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

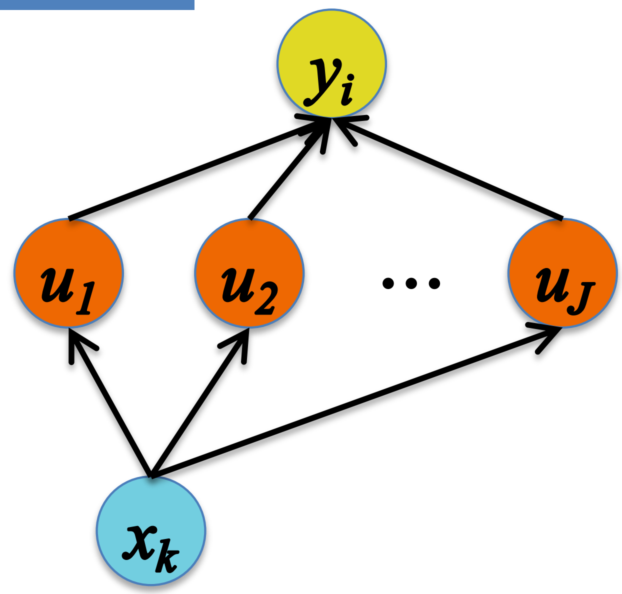
Training

Chain Rule

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

Chain Rule:

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



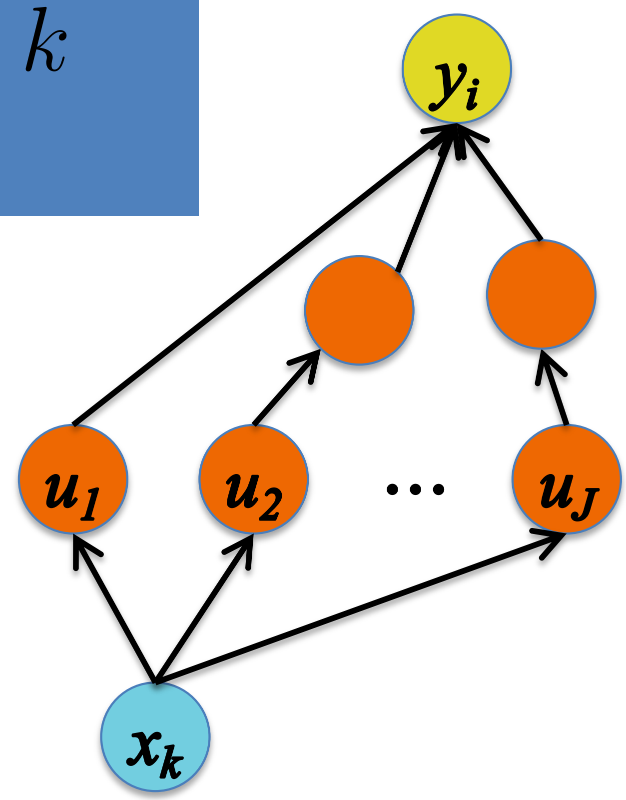
Training

Chain Rule

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

Chain Rule:

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

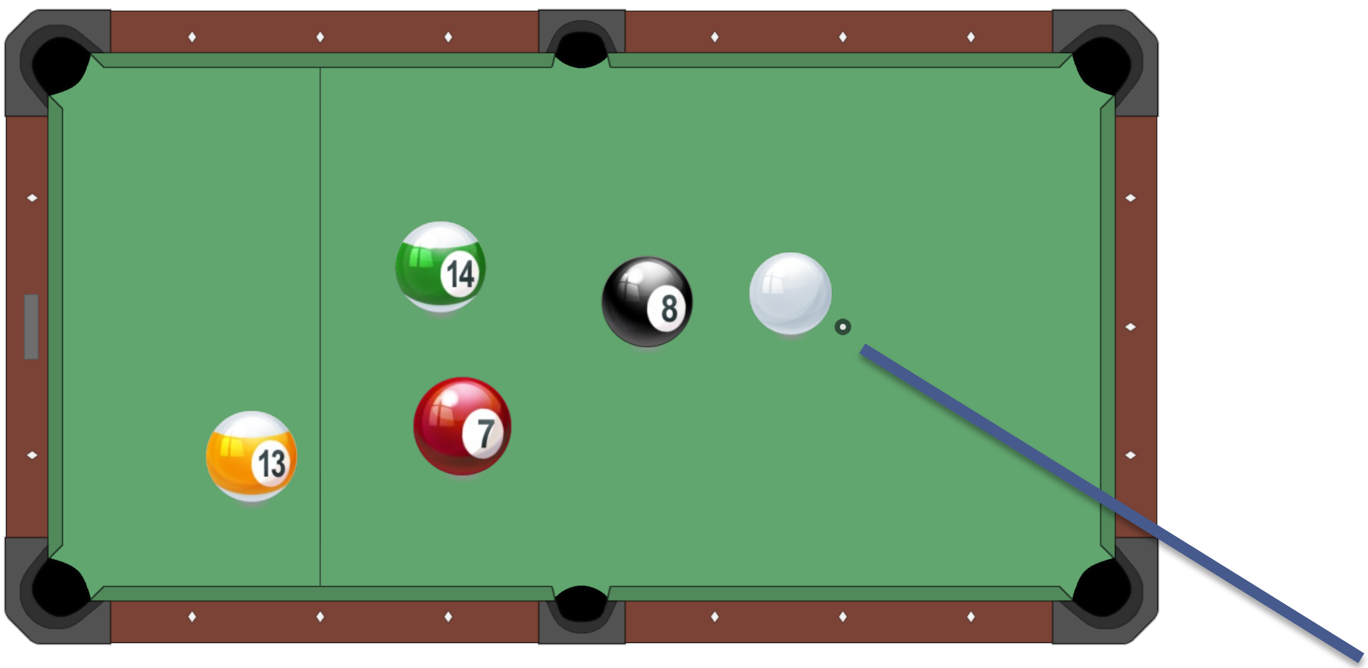


Backpropagation is just repeated application of the **chain rule** from Calculus 101.

