10-601 Introduction to Machine Learning

Machine Learning Department
School of Computer Science
Carnegie Mellon University

Backpropagation

Matt Gormley
Lecture 13
Mar 1, 2018

Reminders

- Homework 5: Neural Networks
  - Out: Tue, Feb 28
  - Due: Fri, Mar 9 at 11:59pm

Q&A

BACKPROPAGATION

Background

A Recipe for Machine Learning

1. Given training data:

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

2. Choose each of these:
   - Decision function

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

   - Loss function

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

3. Define goal:

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

4. Train with SGD:
(take small steps opposite the gradient)

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

Approaches to Differentiation

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

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

Approaches to Differentiation

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

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

1. Finite Difference Method
   - Pro: Great for testing implementations of backpropagation
   - Con: Slow for high dimensional inputs / outputs
   - Required: Ability to call the function f(**x**) on any input **x**
2. Symbolic Differentiation
   - Note: The method you learned in high-school
   - Note: Used by Mathematica / Wolfram Alpha / Maple
   - Pro: Yields easily interpretable derivatives
   - Con: Leads to exponential computation time if not carefully implemented
   - Required: Mathematical expression that defines f(**x**)
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(**x**)
   - Con: Slow for high dimensional outputs (e.g. vector-valued functions)
   - Required: Algorithm for computing f(**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(**x**)
   - Con: Slow for high dimensional inputs (e.g. vector-valued **x**)
   - Required: Algorithm for computing f(**x**)

Training

Finite Difference Method

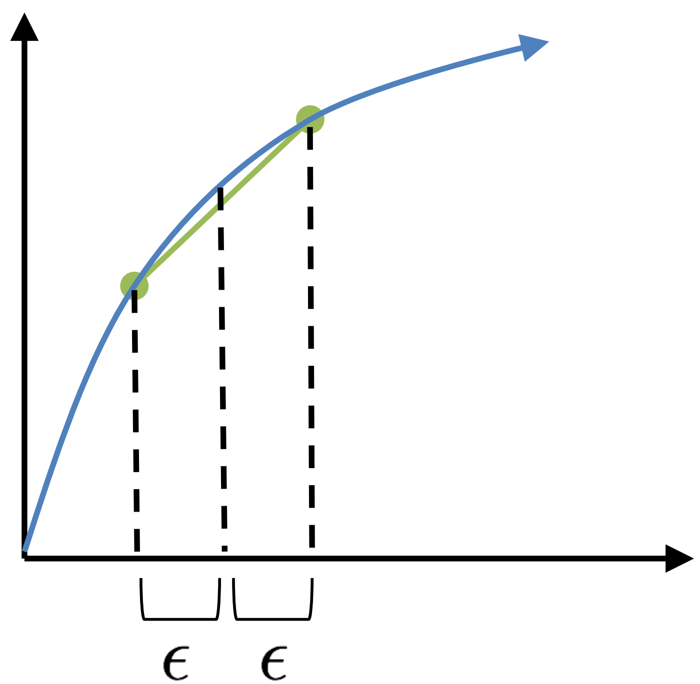
The *centered* finite difference approximation is:

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

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

Notes:

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



Training

Symbolic Differentiation

Differentiation Quiz #1:

Suppose x = 2 and z = 3, what are dy/dx and dy/dz for the function below?

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

Symbolic Differentiation

Differentiation Quiz #2:

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

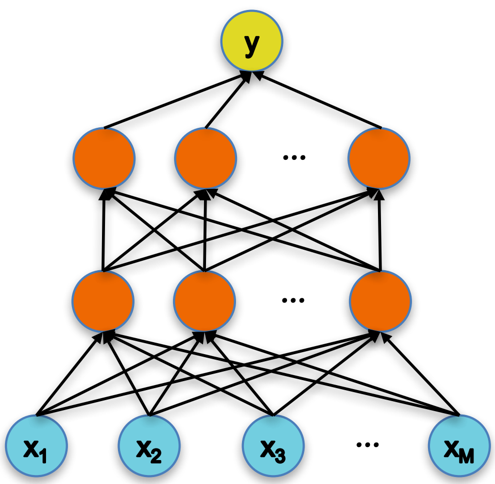
$$y = \sigma(\boldsymbol{\beta}^T \sigma((\boldsymbol{\alpha}^{(2)})^T \sigma((\boldsymbol{\alpha}^{(1)})^T \mathbf{x}))$$

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

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

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

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



Training

Chain Rule

Whiteboard

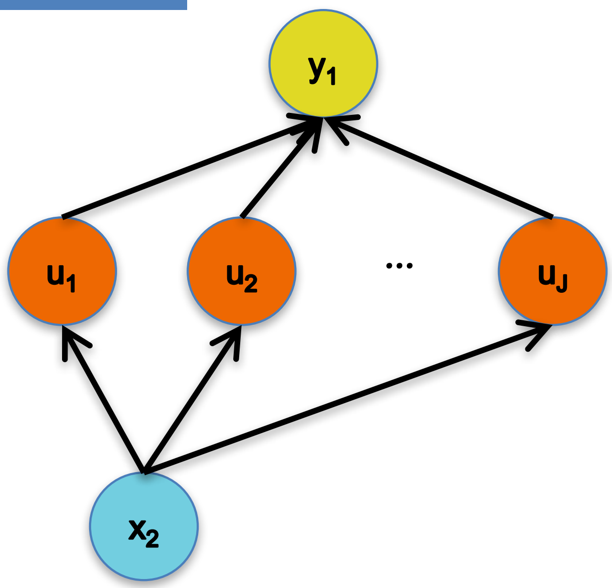
- Chain Rule of Calculus

Chain Rule

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

Chain Rule:

