Machine Learning Department
School of Computer Science
Carnegie Mellon University

Backpropagation

Matt Gormley
Lecture 13
Feb. 27, 2019

Reminders

- Homework 4: Logistic Regression
  - Out: Fri, Feb 15
  - Due: Fri, Mar 1 at 11:59pm
- Homework 5: Neural Networks
  - Out: Fri, Mar 1
  - Due: Fri, Mar 22 at 11:59pm
- Today's In-Class Poll
  - http://p13.mlcourse.org
  - Also linked from Schedule page on mlcourse.org

Q&A

Q: What is mini-batch SGD?

A: A variant of SGD…

Mini-Batch SGD

- **Gradient Descent:**
  Compute true gradient exactly from all N examples
- **Mini-Batch SGD**:
  Approximate true gradient by the average gradient of K randomly chosen examples
- **Stochastic Gradient Descent (SGD)**:
  Approximate true gradient by the gradient of one randomly chosen example

Mini-Batch SGD

while not converged: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \lambda \mathbf{g}$

Three variants of first-order optimization:

Gradient Descent: $\mathbf{g} = \nabla J(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} \nabla J^{(i)}(\boldsymbol{\theta})$

SGD: $\mathbf{g} = \nabla J^{(i)}(\boldsymbol{\theta})$ where i sampled uniformly

Mini-batch SGD: $\mathbf{g} = \frac{1}{S} \sum_{s=1}^{S} \nabla J^{(i_s)}(\boldsymbol{\theta})$ where i_s sampled uniformly $\forall s$

Computing Gradients

DIFFERENTIATION

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 for an arbitrary neural network?

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

Training

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

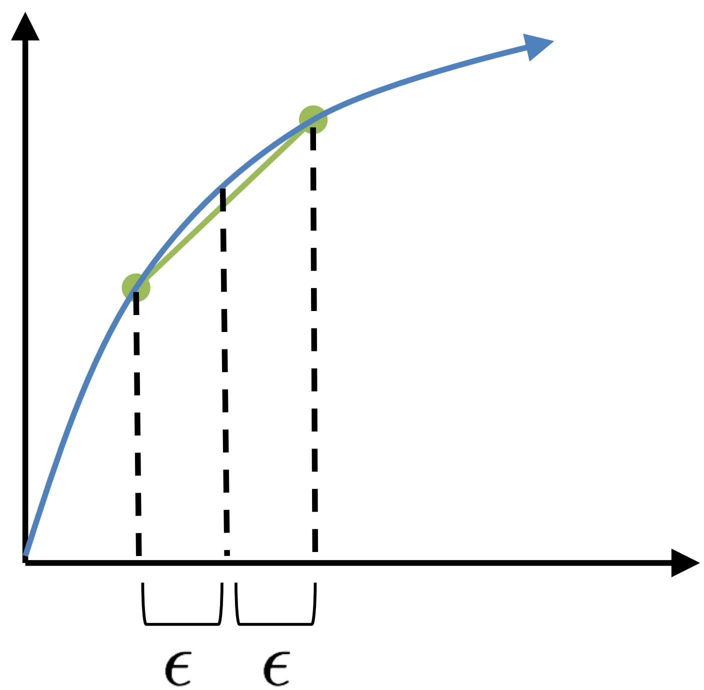
The *centered* finite difference approximation is:

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

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



Symbolic Differentiation

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]
B. [72, -42]
C. [100, 127]
D. [127, 100]
E. [1208, 810]
F. [810, 1208]
G. [1505, 94]
H. [94, 1505]

Differentiation Quiz #2:

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

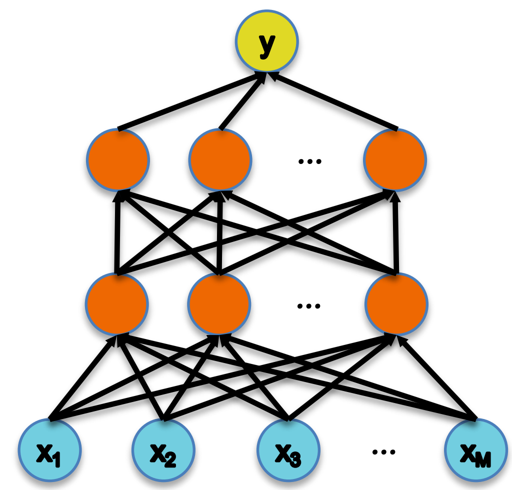
$$y = \sigma(\boldsymbol{\beta}^T \sigma((\boldsymbol{\alpha}^{(2)})^T \sigma((\boldsymbol{\alpha}^{(1)})^T \mathbf{x}))$$

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:

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

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.



CHAIN RULE

Training

Chain Rule

Chalkboard

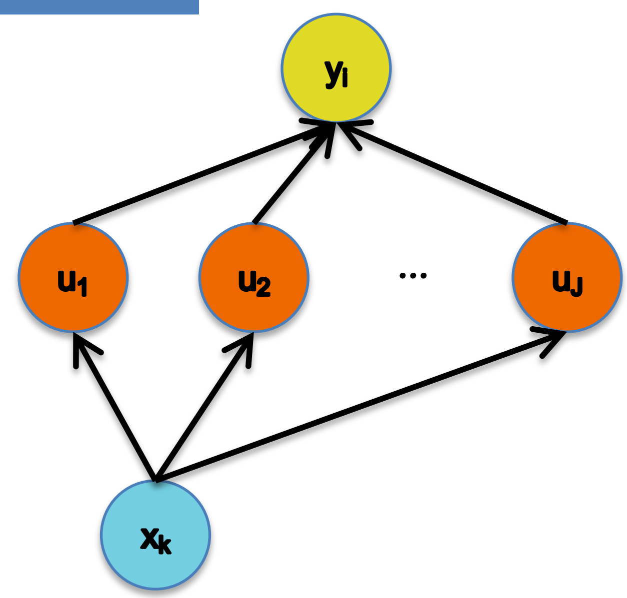
– Chain Rule of Calculus

Chain Rule

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

Chain Rule:

