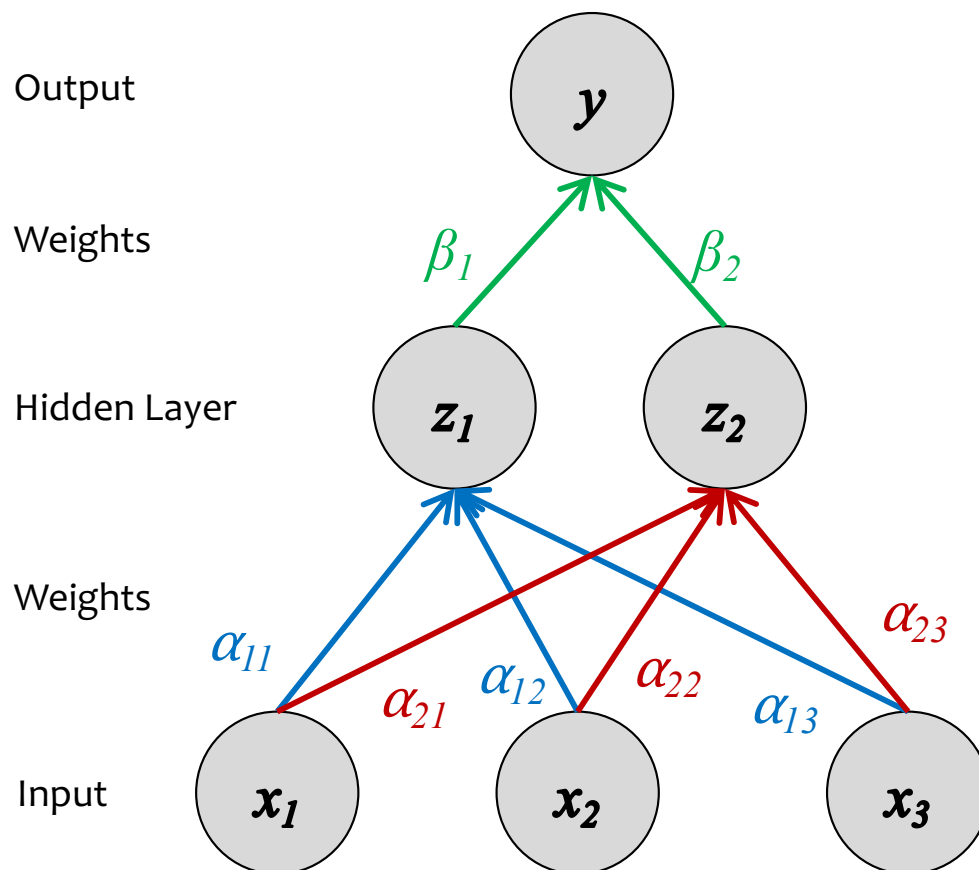
```

1: procedure SGD(Training data  $\mathcal{D}$ , test data  $\mathcal{D}_t$ )
2:   Initialize parameters  $\alpha, \beta$ 
3:   for  $e \in \{1, 2, \dots, E\}$  do
4:     for  $(\mathbf{x}, \mathbf{y}) \in \mathcal{D}$  do
5:       Compute neural network layers:
6:        $\mathbf{o} = \text{object}(\mathbf{x}, \mathbf{a}, \mathbf{b}, \mathbf{z}, \hat{\mathbf{y}}, J) = \text{NNFORWARD}(\mathbf{x}, \mathbf{y}, \alpha, \beta)$ 
7:       Compute gradients via backprop:
8:        $\left. \begin{array}{l} \mathbf{g}_\alpha = \nabla_\alpha J \\ \mathbf{g}_\beta = \nabla_\beta J \end{array} \right\} = \text{NNBACKWARD}(\mathbf{x}, \mathbf{y}, \alpha, \beta, \mathbf{o})$ 
9:       Update parameters:
10:       $\alpha \leftarrow \alpha - \gamma \mathbf{g}_\alpha$ 
11:       $\beta \leftarrow \beta - \gamma \mathbf{g}_\beta$ 
12:      Evaluate training mean cross-entropy  $J_{\mathcal{D}}(\alpha, \beta)$ 
13:      Evaluate test mean cross-entropy  $J_{\mathcal{D}_t}(\alpha, \beta)$ 
14:   return parameters  $\alpha, \beta$ 