Intuitions

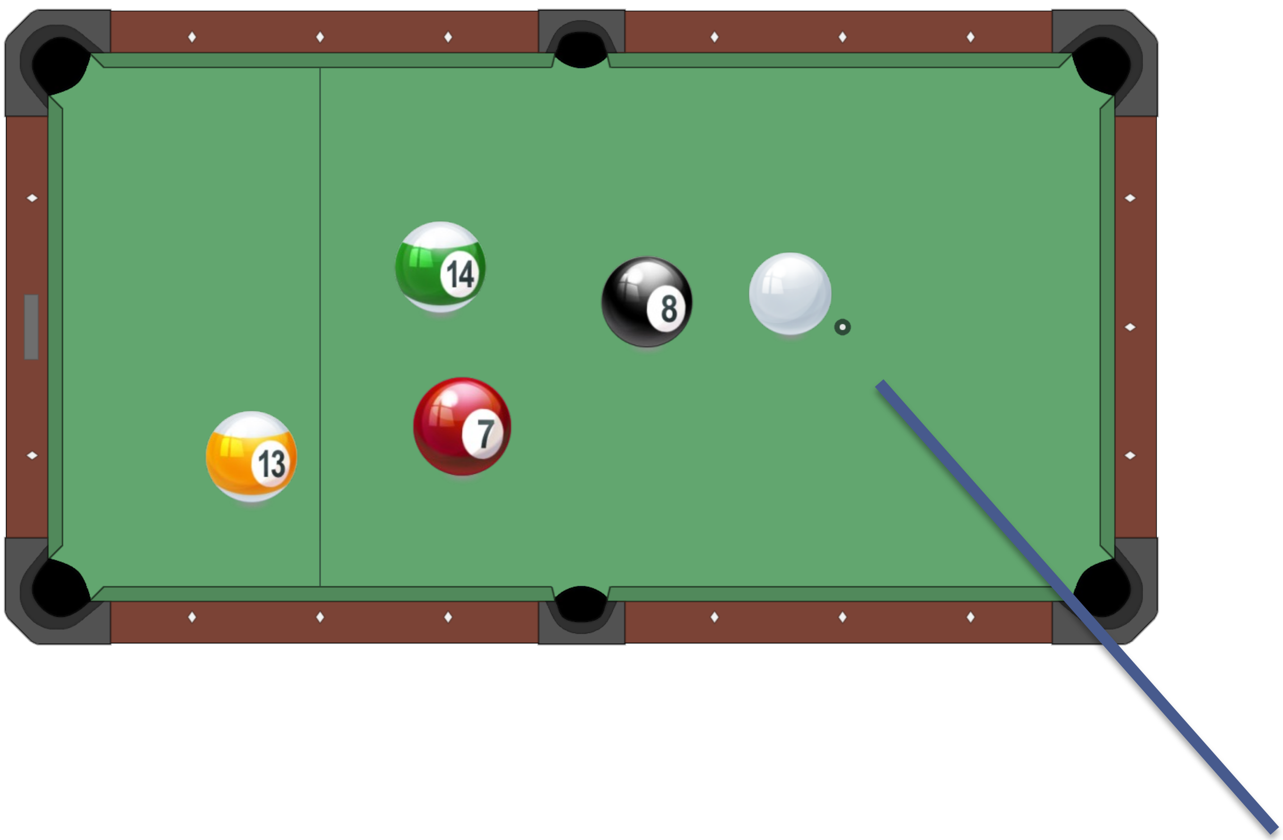
BACKPROPAGATION OF ERRORS

Error Back-Propagation



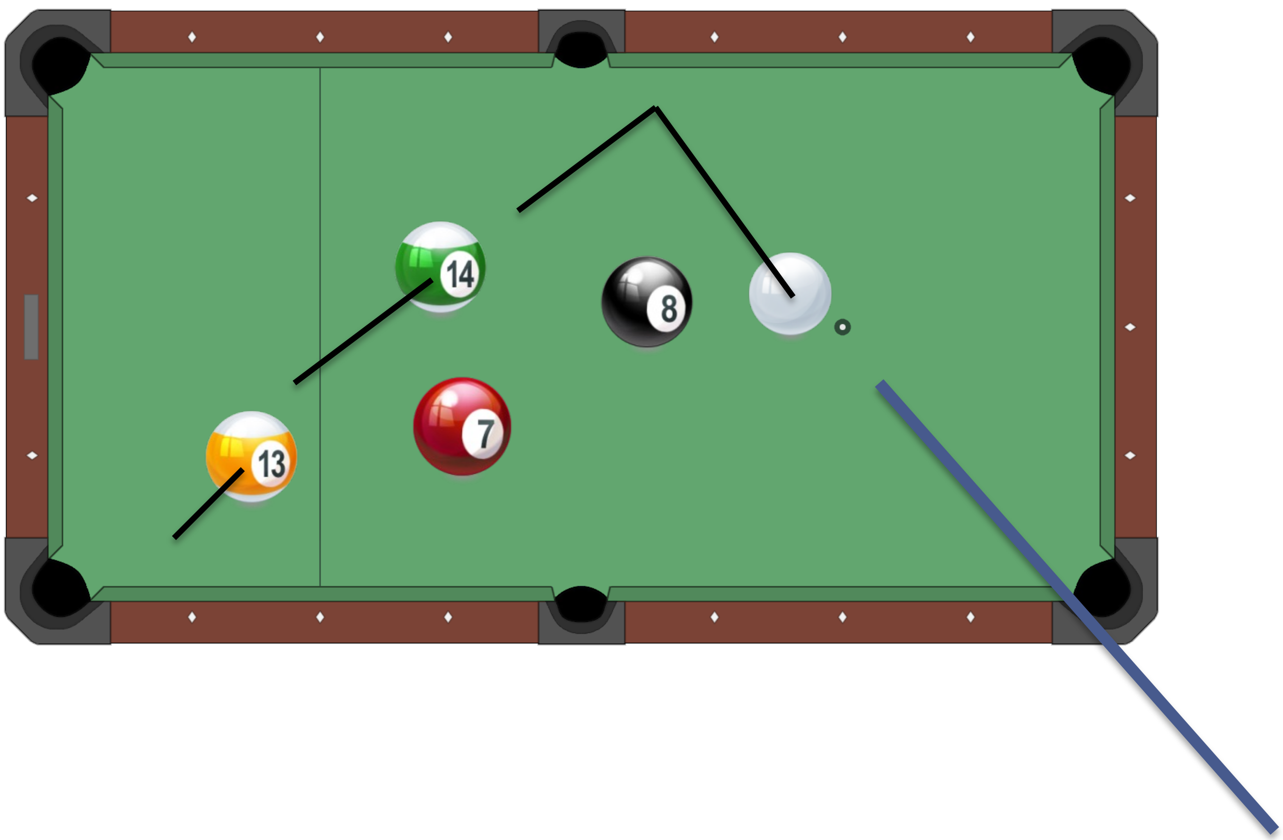
Slide from (Stoyanov & Eisner, 2012)

Error Back-Propagation



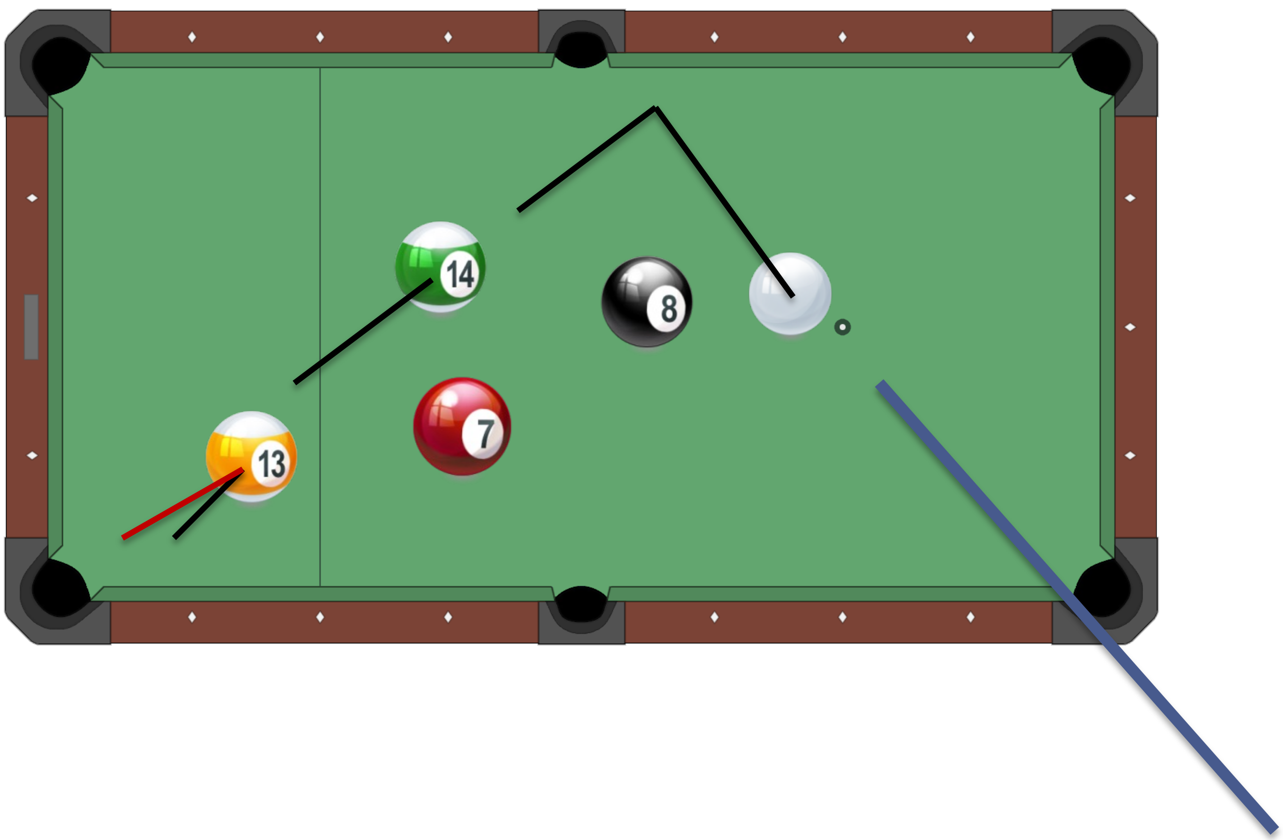
Slide from (Stoyanov & Eisner, 2012)

Error Back-Propagation



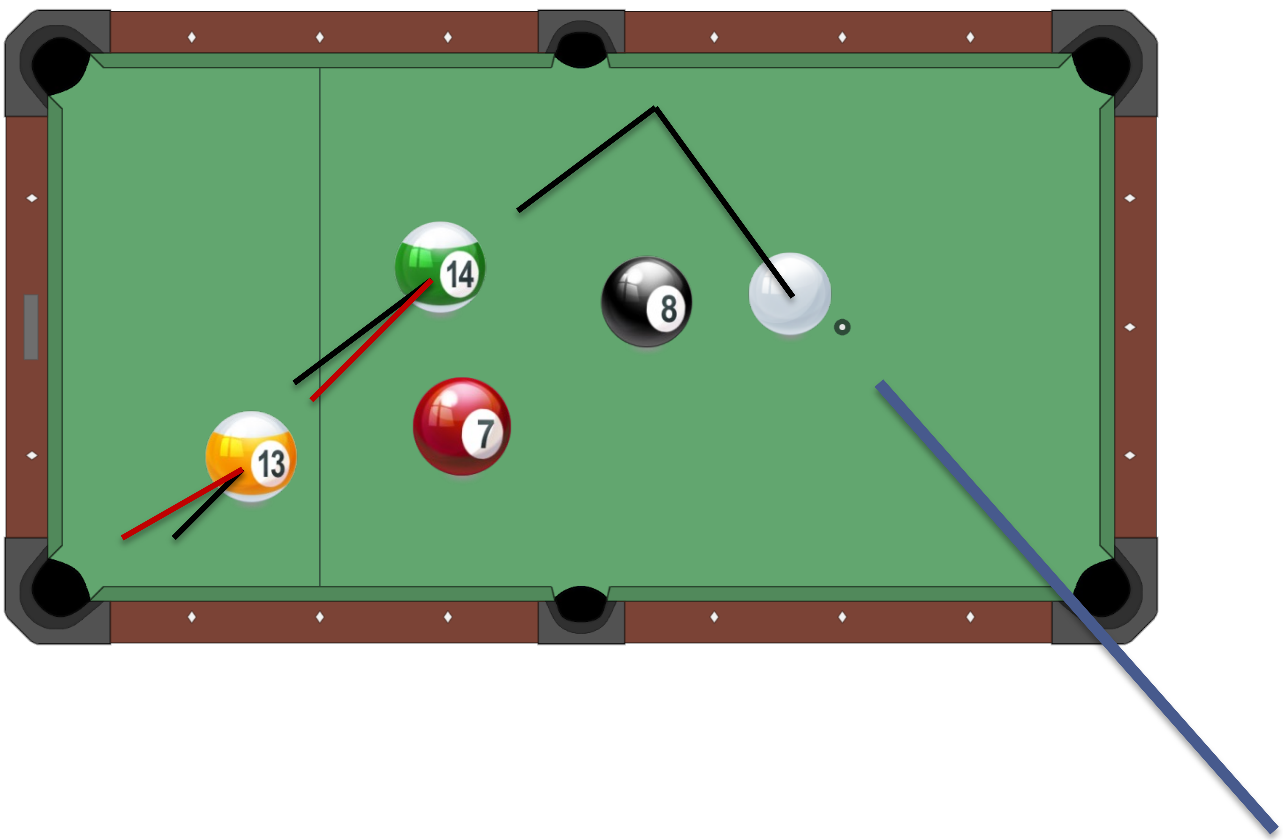
Slide from (Stoyanov & Eisner, 2012)