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



Training

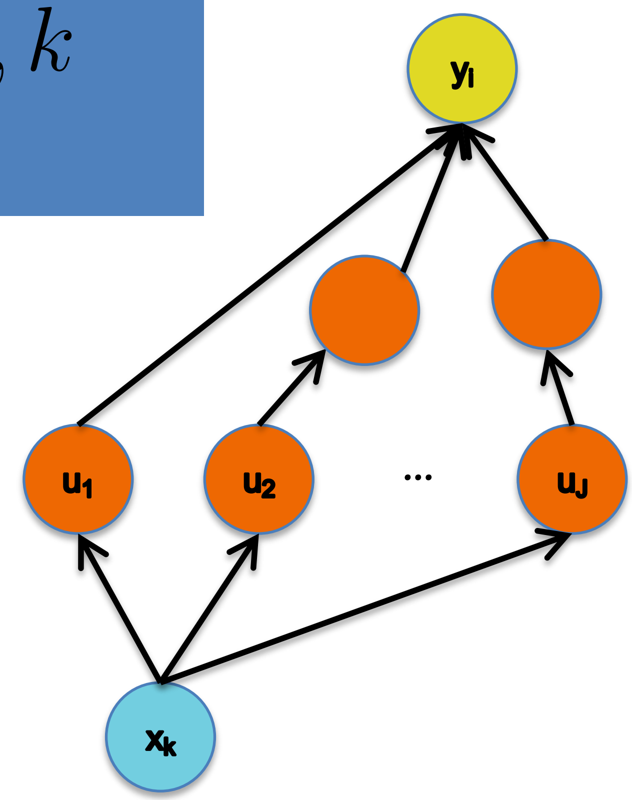
Chain Rule

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

Chain Rule:

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

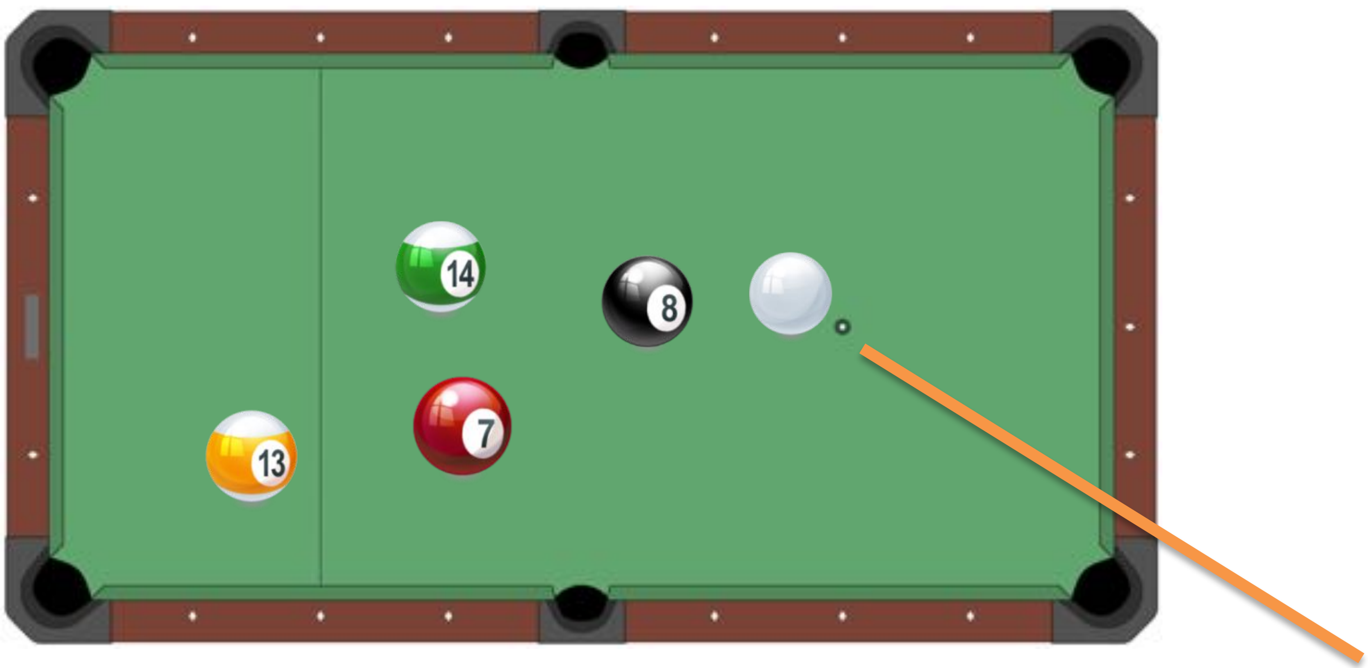
Backpropagation is just repeated application of the **chain rule** from Calculus 101.



