Neural Networks
Reminders

• Homework 4: Logistic Regression
  – Out: Sun, Sep 30
  – Due: Tue, Oct 9 at 11:59pm

• Homework 5: Neural Networks
  – Out: Tue, Oct 9
  – Due: Sat, Oct 20 at 11:59pm
Q&A
Neural Networks Outline

• Logistic Regression (Recap)
  – Data, Model, Learning, Prediction

• Neural Networks
  – A Recipe for Machine Learning
  – Visual Notation for Neural Networks
  – Example: Logistic Regression Output Surface
  – 2-Layer Neural Network
  – 3-Layer Neural Network

• Neural Net Architectures
  – Objective Functions
  – Activation Functions

• Backpropagation
  – Basic Chain Rule (of calculus)
  – Chain Rule for Arbitrary Computation Graph
  – Backpropagation Algorithm
  – Module-based Automatic Differentiation (Autodiff)
NEURAL NETWORKS
1. Given training data:
\[ \{ x_i, y_i \}_{i=1}^{N} \]

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

**Examples:** Linear regression, Logistic regression, Neural Network

**Examples:** Mean-squared error, Cross Entropy
A Recipe for Machine Learning

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

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

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

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

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

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

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

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

3. Define goal:

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

**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!

\[ \theta^{(t)} \rightarrow \theta^{(t)} - \eta_t \nabla \ell(f_\theta(x_i), y_i) \]
A Recipe for Machine Learning

1. Given training data:
2. Choose each of these:
   - Decision function
   - Loss function
3. Define goal:
4. Train with SGD:
   (take small steps opposite the gradient)

Goals for Today’s Lecture

1. Explore a **new class of decision functions** (Neural Networks)
2. Consider **variants of this recipe** for training
Linear Regression

\[ y = h_\theta(x) = \sigma(\theta^T x) \]

where \( \sigma(a) = a \)
Logistic Regression

Decision Functions

Output

\[ y = h_\theta(x) = \sigma(\theta^T x) \]

where \( \sigma(a) = \frac{1}{1 + \exp(-a)} \)

Input

\( \theta_1 \)

\( \theta_2 \)

\( \theta_3 \)

\( \theta_M \)

\( x_1 \)

\( x_2 \)

\( x_3 \)

\( \ldots \)

\( x_M \)
$y = h_\theta(x) = \sigma(\theta^T x)$

where $\sigma(a) = \frac{1}{1 + \exp(-a)}$
Logistic Regression

Output

\[
y = h_\theta(x) = \sigma(\theta^T x)
\]

In-Class Example

Decision Functions

Input

\[
\begin{align*}
\theta_1 & \\
\theta_2 & \\
\theta_3 & \\
\end{align*}
\]

\[
\begin{align*}
x_1 & \\
x_2 & \\
x_3 & \\
\end{align*}
\]
Perceptron

\[ y = h_\theta(x) = \sigma(\theta^T x) \]

where \( \sigma(a) = \text{sign}(a) \)
From Biological to Artificial

The motivation for Artificial Neural Networks comes from biology...

Biological “Model”
- **Neuron**: an excitable cell
- **Synapse**: connection between neurons
- A neuron sends an electrochemical pulse along its synapses when a sufficient voltage change occurs
- **Biological Neural Network**: collection of neurons along some pathway through the brain

Biological “Computation”
- Neuron switching time: \( \sim 0.001 \text{ sec} \)
- Number of neurons: \( \sim 10^{10} \)
- Connections per neuron: \( \sim 10^{4.5} \)
- Scene recognition time: \( \sim 0.1 \text{ sec} \)

Artificial Model
- **Neuron**: node in a directed acyclic graph (DAG)
- **Weight**: multiplier on each edge
- **Activation Function**: nonlinear thresholding function, which allows a neuron to “fire” when the input value is sufficiently high
- **Artificial Neural Network**: collection of neurons into a DAG, which define some differentiable function

Artificial Computation
- Many neuron-like threshold switching units
- Many weighted interconnections among units
- Highly parallel, distributed processes