Error Back-Propagation



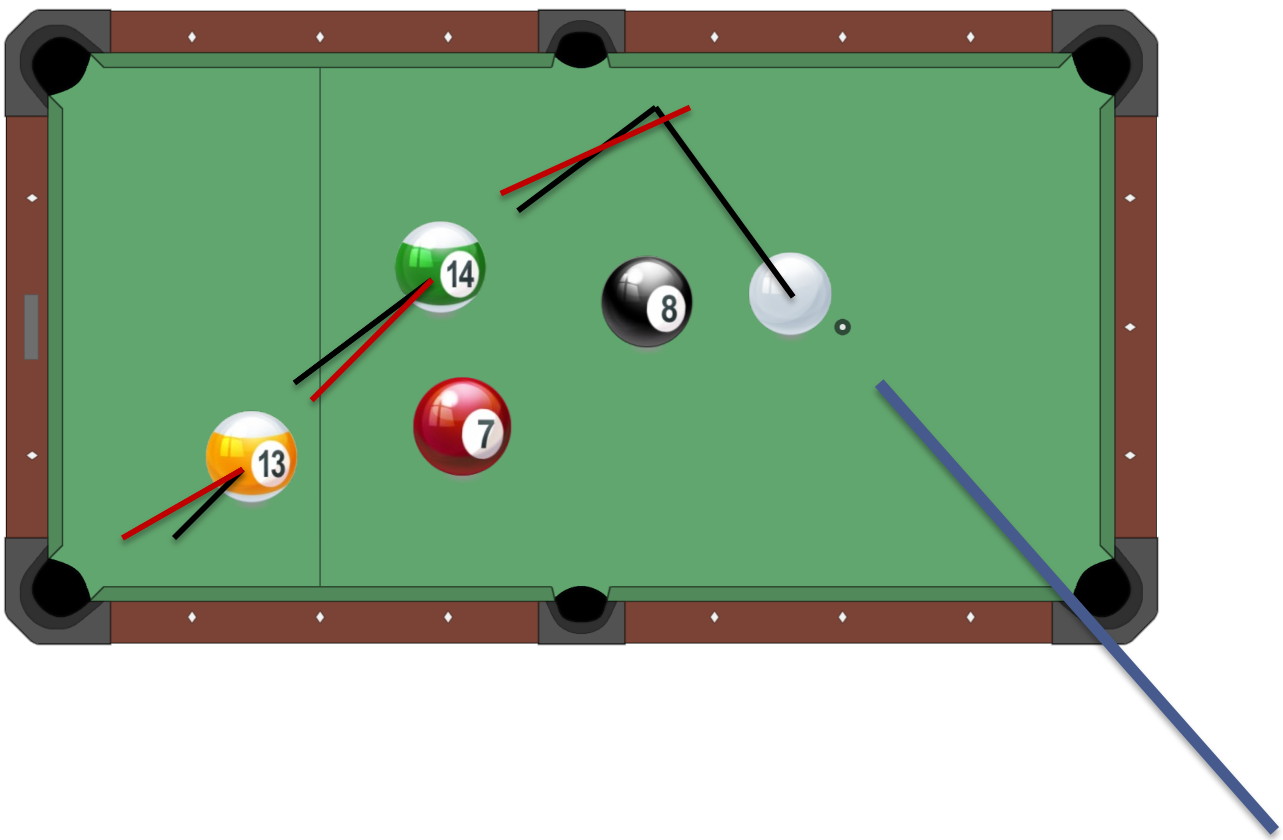
Slide from (Stoyanov & Eisner, 2012)

Error Back-Propagation



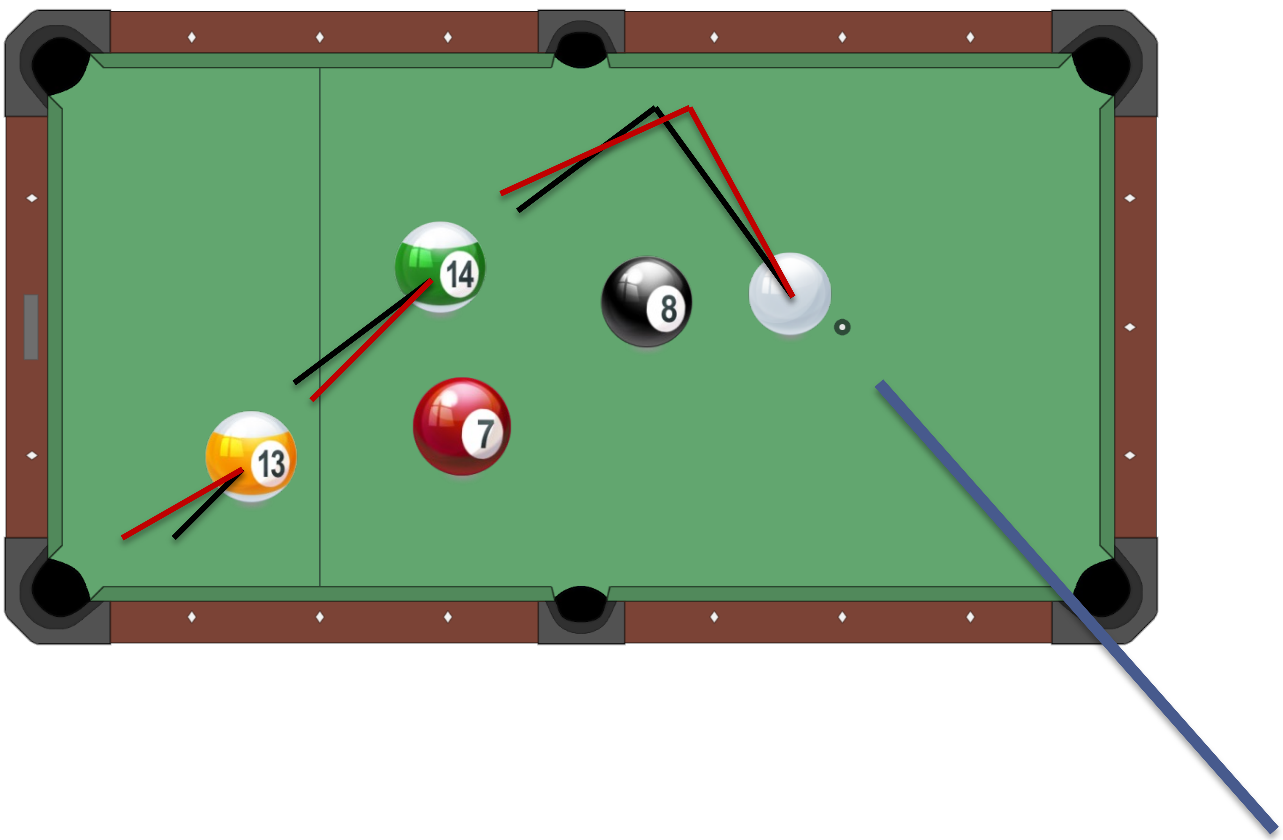
Slide from (Stoyanov & Eisner, 2012)

Error Back-Propagation



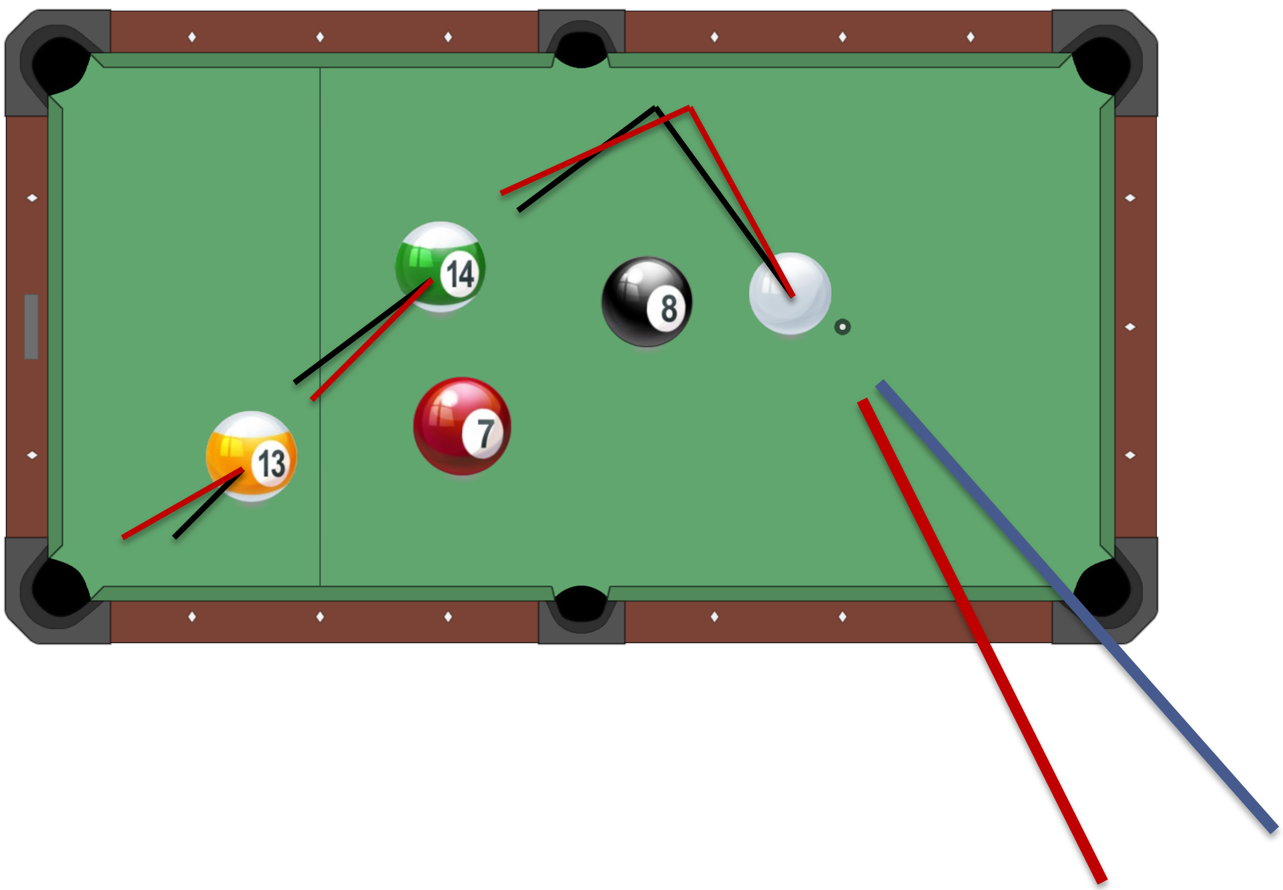
Slide from (Stoyanov & Eisner, 2012)