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

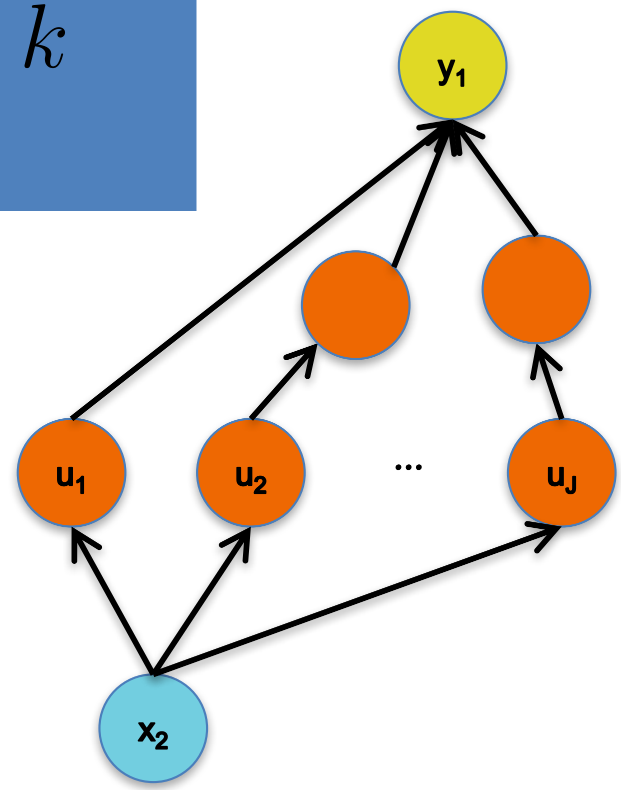


Chain Rule

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

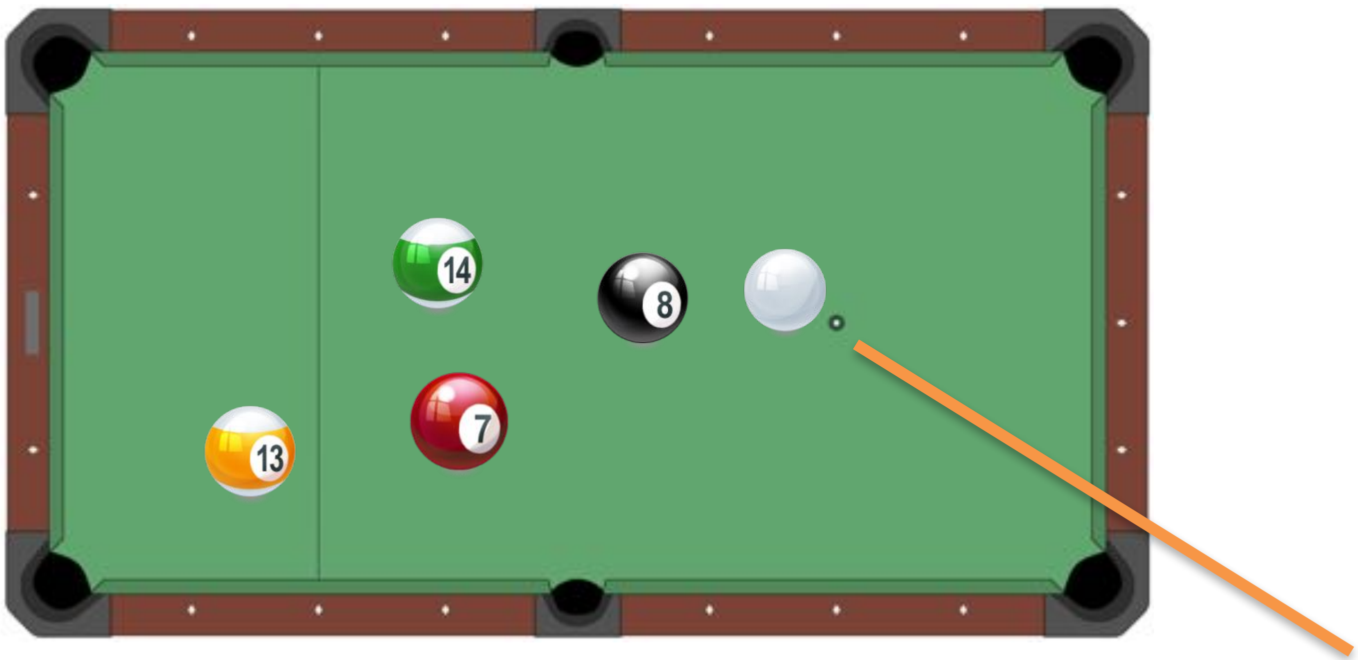
Chain Rule:

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



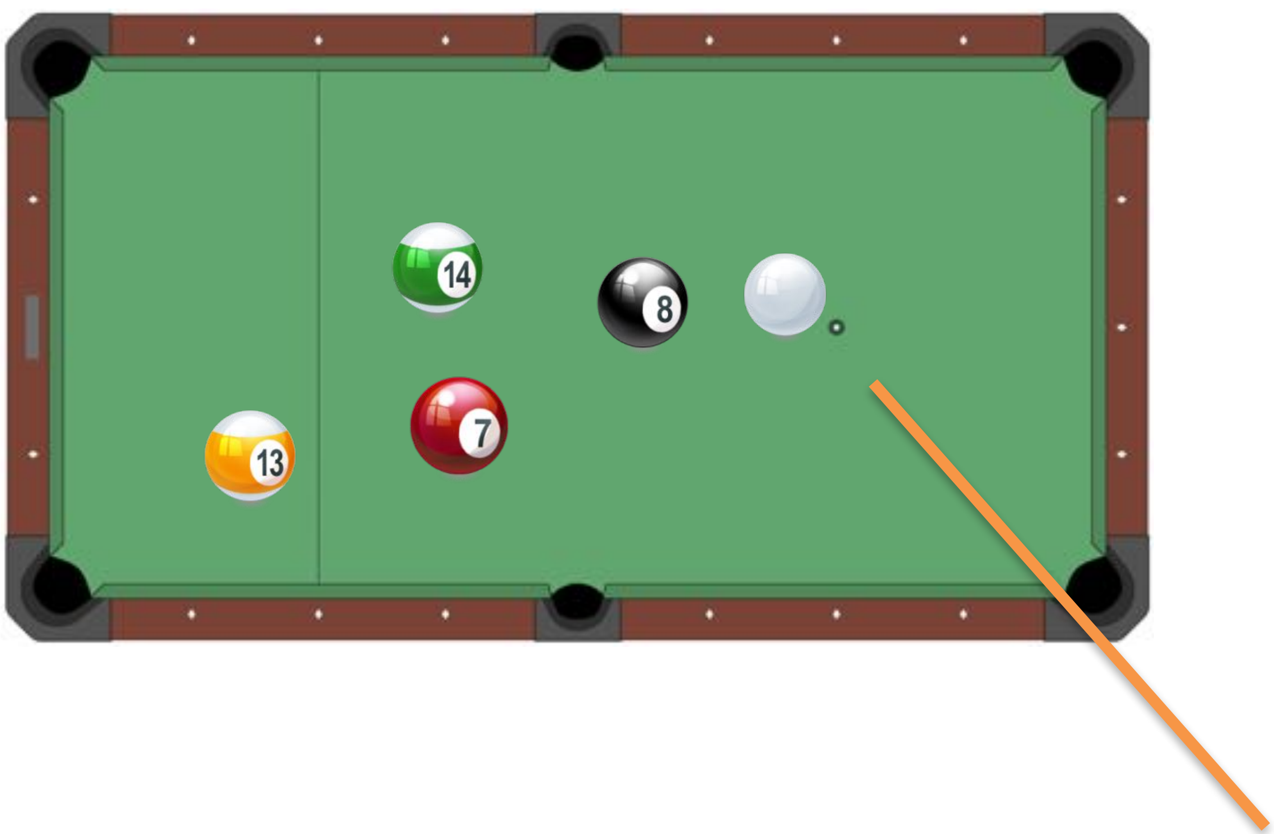
Backpropagation is just repeated application of the **chain rule** from Calculus 101.

