Neural Networks + Backpropagation
Reminders

• Post-Exam Followup:
  – Exam Viewing
  – Exit Poll: Exam 1
  – Grade Summary 1

• Homework 4: Logistic Regression
  – Out: Fri, Oct. 1
  – Due: Mon, Oct. 11 at 11:59pm
ARCHITECTURES
Neural Networks

Whiteboard

– Example: Neural Network w/2 Hidden Layers
– Example: Feed Forward Neural Network (matrix form)
Neural Network for Classification

Neural Network

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

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

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

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

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

Input
$x_1, x_2, x_3, \ldots, x_M$

Hidden Layer
$z_1, z_2, \ldots, z_D$

Output
$y$

Neural Network for Classification

Given $x_i, \forall i$
Neural Network Architectures

Even for a basic Neural Network, there are many design decisions to make:

1. # of hidden layers (depth)
2. # of units per hidden layer (width)
3. Type of activation function (nonlinearity)
4. Form of objective function
5. How to initialize the parameters
BUILDING DEEPER NETWORKS
Building a Neural Net

Q: How many hidden units, \( D \), should we use?

![Neural network diagram]

Output

Hidden Layer

Input

\( D = M \)
Building a Neural Net

Q: How many hidden units, $D$, should we use?

Output

Hidden Layer

Input

$D = M$
Building a Neural Net

**Q: How many hidden units, D, should we use?**

What method(s) is this setting similar to?

Input: $x_1, x_2, x_3, \ldots, x_M$

Hidden Layer: $z_1, z_2, \ldots, z_D$

Output: $y$

$D < M$
Building a Neural Net

**Q: How many hidden units, \( D \), should we use?**

$$
\begin{align*}
\text{Input:} & \quad \{x_1, \ldots, x_M\} \\
\text{Hidden Layer:} & \quad \{z_1, z_2, \ldots, z_D\} \\
\text{Output:} & \quad y
\end{align*}
$$

What method(s) is this setting similar to?

\( D > M \)
Deeper Networks

Q: How many layers should we use?
Deeper Networks

Q: How many layers should we use?
Deeper Networks

Q: How many layers should we use?
Deeper Networks

Q: How many layers should we use?

- **Theoretical answer:**
  - A neural network with 1 hidden layer is a **universal function approximator**
  - Cybenko (1989): For any continuous function $g(x)$, there exists a 1-hidden-layer neural net $h_\theta(x)$ s.t. $|h_\theta(x) - g(x)| < \epsilon$ for all $x$, assuming sigmoid activation functions

- **Empirical answer:**
  - Before 2006: “Deep networks (e.g. 3 or more hidden layers) are too hard to train”
  - After 2006: “Deep networks are easier to train than shallow networks (e.g. 2 or fewer layers) for many problems”

Big caveat: You need to know and use the right tricks.
Feature Learning

- **Traditional feature engineering**: build up levels of abstraction by hand
- **Deep networks** (e.g. convolution networks): learn the increasingly higher levels of abstraction from data
  - each layer is a learned feature representation
  - sophistication increases in higher layers

Figures from Lee et al. (ICML 2009)
Feature Learning

- **Traditional feature engineering**: build up levels of abstraction by hand
- **Deep networks** (e.g. convolution networks): learn the increasingly higher levels of abstraction from data
  - each layer is a learned feature representation
  - sophistication increases in higher layers

Figures from Lee et al. (ICML 2009)
Feature Learning

- **Traditional feature engineering:** build up levels of abstraction by hand
- **Deep networks** (e.g. convolution networks): learn the increasingly higher levels of abstraction from data
  - each layer is a learned feature representation
  - sophistication increases in higher layers

Figures from Lee et al. (ICML 2009)
ACTIVATION FUNCTIONS
Activation Functions

Neural Network with sigmoid activation functions

(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 \( x_i, \forall i \)
Activation Functions

Neural Network with arbitrary nonlinear activation functions

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

(E) Output (nonlinear)
\[ y = \sigma(b) \]

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

(C) Hidden (nonlinear)
\[ z_j = \sigma(a_j), \ \forall j \]

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

(A) Input
Given \( x_i, \ \forall i \)
So far, we’ve assumed that the activation function (nonlinearity) is always the sigmoid function...