Error Back-Propagation



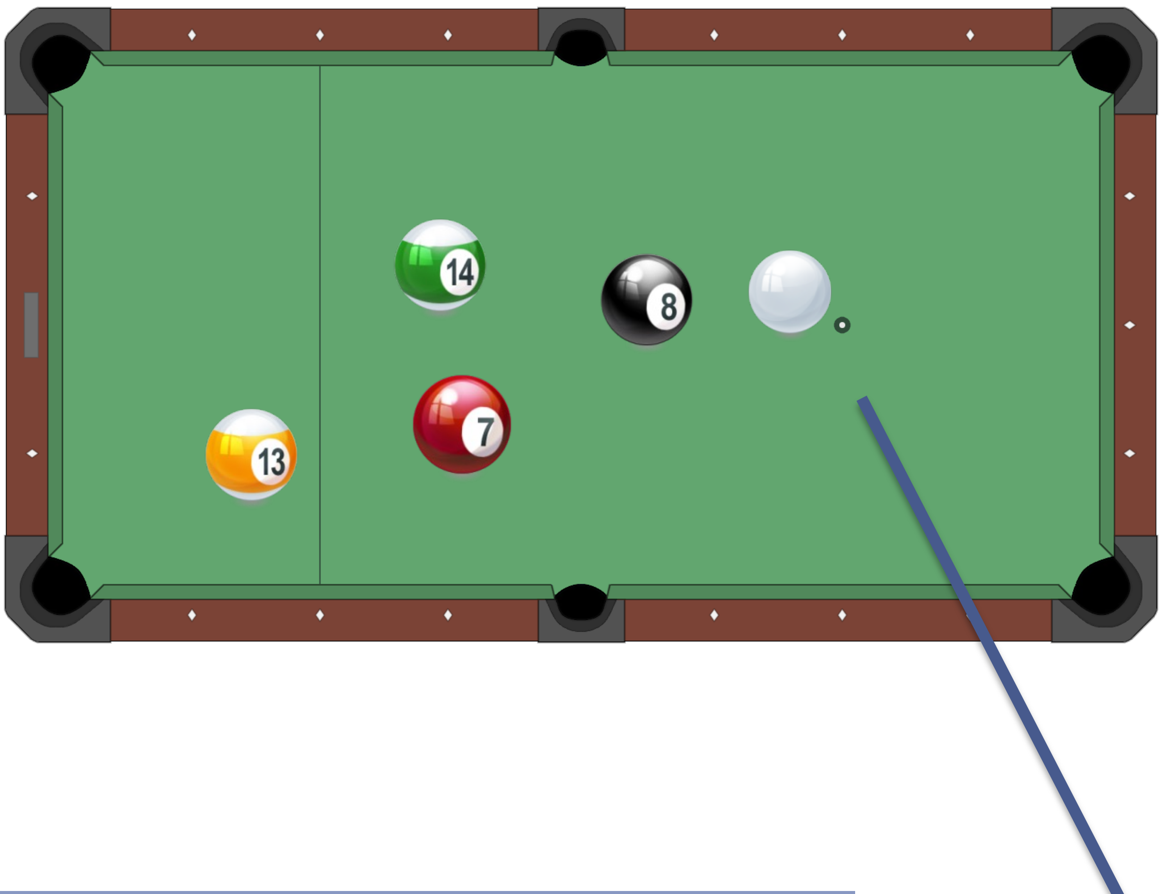
Slide from (Stoyanov & Eisner, 2012)

Error Back-Propagation



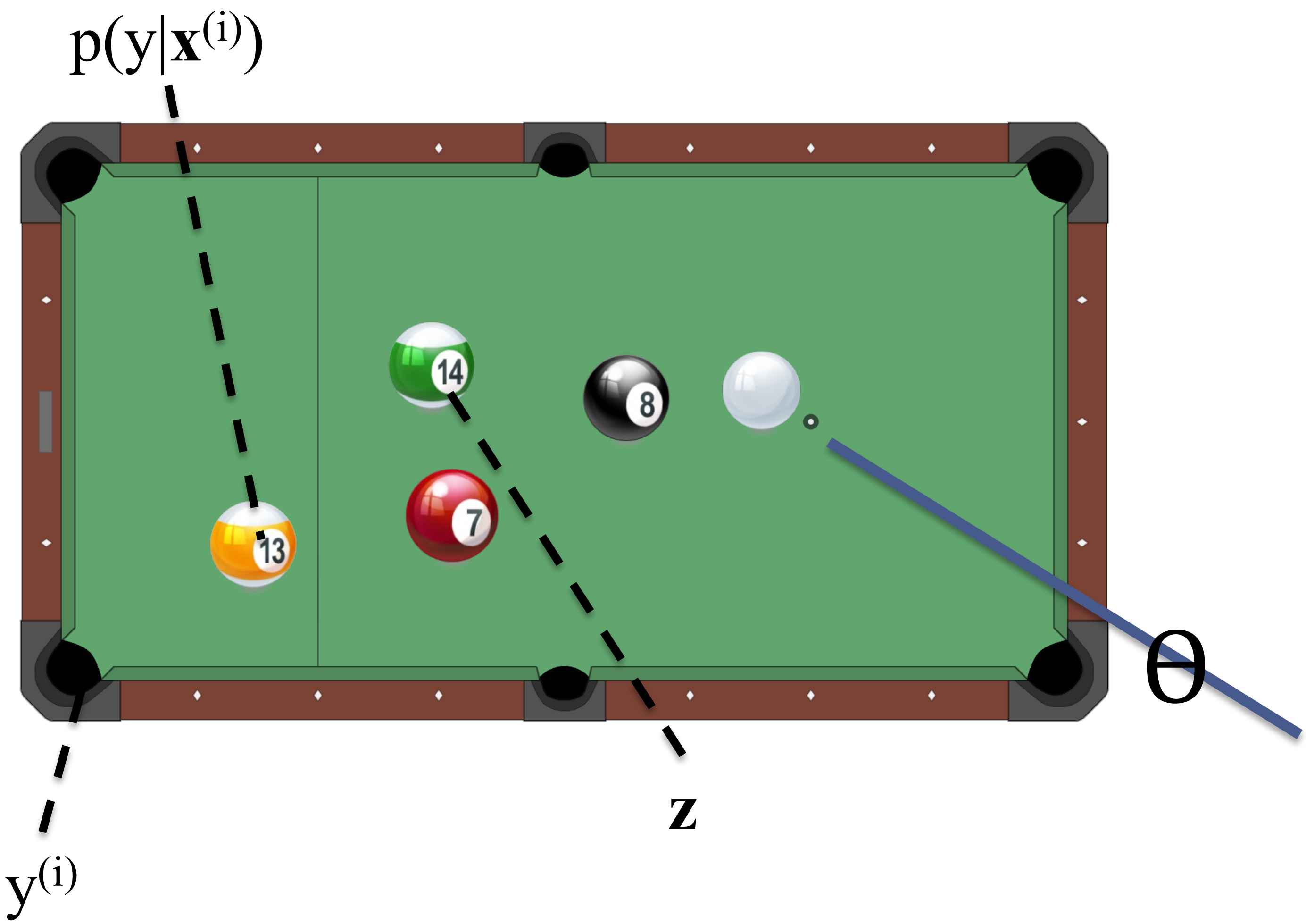
Slide from (Stoyanov & Eisner, 2012)

Error Back-Propagation



Slide from (Stoyanov & Eisner, 2012)

Error Back-Propagation



Slide from (Stoyanov & Eisner, 2012)

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

Training

Backpropagation

Whiteboard

– Backprogation 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**.

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

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

$J = cos(u)$

$u = u_1 + u_2$

$u_1 = sin(t)$

$u_2 = 3t$

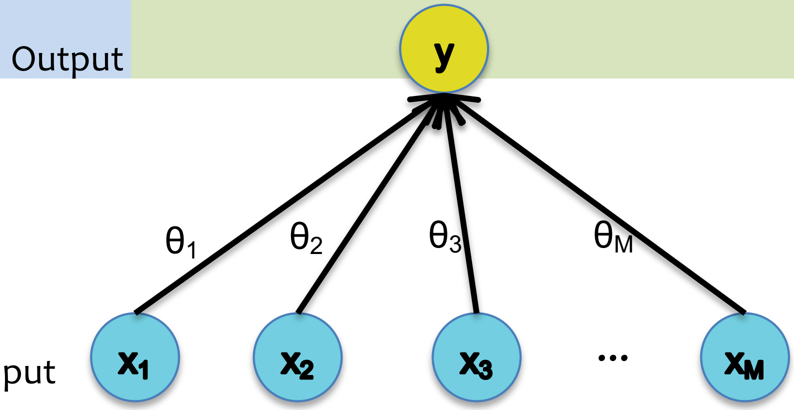
$t = x^2$

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
$J = cos(u)$	$\frac{dJ}{du} \mathrel{+}= -sin(u)$
$u = u_1 + u_2$	$\frac{dJ}{du_1} \mathrel{+}= \frac{dJ}{du}\frac{du}{du_1}, \quad \frac{du}{du_1} = 1 \qquad \frac{dJ}{du_2} \mathrel{+}= \frac{dJ}{du}\frac{du}{du_2}, \quad \frac{du}{du_2} = 1$
$u_1 = sin(t)$	$\frac{dJ}{dt} \mathrel{+}= \frac{dJ}{du_1}\frac{du_1}{dt}, \quad \frac{du_1}{dt} = \cos(t)$
$u_2 = 3t$	$\frac{dJ}{dt} \mathrel{+}= \frac{dJ}{du_2}\frac{du_2}{dt}, \quad \frac{du_2}{dt} = 3$
$t = x^2$	$\frac{dJ}{dx} \mathrel{+}= \frac{dJ}{dt}\frac{dt}{dx}, \quad \frac{dt}{dx} = 2x$