```

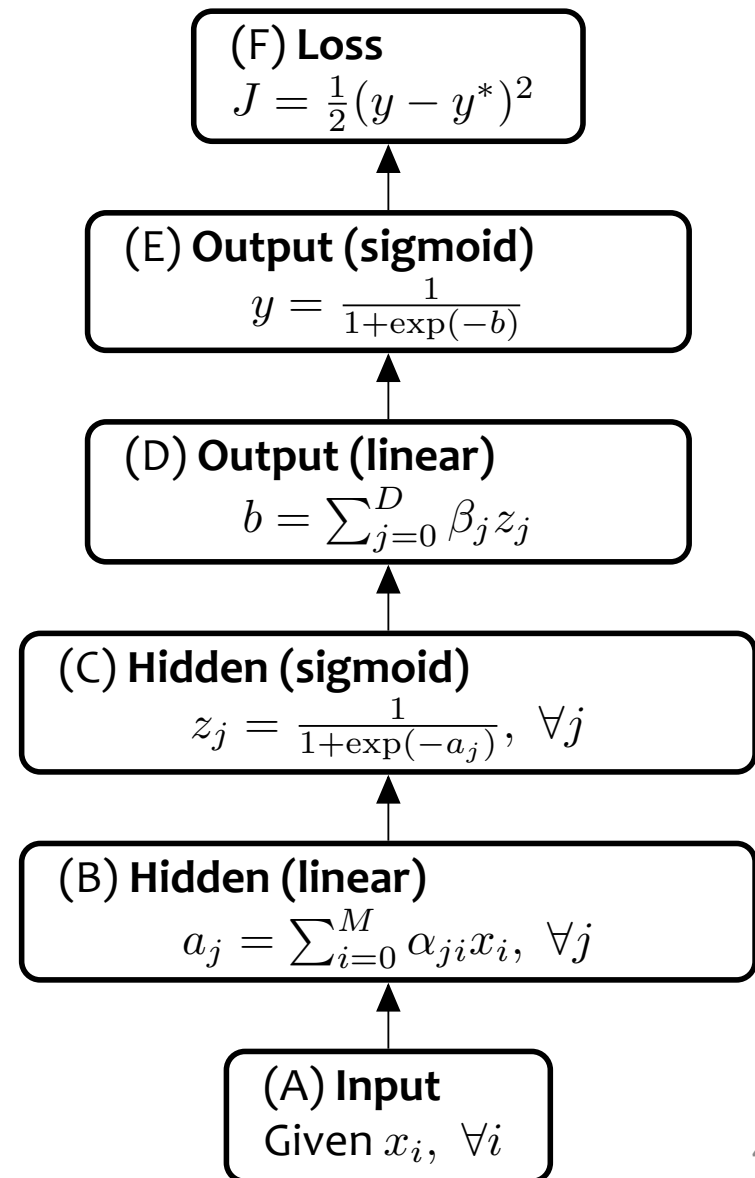
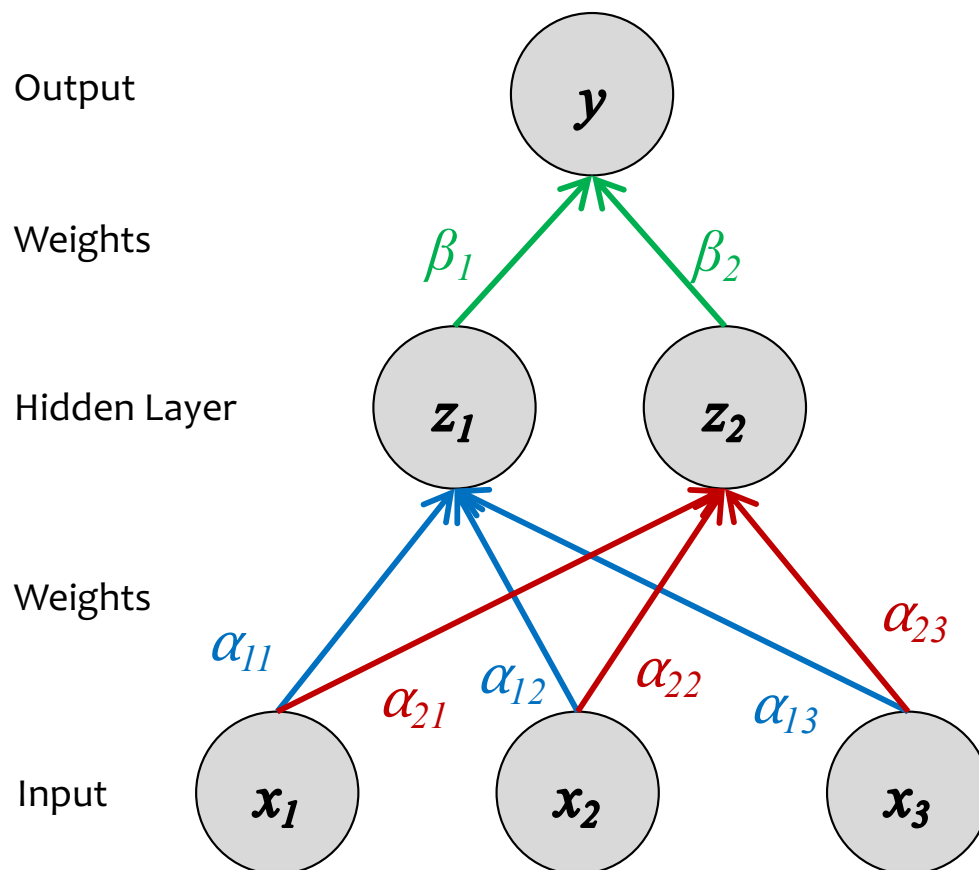
Training