...but the sigmoid is not widely used in modern neural networks
Activation Functions

• sigmoid, $\sigma(x)$
  – output in range $(0,1)$
  – good for probabilistic outputs

• hyperbolic tangent, $\tanh(x)$
  – similar shape to sigmoid, but output in range $(-1,+1)$
Understanding the difficulty of training deep feedforward neural networks

Figure from Glorot & Bentio (2010)
Activation Functions

• Rectified Linear Unit (ReLU)
  – avoids the vanishing gradient problem
  – derivative is fast to compute

\[ \text{ReLU}(x) = \max(0, x) \]
Activation Functions

• Rectified Linear Unit (ReLU)
  – avoids the vanishing gradient problem
  – derivative is fast to compute

\[ \text{ReLU}(x) = \max(0, x) \]

• Exponential Linear Unit (ELU)
  – same as ReLU on positive inputs
  – unlike ReLU, allows negative outputs and smoothly transitions for \( x < 0 \)

\[ \text{ELU}(x) = \begin{cases} 
  x, & \text{if } x > 0 \\
  \alpha(\exp(x) - 1), & \text{if } x \leq 0 
\end{cases} \]
Activation Functions

1. Training loss converges fastest with ELU
2. ELU(x) yields lower test error than ReLU(x) on CIFAR-10

Figure from Clevert et al. (2016)
LOSS FUNCTIONS & OUTPUT LAYERS
Neural Network

Decision Functions

Neural Network for Classification

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

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

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

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

(E) Output (sigmoid)
\[ y = \frac{1}{1 + \exp(-b)} \]
Neural Network for Regression

Output

Hidden Layer

Input

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

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

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

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

$\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_D$

$\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \ldots, \mathbf{x}_M$

$y$
Objective Functions for NNs

1. **Quadratic Loss:**
   - the same objective as Linear Regression
   - i.e. mean squared error
   - add an additional “softmax” layer at the end of our network

   \[
   J = \frac{1}{2} (y - y^*)^2 \\
   \frac{dJ}{dy} = y - y^*
   \]

2. **Cross-Entropy:**
   - the same objective as Logistic Regression
   - i.e. negative log likelihood
   - This requires probabilities, so we add an additional “softmax” layer at the end of our network

   \[
   J = y^* \log(y) + (1 - y^*) \log(1 - y) \\
   \frac{dJ}{dy} = y^* \frac{1}{y} + (1 - y^*) \frac{1}{y - 1}
   \]
Objective Functions for NNs

Cross-entropy vs. Quadratic loss

Figure 5: Cross entropy (black, surface on top) and quadratic (red, bottom surface) cost as a function of two weights (one at each layer) of a network with two layers, $W_1$ respectively on the first layer and $W_2$ on the second, output layer.

Figure from Glorot & Bentio (2010)
Multi-Class Output

Output

Hidden Layer

Input

\( y_1 \quad \cdots \quad y_K \)

\( z_1 \quad z_2 \quad \cdots \quad z_D \)

\( x_1 \quad x_2 \quad x_3 \quad \cdots \quad x_M \)
Multi-Class Output

Softmax:

\[ y_k = \frac{\exp(b_k)}{\sum_{l=1}^{K} \exp(b_l)} \]

(F) Loss

\[ J = \sum_{k=1}^{K} y_k^* \log(y_k) \]

(E) Output (softmax)

\[ y_k = \frac{\exp(b_k)}{\sum_{l=1}^{K} \exp(b_l)} \]

(D) Output (linear)

\[ b_k = \sum_{j=0}^{D} \beta_{kj} z_j \forall k \]

(C) Hidden (nonlinear)

\[ z_j = \sigma(a_j), \forall j \]

(B) Hidden (linear)

\[ a_j = \sum_{i=0}^{M} \alpha_{ji} x_i, \forall j \]

(A) Input

Given \( x_i, \forall i \)
**Question X:** For which of the datasets below does there exist a one-hidden layer neural network that achieves zero classification error? **Select all that apply.**

A) ![Dataset A](image1.png)
B) ![Dataset B](image2.png)
C) ![Dataset C](image3.png)
D) ![Dataset D](image4.png)

**Question Y:** For which of the datasets below does there exist a one-hidden layer neural network for regression that achieves nearly zero MSE? **Select all that apply.**

A) ![Regression A](image5.png)
B) ![Regression B](image6.png)
C) ![Regression C](image7.png)
D) ![Regression D](image8.png)
Lecture 12: In-Class Poll
@ When poll is active, respond at pollev.com/10301601polls

Question 1

A
B
C
D
E
@ When poll is active, respond at pollev.com/10301601polls

Question 2

A

B

C

D

E
Examples 3 and 4

DECISION BOUNDARY EXAMPLES
Example #1: Diagonal Band

Example #2: One Pocket

Example #3: Four Gaussians

Example #4: Two Pockets
Example #3: Four Gaussians
Example #3: Four Gaussians
Example #3: Four Gaussians

K-NN (k=5, metric=euclidean)
Example #3: Four Gaussians

Tuned Neural Network (hidden=2, activation=logistic)
Example #3: Four Gaussians

LR1 for Tuned Neural Network (hidden=2, activation=logistic)
Example #3: Four Gaussians
Example #3: Four Gaussians
Example #4: Two Pockets
Example #4: Two Pockets

Logistic Regression
Example #4: Two Pockets

SVM (kernel=linear)
Example #4: Two Pockets

SVM (kernel=rbf, gamma=80.000000)
Example #4: Two Pockets

K-NN ($k=5$, metric=euclidean)
Example #4: Two Pockets

Tuned Neural Network (hidden=2, activation=logistic)
Example #4: Two Pockets

Tuned Neural Network (hidden=3, activation=logistic)
Example #4: Two Pockets

Tuned Neural Network (hidden=4, activation=logistic)
Example #4: Two Pockets

Tuned Neural Network (hidden=10, activation=logistic)
Neural Networks Objectives

You should be able to...