Intuitions

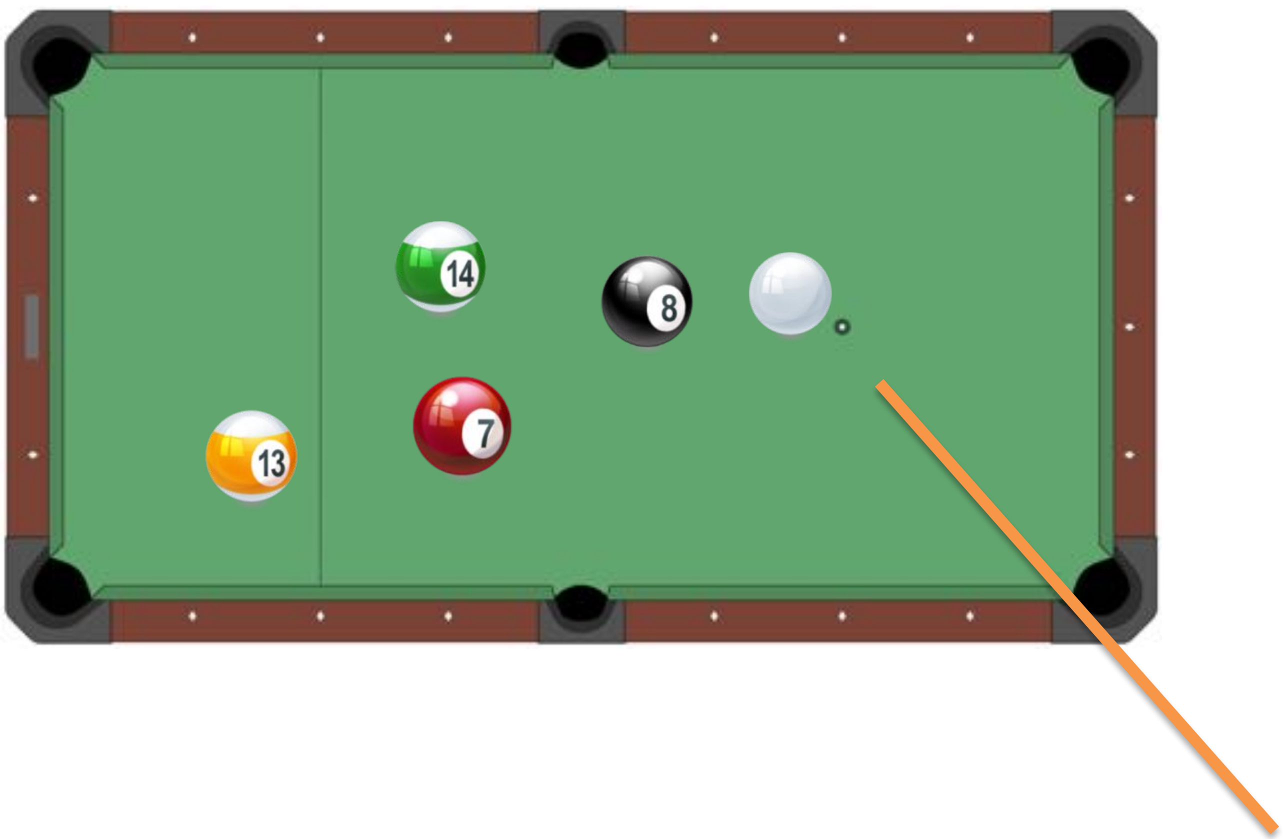
BACKPROPAGATION

Error Back-Propagation



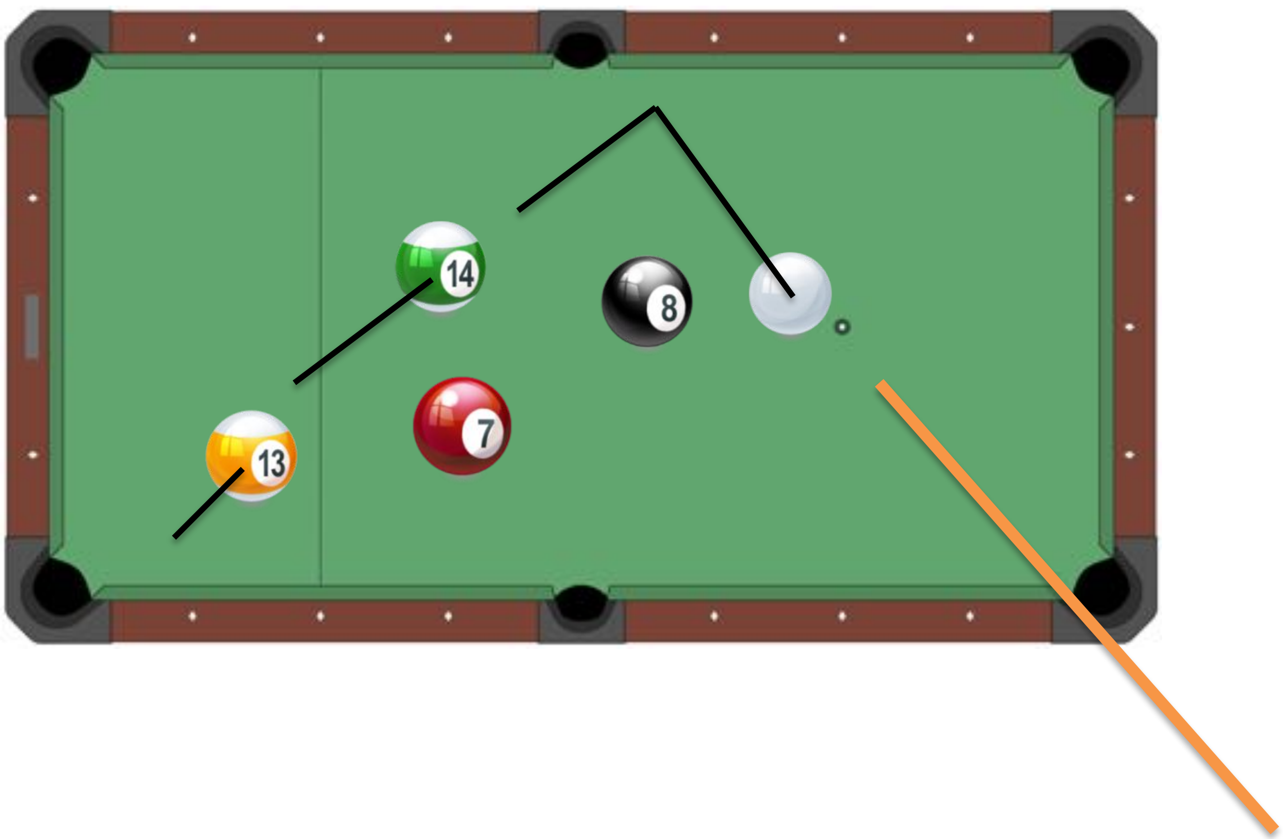
Slide from (Stoyanov & Eisner, 2012)

