



10-301/10-601 Introduction to Machine Learning

Machine Learning Department
School of Computer Science
Carnegie Mellon University

Backpropagation

Hoda Heidari, Henry Chai, Matt Gormley

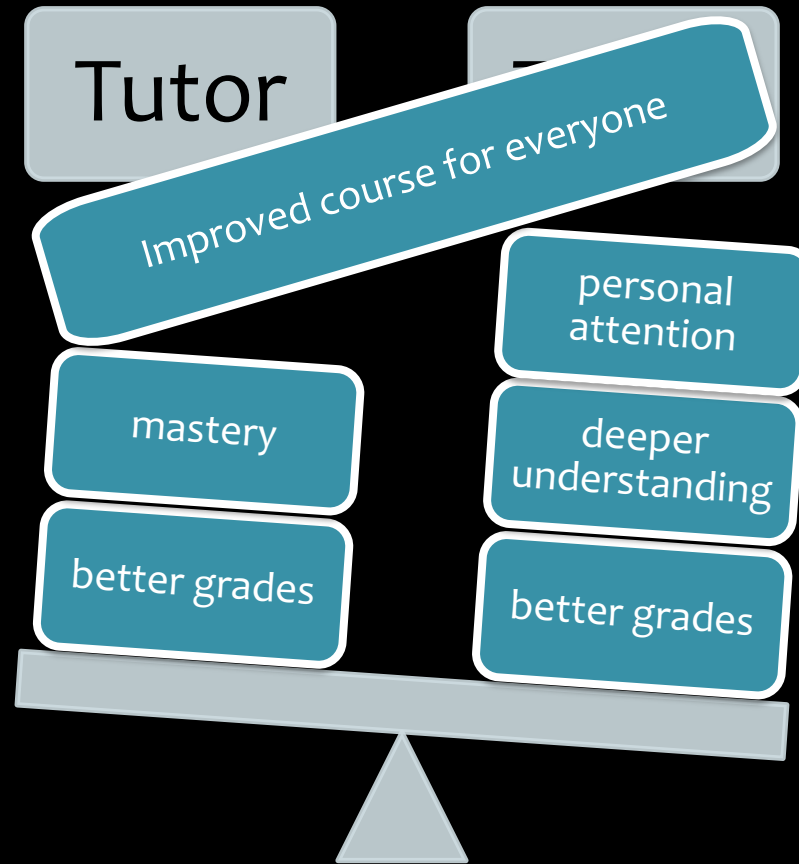
Lecture 13

Feb. 28, 2024

Reminders

- **Homework 4: Logistic Regression**
 - Out: Fri, Sep 29
 - Due: Mon, Oct 9 at 11:59pm
- **Homework 5: Neural Networks**
 - Out: Mon, Oct 9
 - Due: Fri, Oct 27 at 11:59pm
- **Exam viewings**

Peer Tutoring



Algorithm

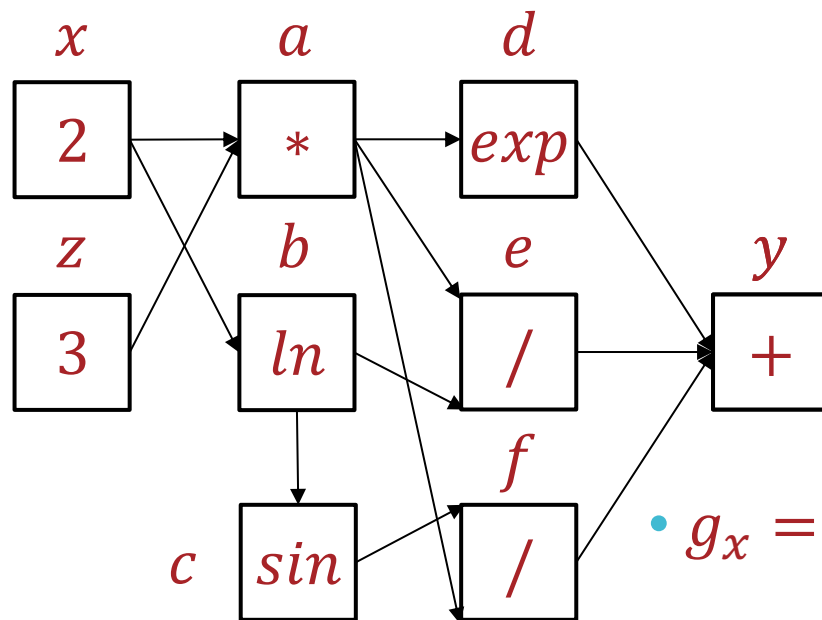
BACKPROPAGATION FOR A SIMPLE COMPUTATION GRAPH

- Given

$$y = f(x, z) = e^{xz} + \frac{xz}{\ln(x)} + \frac{\sin(\ln(x))}{xz}$$

what are $\frac{\partial y}{\partial x}$ and $\frac{\partial y}{\partial z}$ at $x = 2, z = 3$?

- Then compute partial derivatives, starting from y and working back



- $g_y = \frac{\partial y}{\partial y} = 1$

- $g_d = g_e = g_f = 1$

- $g_c = \frac{\partial y}{\partial c} = \frac{\partial y}{\partial f} \frac{\partial f}{\partial c} = g_f \left(\frac{1}{a} \right)$

- $g_b = \frac{\partial y}{\partial b} = \frac{\partial y}{\partial e} \frac{\partial e}{\partial b} + \frac{\partial y}{\partial c} \frac{\partial c}{\partial b}$
 $= g_e \left(-\frac{a}{b^2} \right) + g_c (\cos(b))$

- $g_a = \frac{\partial y}{\partial a} = \frac{\partial y}{\partial f} \frac{\partial f}{\partial a} + \frac{\partial y}{\partial e} \frac{\partial e}{\partial a} + \frac{\partial y}{\partial d} \frac{\partial d}{\partial a}$
 $= g_f \left(\frac{-c}{a^2} \right) + g_e \left(\frac{1}{b} \right) + g_d (e^a)$

- $g_x = \frac{\partial y}{\partial x} = \frac{\partial y}{\partial b} \frac{\partial b}{\partial x} + \frac{\partial y}{\partial a} \frac{\partial a}{\partial x} = g_b \left(\frac{1}{x} \right) + g_a (z)$

- $g_z = \frac{\partial y}{\partial z} = \frac{\partial y}{\partial a} \frac{\partial a}{\partial z} = g_a (x)$

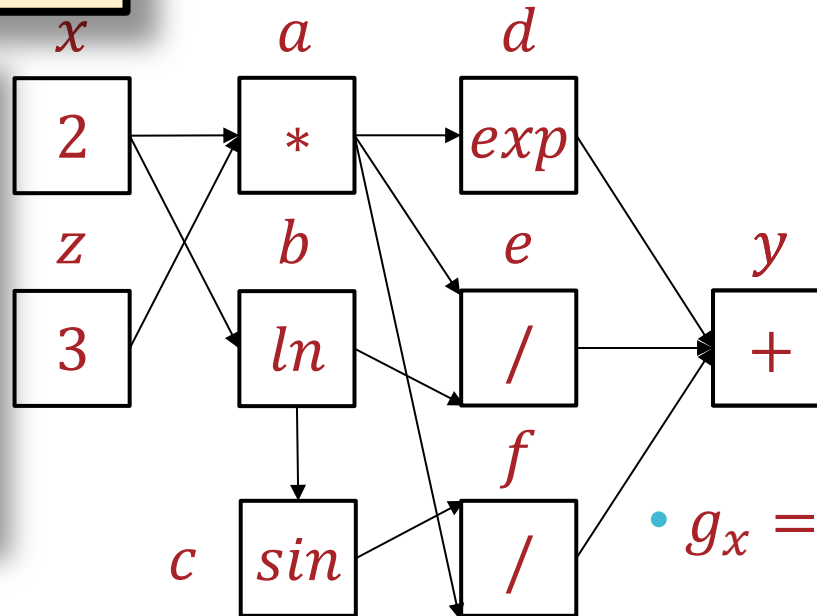
Approach 3: Automatic Differentiation (reverse mode)

Updates for Backpropagation:

$$g_x = \frac{\partial y}{\partial x} = \sum_{k=1}^K \frac{\partial y}{\partial u_k} \frac{\partial u_k}{\partial x}$$

$$= \sum_{k=1}^K g_{u_k} \frac{\partial u_k}{\partial x}$$

Backprop is efficient b/c of reuse in the forward pass and the backward pass.



$$y = f(x, z) = e^{xz} + \frac{xz}{\ln(x)} + \frac{\sin(\ln(x))}{xz}$$

What are $\frac{\partial y}{\partial x}$ and $\frac{\partial y}{\partial z}$ at $x = 2, z = 3$?

When compute partial derivatives, starting from y and working back

- $g_y = \frac{\partial y}{\partial y} = 1$

- $g_d = g_e = g_f = 1$

- $g_c = \frac{\partial y}{\partial c} = \frac{\partial y}{\partial f} \frac{\partial f}{\partial c} = g_f \left(\frac{1}{a} \right)$

- $g_b = \frac{\partial y}{\partial b} = \frac{\partial y}{\partial e} \frac{\partial e}{\partial b} + \frac{\partial y}{\partial c} \frac{\partial c}{\partial b}$
 $= g_e \left(-\frac{a}{b^2} \right) + g_c (\cos(b))$

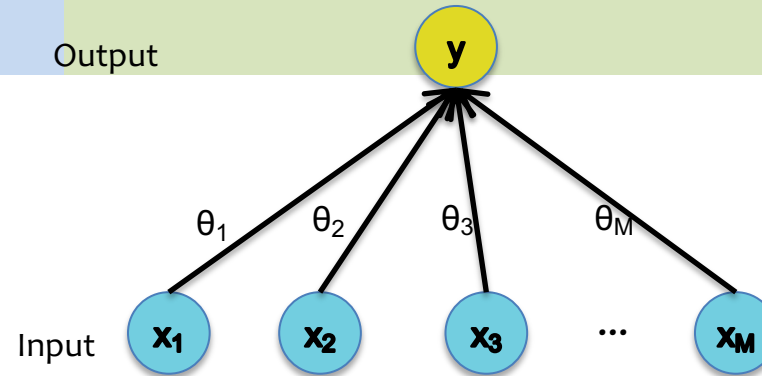
- $g_a = \frac{\partial y}{\partial a} = \frac{\partial y}{\partial f} \frac{\partial f}{\partial a} + \frac{\partial y}{\partial e} \frac{\partial e}{\partial a} + \frac{\partial y}{\partial d} \frac{\partial d}{\partial a}$
 $= g_f \left(\frac{-c}{a^2} \right) + g_e \left(\frac{1}{b} \right) + g_d (e^a)$