Training Backpropagation



Case 1: Logistic Regression

Forward

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

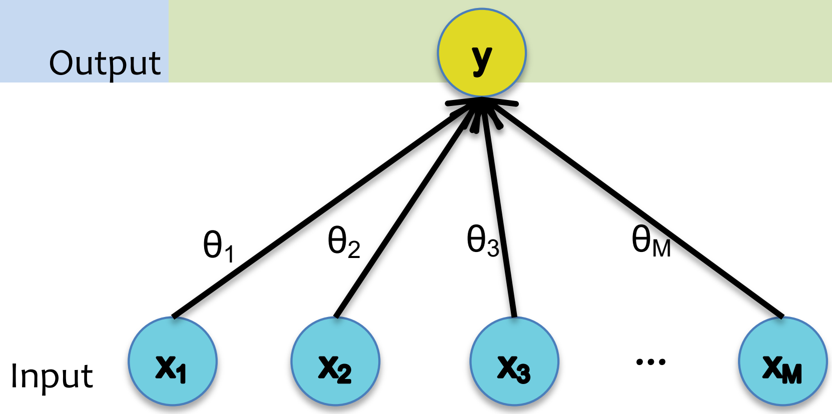
$$y = \frac{1}{1 + \exp(-a)}$$

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

Training

Backpropagation

Case 1: Logistic Regression



Forward	Backward
$J = y^* \log y + (1 - y^*) \log(1 - y)$	$\frac{dJ}{dy} = \frac{y^*}{y} + \frac{(1 - y^*)}{y - 1}$
$y = \frac{1}{1 + \exp(-a)}$	$\frac{dJ}{da} = \frac{dJ}{dy}\frac{dy}{da}, \; \frac{dy}{da} = \frac{\exp(-a)}{(\exp(-a) + 1)^2}$
$a = \sum_{j=0}^{D} \theta_j x_j$	$\frac{dJ}{d\theta_j} = \frac{dJ}{da}\frac{da}{d\theta_j}, \; \frac{da}{d\theta_j} = x_j$ $\frac{dJ}{dx_j} = \frac{dJ}{da}\frac{da}{dx_j}, \; \frac{da}{dx_j} = \theta_j$

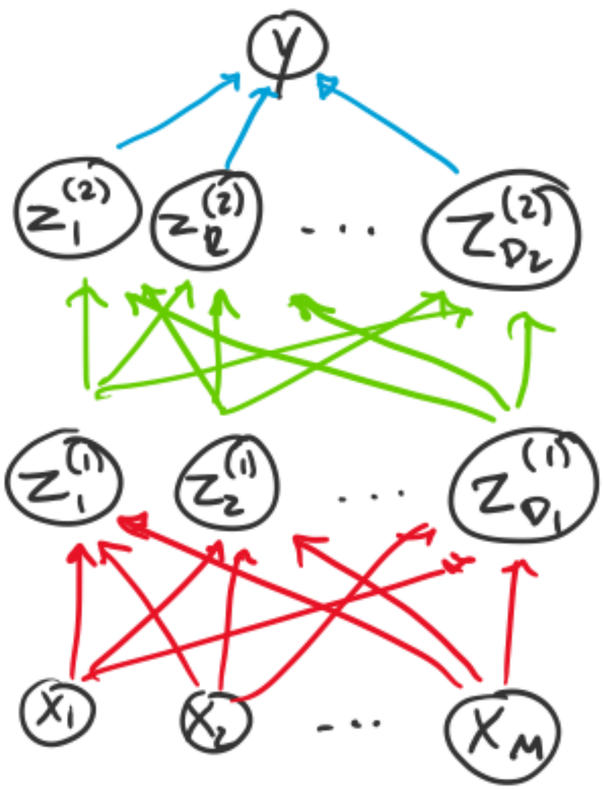
A 2-Hidden Layer Neural Network

TRAINING / FORWARD COMPUTATION / BACKWARD COMPUTATION

Backpropagation

Recall: Our 2-Hidden Layer Neural Network

Question: How do we train this model?



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

$\beta_0 \in \mathbb{R}$

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

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

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

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

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

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

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

Backpropagation

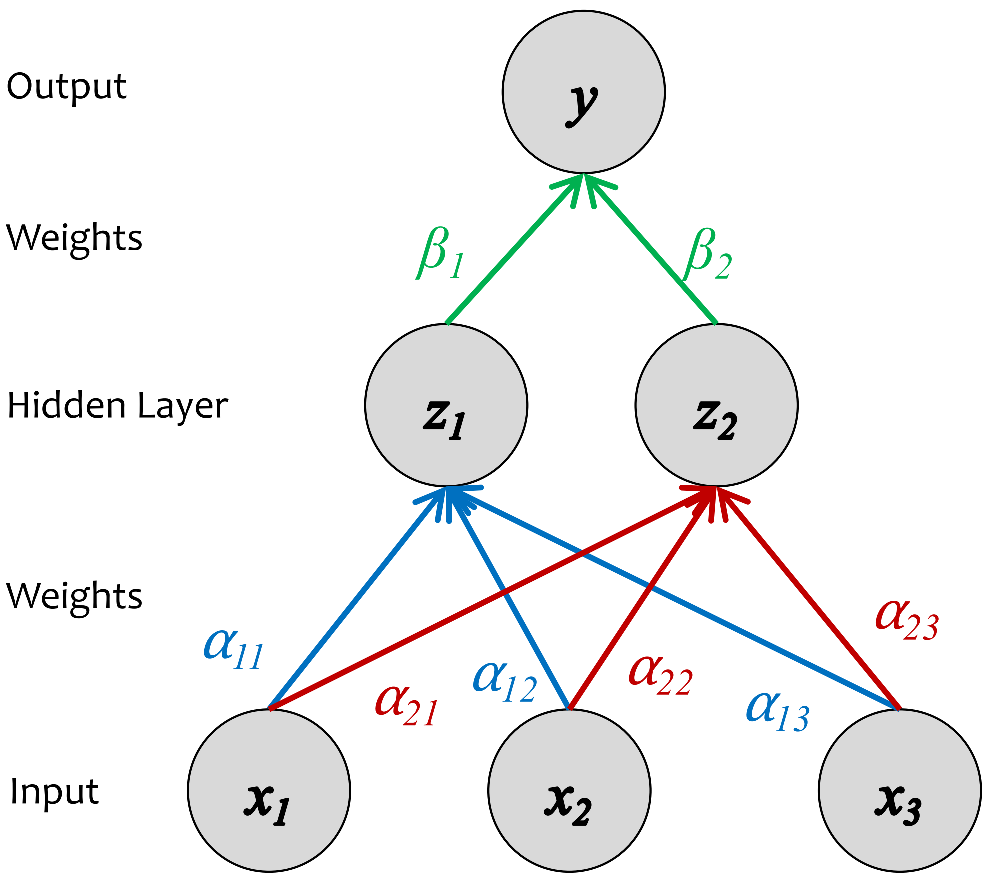
Whiteboard

- 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



(E) **Output (sigmoid)**
$y = \frac{1}{1+\exp(-b)}$

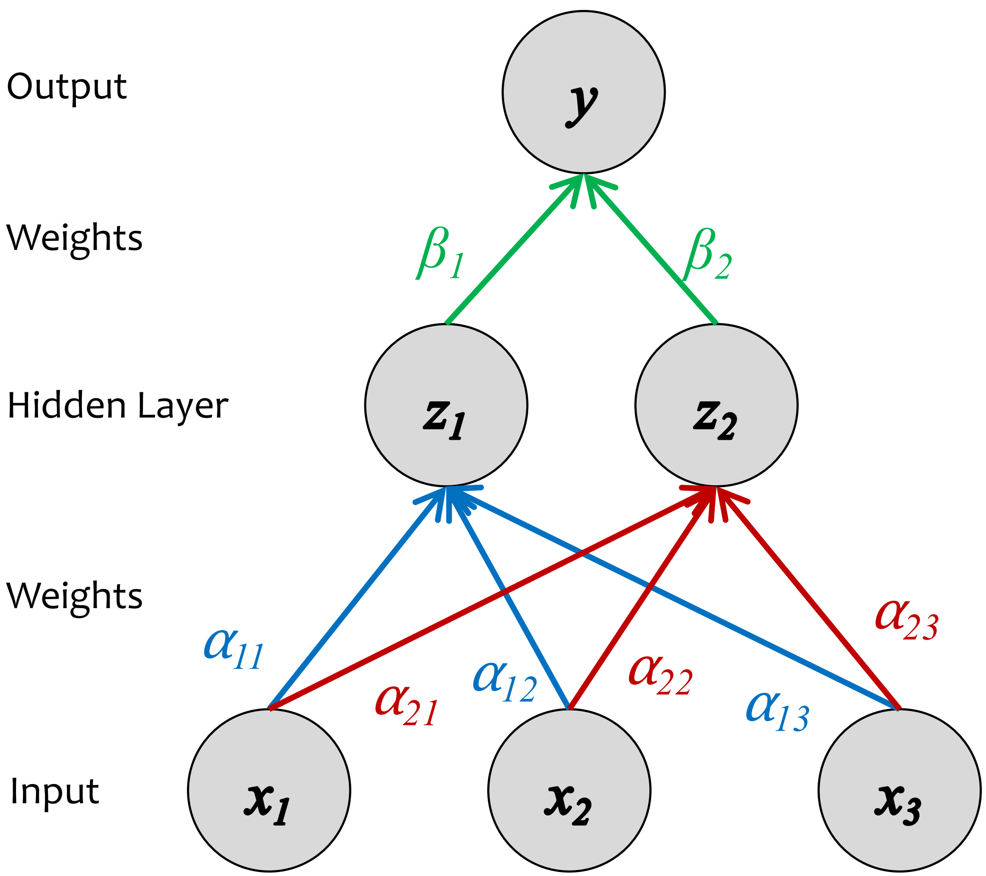
(D) **Output (linear)**
$b = \sum_{j=0}^{D} \beta_j z_j$