Error Back-Propagation



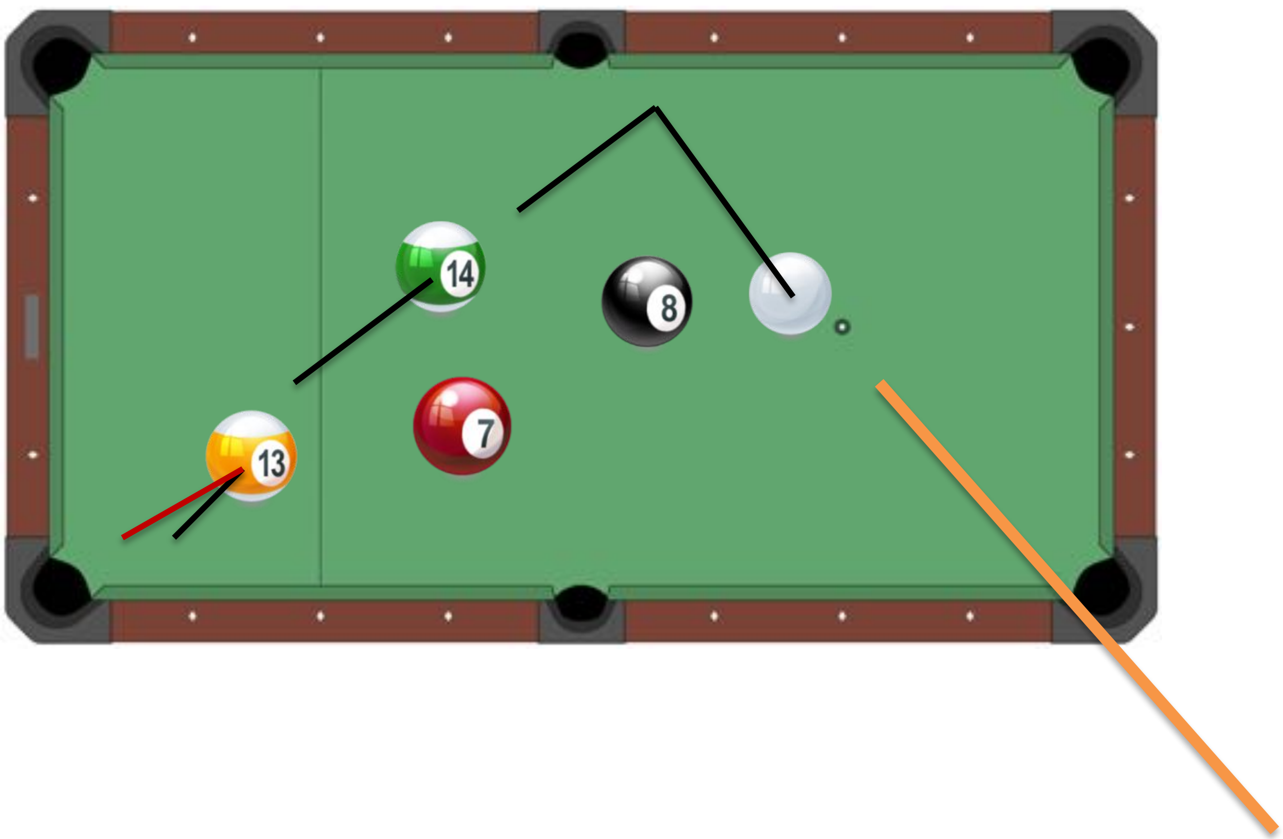
Slide from (Stoyanov & Eisner, 2012)