- $g_x = \frac{\partial y}{\partial x} = \frac{\partial y}{\partial b} \frac{\partial b}{\partial x} + \frac{\partial y}{\partial a} \frac{\partial a}{\partial x} = g_b \left(\frac{1}{x} \right) + g_a (z)$

- $g_z = \frac{\partial y}{\partial z} = \frac{\partial y}{\partial a} \frac{\partial a}{\partial z} = g_a (x)$

Algorithm

BACKPROPAGATION FOR BINARY LOGISTIC REGRESSION

**Case 1:
Logistic
Regression**

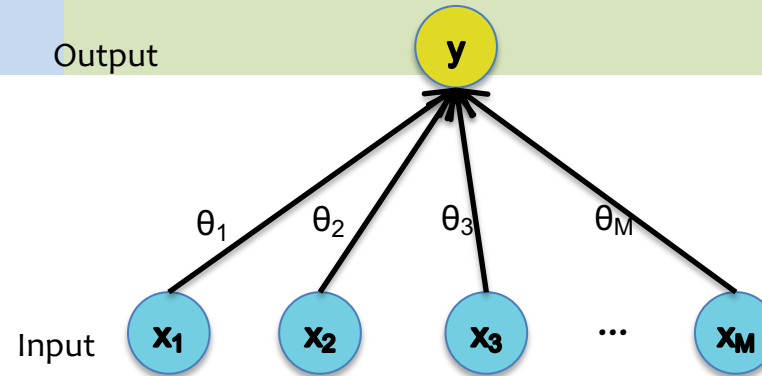
Forward

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

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

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

**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

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

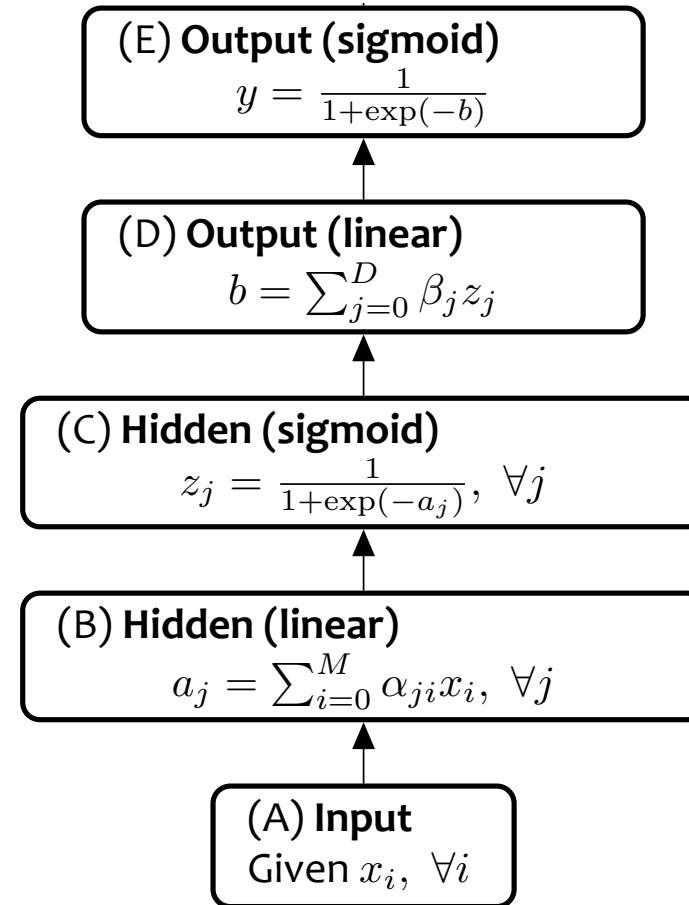
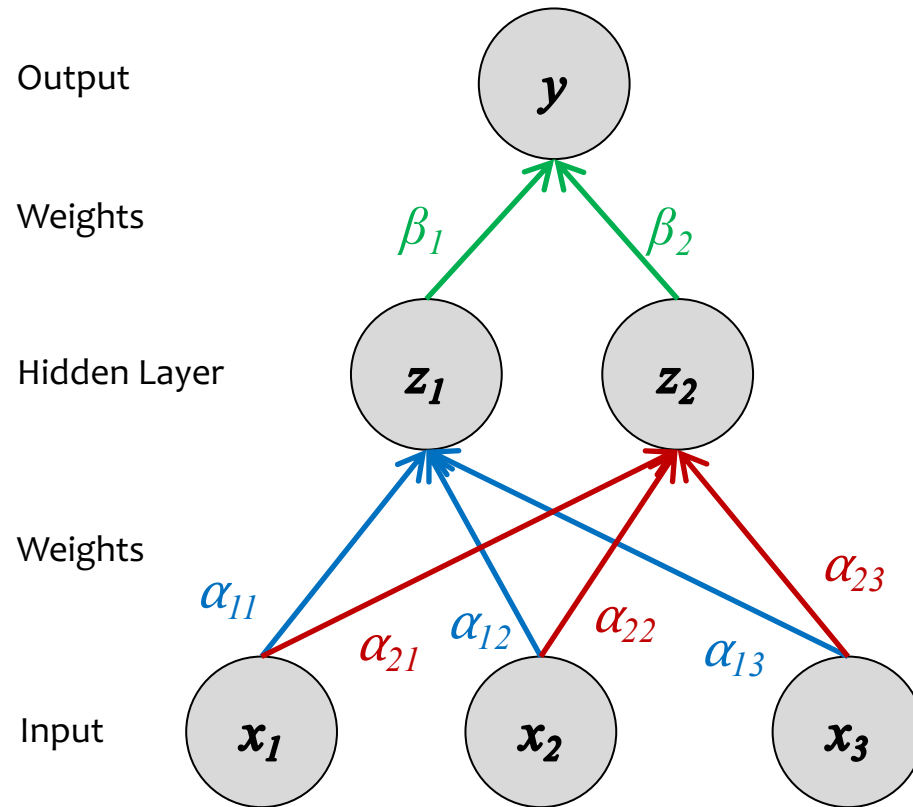
$$g_a = g_y \frac{\partial y}{\partial a}, \quad \frac{\partial y}{\partial a} = \frac{\exp(-a)}{(\exp(-a) + 1)^2}$$

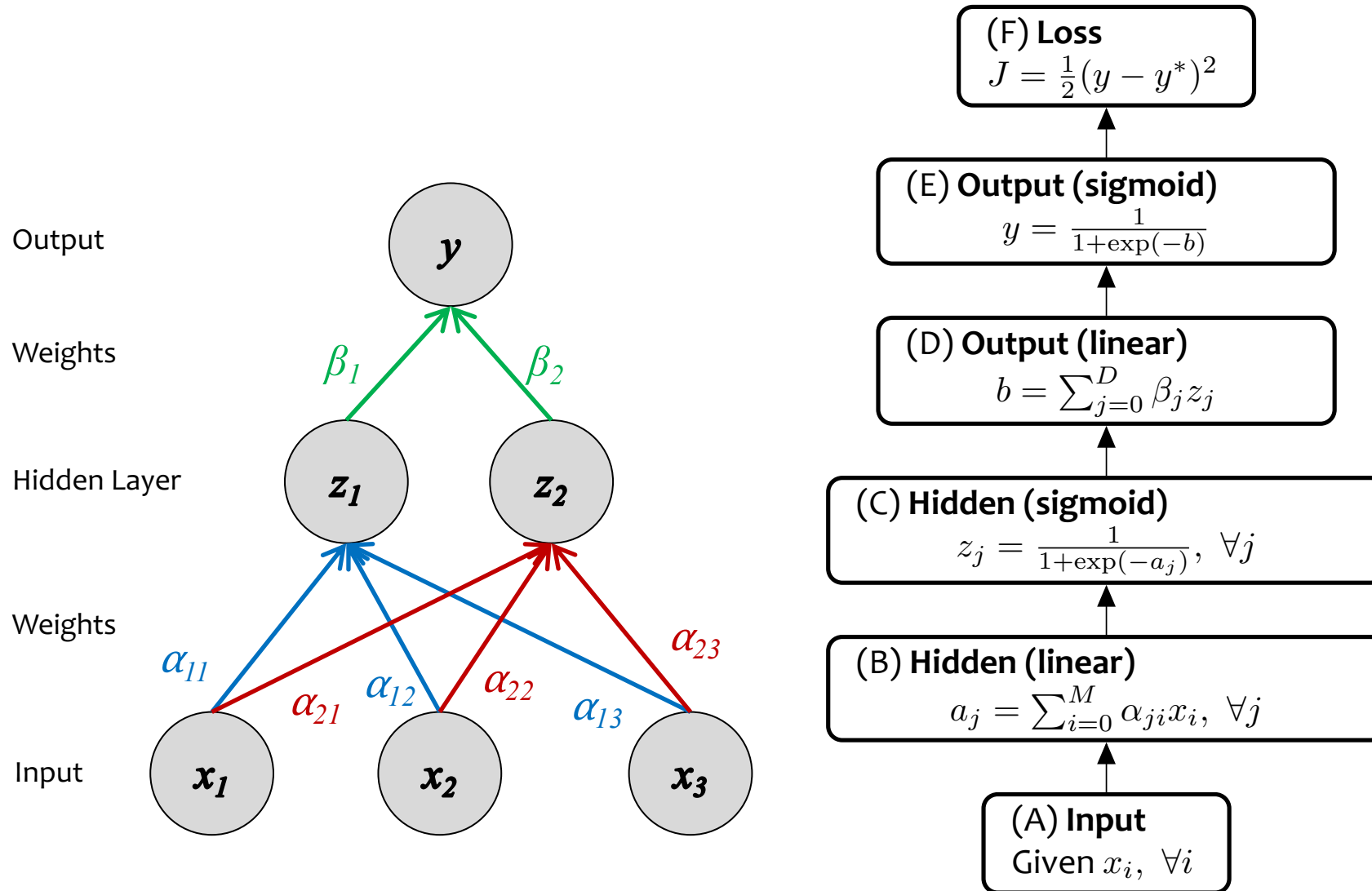
$$g_{\theta_j} = g_a \frac{\partial a}{\partial \theta_j}, \quad \frac{\partial a}{\partial \theta_j} = x_j$$