Backpropagation



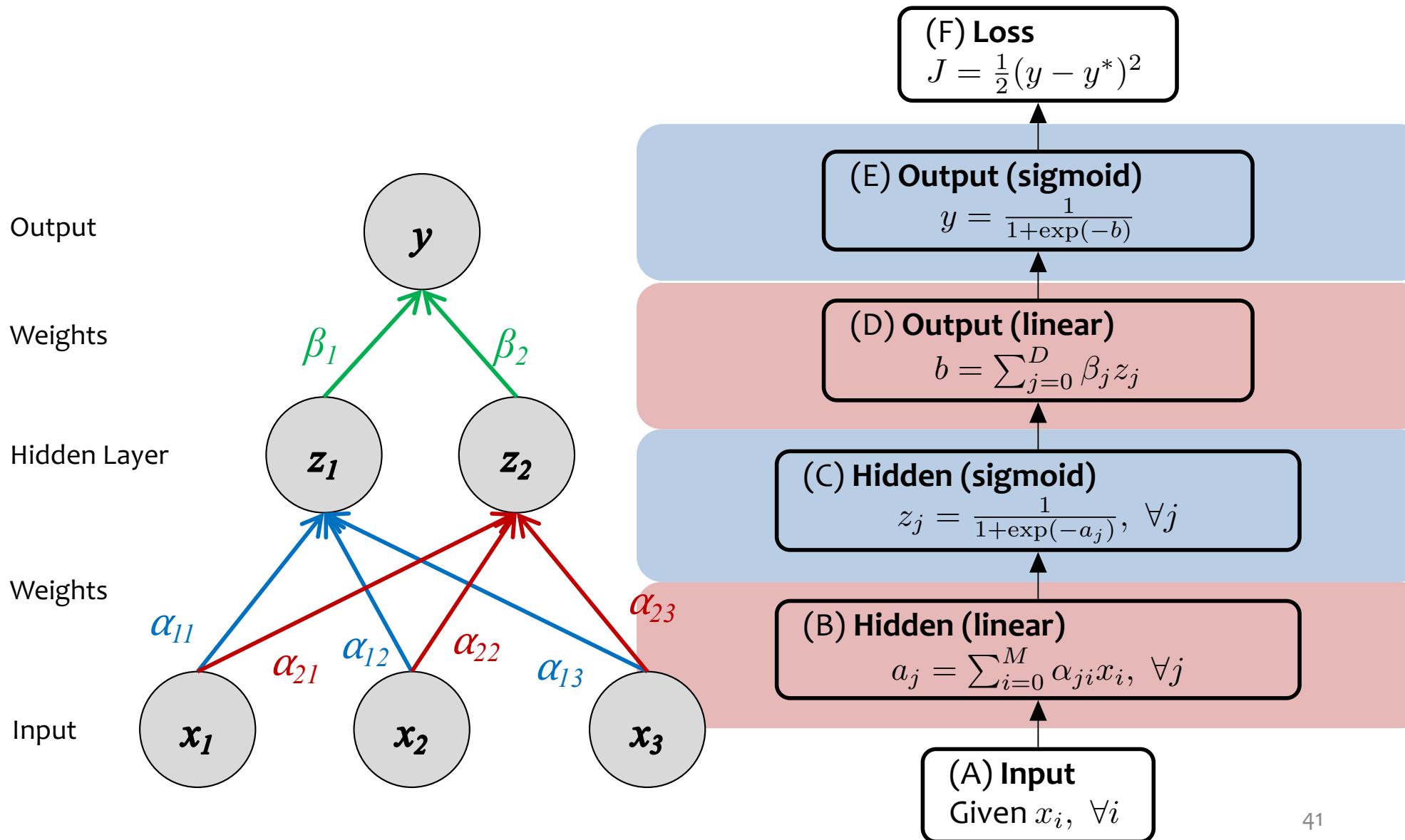
Training

Backpropagation



Training

Backpropagation

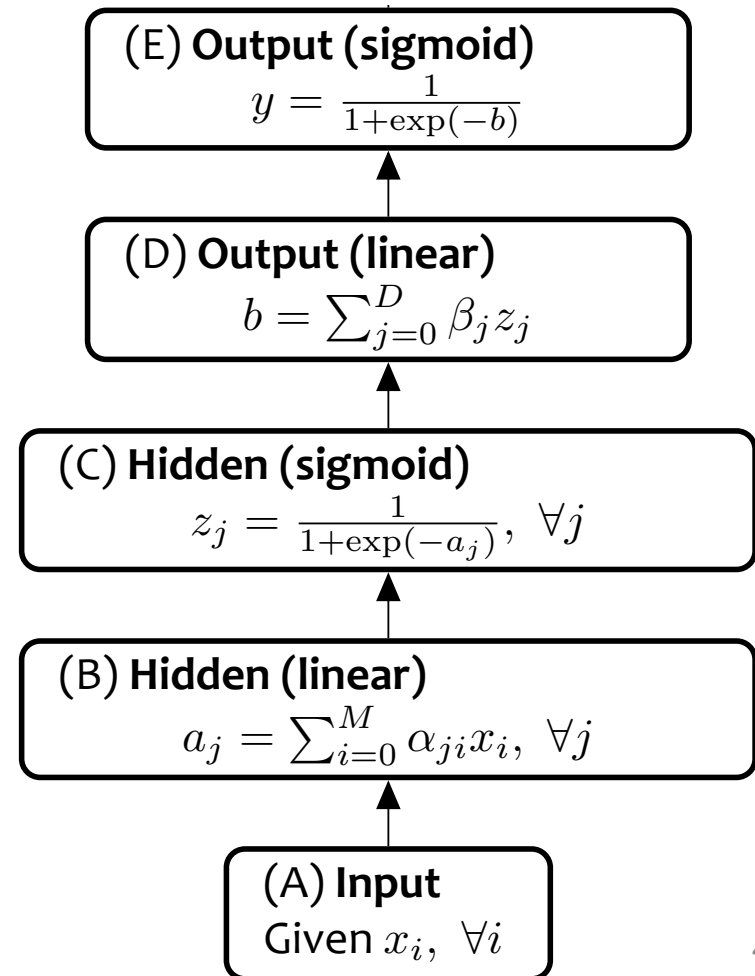
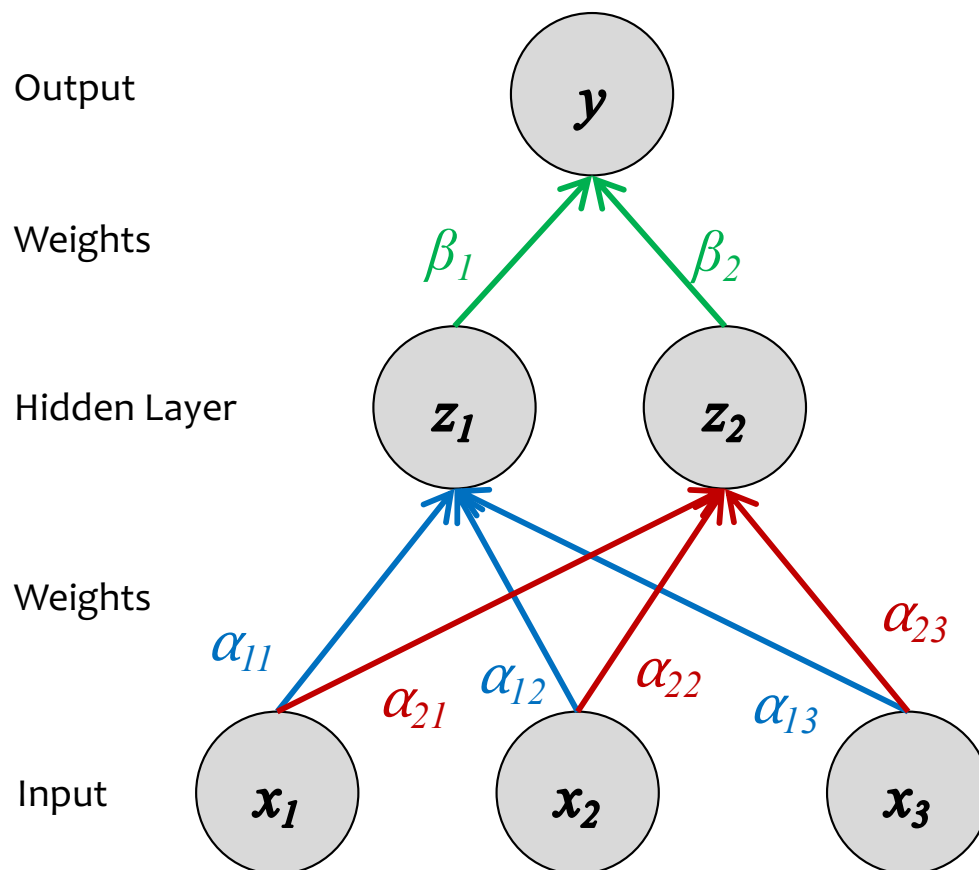