Error Back-Propagation



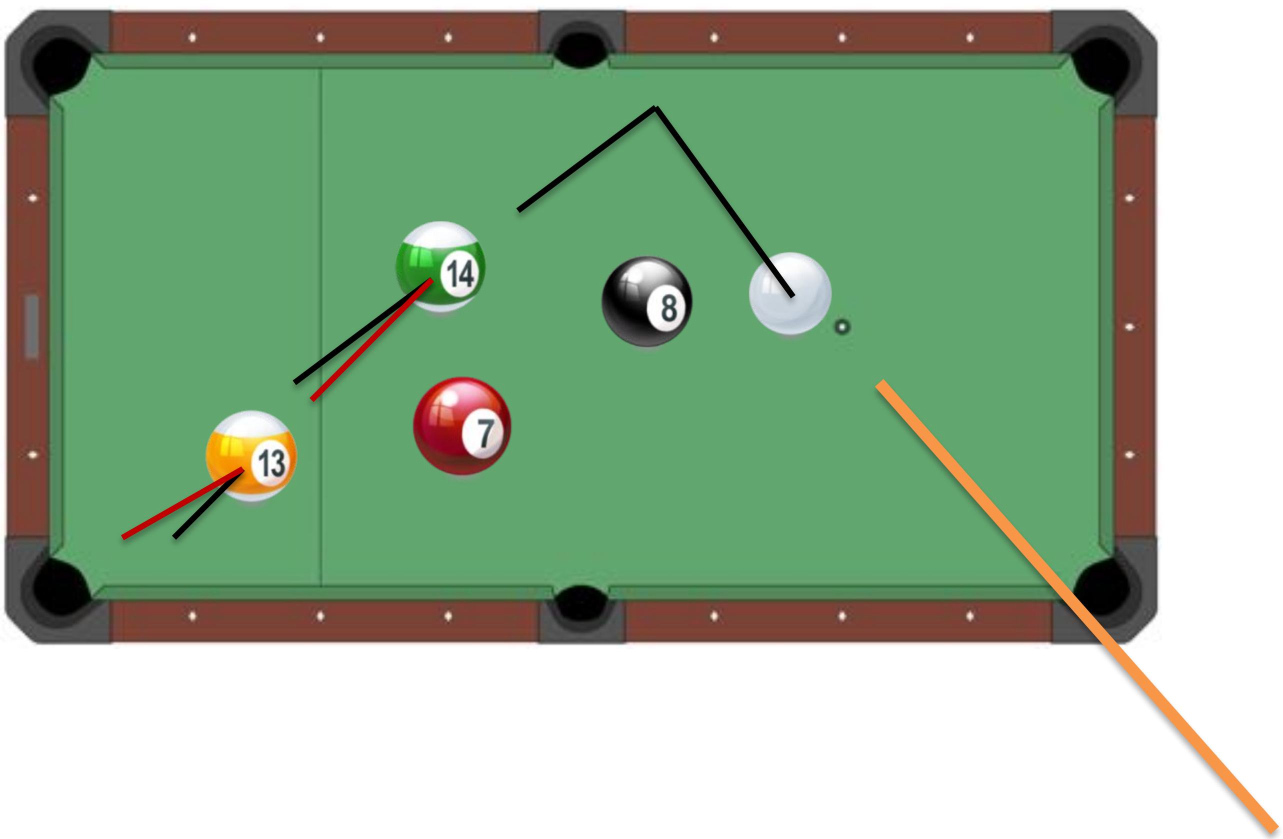
Slide from (Stoyanov & Eisner, 2012)

Error Back-Propagation



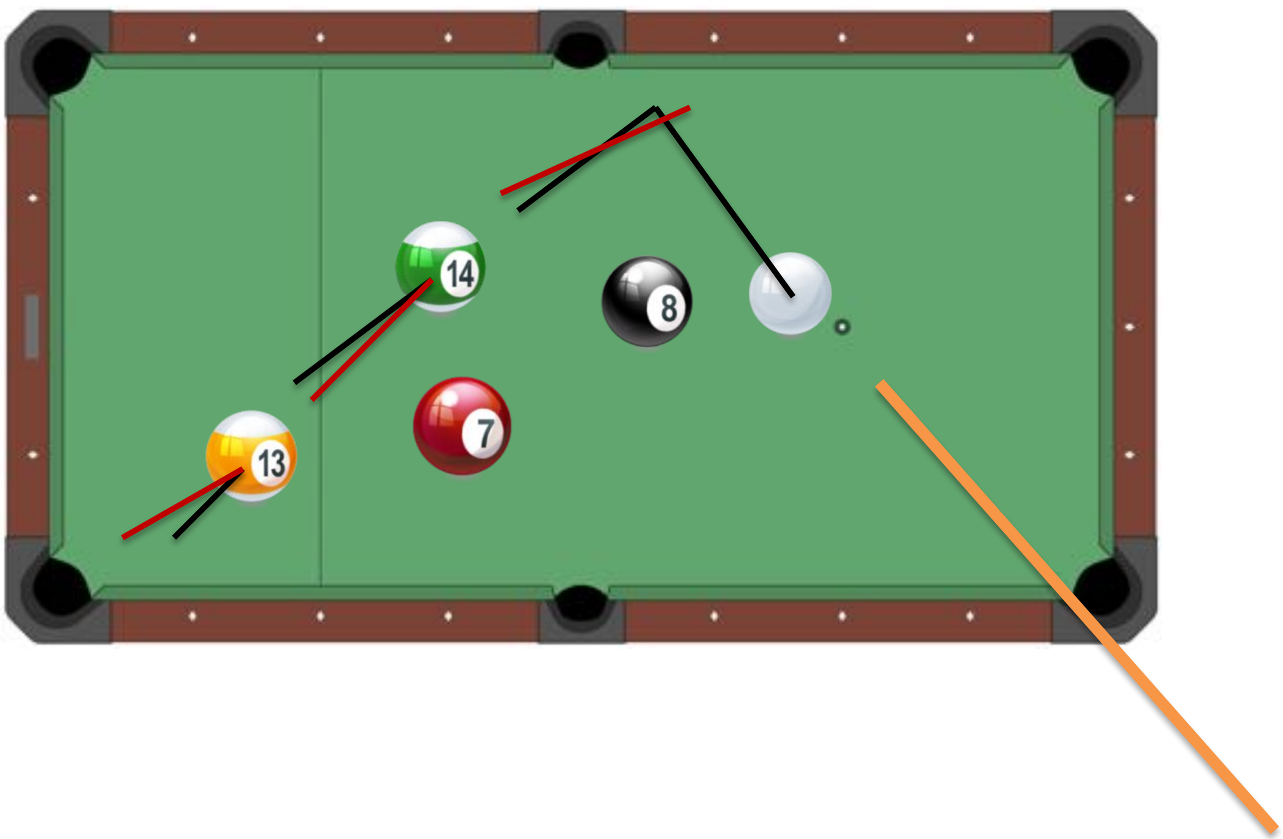
Slide from (Stoyanov & Eisner, 2012)