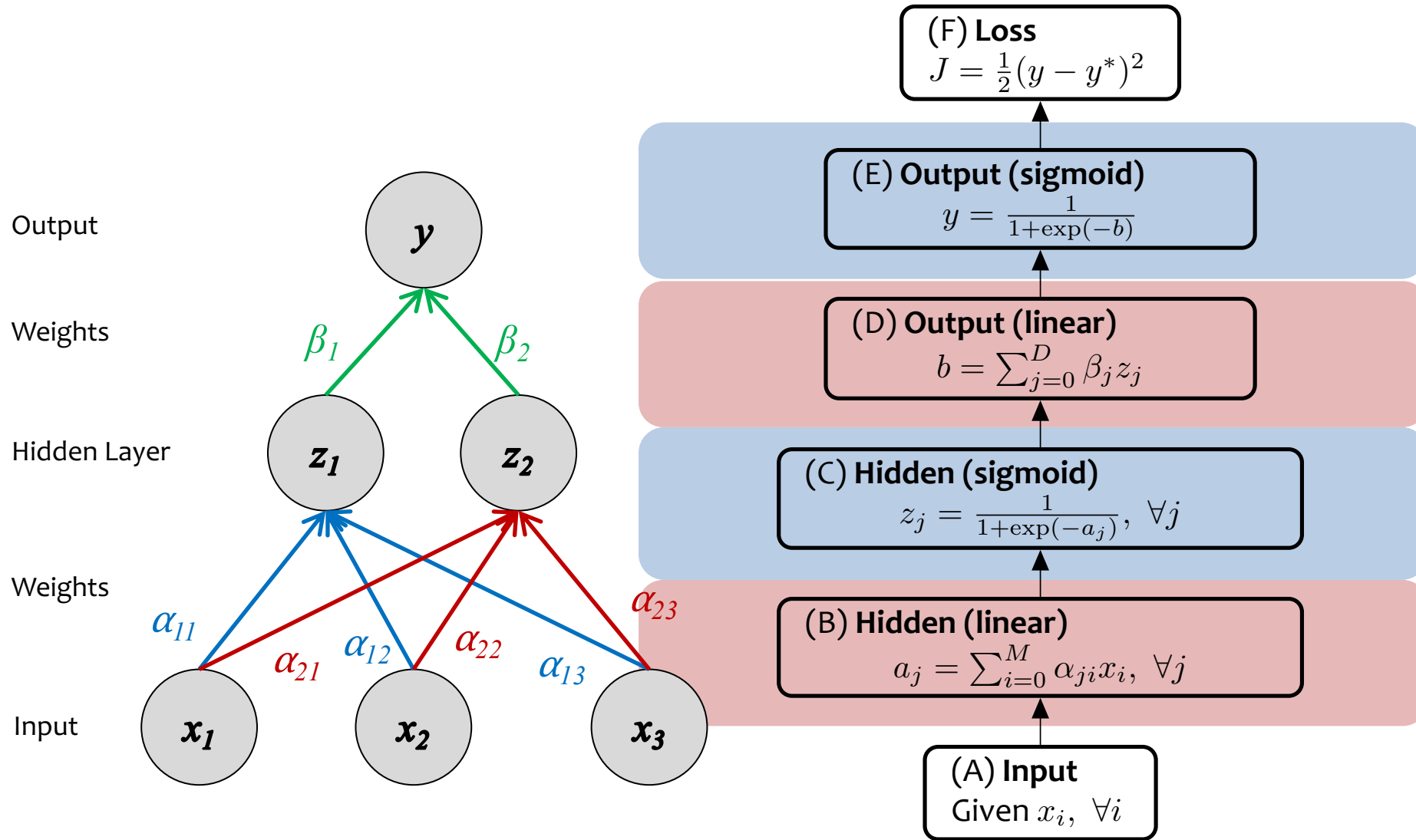
$$g_{x_j} = g_a \frac{\partial a}{\partial x_j}, \quad \frac{\partial a}{\partial x_j} = \theta_j$$

A 1-Hidden Layer Neural Network

TRAINING / FORWARD COMPUTATION / BACKWARD COMPUTATION





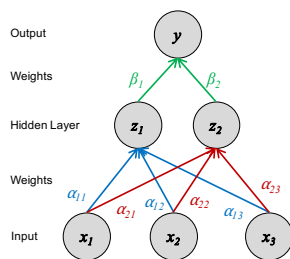


Example: 1-Hidden Layer Neural Network

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  $\alpha, \beta$ 
```

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

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

$$g_b = g_y \frac{\partial y}{\partial b}, \quad \frac{\partial y}{\partial b} = \frac{\exp(-b)}{(\exp(-b) + 1)^2}$$

$$g_{\beta_j} = g_b \frac{\partial b}{\partial \beta_j}, \quad \frac{\partial b}{\partial \beta_j} = z_j$$

$$g_{z_j} = g_b \frac{\partial b}{\partial z_j}, \quad \frac{\partial b}{\partial z_j} = \beta_j$$

$$g_{a_j} = g_{z_j} \frac{\partial z_j}{\partial a_j}, \quad \frac{\partial z_j}{\partial a_j} = \frac{\exp(-a_j)}{(\exp(-a_j) + 1)^2}$$

$$g_{\alpha_{ji}} = g_{a_j} \frac{\partial a_j}{\partial \alpha_{ji}}, \quad \frac{\partial a_j}{\partial \alpha_{ji}} = x_i$$

$$g_{x_i} = \sum_{j=0}^D g_{a_j} \frac{\partial a_j}{\partial x_i}, \quad \frac{\partial a_j}{\partial x_i} = \alpha_{ji}$$

Case 2:	Forward	Backward
Loss	$J = y^* \log y + (1 - y^*) \log(1 - y)$	$g_y = \frac{y^*}{y} + \frac{(1 - y^*)}{y - 1}$
Sigmoid	$y = \frac{1}{1 + \exp(-b)}$	$g_b = g_y \frac{\partial y}{\partial b}, \frac{\partial y}{\partial b} = \frac{\exp(-b)}{(\exp(-b) + 1)^2}$
Linear	$b = \sum_{j=0}^D \beta_j z_j$	$g_{\beta_j} = g_b \frac{\partial b}{\partial \beta_j}, \frac{\partial b}{\partial \beta_j} = z_j$ $g_{z_j} = g_b \frac{\partial b}{\partial z_j}, \frac{\partial b}{\partial z_j} = \beta_j$
Sigmoid	$z_j = \frac{1}{1 + \exp(-a_j)}$	$g_{a_j} = g_{z_j} \frac{\partial z_j}{\partial a_j}, \frac{\partial z_j}{\partial a_j} = \frac{\exp(-a_j)}{(\exp(-a_j) + 1)^2}$
Linear	$a_j = \sum_{i=0}^M \alpha_{ji} x_i$	$g_{\alpha_{ji}} = g_{a_j} \frac{\partial a_j}{\partial \alpha_{ji}}, \frac{\partial a_j}{\partial \alpha_{ji}} = x_i$ $g_{x_i} = \sum_{j=0}^D g_{a_j} \frac{\partial a_j}{\partial x_i}, \frac{\partial a_j}{\partial x_i} = \alpha_{ji}$

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

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.

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

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

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

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

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

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

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

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

Case 2:	Forward	Backward
Loss	$J = y^* \log y + (1 - y^*) \log(1 - y)$	$g_y = \frac{y^*}{y} + \frac{(1 - y^*)}{y - 1}$
Sigmoid	$y = \frac{1}{1 + \exp(-b)}$	$g_b = g_y \frac{\partial y}{\partial b}, \frac{\partial y}{\partial b} = \frac{\exp(-b)}{(\exp(-b) + 1)^2}$
Linear	$b = \sum_{j=0}^D \beta_j z_j$	$g_{\beta_j} = g_b \frac{\partial b}{\partial \beta_j}, \frac{\partial b}{\partial \beta_j} = z_j$
Sigmoid	$z_j = \frac{1}{1 + \exp(-a_j)}$	$g_{a_j} = g_{z_j} \frac{\partial z_j}{\partial a_j}, \frac{\partial z_j}{\partial a_j} = \frac{\exp(-a_j)}{(\exp(-a_j) + 1)^2}$
Linear	$a_j = \sum_{i=0}^M \alpha_{ji} x_i$	$g_{\alpha_{ji}} = g_{a_j} \frac{\partial a_j}{\partial \alpha_{ji}}, \frac{\partial a_j}{\partial \alpha_{ji}} = x_i$
		$g_{x_i} = \sum_{j=0}^D g_{a_j} \frac{\partial a_j}{\partial x_i}, \frac{\partial a_j}{\partial x_i} = \alpha_{ji}$

Case 2:	Forward	Backward
Loss	$J = y^* \log y + (1 - y^*) \log(1 - y)$	$g_y = \frac{y^*}{y} + \frac{(1 - y^*)}{y - 1}$
Sigmoid	$y = \frac{1}{1 + \exp(-b)}$	$g_b = g_y \frac{\partial y}{\partial b}, \frac{\partial y}{\partial b} = y(1 - y)$
Linear	$b = \sum_{j=0}^D \beta_j z_j$	$g_{\beta_j} = g_b \frac{\partial b}{\partial \beta_j}, \frac{\partial b}{\partial \beta_j} = z_j$
Sigmoid	$z_j = \frac{1}{1 + \exp(-a_j)}$	$g_{a_j} = g_{z_j} \frac{\partial z_j}{\partial a_j}, \frac{\partial z_j}{\partial a_j} = z_j(1 - z_j)$
Linear	$a_j = \sum_{i=0}^M \alpha_{ji} x_i$	$g_{\alpha_{ji}} = g_{a_j} \frac{\partial a_j}{\partial \alpha_{ji}}, \frac{\partial a_j}{\partial \alpha_{ji}} = x_i$
		$g_{x_i} = \sum_{j=0}^D g_{a_j} \frac{\partial a_j}{\partial x_i}, \frac{\partial a_j}{\partial x_i} = \alpha_{ji}$

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  $\alpha, \beta$ 
```

In-Class Poll

Question:

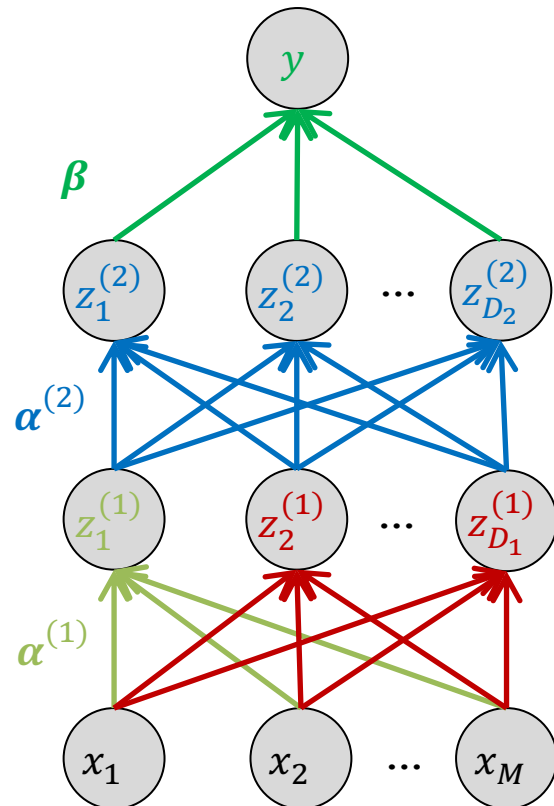
What questions do *you* have?