Error Back-Propagation



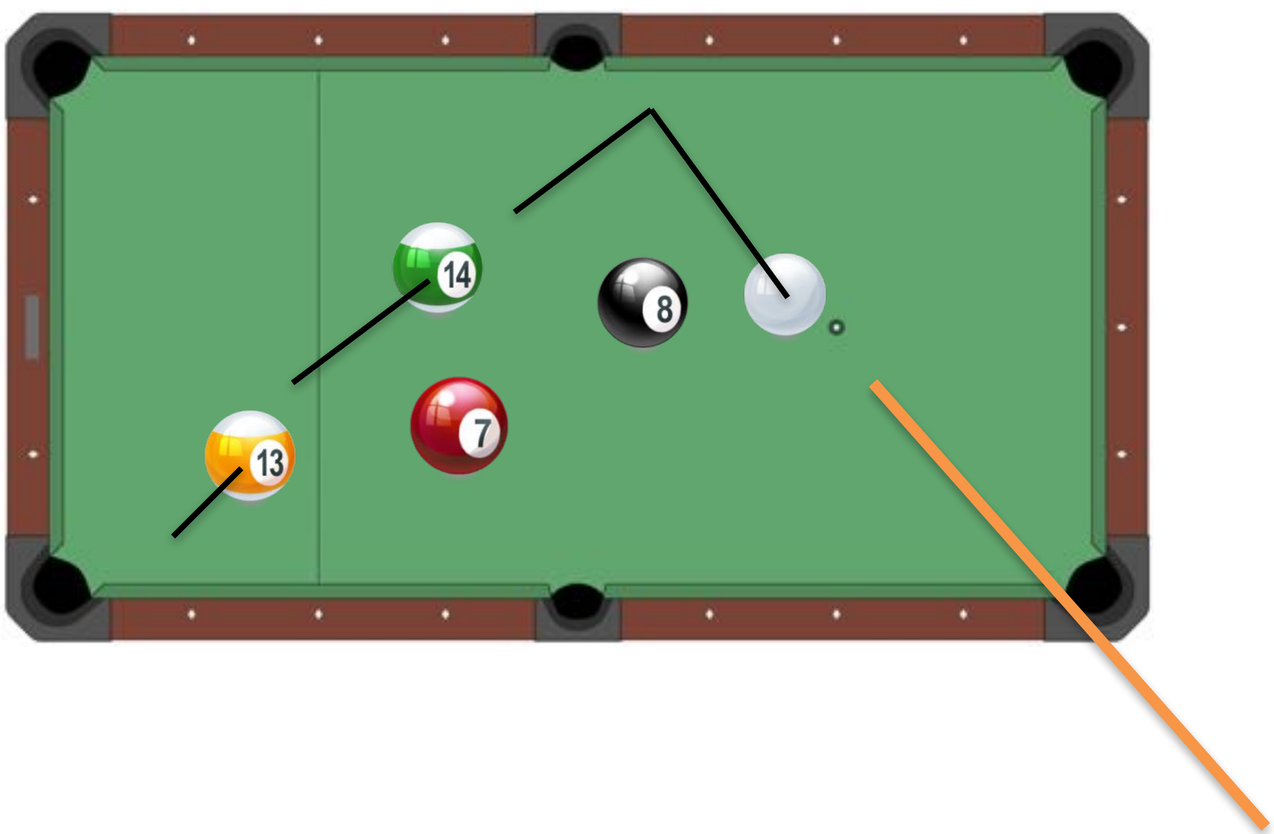
Slide from (Stoyanov & Eisner, 2012)