A 1-Hidden Layer Neural Network

BACKPROPAGATION FOR A NEURAL NETWORK

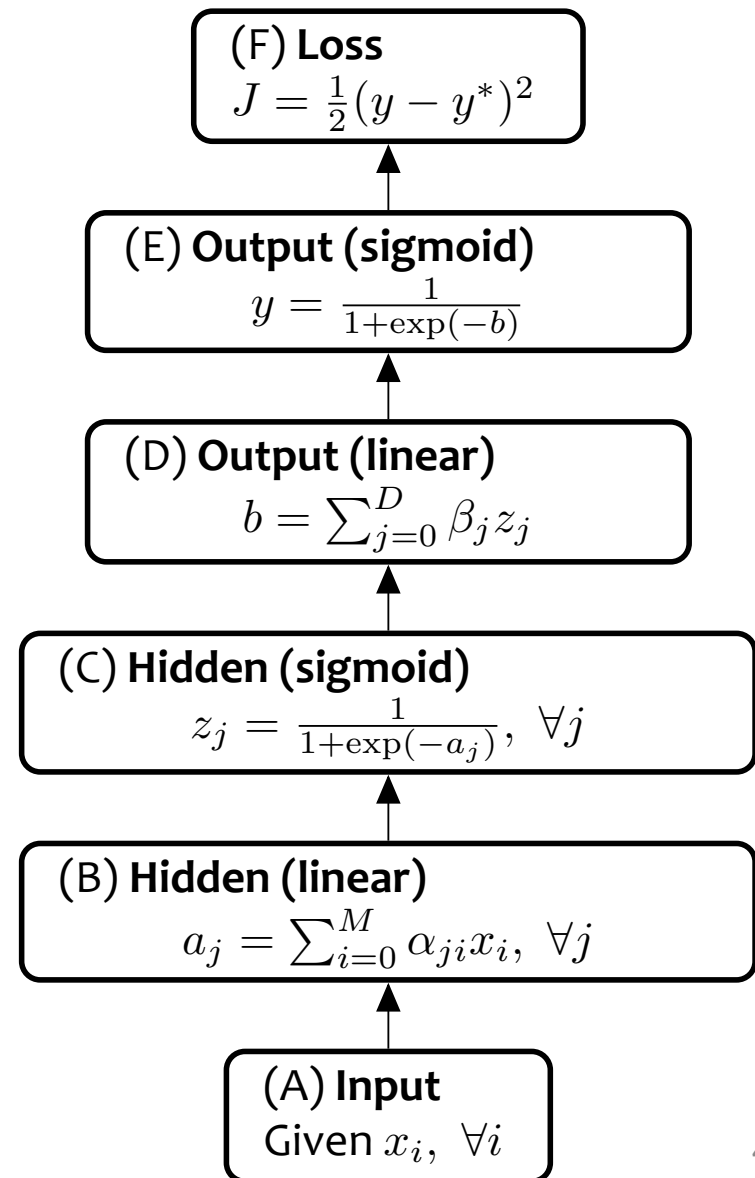
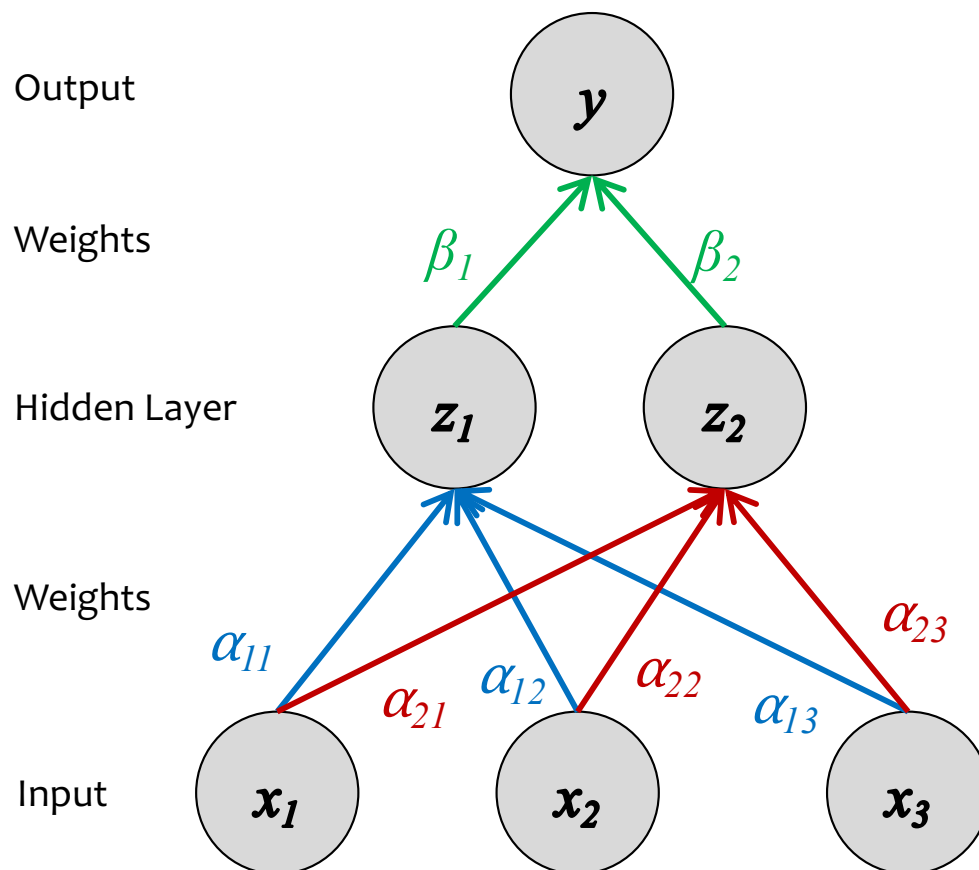
Training