A 2-Hidden Layer Neural Network

TRAINING / FORWARD COMPUTATION / BACKWARD COMPUTATION

Recall: Our 2-Hidden Layer Neural Network

Question: How do we train this model?



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

$$\beta_0 \in \mathbb{R}$$

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

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

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

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

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

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

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

Example: Neural Net Training (2-Hidden Layers)

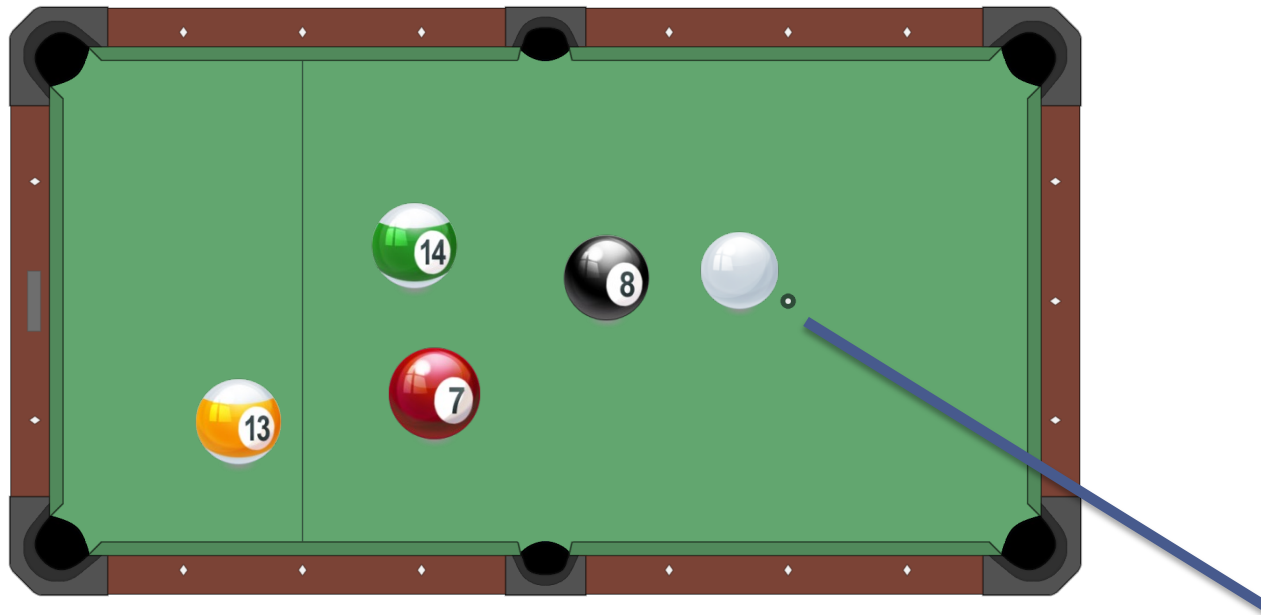
Example: Backpropagation (2-Hidden Layers)

Example: Backpropagation (2-Hidden Layers)