Error Back-Propagation



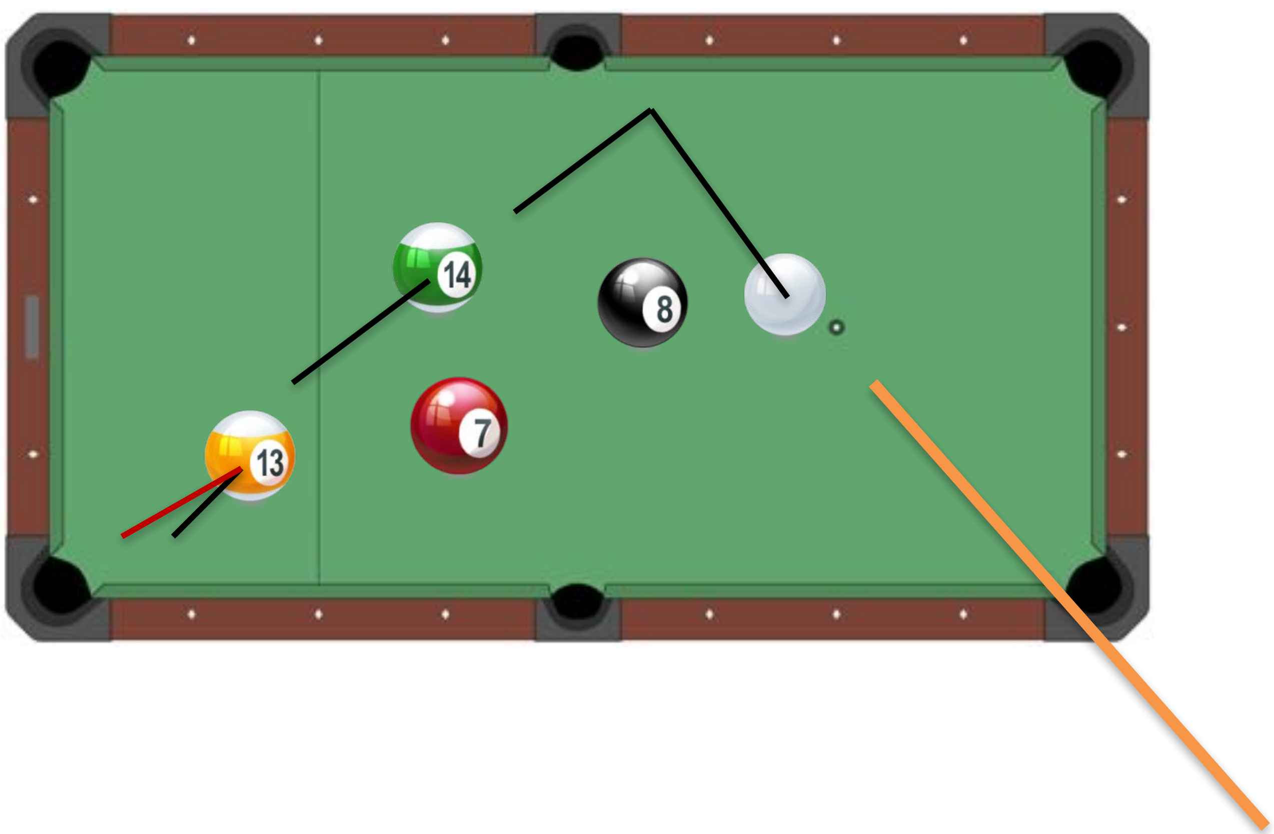
Slide from (Stoyanov & Eisner, 2012)

Error Back-Propagation



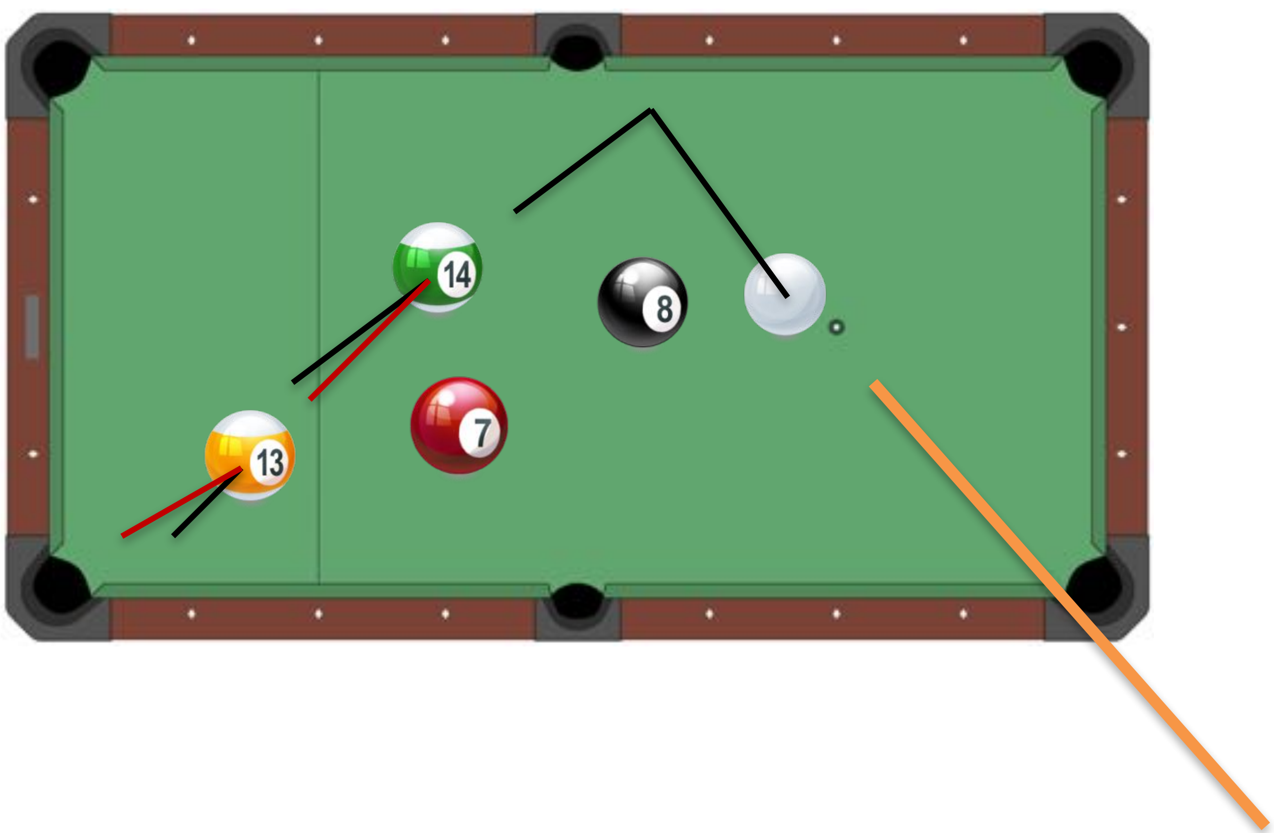
Slide from (Stoyanov & Eisner, 2012)