• Explain the biological motivations for a neural network
• Combine simpler models (e.g. linear regression, binary logistic regression, multinomial logistic regression) as components to build up feed-forward neural network architectures
• Explain the reasons why a neural network can model nonlinear decision boundaries for classification
• Compare and contrast feature engineering with learning features
• Identify (some of) the options available when designing the architecture of a neural network
• Implement a feed-forward neural network
Computing Gradients

APPROACHES TO DIFFERENTIATION
1. Given training data:
   \[ \{ \mathbf{x}_i, y_i \}_{i=1}^{N} \]

2. Choose each of these:
   – Decision function
   \[ \hat{y} = f_{\theta}(\mathbf{x}_i) \]
   – Loss function
   \[ \ell(\hat{y}, y_i) \in \mathbb{R} \]

3. Define goal:
\[ \theta^* = \arg \min_{\theta} \sum_{i=1}^{N} \ell(f_{\theta}(\mathbf{x}_i), y_i) \]

4. Train with SGD:
(take small steps opposite the gradient)
\[ \theta^{(t+1)} = \theta^{(t)} - \eta_t \nabla \ell(f_{\theta}(\mathbf{x}_i), y_i) \]
A Recipe for Machine Learning

Background

1. Given training data:
\[
\{x_i, y_i\}_{i=1}^{N}
\]

2. Choose each of these:
- Decision function
- Loss function

\[\hat{y} = f_\theta(x_i)\]

\[\ell(\hat{y}, y_i) \in \mathbb{R}\]

Gradients

Backpropagation can compute this gradient!

And it’s a **special case of a more general algorithm** called reverse-mode automatic differentiation that can compute the gradient of any differentiable function efficiently!

\[\theta^{(t)} \leftarrow \theta^{(t)} - \eta_t \nabla \ell(f_\theta(x_i), y_i)\]
• **Question 1:**
  When can we compute the gradients for an arbitrary neural network?

• **Question 2:**
  When can we make the gradient computation efficient?
Approaches to Differentiation

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 $f(x)$
Approaches to Differentiation

3. Automatic Differentiation - Reverse Mode
   - Note: Called Backpropagation when applied to Neural Nets
   - Pro: Computes partial derivatives of one output $f(x)_i$ 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 $f(x)_i$ 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 $f(x)$

\[ A = \# \text{params} \quad B = 1 \]

Given $f : \mathbb{R}^A \rightarrow \mathbb{R}^B, f(x)$
Compute $\frac{\partial f(x)_i}{\partial x_j} \forall i, j$
THE FINITE DIFFERENCE METHOD
The centered finite difference approximation is:

$$\frac{\partial}{\partial \theta_i} J(\theta) \approx \frac{(J(\theta + \epsilon \cdot d_i) - J(\theta - \epsilon \cdot d_i))}{2\epsilon}$$

(1)

where $d_i$ is a 1-hot vector consisting of all zeros except for the $i$th entry of $d_i$, 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
Differentiation Quiz #1:
Suppose \( x = 2 \) and \( z = 3 \), what are \( \frac{dy}{dx} \) and \( \frac{dy}{dz} \) 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

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

\[ y = \sigma(\beta^T \sigma((\alpha^{(2)})^T \sigma((\alpha^{(1)})^T x))) \]

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

\[ \sigma(a) = [\sigma(a_1), \ldots, \sigma(a_K)]^T \]

Let \( \sigma \) be sigmoid: \( \sigma(a) = \frac{1}{1 + e^{x-a}} \)

What is \( \frac{\partial y}{\partial \beta_j} \) and \( \frac{\partial y}{\partial \alpha_{j}^{(i)}} \) for all \( i, j \).
THE CHAIN RULE OF CALCULUS
Whiteboard

– Chain Rule of Calculus
Given: $y = g(u)$ and $u = h(x)$.

Chain Rule:
\[
\frac{dy_i}{dx_k} = \sum_{j=1}^{J} \frac{dy_i}{du_j} \frac{du_j}{dx_k}, \quad \forall i, k
\]
Given: $y = g(u)$ and $u = h(x)$.

Chain Rule:

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

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

BACKPROPAGATION OF ERRORS
Error Back-Propagation

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)
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 $\frac{dy}{dx}$ and $\frac{dy}{dz}$ 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
Differentiation Quiz #1:
Suppose $x = 2$ and $z = 3$, what are $\frac{dy}{dx}$ and $\frac{dy}{dz}$ for the function below? Round your answer to the nearest integer.

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