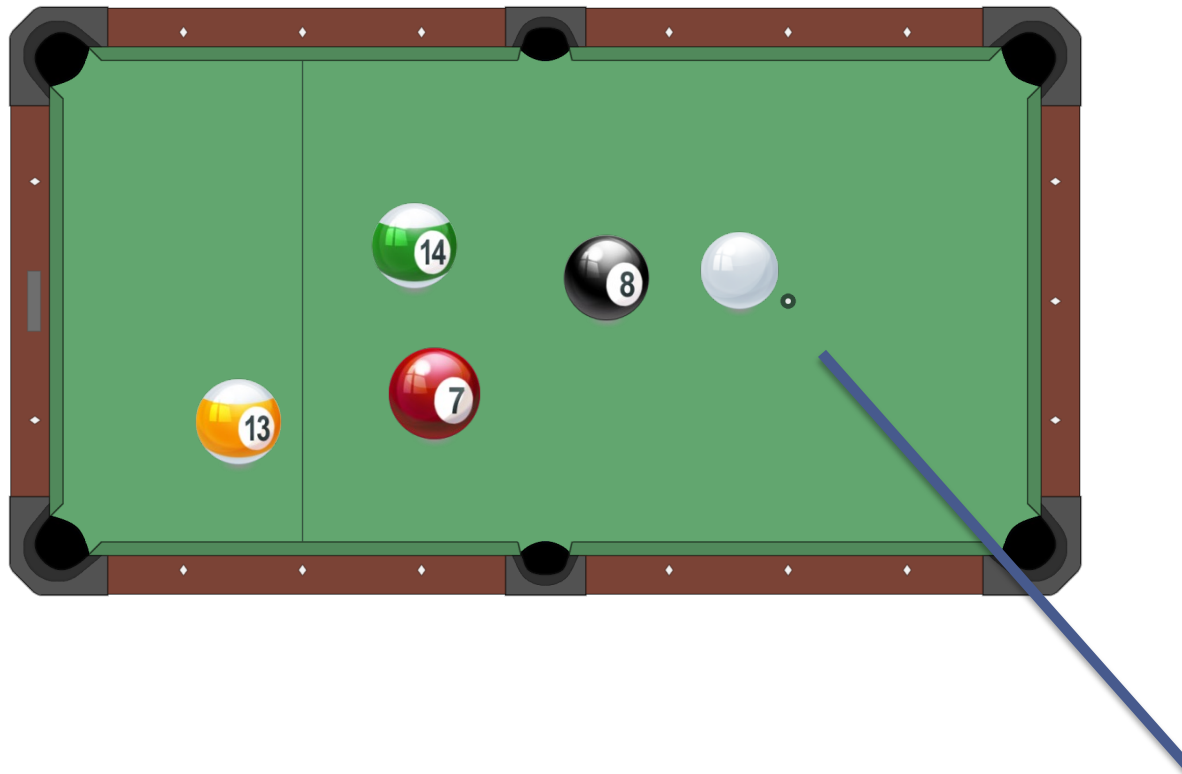
Intuitions

BACKPROPAGATION OF ERRORS

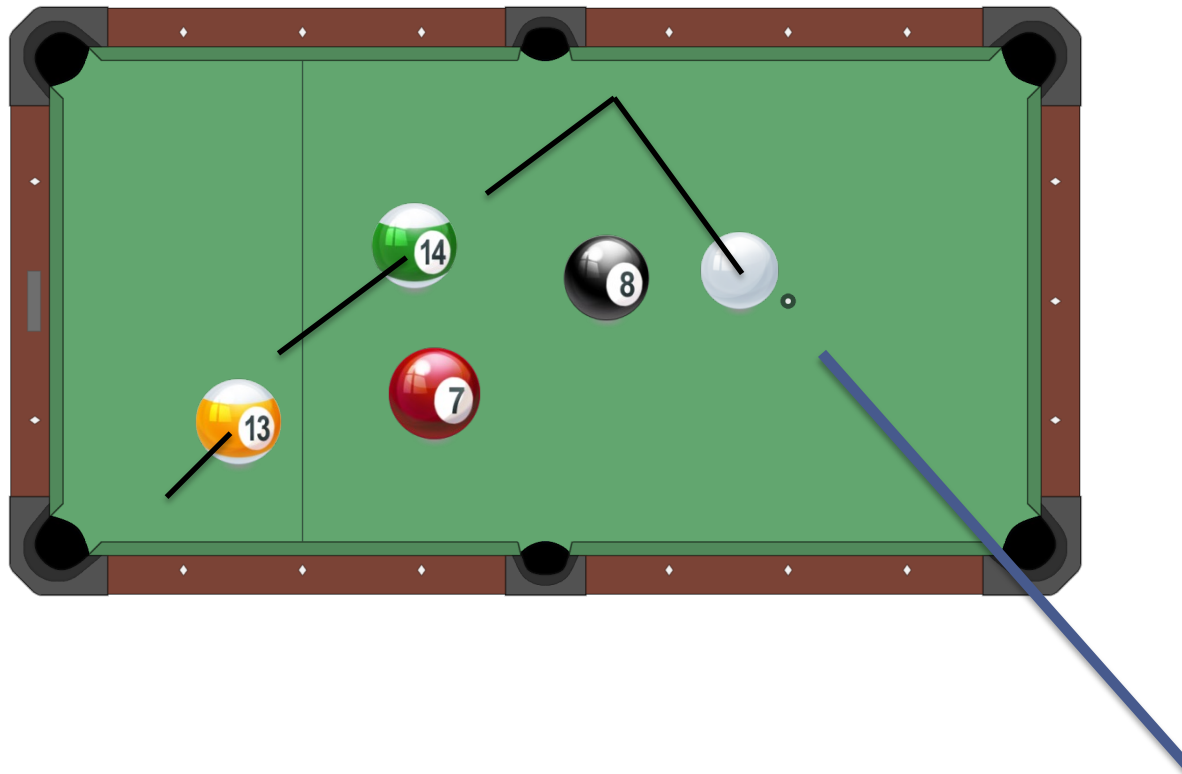
Error Back-Propagation



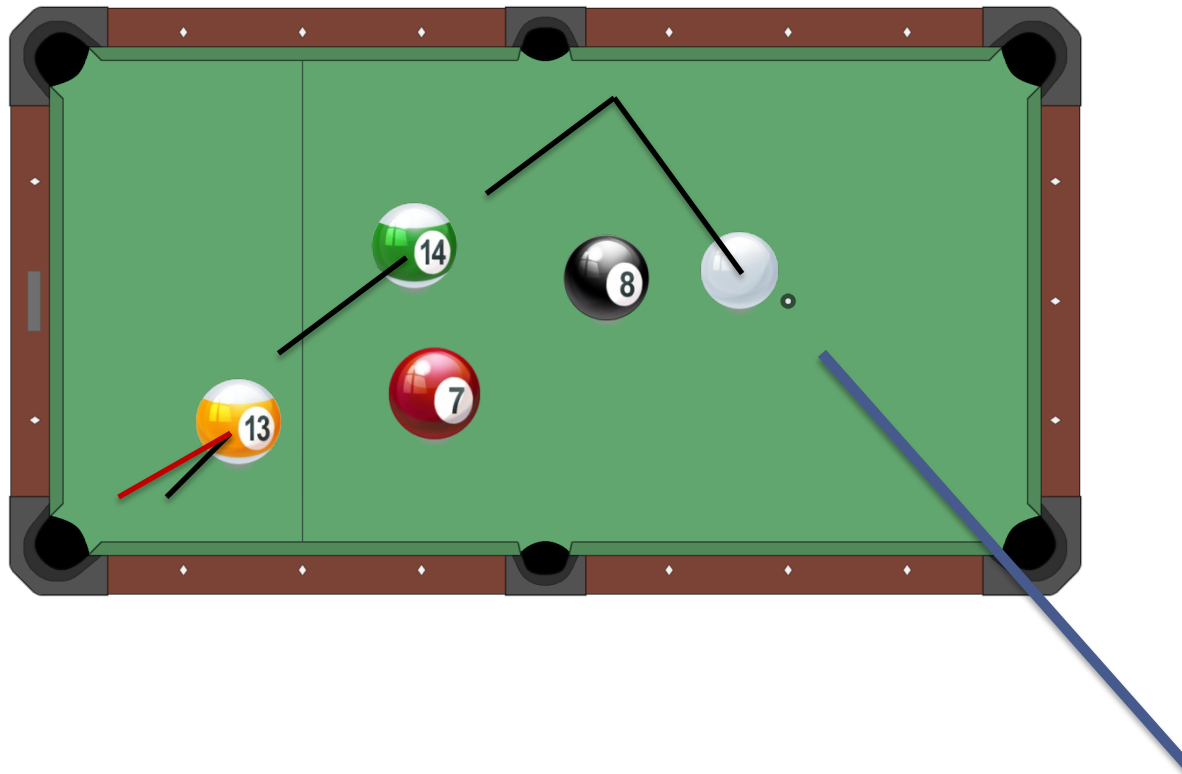
Error Back-Propagation



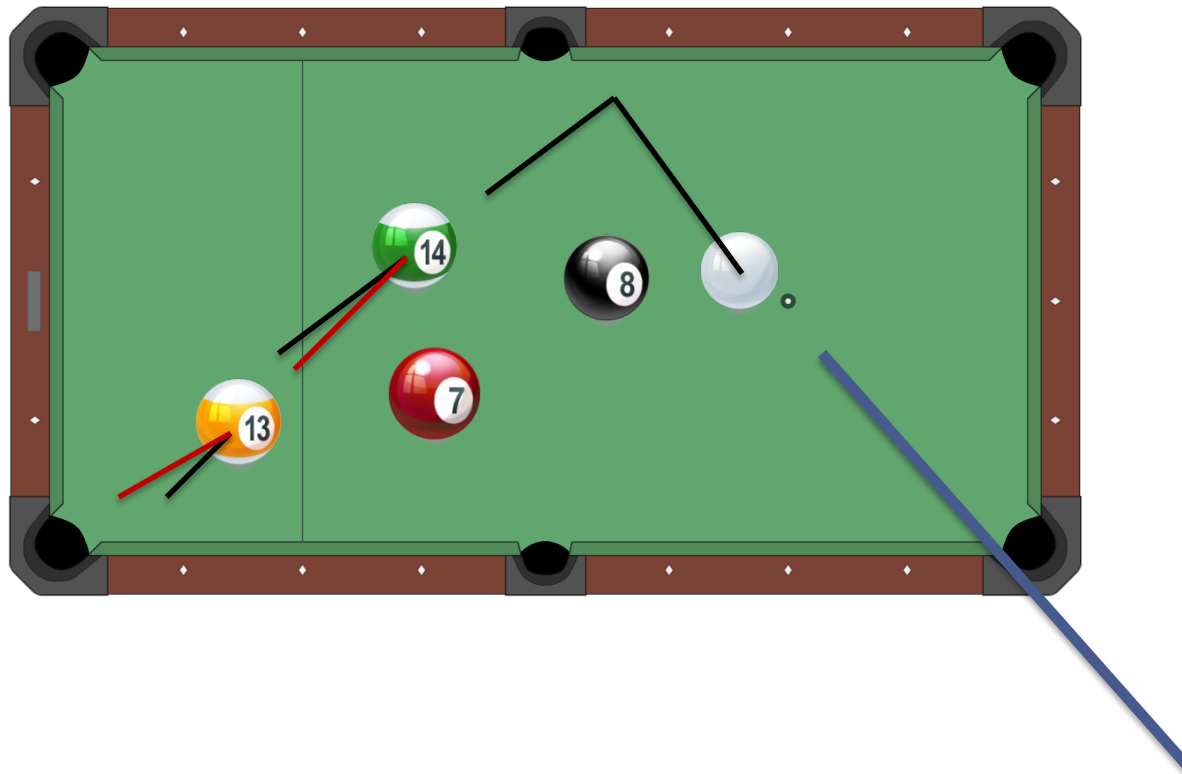
Error Back-Propagation



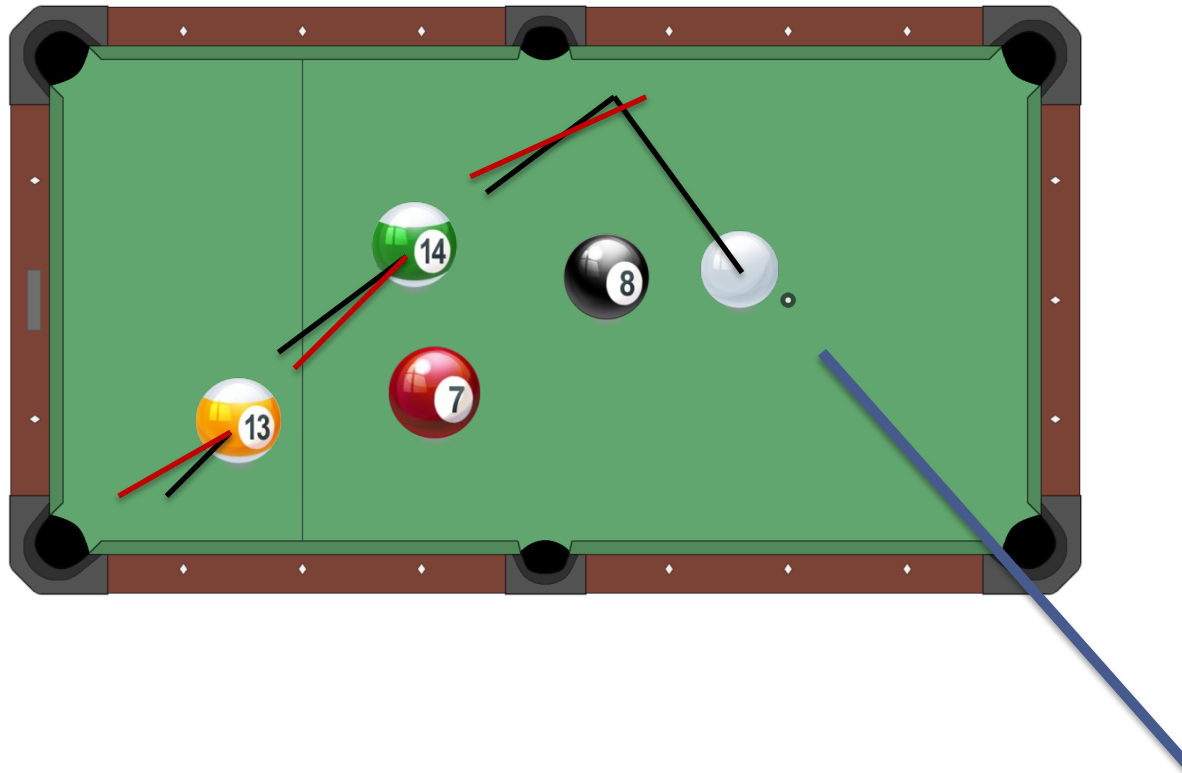
Error Back-Propagation