Error Back-Propagation



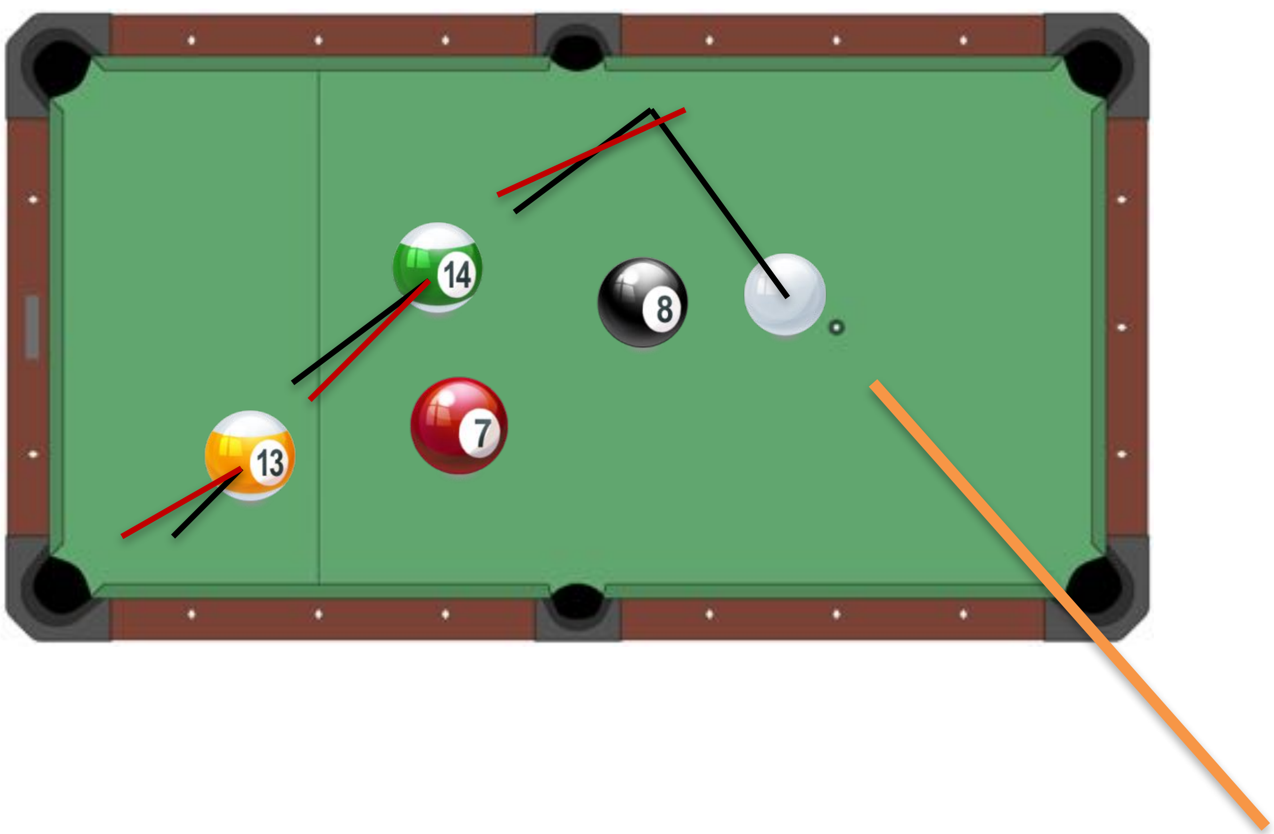
Slide from (Stoyanov & Eisner, 2012)

Error Back-Propagation



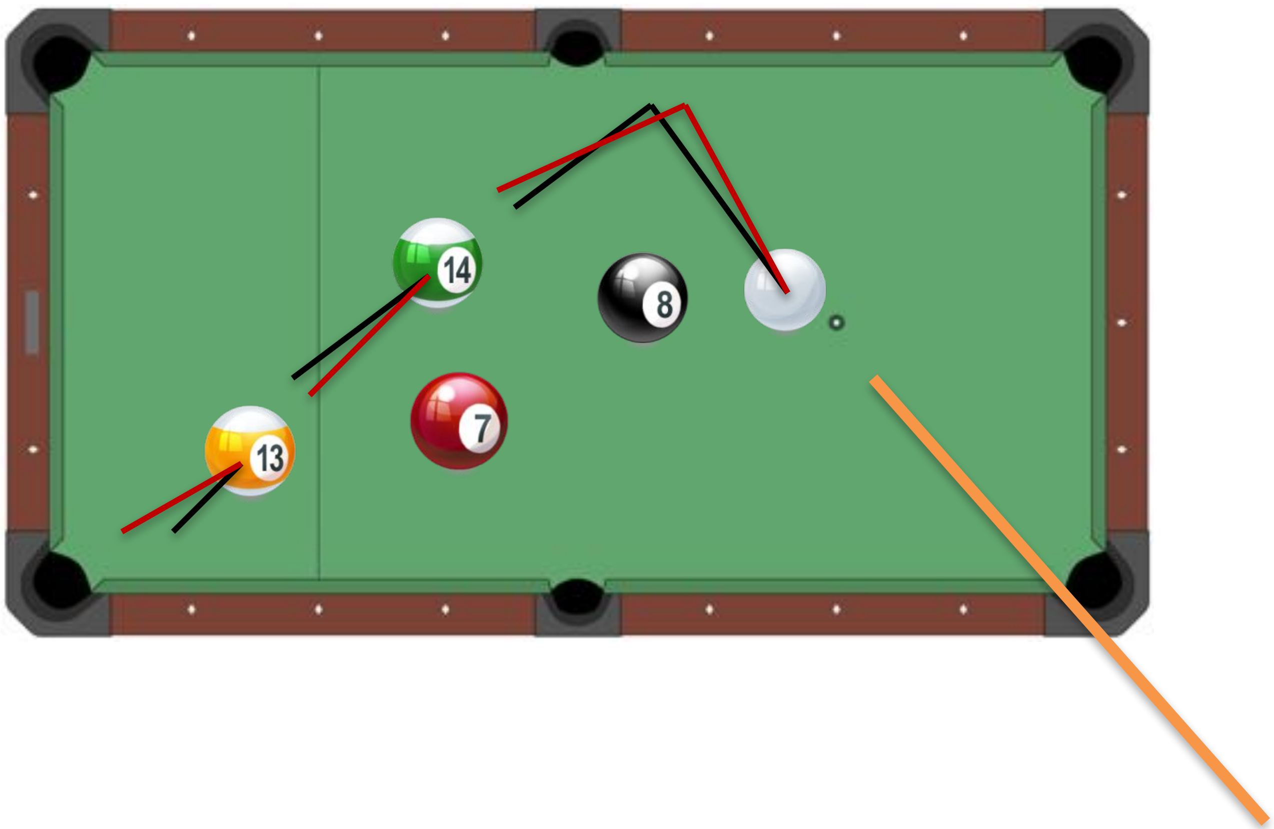
Slide from (Stoyanov & Eisner, 2012)