Error Back-Propagation



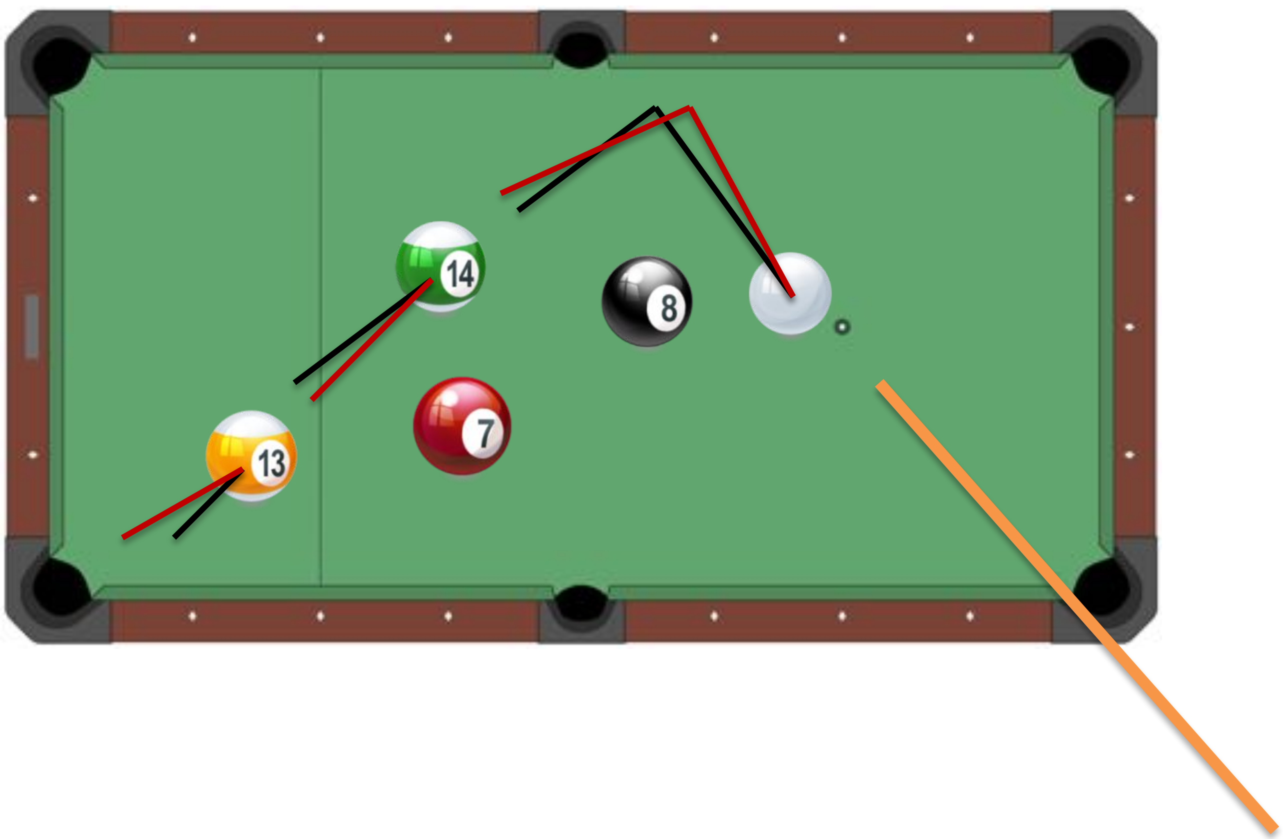
Slide from (Stoyanov & Eisner, 2012)