Backpropagation



Training

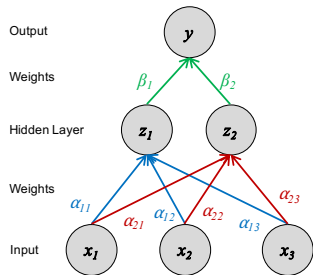
Backpropagation



Training

Backpropagation

Case 2: Neural Network



Forward

$$J = y^* \log y + (1 - y^*) \log(1 - y)$$

$$y = \frac{1}{1 + \exp(-b)}$$

$$b = \sum_{j=0}^D \beta_j z_j$$

$$z_j = \frac{1}{1 + \exp(-a_j)}$$

$$a_j = \sum_{i=0}^M \alpha_{ji} x_i$$

Backward

$$\frac{dJ}{dy} = \frac{y^*}{y} + \frac{(1 - y^*)}{y - 1}$$

$$\frac{dJ}{db} = \frac{dJ}{dy} \frac{dy}{db}, \quad \frac{dy}{db} = \frac{\exp(-b)}{(\exp(-b) + 1)^2}$$

$$\frac{dJ}{d\beta_j} = \frac{dJ}{db} \frac{db}{d\beta_j}, \quad \frac{db}{d\beta_j} = z_j$$

$$\frac{dJ}{dz_j} = \frac{dJ}{db} \frac{db}{dz_j}, \quad \frac{db}{dz_j} = \beta_j$$

$$\frac{dJ}{da_j} = \frac{dJ}{dz_j} \frac{dz_j}{da_j}, \quad \frac{dz_j}{da_j} = \frac{\exp(-a_j)}{(\exp(-a_j) + 1)^2}$$

$$\frac{dJ}{d\alpha_{ji}} = \frac{dJ}{da_j} \frac{da_j}{d\alpha_{ji}}, \quad \frac{da_j}{d\alpha_{ji}} = x_i$$

$$\frac{dJ}{dx_i} = \sum_{j=0}^D \frac{dJ}{da_j} \frac{da_j}{dx_i}, \quad \frac{da_j}{dx_i} = \alpha_{ji}$$

Training

Backpropagation

Case 2:

Forward

Backward

Loss

$$J = y^* \log y + (1 - y^*) \log(1 - y)$$

$$\frac{dJ}{dy} = \frac{y^*}{y} + \frac{(1 - y^*)}{y - 1}$$

Sigmoid

$$y = \frac{1}{1 + \exp(-b)}$$

$$\frac{dJ}{db} = \frac{dJ}{dy} \frac{dy}{db}, \quad \frac{dy}{db} = \frac{\exp(-b)}{(\exp(-b) + 1)^2}$$

Linear

$$b = \sum_{j=0}^D \beta_j z_j$$

$$\frac{dJ}{d\beta_j} = \frac{dJ}{db} \frac{db}{d\beta_j}, \quad \frac{db}{d\beta_j} = z_j$$

$$\frac{dJ}{dz_j} = \frac{dJ}{db} \frac{db}{dz_j}, \quad \frac{db}{dz_j} = \beta_j$$

Sigmoid

$$z_j = \frac{1}{1 + \exp(-a_j)}$$

$$\frac{dJ}{da_j} = \frac{dJ}{dz_j} \frac{dz_j}{da_j}, \quad \frac{dz_j}{da_j} = \frac{\exp(-a_j)}{(\exp(-a_j) + 1)^2}$$