(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 $x_i, \;\; \forall i$

Training Backpropagation



(F) **Loss**
$J = \frac{1}{2}(y - y^*)^2$

↑

(E) **Output (sigmoid)**
$$y = \frac{1}{1+\exp(-b)}$$

↑

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

↑

(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 $x_i, \ \ \forall i$

Training

SGD with Backprop

Example: 1-Hidden Layer Neural Network

Algorithm 1 Stochastic Gradient Descent (SGD)

procedure SGD(Training data $\mathcal{D}$, test data $\mathcal{D}_t$)
  Initialize parameters $\boldsymbol{\alpha}, \boldsymbol{\beta}$
  for $e \in \{1, 2, \ldots, E\}$ **do**
    for $(\mathbf{x}, \mathbf{y}) \in \mathcal{D}$ **do**
      Compute neural network layers:
      $\mathbf{o} = \texttt{object}(\mathbf{x}, \mathbf{a}, \mathbf{b}, \mathbf{z}, \hat{\mathbf{y}}, J) = \text{NNFORWARD}(\mathbf{x}, \mathbf{y}, \boldsymbol{\alpha}, \boldsymbol{\beta})$
      Compute gradients via backprop:
      $$\left.\begin{aligned} \mathbf{g}_{\boldsymbol{\alpha}} &= \nabla_{\boldsymbol{\alpha}} J \\ \mathbf{g}_{\boldsymbol{\beta}} &= \nabla_{\boldsymbol{\beta}} J \end{aligned}\right\} = \text{NNBACKWARD}(\mathbf{x}, \mathbf{y}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \mathbf{o})$$
      Update parameters:
      $\boldsymbol{\alpha} \leftarrow \boldsymbol{\alpha} - \gamma \mathbf{g}_{\boldsymbol{\alpha}}$
      $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} - \gamma \mathbf{g}_{\boldsymbol{\beta}}$
    Evaluate training mean cross-entropy $J_{\mathcal{D}}(\boldsymbol{\alpha}, \boldsymbol{\beta})$
    Evaluate test mean cross-entropy $J_{\mathcal{D}_t}(\boldsymbol{\alpha}, \boldsymbol{\beta})$
  return parameters $\boldsymbol{\alpha}, \boldsymbol{\beta}$

A 1-Hidden Layer Neural Network

FORWARD COMPUTATION FOR A NEURAL NETWORK

Training

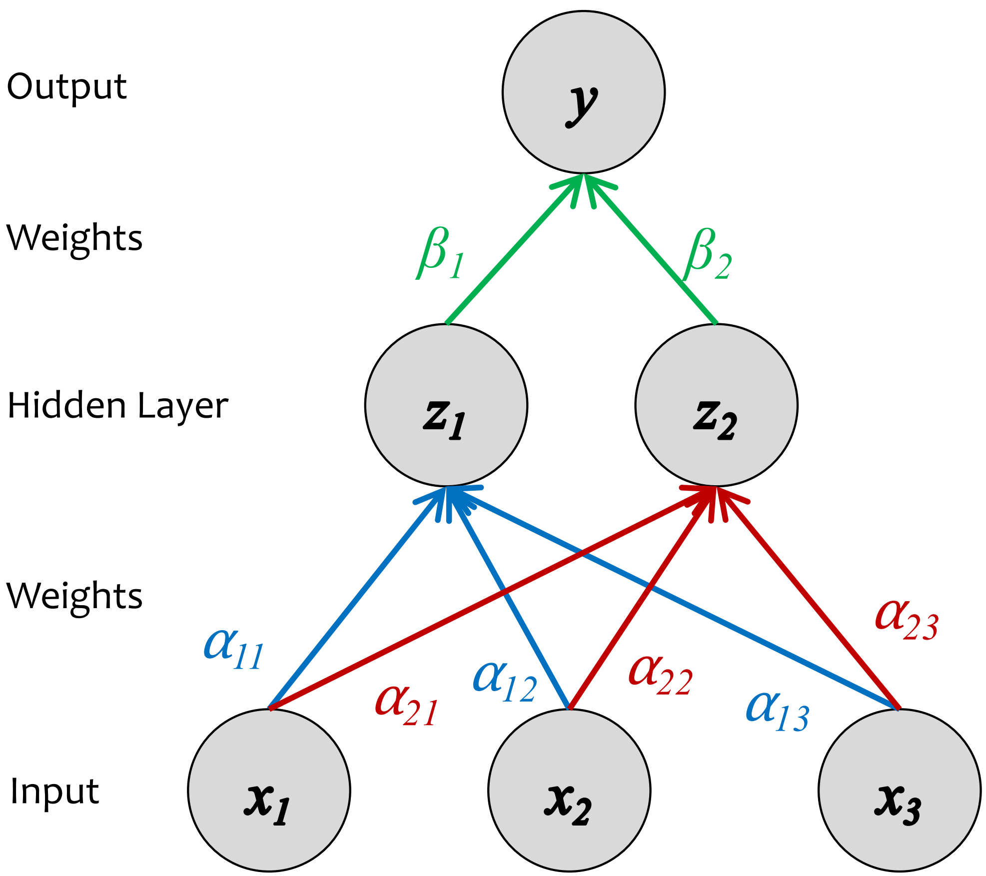
SGD with Backprop

Example: 1-Hidden Layer Neural Network

Algorithm 1 Stochastic Gradient Descent (SGD)

procedure SGD(Training data $\mathcal{D}$, test data $\mathcal{D}_t$)
    Initialize parameters $\boldsymbol{\alpha}, \boldsymbol{\beta}$
    for $e \in \{1, 2, \ldots, E\}$ **do**
        for $(\mathbf{x}, \mathbf{y}) \in \mathcal{D}$ **do**
            Compute neural network layers:
            $\mathbf{o} = \texttt{object}(\mathbf{x}, \mathbf{a}, \mathbf{b}, \mathbf{z}, \hat{\mathbf{y}}, J) = \text{NNForward}(\mathbf{x}, \mathbf{y}, \boldsymbol{\alpha}, \boldsymbol{\beta})$
            Compute gradients via backprop:
            $$\left.\begin{aligned} \mathbf{g}_{\boldsymbol{\alpha}} &= \nabla_{\boldsymbol{\alpha}} J \\ \mathbf{g}_{\boldsymbol{\beta}} &= \nabla_{\boldsymbol{\beta}} J \end{aligned}\right\} = \text{NNBackward}(\mathbf{x}, \mathbf{y}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \mathbf{o})$$
            Update parameters:
            $\boldsymbol{\alpha} \leftarrow \boldsymbol{\alpha} - \gamma \mathbf{g}_{\boldsymbol{\alpha}}$
            $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} - \gamma \mathbf{g}_{\boldsymbol{\beta}}$
        Evaluate training mean cross-entropy $J_{\mathcal{D}}(\boldsymbol{\alpha}, \boldsymbol{\beta})$
        Evaluate test mean cross-entropy $J_{\mathcal{D}_t}(\boldsymbol{\alpha}, \boldsymbol{\beta})$
    return parameters $\boldsymbol{\alpha}, \boldsymbol{\beta}$

Backpropagation



(E) **Output (sigmoid)**
$y = \frac{1}{1+\exp(-b)}$

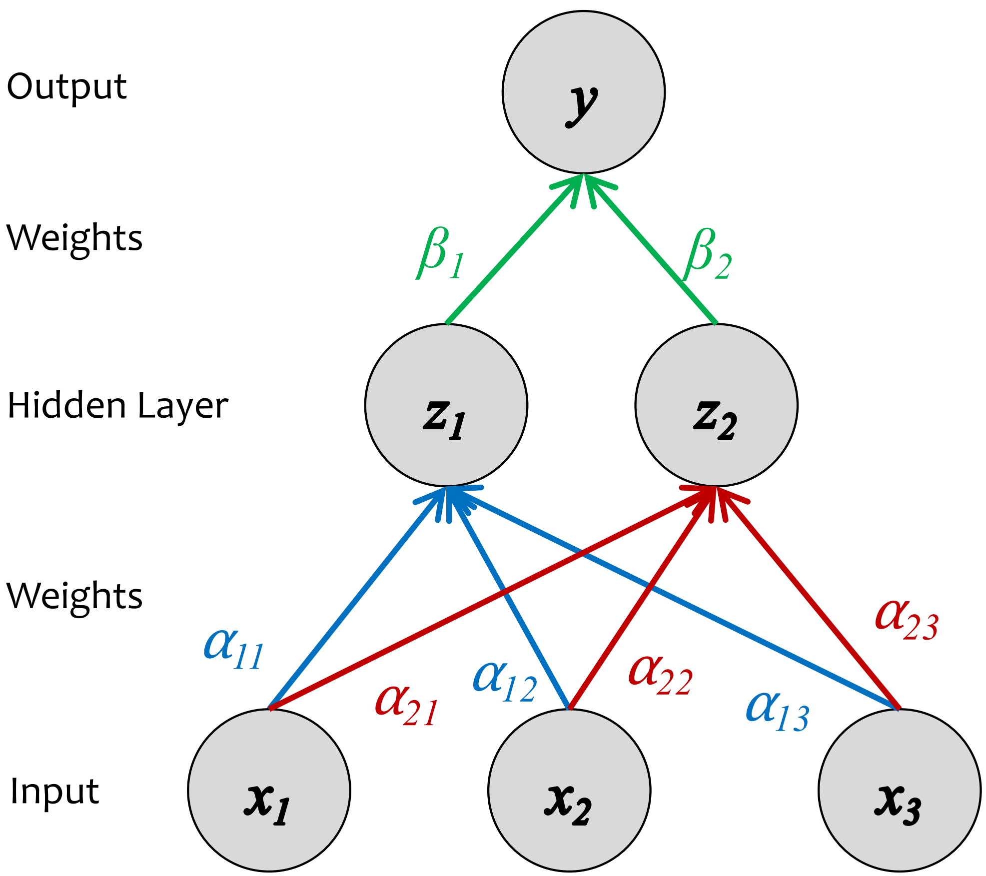
(D) **Output (linear)**
$b = \sum_{j=0}^{D} \beta_j z_j$