Error Back-Propagation



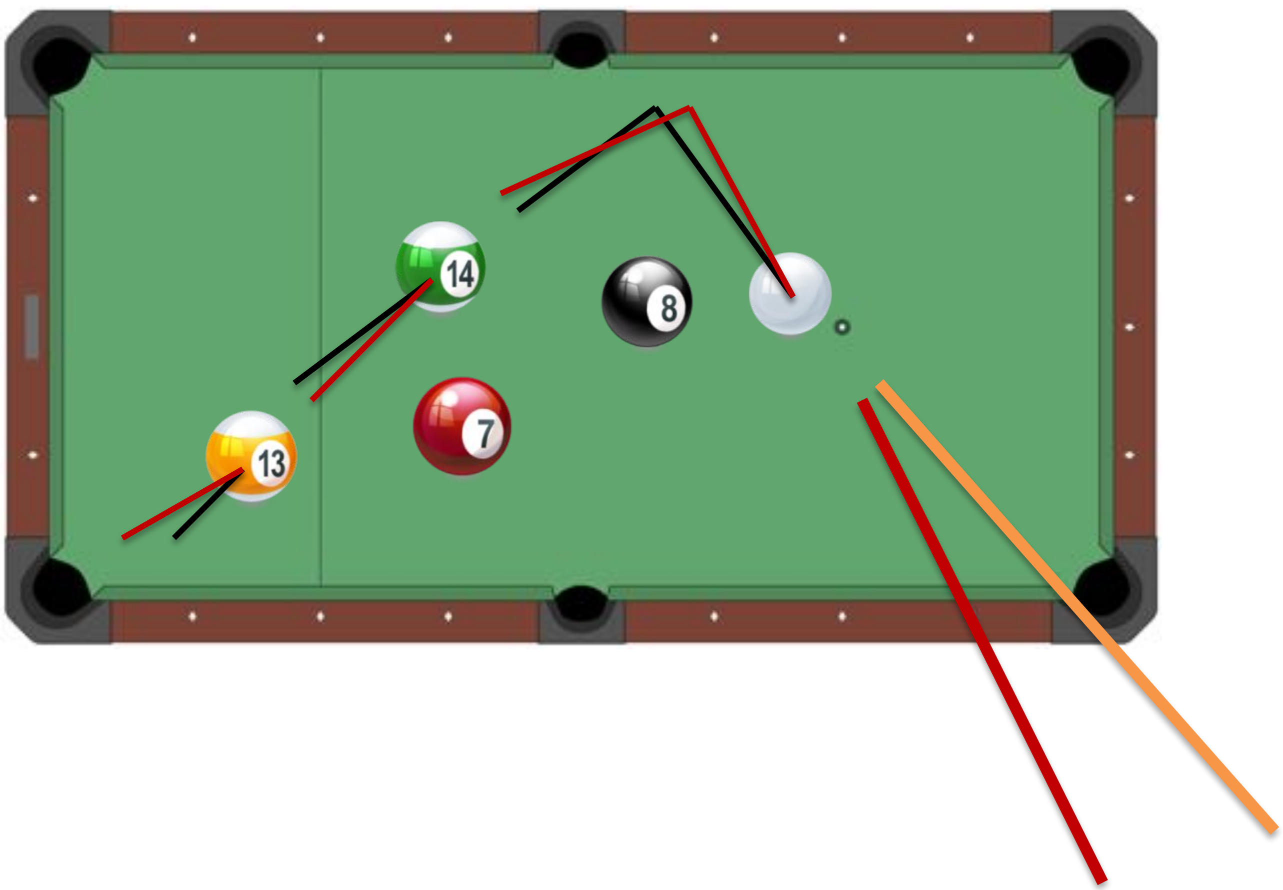
Slide from (Stoyanov & Eisner, 2012)

Error Back-Propagation



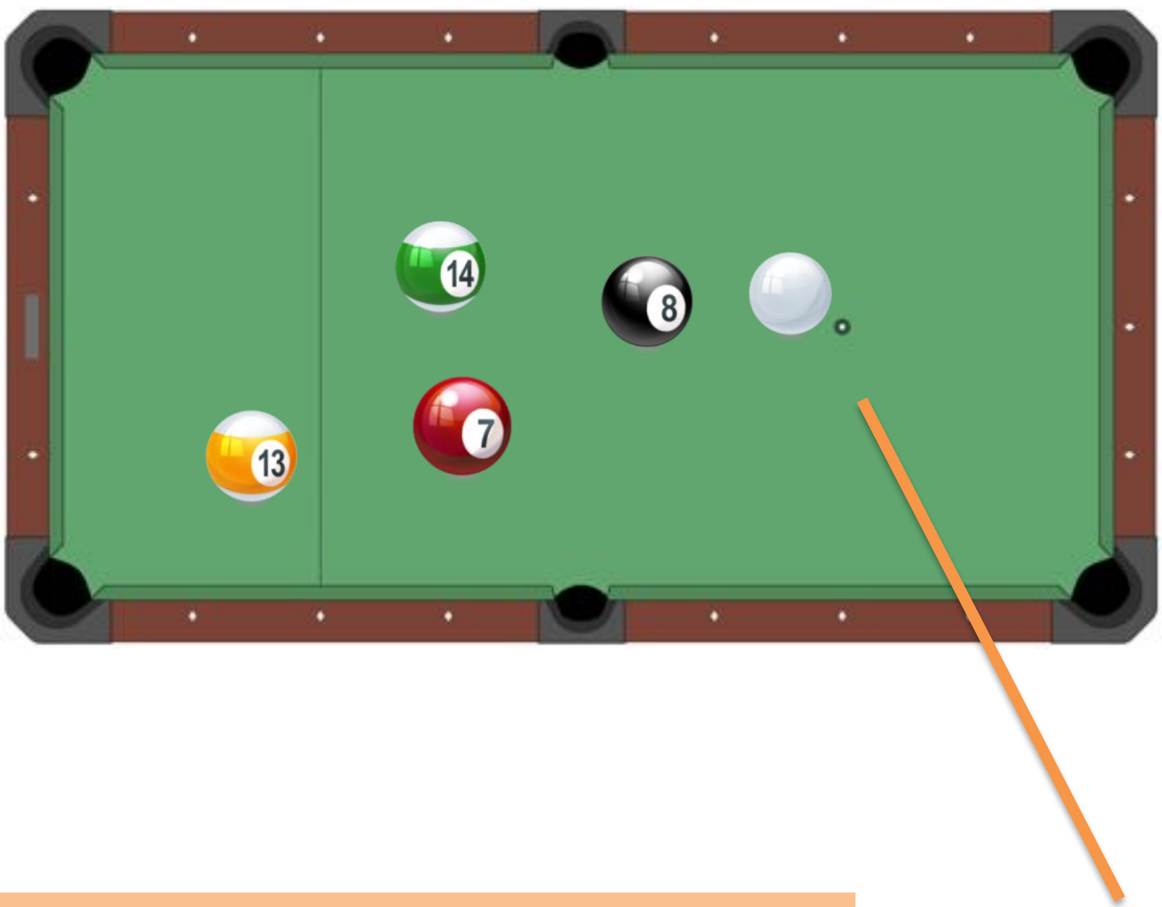
Slide from (Stoyanov & Eisner, 2012)