Linear

$$a_j = \sum_{i=0}^M \alpha_{ji} x_i$$

$$\frac{dJ}{d\alpha_{ji}} = \frac{dJ}{da_j} \frac{da_j}{d\alpha_{ji}}, \quad \frac{da_j}{d\alpha_{ji}} = x_i$$

$$\frac{dJ}{dx_i} = \sum_{j=0}^D \frac{dJ}{da_j} \frac{da_j}{dx_i}, \quad \frac{da_j}{dx_i} = \alpha_{ji}$$

Derivative of a Sigmoid

First suppose that

$$s = \frac{1}{1 + \exp(-b)} \quad (1)$$

To obtain the simplified form of the derivative of a sigmoid.

$$\frac{ds}{db} = \frac{\exp(-b)}{(\exp(-b) + 1)^2} \quad (2)$$

$$= \frac{\exp(-b) + 1 - 1}{(\exp(-b) + 1 + 1 - 1)^2} \quad (3)$$

$$= \frac{\exp(-b) + 1 - 1}{(\exp(-b) + 1)^2} \quad (4)$$

$$= \frac{\exp(-b) + 1}{(\exp(-b) + 1)^2} - \frac{1}{(\exp(-b) + 1)^2} \quad (5)$$

$$= \frac{1}{(\exp(-b) + 1)} - \frac{1}{(\exp(-b) + 1)^2} \quad (6)$$

$$= \frac{1}{(\exp(-b) + 1)} - \left(\frac{1}{(\exp(-b) + 1)} \frac{1}{(\exp(-b) + 1)} \right) \quad (7)$$

$$= \frac{1}{(\exp(-b) + 1)} \left(1 - \frac{1}{(\exp(-b) + 1)} \right) \quad (8)$$

$$= s(1 - s) \quad (9)$$

Training

Backpropagation

Case 2:

Forward

Backward

Loss

$$J = y^* \log y + (1 - y^*) \log(1 - y)$$

$$\frac{dJ}{dy} = \frac{y^*}{y} + \frac{(1 - y^*)}{y - 1}$$

Sigmoid

$$y = \frac{1}{1 + \exp(-b)}$$

$$\frac{dJ}{db} = \frac{dJ}{dy} \frac{dy}{db}, \quad \frac{dy}{db} = \frac{\exp(-b)}{(\exp(-b) + 1)^2}$$

Linear

$$b = \sum_{j=0}^D \beta_j z_j$$

$$\frac{dJ}{d\beta_j} = \frac{dJ}{db} \frac{db}{d\beta_j}, \quad \frac{db}{d\beta_j} = z_j$$

Sigmoid

$$z_j = \frac{1}{1 + \exp(-a_j)}$$

$$\frac{dJ}{da_j} = \frac{dJ}{dz_j} \frac{dz_j}{da_j}, \quad \frac{dz_j}{da_j} = \frac{\exp(-a_j)}{(\exp(-a_j) + 1)^2}$$

Linear

$$a_j = \sum_{i=0}^M \alpha_{ji} x_i$$

$$\frac{dJ}{d\alpha_{ji}} = \frac{dJ}{da_j} \frac{da_j}{d\alpha_{ji}}, \quad \frac{da_j}{d\alpha_{ji}} = x_i$$

$$\frac{dJ}{dx_i} = \sum_{j=0}^D \frac{dJ}{da_j} \frac{da_j}{dx_i}, \quad \frac{da_j}{dx_i} = \alpha_{ji}$$

Training

Backpropagation

Case 2:

Forward

Backward

Loss

$$J = y^* \log y + (1 - y^*) \log(1 - y)$$

$$\frac{dJ}{dy} = \frac{y^*}{y} + \frac{(1 - y^*)}{y - 1}$$

Sigmoid

$$y = \frac{1}{1 + \exp(-b)}$$

$$\frac{dJ}{db} = \frac{dJ}{dy} \frac{dy}{db}, \quad \frac{dy}{db} = y(1 - y)$$

Linear

$$b = \sum_{j=0}^D \beta_j z_j$$

$$\frac{dJ}{d\beta_j} = \frac{dJ}{db} \frac{db}{d\beta_j}, \quad \frac{db}{d\beta_j} = z_j$$

Sigmoid

$$z_j = \frac{1}{1 + \exp(-a_j)}$$

$$\frac{dJ}{da_j} = \frac{dJ}{dz_j} \frac{dz_j}{da_j}, \quad \frac{dz_j}{da_j} = z_j(1 - z_j)$$

Linear

$$a_j = \sum_{i=0}^M \alpha_{ji} x_i$$

$$\frac{dJ}{d\alpha_{ji}} = \frac{dJ}{da_j} \frac{da_j}{d\alpha_{ji}}, \quad \frac{da_j}{d\alpha_{ji}} = x_i$$

$$\frac{dJ}{dx_i} = \sum_{j=0}^D \frac{dJ}{da_j} \frac{da_j}{dx_i}, \quad \frac{da_j}{dx_i} = \alpha_{ji}$$