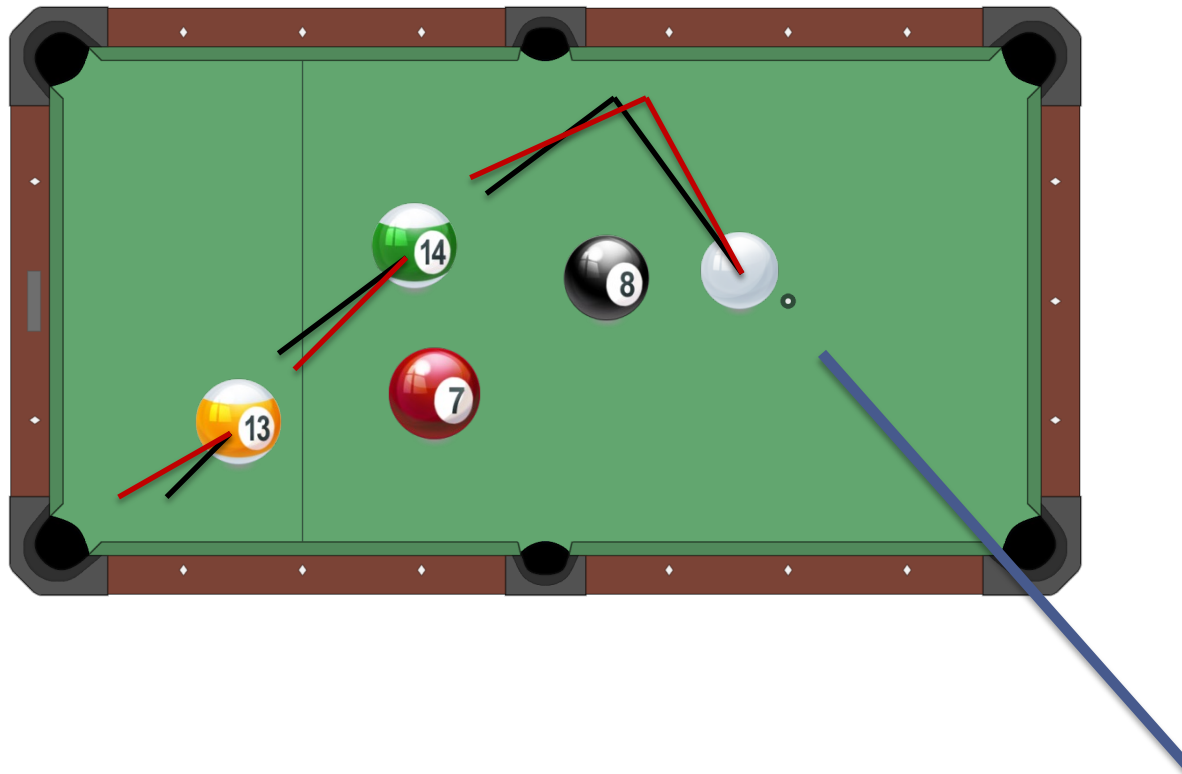
Error Back-Propagation



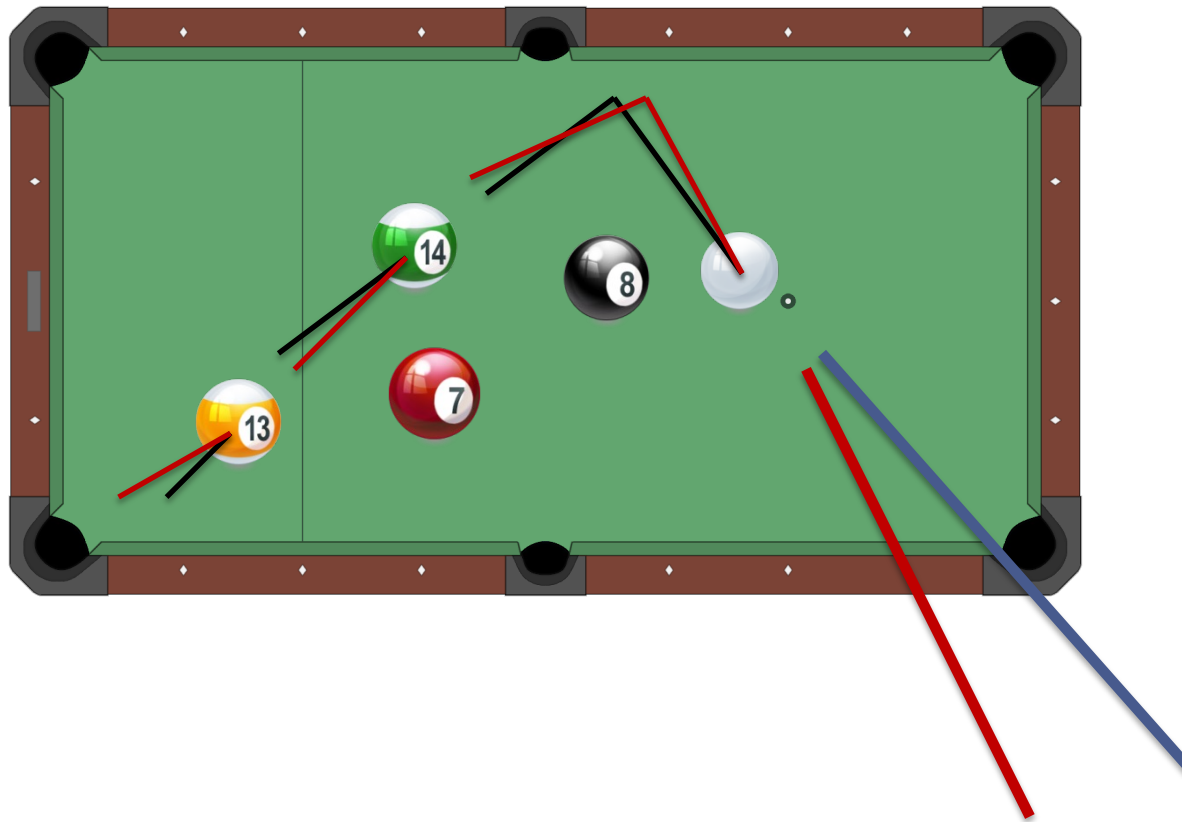
Error Back-Propagation



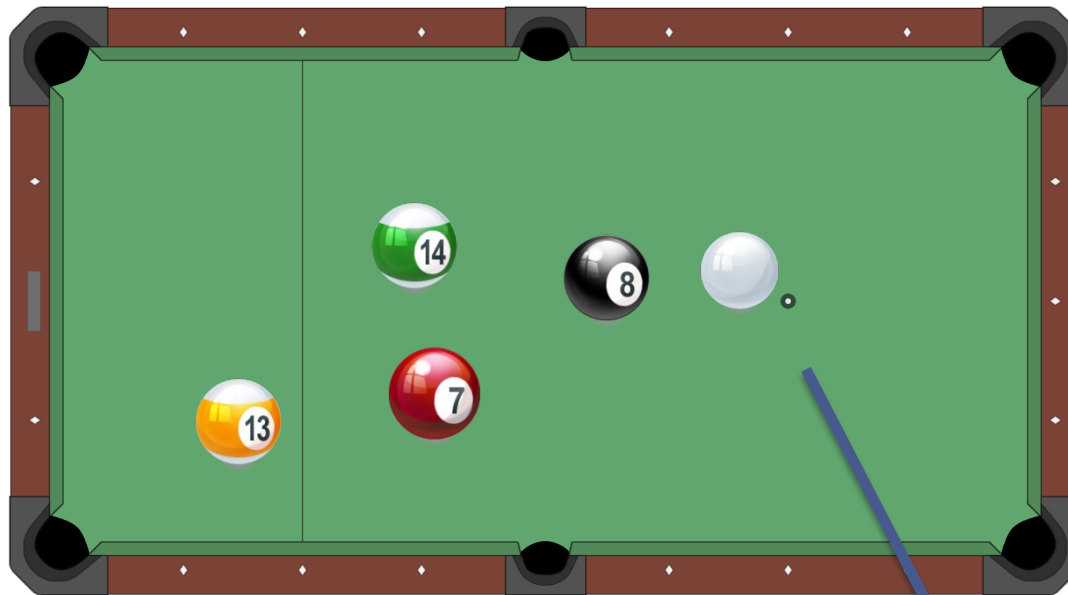
Error Back-Propagation



Error Back-Propagation

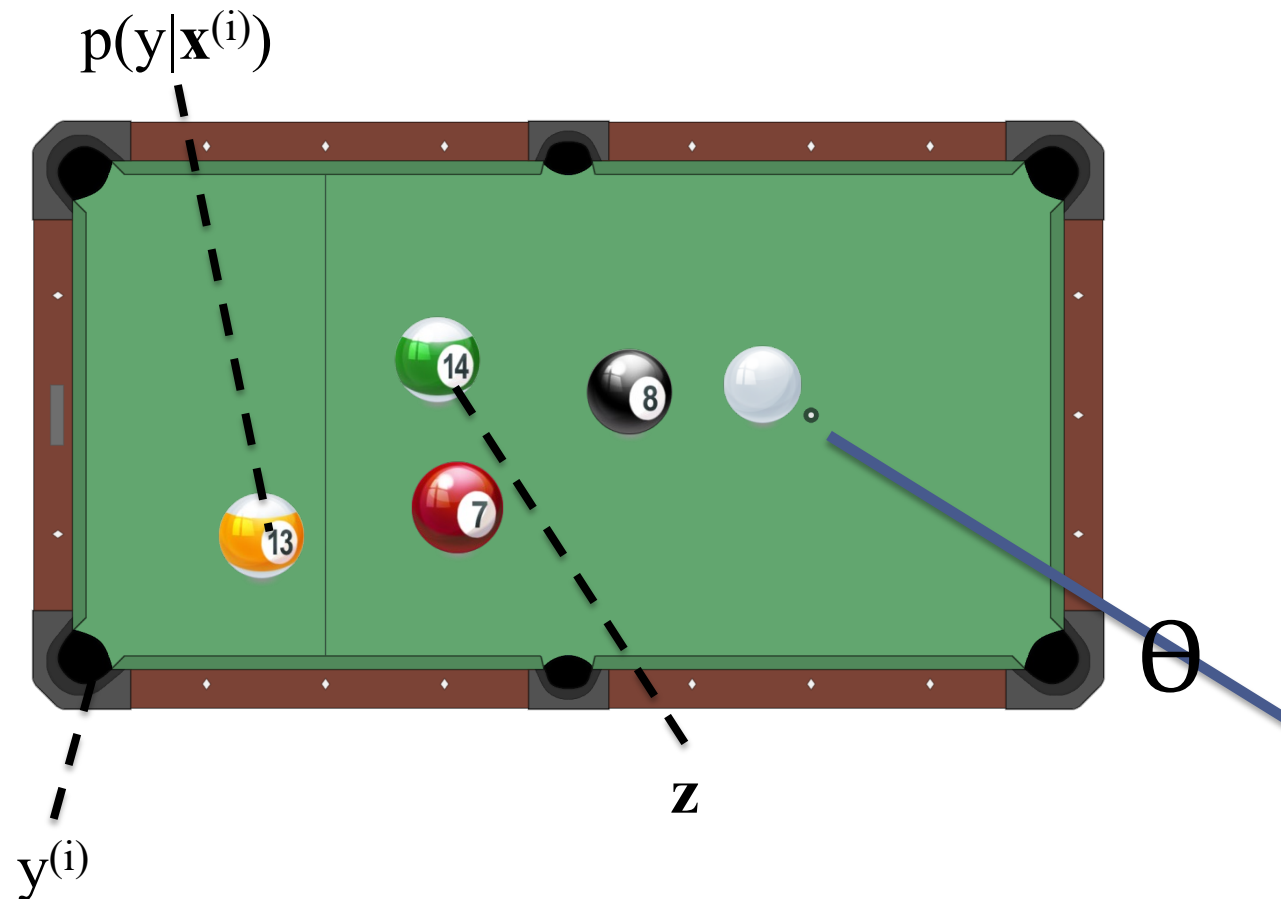


Error Back-Propagation



Slide from (Stoyanov & Eisner, 2012)