Error Back-Propagation



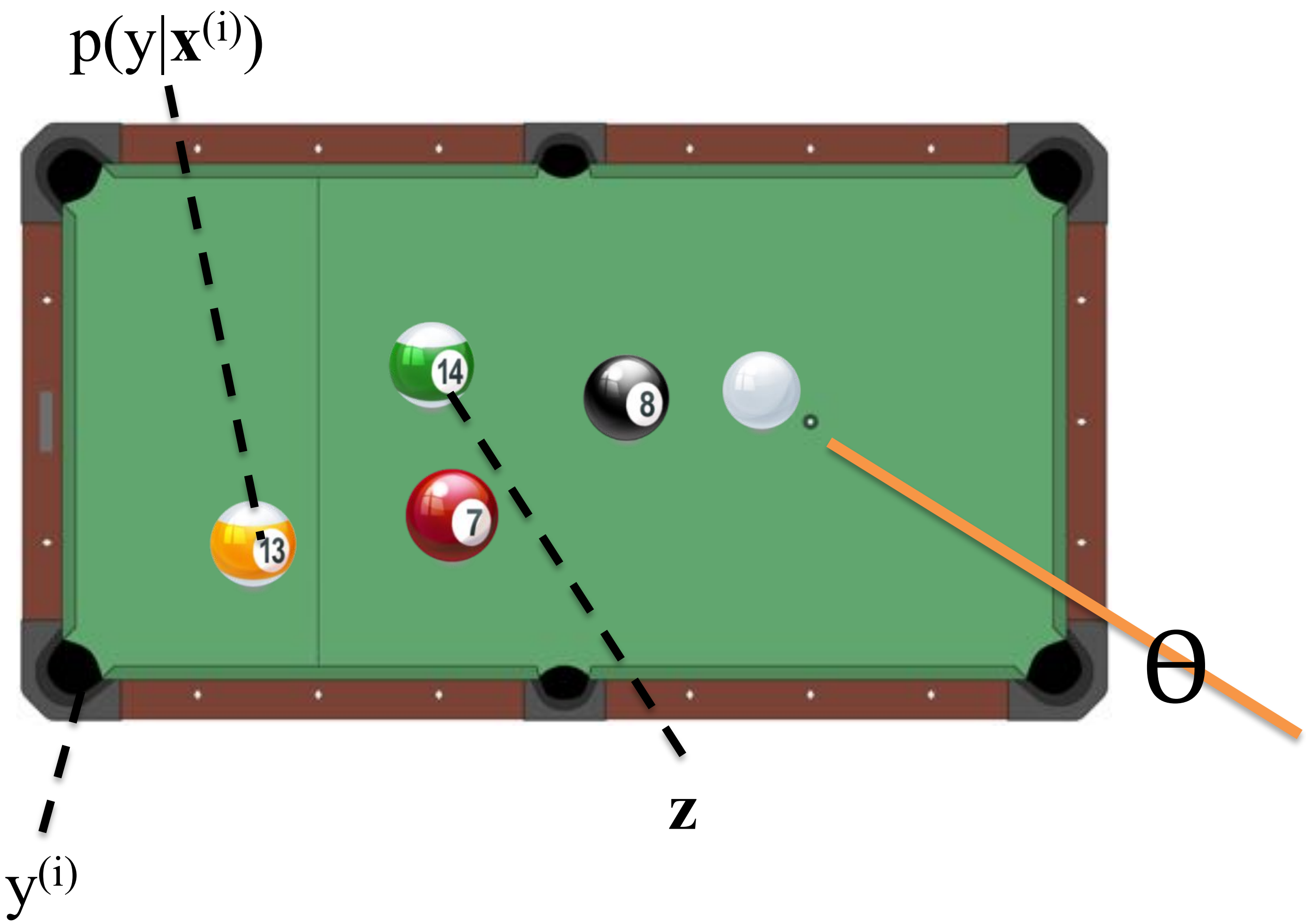
Slide from (Stoyanov & Eisner, 2012)

Error Back-Propagation



Slide from (Stoyanov & Eisner, 2012)

Error Back-Propagation



Slide from (Stoyanov & Eisner, 2012)

Algorithm

BACKPROPAGATION

Backpropagation

Chalkboard

- 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? **Round your answer to the nearest integer**.

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

Backpropagation

Automatic Differentiation – Reverse Mode (aka. Backpropagation)

<u>Forward Computation</u>

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 $v_1, \dots, v_N$
   a. Compute $u_i = g_i(v_1, \dots, v_N)$
   b. Store the result at the node

<u>Backward Computation</u>

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