(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 $x_i, \ \forall i$

Backpropagation



(F) **Loss**
$J = \frac{1}{2}(y - y^*)^2$

↑

(E) **Output (sigmoid)**
$y = \frac{1}{1+\exp(-b)}$

↑

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

↑

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

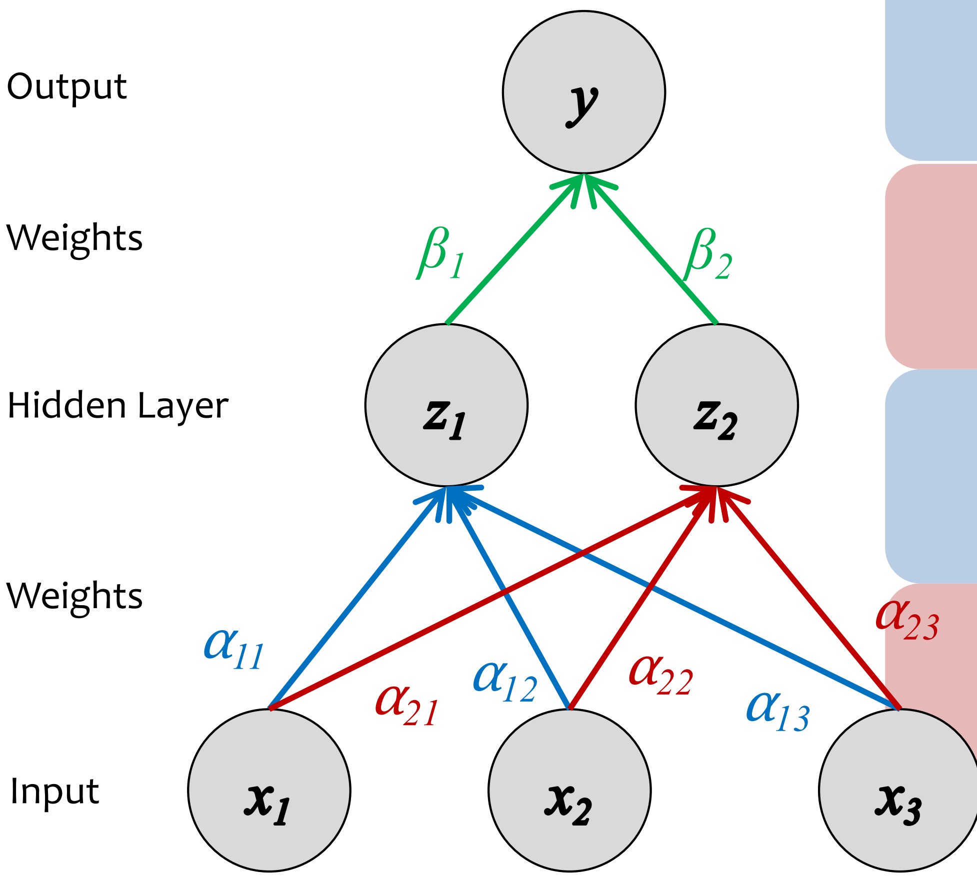
↑

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

↑

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

Training — Backpropagation



(F) **Loss**
$J = \frac{1}{2}(y - y^*)^2$

(E) **Output (sigmoid)**
$y = \frac{1}{1+\exp(-b)}$

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

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

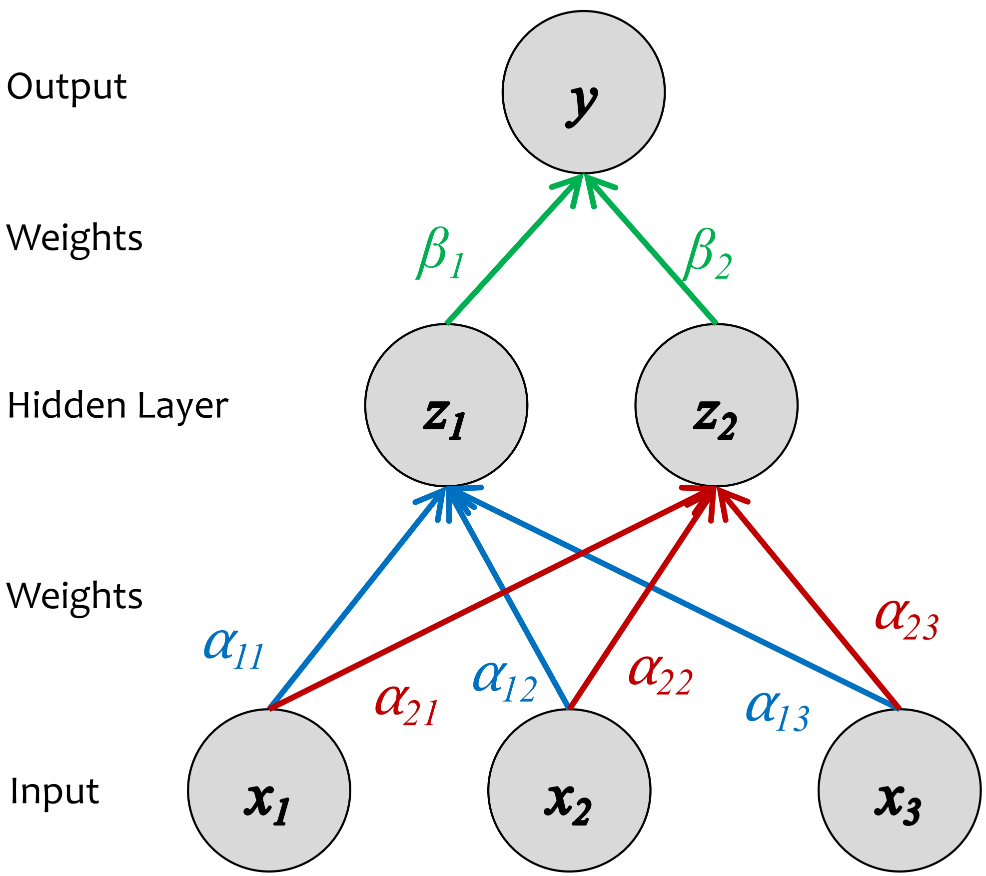
(B) **Hidden (linear)**
$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i, \ \forall j$

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

A 1-Hidden Layer Neural Network

BACKPROPAGATION FOR A NEURAL NETWORK

Training | Backpropagation



(E) **Output (sigmoid)**
$y = \frac{1}{1+\exp(-b)}$

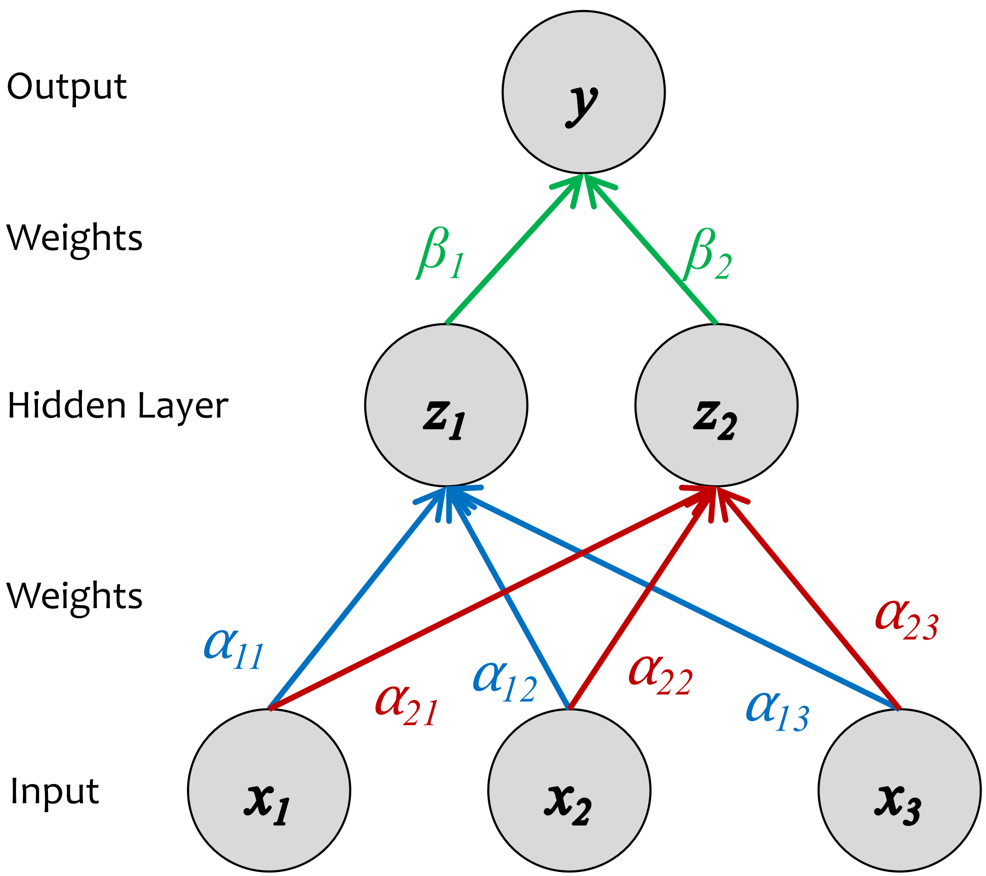
(D) **Output (linear)**
$b = \sum_{j=0}^{D} \beta_j z_j$