Error Back-Propagation



THE BACKPROPAGATION ALGORITHM

Automatic Differentiation – Reverse Mode (aka. Backpropagation)

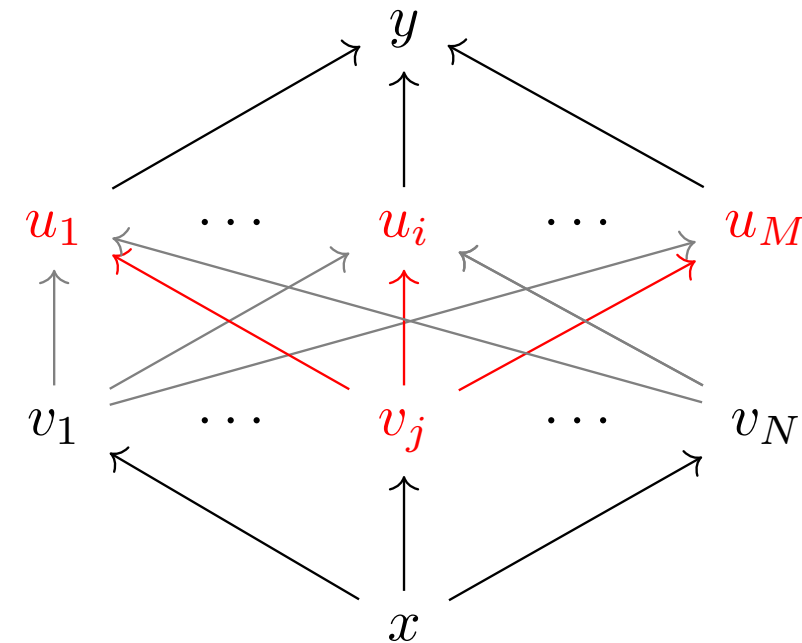
Forward Computation

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, \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

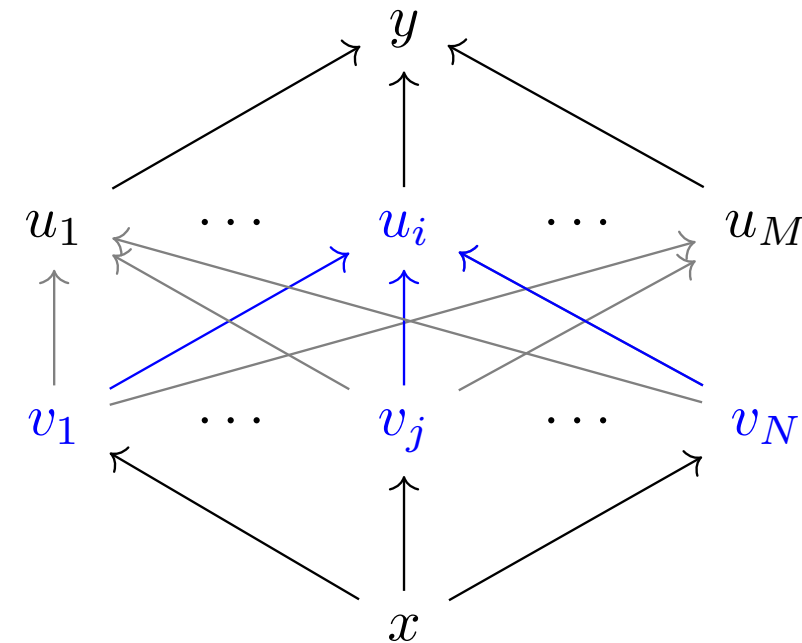
Automatic Differentiation – Reverse Mode (aka. Backpropagation)

Forward Computation

1. Write an **algorithm** for evaluating the function $y = f(x)$. 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_i 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

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 (Version B)

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
	$g_u = 0, g_{u_1} = 0, g_{u_2} = 0, g_t = 0, g_x = 0$ ← Initialize all the adjoints to zero
$J = \cos(u)$	$g_u = -\sin(u)$
$u = u_1 + u_2$	$g_{u_1} += g_u \frac{du}{du_1}, \frac{du}{du_1} = 1$ $g_{u_2} += g_u \frac{du}{du_2}, \frac{du}{du_2} = 1$
$u_1 = \sin(t)$	$g_t += g_{u_1} \frac{du_1}{dt}, \frac{du_1}{dt} = \cos(t)$ ← Notice that we increment the partial derivative for $\frac{dJ}{dt}$ in two places!
$u_2 = 3t$	$g_t += g_{u_2} \frac{du_2}{dt}, \frac{du_2}{dt} = 3$ ←
$t = x^2$	$g_x += g_t \frac{dt}{dx}, \frac{dt}{dx} = 2x$

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)

Gradients

1. Given training data

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

2. Choose each of the

– Decision function

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

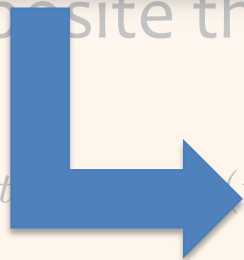
– Loss function

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

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)


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

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

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

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

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.



Vector Derivatives

Scalar Derivatives

Suppose $x \in \mathbb{R}$
and $f : \mathbb{R} \rightarrow \mathbb{R}$

$f(x)$	$\frac{\partial f(x)}{\partial x}$
bx	b
xb	b
x^2	$2x$
bx^2	$2bx$

Vector Derivatives

Suppose $\mathbf{x} \in \mathbb{R}^m$, $\mathbf{b} \in \mathbb{R}^m$,
 $\mathbf{B} \in \mathbb{R}^{m \times n}$, $\mathbf{Q} \in \mathbb{R}^{m \times m}$
and \mathbf{Q} is symmetric.

$f(\mathbf{x})$	$\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}}$	type of f
$\mathbf{b}^T \mathbf{x}$	\mathbf{b}	$f : \mathbb{R}^m \rightarrow \mathbb{R}$
$\mathbf{x}^T \mathbf{b}$	\mathbf{b}	$f : \mathbb{R}^m \rightarrow \mathbb{R}$
$\mathbf{x}^T \mathbf{B}$	\mathbf{B}	$f : \mathbb{R}^m \rightarrow \mathbb{R}^n$
$\mathbf{B}^T \mathbf{x}$	\mathbf{B}^T	$f : \mathbb{R}^m \rightarrow \mathbb{R}^n$
$\mathbf{x}^T \mathbf{x}$	$2\mathbf{x}$	$f : \mathbb{R}^m \rightarrow \mathbb{R}$
$\mathbf{x}^T \mathbf{Q} \mathbf{x}$	$2\mathbf{Q} \mathbf{x}$	$f : \mathbb{R}^m \rightarrow \mathbb{R}$

Vector Derivatives

Scalar Derivatives

Suppose $x \in \mathbb{R}^m$ and we have constants $a \in \mathbb{R}, b \in \mathbb{R}$

$f(x)$	$\frac{\partial f(x)}{\partial x}$
$g(x) + h(x)$	$\frac{\partial g(x)}{\partial x} + \frac{\partial h(x)}{\partial x}$
$ag(x)$	$a \frac{\partial g(x)}{\partial x}$
$g(x)b$	$\frac{\partial g(x)}{\partial x} b$

Vector Derivatives

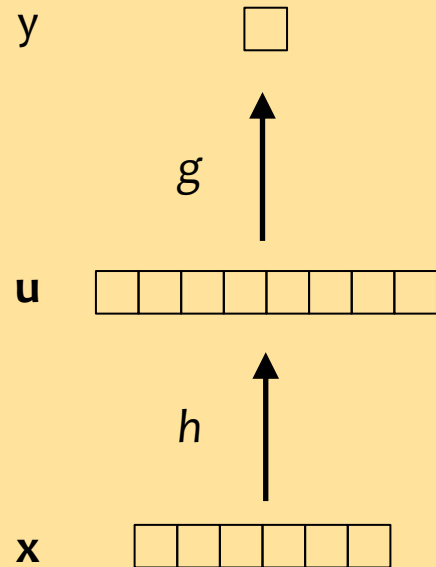
Suppose $\mathbf{x} \in \mathbb{R}^m$ and we have constants $a \in \mathbb{R}, \mathbf{b} \in \mathbb{R}^n$

$f(\mathbf{x})$	$\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}}$
$g(\mathbf{x}) + h(\mathbf{x})$	$\frac{\partial g(\mathbf{x})}{\partial \mathbf{x}} + \frac{\partial h(\mathbf{x})}{\partial \mathbf{x}}$
$ag(\mathbf{x})$	$a \frac{\partial g(\mathbf{x})}{\partial \mathbf{x}}$
$g(\mathbf{x})\mathbf{b}$	$\frac{\partial g(\mathbf{x})}{\partial \mathbf{x}} \mathbf{b}^T$

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:

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

Answer:

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

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

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

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

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

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

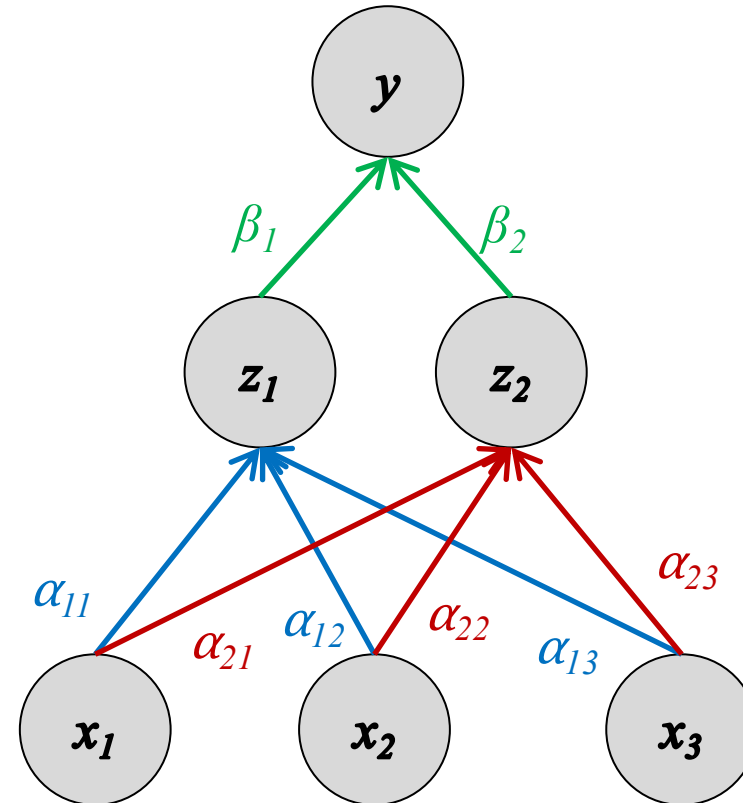
F. None of the above

DRAWING A NEURAL NETWORK

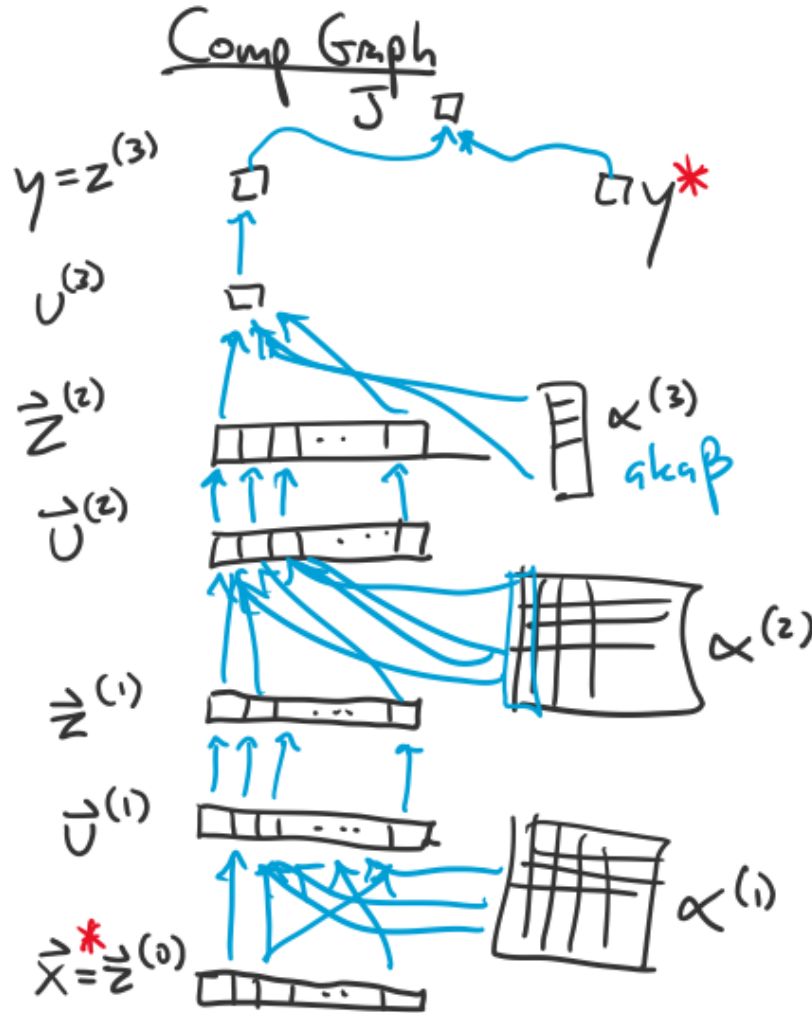
Ways of Drawing Neural Networks

Neural Network Diagram

- The diagram represents a neural network
- Nodes are **circles**
- One node per **hidden unit**
- Node is labeled with the **variable** corresponding to the hidden unit
- For a fully connected feed-forward neural network, a hidden unit is a nonlinear function of nodes in the previous layer
- *Edges are directed*
- Each **edge is labeled with its weight** (side note: we should be careful about ascribing how a matrix can be used to indicate the labels of the edges and pitfalls there)
- Other details:
 - Following standard convention, the **intercept term is NOT shown** as a node, but rather is assumed to be part of the non-linear function that yields a hidden unit. (i.e. its weight does NOT appear in the picture anywhere)
 - The diagram does **NOT include any nodes related to the loss computation**



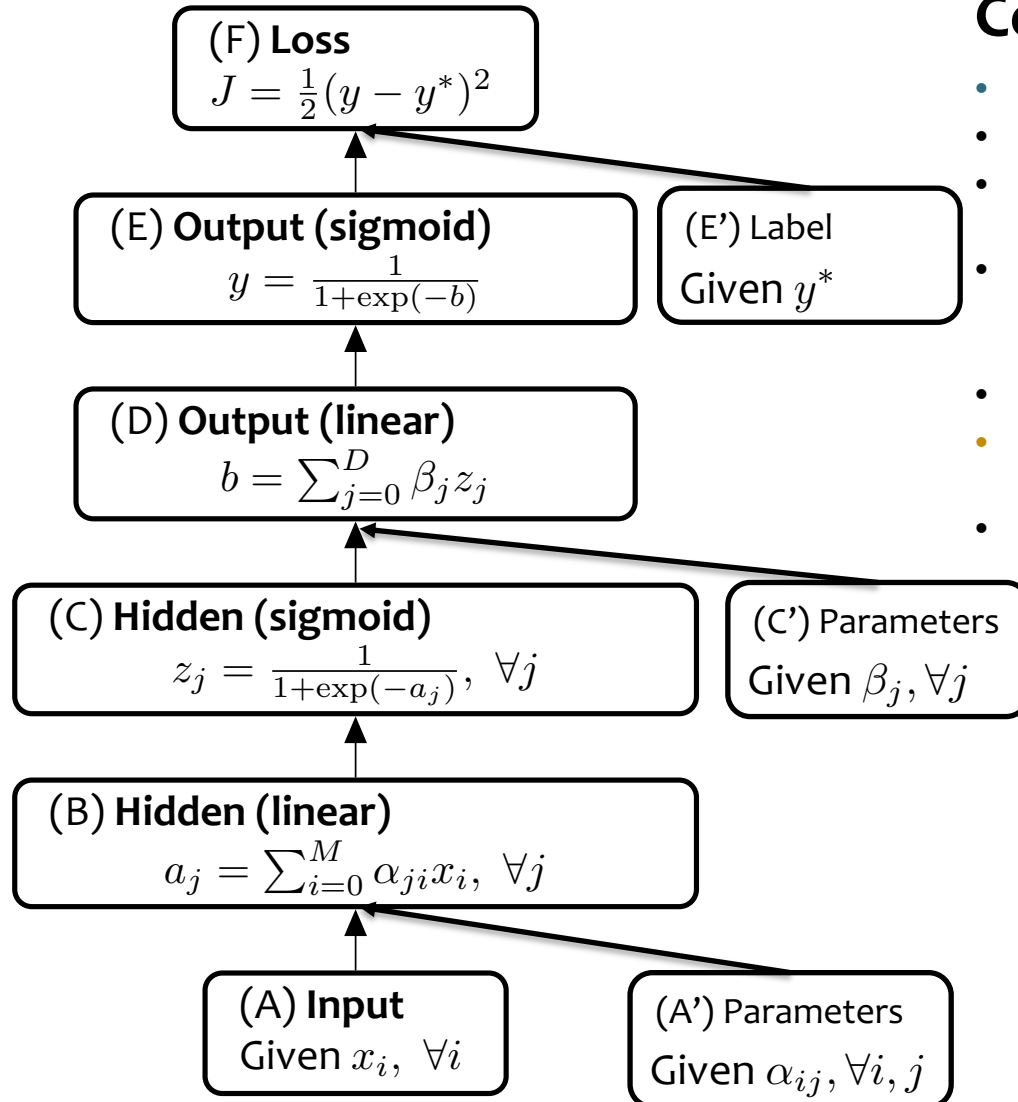
Ways of Drawing Neural Networks



Computation Graph

- The diagram represents an algorithm
- Nodes are **rectangles**
- One node per **intermediate variable in the algorithm**
- Node is labeled with the **function** that it computes (inside the box) and also the **variable** name (outside the box)
- Edges are directed
- Edges do not have labels (since they don't need them)
- For neural networks:
 - Each **intercept term should appear as a node** (if it's not folded in somewhere)
 - Each parameter should appear as a node
 - Each constant, e.g. a true label or a feature vector should appear in the graph
 - It's perfectly fine to include the loss

Ways of Drawing Neural Networks



Computation Graph

- The diagram represents an algorithm
- Nodes are **rectangles**
- One node per **intermediate variable in the algorithm**
- Node is labeled with the **function** that it computes (inside the box) and also the **variable** name (outside the box)
- Edges are directed
- Edges do not have labels (since they don't need them)
- For neural networks:
 - Each **intercept term** should appear as a node (if it's not folded in somewhere)
 - Each parameter should appear as a node
 - Each constant, e.g. a true label or a feature vector should appear in the graph
 - It's perfectly fine to include the loss

Ways of Drawing Neural Networks

Neural Network Diagram

- The diagram represents a neural network
- Nodes are **circles**
- One node per **hidden unit**
- Node is labeled with the **variable** corresponding to the hidden unit
- For a fully connected feed-forward neural network, a hidden unit is a nonlinear function of nodes in the previous layer
- *Edges are directed*
- Each **edge is labeled with its weight** (side note: we should be careful about ascribing how a matrix can be used to indicate the labels of the edges and pitfalls there)
- Other details:
 - Following standard convention, the **intercept term is NOT shown** as a node, but rather is assumed to be part of the non-linear function that yields a hidden unit. (i.e. its weight does NOT appear in the picture anywhere)
 - The diagram does **NOT include any nodes related to the loss computation**

Computation Graph

- The diagram represents an algorithm
- Nodes are **rectangles**
- One node per **intermediate variable in the algorithm**
- Node is labeled with the **function** that it computes (inside the box) and also the **variable** name (outside the box)
- *Edges are directed*
- **Edges do not have labels** (since they don't need them)
- For neural networks:
 - Each **intercept term should appear as a node** (if it's not folded in somewhere)
 - Each parameter should appear as a node
 - Each constant, e.g. a true label or a feature vector should appear in the graph
 - It's **perfectly fine to include the loss**

Important!

Some of these conventions are specific to 10-301/601. The literature abounds with variations on these conventions, but it's helpful to have some distinction nonetheless.

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...

- Differentiate between a neural network diagram and a computation graph
- 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
- Employ basic matrix calculus to compute vector/matrix/tensor derivatives.