Error Back-Propagation



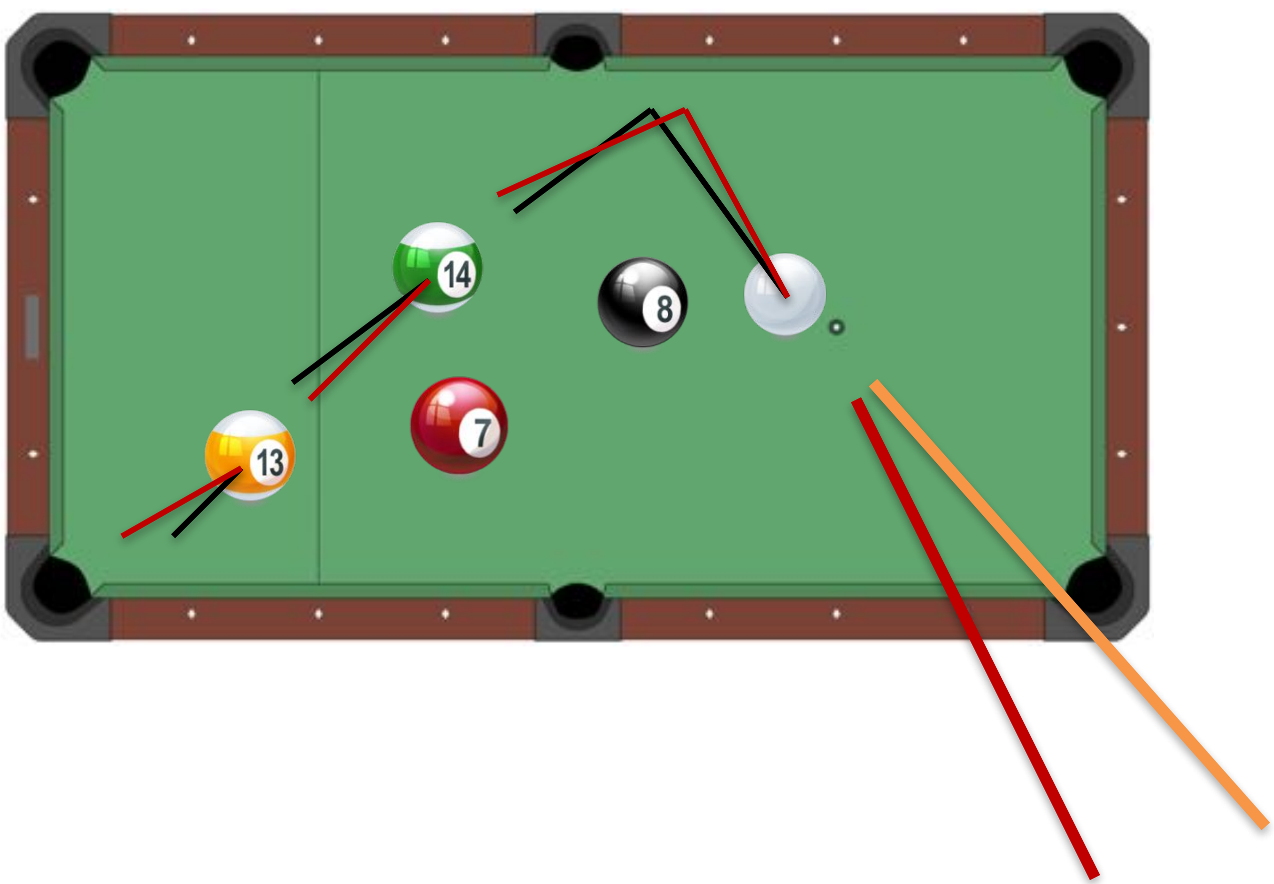
Slide from (Stoyanov & Eisner, 2012)

Error Back-Propagation



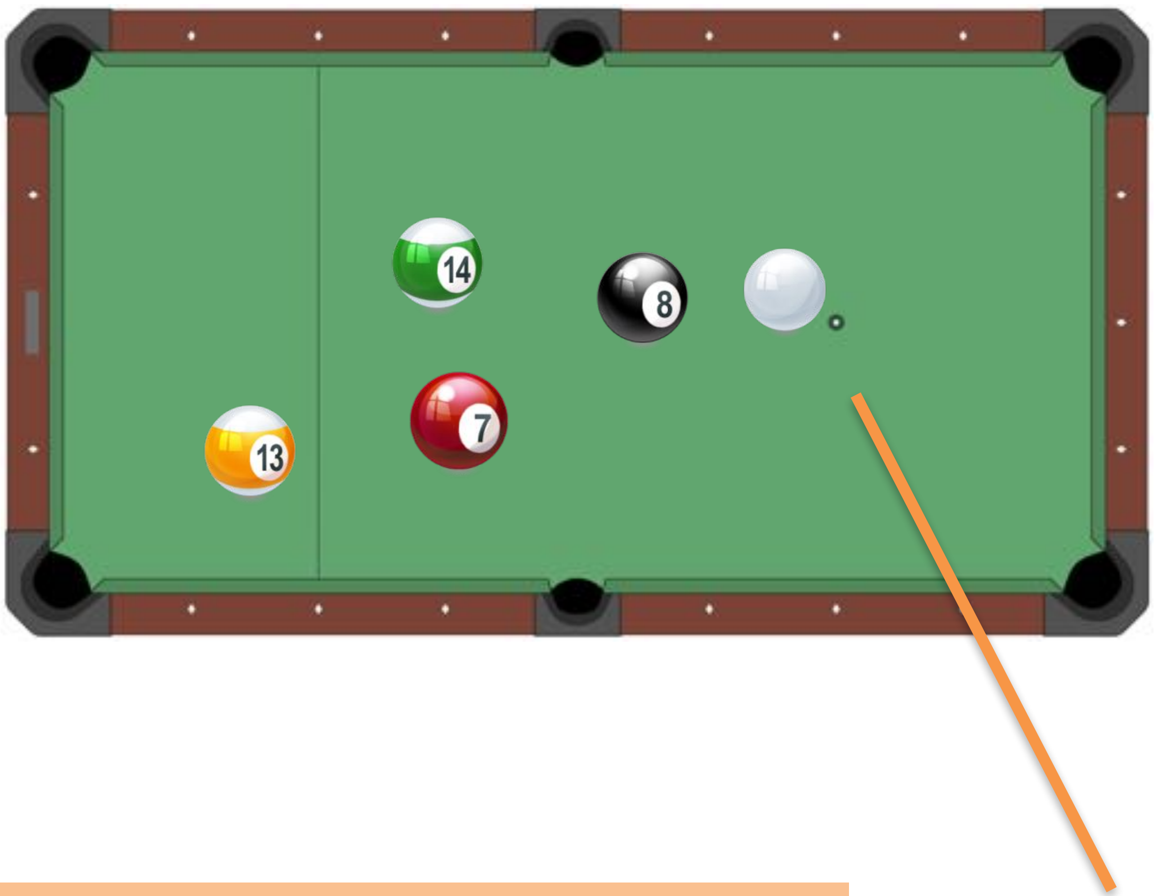
Slide from (Stoyanov & Eisner, 2012)