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

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

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

Training Backpropagation



(F) **Loss**
$J = \frac{1}{2}(y - y^*)^2$

(E) **Output (sigmoid)**
$y = \frac{1}{1+\exp(-b)}$

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

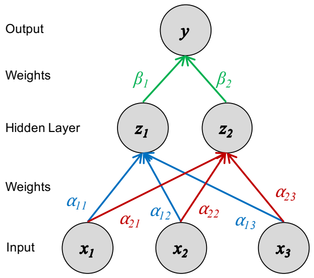
(C) **Hidden (sigmoid)**
$z_j = \frac{1}{1+\exp(-a_j)}, \ \forall j$

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

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

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

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

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

$$\begin{aligned}
\frac{ds}{db} &= \frac{\exp(-b)}{(\exp(-b) + 1)^2} && (2)\\
&= \frac{\exp(-b) + 1 - 1}{(\exp(-b) + 1 + 1 - 1)^2} && (3)\\
&= \frac{\exp(-b) + 1 - 1}{(\exp(-b) + 1)^2} && (4)\\
&= \frac{\exp(-b) + 1}{(\exp(-b) + 1)^2} - \frac{1}{(\exp(-b) + 1)^2} && (5)\\
&= \frac{1}{(\exp(-b) + 1)} - \frac{1}{(\exp(-b) + 1)^2} && (6)\\
&= \frac{1}{(\exp(-b) + 1)} - \left(\frac{1}{(\exp(-b) + 1)} \frac{1}{(\exp(-b) + 1)}\right) && (7)\\
&= \frac{1}{(\exp(-b) + 1)} \left(1 - \frac{1}{(\exp(-b) + 1)}\right) && (8)\\
&= s(1 - s) && (9)
\end{aligned}$$

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	$y = \frac{1}{1 + \exp(-b)}$	$\frac{dJ}{db} = \frac{dJ}{dy}\frac{dy}{db}, \boxed{\frac{dy}{db} = \frac{\exp(-b)}{(\exp(-b) + 1)^2}}$
Linear	$b = \sum_{j=0}^{D} \beta_j z_j$	$\frac{dJ}{d\beta_j} = \frac{dJ}{db}\frac{db}{d\beta_j}, \frac{db}{d\beta_j} = z_j$ $\frac{dJ}{dz_j} = \frac{dJ}{db}\frac{db}{dz_j}, \frac{db}{dz_j} = \beta_j$
Sigmoid	$z_j = \frac{1}{1 + \exp(-a_j)}$	$\frac{dJ}{da_j} = \frac{dJ}{dz_j}\frac{dz_j}{da_j}, \boxed{\frac{dz_j}{da_j} = \frac{\exp(-a_j)}{(\exp(-a_j) + 1)^2}}$
Linear	$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i$	$\frac{dJ}{d\alpha_{ji}} = \frac{dJ}{da_j}\frac{da_j}{d\alpha_{ji}}, \frac{da_j}{d\alpha_{ji}} = x_i$ $\frac{dJ}{dx_i} = \sum_{j=0}^{D} \frac{dJ}{da_j}\frac{da_j}{dx_i}, \frac{da_j}{dx_i} = \alpha_{ji}$

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	$y = \frac{1}{1 + \exp(-b)}$	$\frac{dJ}{db} = \frac{dJ}{dy}\frac{dy}{db}, \boxed{\frac{dy}{db} = y(1 - y)}$
Linear	$b = \sum_{j=0}^{D} \beta_j z_j$	$\frac{dJ}{d\beta_j} = \frac{dJ}{db}\frac{db}{d\beta_j}, \frac{db}{d\beta_j} = z_j$ $\frac{dJ}{dz_j} = \frac{dJ}{db}\frac{db}{dz_j}, \frac{db}{dz_j} = \beta_j$
Sigmoid	$z_j = \frac{1}{1 + \exp(-a_j)}$	$\frac{dJ}{da_j} = \frac{dJ}{dz_j}\frac{dz_j}{da_j}, \boxed{\frac{dz_j}{da_j} = z_j(1 - z_j)}$
Linear	$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i$	$\frac{dJ}{d\alpha_{ji}} = \frac{dJ}{da_j}\frac{da_j}{d\alpha_{ji}}, \frac{da_j}{d\alpha_{ji}} = x_i$ $\frac{dJ}{dx_i} = \sum_{j=0}^{D} \frac{dJ}{da_j}\frac{da_j}{dx_i}, \frac{da_j}{dx_i} = \alpha_{ji}$

Training

SGD with Backprop

Example: 1-Hidden Layer Neural Network

Algorithm 1 Stochastic Gradient Descent (SGD)

procedure SGD(Training data $\mathcal{D}$, test data $\mathcal{D}_t$)
    Initialize parameters $\boldsymbol{\alpha}, \boldsymbol{\beta}$
    for $e \in \{1, 2, \ldots, E\}$ **do**
        for $(\mathbf{x}, \mathbf{y}) \in \mathcal{D}$ **do**
            Compute neural network layers:
            $\mathbf{o} = \texttt{object}(\mathbf{x}, \mathbf{a}, \mathbf{b}, \mathbf{z}, \hat{\mathbf{y}}, J) = \text{NNFORWARD}(\mathbf{x}, \mathbf{y}, \boldsymbol{\alpha}, \boldsymbol{\beta})$
            Compute gradients via backprop:
            $$\left.\begin{aligned} \mathbf{g}_{\boldsymbol{\alpha}} &= \nabla_{\boldsymbol{\alpha}} J \\ \mathbf{g}_{\boldsymbol{\beta}} &= \nabla_{\boldsymbol{\beta}} J \end{aligned}\right\} = \text{NNBACKWARD}(\mathbf{x}, \mathbf{y}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \mathbf{o})$$
            Update parameters:
            $\boldsymbol{\alpha} \leftarrow \boldsymbol{\alpha} - \gamma \mathbf{g}_{\boldsymbol{\alpha}}$
            $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} - \gamma \mathbf{g}_{\boldsymbol{\beta}}$
        Evaluate training mean cross-entropy $J_{\mathcal{D}}(\boldsymbol{\alpha}, \boldsymbol{\beta})$
        Evaluate test mean cross-entropy $J_{\mathcal{D}_t}(\boldsymbol{\alpha}, \boldsymbol{\beta})$
    return parameters $\boldsymbol{\alpha}, \boldsymbol{\beta}$

THE BACKPROPAGATION ALGORITHM

Backpropagation

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 $v_1, \ldots, v_N$
   a. Compute $u_i = g_i(v_1, \ldots, v_N)$
   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 $u_1, \ldots, u_M$ denote all the nodes with v_j as an input
   Assuming that $y = h(\mathbf{u}) = h(u_1, \ldots, u_M)$
   and $\mathbf{u} = g(\mathbf{v})$ or equivalently $u_i = g_i(v_1, \ldots, v_j, \ldots, v_N)$ 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

Backpropagation

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 $v_1, \ldots, v_N$
   a. Compute $u_i = g_i(v_1, \ldots, v_N)$
   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 $u_i = g_i(v_1, \ldots, v_N)$
   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

Backpropagation

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

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 th

– Decision functio

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

– Loss function

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

opp site the gradient)

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

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

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

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	$\frac{\partial y}{\partial \mathbf{X}}$	$\frac{\partial \mathbf{y}}{\partial \mathbf{X}}$	$\frac{\partial \mathbf{Y}}{\partial \mathbf{X}}$

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

denominator layout

Types of Derivatives	**scalar**	**vector**
scalar	$\frac{\partial y}{\partial x} = \begin{bmatrix} \frac{\partial y}{\partial x} \end{bmatrix}$	row vec $\frac{\partial \mathbf{y}}{\partial x} = \begin{bmatrix} \frac{\partial y_1}{\partial x} & \frac{\partial y_2}{\partial x} & \dots & \frac{\partial y_N}{\partial x} \end{bmatrix}$
vector	$\frac{\partial y}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y}{\partial x_1} \\ \frac{\partial y}{\partial x_2} \\ \vdots \\ \frac{\partial y}{\partial x_P} \end{bmatrix}$ col vec.	$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_2}{\partial x_1} & \dots & \frac{\partial y_N}{\partial x_1} \\ \frac{\partial y_1}{\partial x_2} & \frac{\partial y_2}{\partial x_2} & \dots & \frac{\partial y_N}{\partial x_2} \\ \vdots & & & \\ \frac{\partial y_1}{\partial x_P} & \frac{\partial y_2}{\partial x_P} & \dots & \frac{\partial y_N}{\partial x_P} \end{bmatrix}$

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:

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

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

2. In denominator layout:

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

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

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, $B \in \mathbb{R}^{m \times n}$

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

Scalar Derivative		Vector Derivative	
$f(x)$	$\rightarrow \frac{\partial f}{\partial x}$	$f(x)$	$\rightarrow \frac{\partial f}{\partial \vec{x}}$
bx	$\rightarrow b$	$x^T B$	$\rightarrow B$
xb	$\rightarrow b$	$x^T b$	$\rightarrow b$
x^2	$\rightarrow 2x$	$x^T x$	$\rightarrow 2x$
bx^2	$\rightarrow 2bx$	$x^T B x$	$\rightarrow 2Bx$

B symmetric

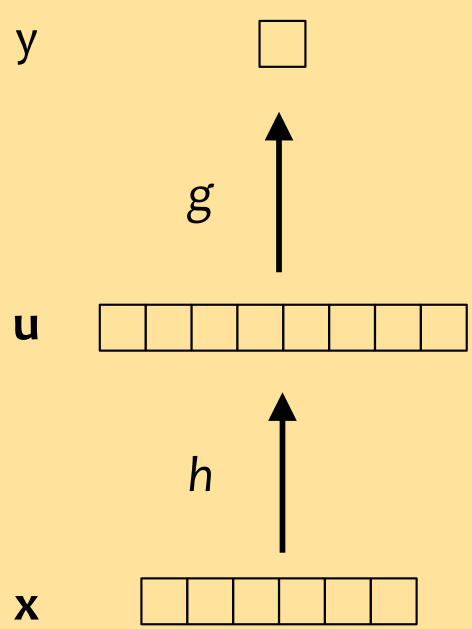
Matrix Calculus

Recall:

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

Question: Q1

Suppose y = g(**u**) and **u** = h(**x**)



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

Answer:

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

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

G = toxic