Example: 1-Hidden Layer Neural Network

Algorithm 1 Stochastic Gradient Descent (SGD)

```
1: procedure SGD(Training data  $\mathcal{D}$ , test data  $\mathcal{D}_t$ )
2:   Initialize parameters  $\alpha, \beta$ 
3:   for  $e \in \{1, 2, \dots, E\}$  do
4:     for  $(\mathbf{x}, \mathbf{y}) \in \mathcal{D}$  do
5:       Compute neural network layers:
6:        $\mathbf{o} = \text{object}(\mathbf{x}, \mathbf{a}, \mathbf{b}, \mathbf{z}, \hat{\mathbf{y}}, J) = \text{NNFORWARD}(\mathbf{x}, \mathbf{y}, \alpha, \beta)$ 
7:       Compute gradients via backprop:
8:        $\left. \begin{array}{l} \mathbf{g}_\alpha = \nabla_\alpha J \\ \mathbf{g}_\beta = \nabla_\beta J \end{array} \right\} = \text{NNBACKWARD}(\mathbf{x}, \mathbf{y}, \alpha, \beta, \mathbf{o})$ 
9:       Update parameters:
10:       $\alpha \leftarrow \alpha - \gamma \mathbf{g}_\alpha$ 
11:       $\beta \leftarrow \beta - \gamma \mathbf{g}_\beta$ 
12:      Evaluate training mean cross-entropy  $J_{\mathcal{D}}(\alpha, \beta)$ 
13:      Evaluate test mean cross-entropy  $J_{\mathcal{D}_t}(\alpha, \beta)$ 
14:   return parameters  $\alpha, \beta$ 
```

THE BACKPROPAGATION ALGORITHM

Automatic Differentiation – Reverse Mode (aka. Backpropagation)

Forward Computation

1. Write an **algorithm** for evaluating the function $y = f(\mathbf{x})$. The algorithm defines a **directed acyclic graph**, where each variable is a node (i.e. the “**computation graph**”)
2. Visit each node in **topological order**.
For variable u_i with inputs $v_1, \dots, v_N$
 - a. Compute $u_i = g_i(v_1, \dots, v_N)$
 - b. Store the result at the node

Backward Computation (Version A)

1. **Initialize** $dy/dy = 1$.
2. Visit each node v_j in **reverse topological order**.
Let $u_1, \dots, u_M$ denote all the nodes with v_j as an input
Assuming that $y = h(\mathbf{u}) = h(u_1, \dots, u_M)$
and $\mathbf{u} = g(\mathbf{v})$ or equivalently $u_i = g_i(v_1, \dots, v_j, \dots, v_N)$ for all i
 - a. We already know dy/du_i for all i
 - b. Compute dy/dv_j as below (Choice of algorithm ensures computing (du_i/dv_j) is easy)

$$\frac{dy}{dv_j} = \sum_{i=1}^M \frac{dy}{du_i} \frac{du_i}{dv_j}$$

Return partial derivatives dy/du_i for all variables

Automatic Differentiation – Reverse Mode (aka. Backpropagation)

Forward Computation

1. Write an **algorithm** for evaluating the function $y = f(\mathbf{x})$. The algorithm defines a **directed acyclic graph**, where each variable is a node (i.e. the “**computation graph**”)
2. Visit each node in **topological order**.
For variable u_i with inputs $v_1, \dots, v_N$
 - a. Compute $u_i = g_i(v_1, \dots, v_N)$
 - b. Store the result at the node

Backward Computation (Version B)

1. **Initialize** all partial derivatives dy/du_j to 0 and $dy/dy = 1$.
2. Visit each node in **reverse topological order**.
For variable $u_i = g_i(v_1, \dots, v_N)$
 - a. We already know dy/du_i
 - b. Increment dy/dv_j by $(dy/du_i)(du_i/dv_j)$
(Choice of algorithm ensures computing (du_i/dv_j) is easy)

Return partial derivatives dy/du_i for all variables

Why is the backpropagation algorithm efficient?

1. Reuses **computation from the forward pass** in the backward pass
2. Reuses **partial derivatives** throughout the backward pass (*but only if the algorithm reuses shared computation in the forward pass*)

(Key idea: partial derivatives in the backward pass should be thought of as variables stored for reuse)

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

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

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)


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

MATRIX CALCULUS

Q&A

Q: Do I need to know **matrix calculus** to derive the backprop algorithms used in this class?

A: Well, we've carefully constructed our assignments so that you do **not** need to know matrix calculus.

That said, it's pretty handy. So we *added matrix calculus to our learning objectives* for backprop.

Matrix Calculus

Numerator

Denominator	Types of Derivatives	scalar	vector	matrix
	scalar	$\frac{\partial y}{\partial x}$	$\frac{\partial \mathbf{y}}{\partial x}$	$\frac{\partial \mathbf{Y}}{\partial x}$
	vector	$\frac{\partial y}{\partial \mathbf{x}}$	$\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$	$\frac{\partial \mathbf{Y}}{\partial \mathbf{x}}$
	matrix	$\frac{\partial y}{\partial \mathbf{X}}$	$\frac{\partial \mathbf{y}}{\partial \mathbf{X}}$	$\frac{\partial \mathbf{Y}}{\partial \mathbf{X}}$

Let $y, x \in \mathbb{R}$ be scalars,
 $\mathbf{y} \in \mathbb{R}^M$ and $\mathbf{x} \in \mathbb{R}^P$
 be vectors, and
 $\mathbf{Y} \in \mathbb{R}^{M \times N}$ and $\mathbf{X} \in \mathbb{R}^{P \times Q}$ be matrices

Matrix Calculus

Types of Derivatives	scalar
scalar	$\frac{\partial y}{\partial x} = \left[\frac{\partial y}{\partial x} \right]$
vector	$\frac{\partial y}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y}{\partial x_1} \\ \frac{\partial y}{\partial x_2} \\ \vdots \\ \frac{\partial y}{\partial x_P} \end{bmatrix}$
matrix	$\frac{\partial y}{\partial \mathbf{X}} = \begin{bmatrix} \frac{\partial y}{\partial X_{11}} & \frac{\partial y}{\partial X_{12}} & \cdots & \frac{\partial y}{\partial X_{1Q}} \\ \frac{\partial y}{\partial X_{21}} & \frac{\partial y}{\partial X_{22}} & \cdots & \frac{\partial y}{\partial X_{2Q}} \\ \vdots & & & \vdots \\ \frac{\partial y}{\partial X_{P1}} & \frac{\partial y}{\partial X_{P2}} & \cdots & \frac{\partial y}{\partial X_{PQ}} \end{bmatrix}$