Error Back-Propagation



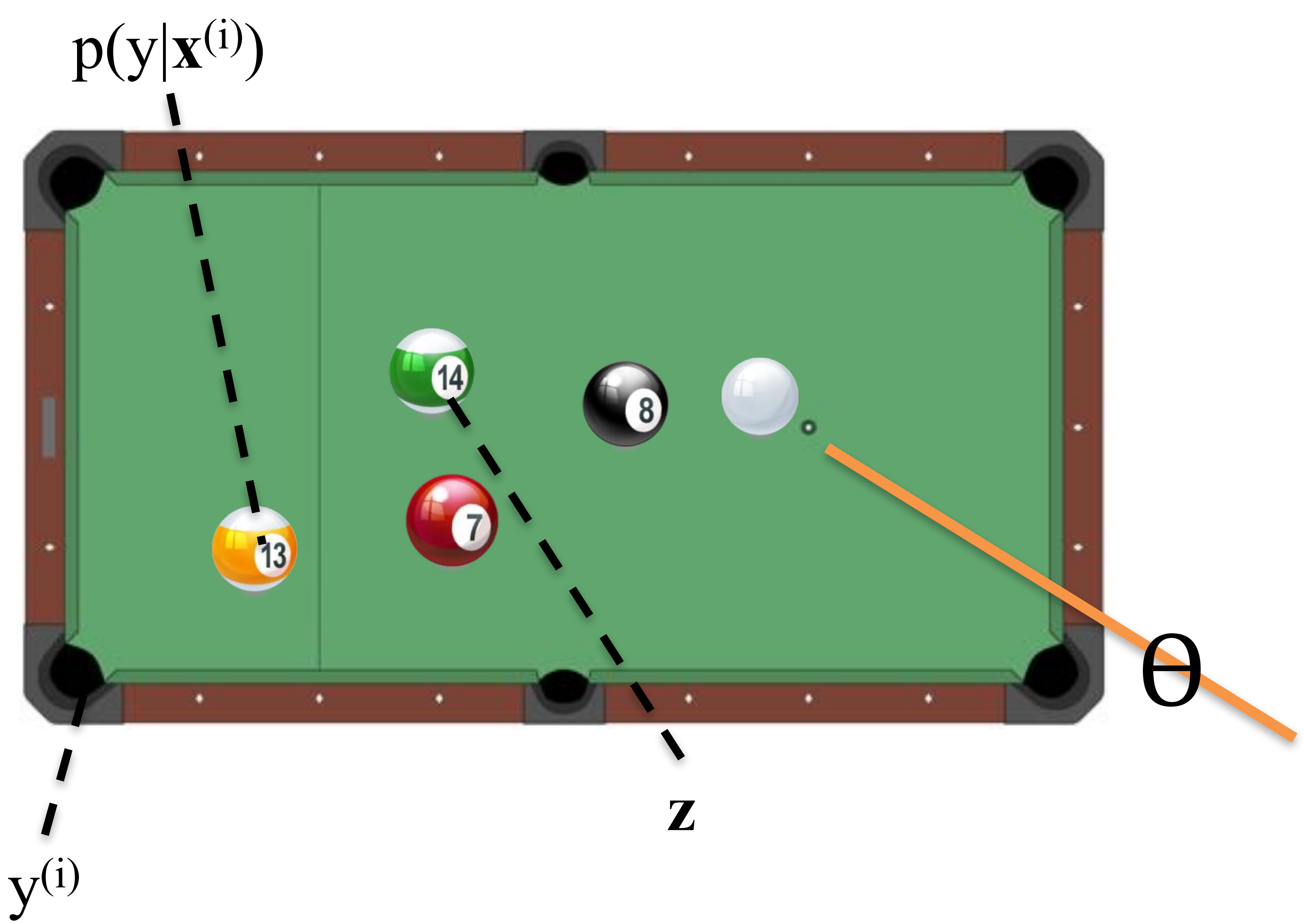
Slide from (Stoyanov & Eisner, 2012)

Error Back-Propagation



Slide from (Stoyanov & Eisner, 2012)

Error Back-Propagation



Slide from (Stoyanov & Eisner, 2012)

Training

Backpropagation

Whiteboard

- Example: Backpropagation for Chain Rule #1

Differentiation Quiz #1:

Suppose x = 2 and z = 3, what are dy/dx and dy/dz for the function below?

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

Backpropagation

Automatic Differentiation – Reverse Mode (aka. Backpropagation)

Forward Computation

1. Write an **algorithm** for evaluating the function y = f(**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

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

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

$J = cos(u)$

$u = u_1 + u_2$

$u_1 = sin(t)$

$u_2 = 3t$

$t = x^2$

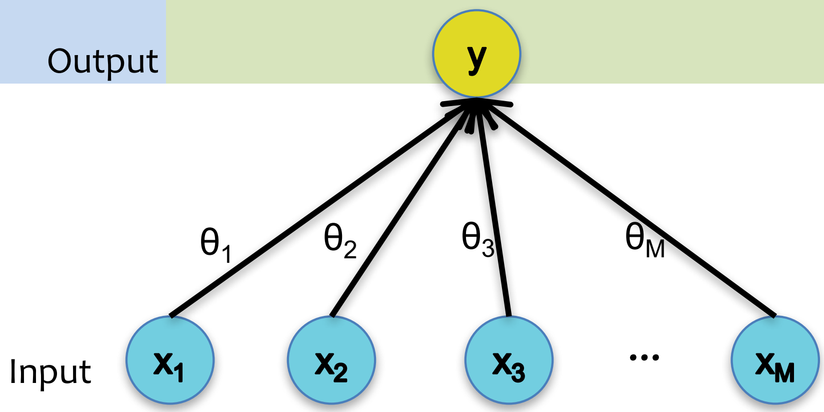
Backpropagation

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

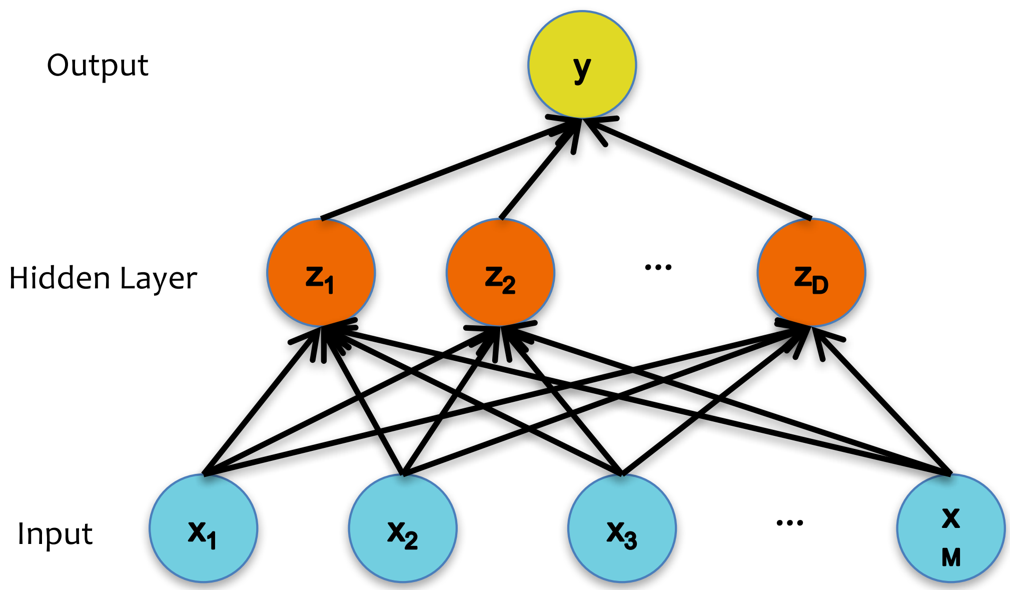
Case 1: Logistic Regression



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

Training

Backpropagation



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

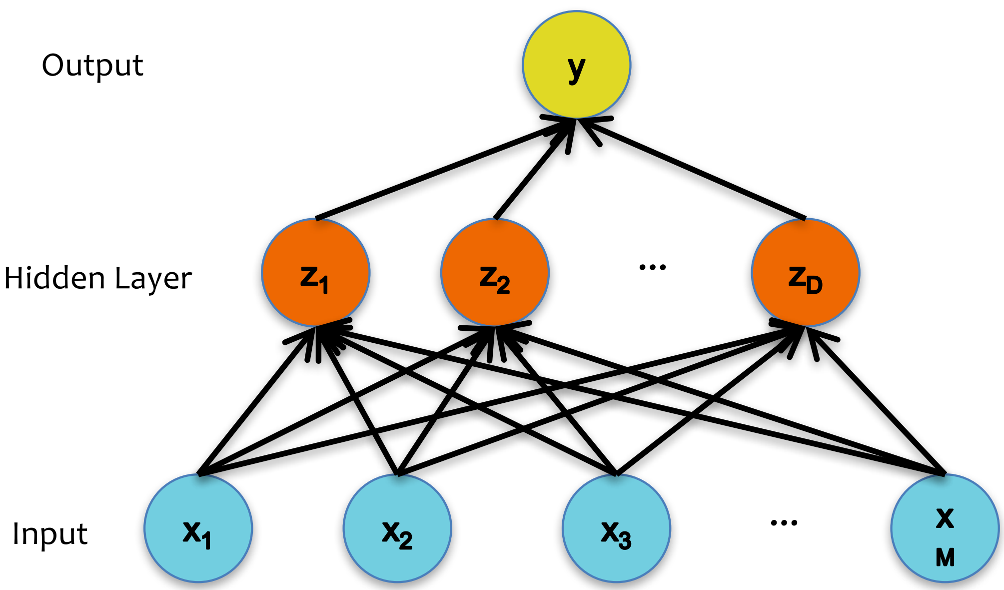
(D) **Output (linear)**
$$b = \sum_{j=0}^{D} \beta_j z_j$$

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

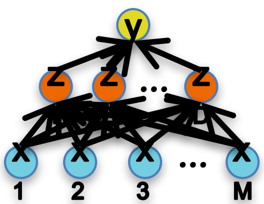
(B) **Hidden (linear)**
$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i, \ \forall j$

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

Training

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

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

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

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

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

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

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

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

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	$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i$	$\frac{dJ}{d\alpha_{ji}} = \frac{dJ}{da_j}\frac{da_j}{d\alpha_{ji}}, \quad \frac{da_j}{d\alpha_{ji}} = x_i$ $\frac{dJ}{dx_i} = \frac{dJ}{da_j}\frac{da_j}{dx_i}, \quad \frac{da_j}{dx_i} = \sum_{j=0}^{D} \alpha_{ji}$

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

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

Training

Backpropagation

Whiteboard

- SGD for Neural Network
- Example: Backpropagation for Neural Network

Backpropagation

Backpropagation (Auto.Diff. - Reverse Mode)

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**.
   a. Compute the corresponding variable's value
   b. Store the result at the node

Backward Computation

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

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

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

opposite the gradient)

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

Summary

1. **Neural Networks…**
   – provide a way of learning features
   – are highly nonlinear prediction functions
   – (can be) a highly parallel network of logistic regression classifiers
   – discover useful hidden representations of the input
2. **Backpropagation…**
   – provides an efficient way to compute gradients
   – is a special case of reverse-mode automatic differentiation

Backprop Objectives

You should be able to…

- Construct a computation graph for a function as specified by an algorithm
- Carry out the backpropagation on an arbitrary computation graph
- Construct a computation graph for a neural network, identifying all the given and intermediate quantities that are relevant
- Instantiate the backpropagation algorithm for a neural network
- Instantiate an optimization method (e.g. SGD) and a regularizer (e.g. L2) when the parameters of a model are comprised of several matrices corresponding to different layers of a neural network
- Apply the empirical risk minimization framework to learn a neural network
- Use the finite difference method to evaluate the gradient of a function
- Identify when the gradient of a function can be computed at all and when it can be computed efficiently