Matrix Calculus

Types of Derivatives	scalar	vector
scalar	$\frac{\partial y}{\partial x} = \left[\frac{\partial y}{\partial x} \right]$	$\frac{\partial \mathbf{y}}{\partial x} = \left[\frac{\partial y_1}{\partial x} \quad \frac{\partial y_2}{\partial x} \quad \dots \quad \frac{\partial y_N}{\partial x} \right]$
vector	$\frac{\partial y}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y}{\partial x_1} \\ \frac{\partial y}{\partial x_2} \\ \vdots \\ \frac{\partial y}{\partial x_P} \end{bmatrix}$	$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_2}{\partial x_1} & \dots & \frac{\partial y_N}{\partial x_1} \\ \frac{\partial y_1}{\partial x_2} & \frac{\partial y_2}{\partial x_2} & \dots & \frac{\partial y_N}{\partial x_2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_1}{\partial x_P} & \frac{\partial y_2}{\partial x_P} & \dots & \frac{\partial y_N}{\partial x_P} \end{bmatrix}$

Matrix Calculus

Whenever you read about matrix calculus, you'll be confronted with two layout conventions:

Let $y, x \in \mathbb{R}$ be scalars, $\mathbf{y} \in \mathbb{R}^M$ and $\mathbf{x} \in \mathbb{R}^P$ be vectors.

1. In numerator layout:

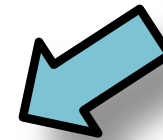
$\frac{\partial y}{\partial \mathbf{x}}$ is a $1 \times P$ matrix, i.e. a row vector

$\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$ is an $M \times P$ matrix

2. In denominator layout:

$\frac{\partial y}{\partial \mathbf{x}}$ is a $P \times 1$ matrix, i.e. a column vector

$\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$ is an $P \times M$ matrix



In this course, we use **denominator layout**.

Why? This ensures that our gradients of the objective function with respect to some subset of parameters are the same shape as those parameters.

Matrix Calculus

Common Vector Derivatives

Let $\frac{\partial f(\vec{x})}{\partial \vec{x}} = \nabla_x f(\vec{x})$ be the vector derivative of f , $B \in \mathbb{R}^{m \times n}$
 $x \in \mathbb{R}^n$

Scalar Derivative
 $f(x) \rightarrow \frac{\partial f}{\partial x}$

$$bx \rightarrow b$$

$$xb \rightarrow b$$

$$x^2 \rightarrow 2x$$

$$bx^2 \rightarrow 2bx$$

Vector Derivative
 $f(x) \rightarrow \frac{\partial f}{\partial \vec{x}}$

$$x^T B \rightarrow B$$

$$x^T b \rightarrow b$$

$$x^T x \rightarrow 2x$$

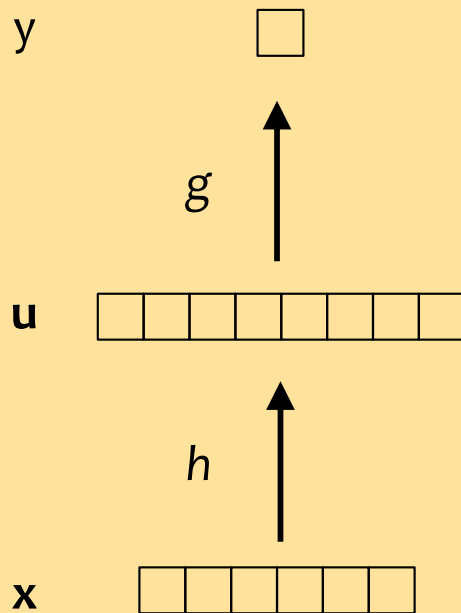
$$x^T B x \rightarrow 2Bx$$

$\nwarrow B$ symmetric

Matrix Calculus

Question:

Suppose $y = g(\mathbf{u})$ and $\mathbf{u} = h(\mathbf{x})$



Which of the following is the correct definition of the chain rule?

Recall:

$$\frac{\partial y}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y}{\partial x_1} \\ \frac{\partial y}{\partial x_2} \\ \vdots \\ \frac{\partial y}{\partial x_P} \end{bmatrix} \quad \frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_2}{\partial x_1} & \dots & \frac{\partial y_N}{\partial x_1} \\ \frac{\partial y_1}{\partial x_2} & \frac{\partial y_2}{\partial x_2} & \dots & \frac{\partial y_N}{\partial x_2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_1}{\partial x_P} & \frac{\partial y_2}{\partial x_P} & \dots & \frac{\partial y_N}{\partial x_P} \end{bmatrix}$$

Answer:

$$\frac{\partial y}{\partial \mathbf{x}} = \dots$$

A. $\frac{\partial y}{\partial \mathbf{u}} \frac{\partial \mathbf{u}}{\partial \mathbf{x}}$

B. $\frac{\partial y}{\partial \mathbf{u}}^T \frac{\partial \mathbf{u}}{\partial \mathbf{x}}$

C. $\frac{\partial y}{\partial \mathbf{u}} \frac{\partial \mathbf{u}}{\partial \mathbf{x}}^T$

D. $\frac{\partial y}{\partial \mathbf{u}}^T \frac{\partial \mathbf{u}}{\partial \mathbf{x}}^T$

E. $(\frac{\partial y}{\partial \mathbf{u}} \frac{\partial \mathbf{u}}{\partial \mathbf{x}})^T$

F. None of the above