Slide adapted from Eric Xing
Logistic Regression

\[ y = h_\theta(x) = \sigma(\theta^T x) \]

where \( \sigma(a) = \frac{1}{1 + \exp(-a)} \)
Neural Networks

Chalkboard

– Example: Neural Network w/1 Hidden Layer
– Example: Neural Network w/2 Hidden Layers
– Example: Feed Forward Neural Network
Decision Functions

Neural Network

Output

Hidden Layer

Input
Decision Functions

Neural Network

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 $y$

$$\text{Loss} \quad J = \frac{1}{2} \sum (y - y(d))^2$$
DECISION BOUNDARY EXAMPLES
Example #1: Diagonal Band

Example #2: One Pocket

Example #3: Four Gaussians

Example #4: Two Pockets
Example #1: Diagonal Band
Example #1: Diagonal Band

Logistic Regression
Example #1: Diagonal Band

Tuned Neural Network (hidden=2, activation=logistic)
Example #1: Diagonal Band

LR1 for Tuned Neural Network (hidden=2, activation=logistic)
Example #1: Diagonal Band

LR2 for Tuned Neural Network (hidden=2, activation=logistic)
Example #1: Diagonal Band

Tuned Neural Network (hidden=2, activation=logistic)
Example #1: Diagonal Band
Example #2: One Pocket
Example #2: One Pocket

Logistic Regression
Example #2: One Pocket

Tuned Neural Network (hidden=3, activation=logistic)
Example #2: One Pocket

LR1 for Tuned Neural Network (hidden=3, activation=logistic)
Example #2: One Pocket

LR2 for Tuned Neural Network (hidden=3, activation=logistic)
Example #2: One Pocket

LR3 for Tuned Neural Network (hidden=3, activation=logistic)
Example #2: One Pocket

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

$\text{SVM (kernel=rbf, gamma=80.000000)}$
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

LR2 for Tuned Neural Network (hidden=2, activation=logistic)
Example #3: Four Gaussians
Example #4: Two Pockets
Example #4: Two Pockets
Example #4: Two Pockets
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)
ARCHITECTURES
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
Q: How many hidden units, $D$, should we use?
Q: How many hidden units, D, should we use?
Building a Neural Net

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

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?

$D > M$
Q: How many layers should we use?
Q: How many layers should we use?
Q: How many layers should we use?
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 $h_\theta(x)$ such that $|h_\theta(x) - g(x)| < \epsilon$ 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.
Different Levels of Abstraction

• We don’t know the “right” levels of abstraction
• So let the model figure it out!

Example from Honglak Lee (NIPS 2010)
Different Levels of Abstraction

Face Recognition:

– Deep Network can build up increasingly higher levels of abstraction

– Lines, parts, regions

Example from Honglak Lee (NIPS 2010)
Different Levels of Abstraction

Decision Functions

Example from Honglak Lee (NIPS 2010)
Activation Functions

Neural Network with sigmoid activation functions

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

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

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

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

Output
$y$

Diagram of a neural network with sigmoid activation functions.
Activation Functions

Neural Network with arbitrary nonlinear activation functions

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

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

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

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

(E) Output (nonlinear)
$$y = \sigma(b)$$

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

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

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

Output
$y$
So far, we’ve assumed that the activation function (nonlinearity) is always the sigmoid function...

\[
\text{logistic}(u) \equiv \frac{1}{1 + e^{-u}}
\]
Activation Functions

• A new change: modifying the nonlinearity
  – The logistic is not widely used in modern ANNs

Alternate 1: tanh
Like logistic function but shifted to range [-1, +1]
Understanding the difficulty of training deep feedforward neural networks

Figure from Glorot & Bentio (2010)
Activation Functions

• A new change: modifying the nonlinearity
  – reLU often used in vision tasks

Alternate 2: rectified linear unit
Linear with a cutoff at zero
(Implementation: clip the gradient when you pass zero)
Activation Functions

• A new change: modifying the nonlinearity
  – reLU often used in vision tasks

  Alternate 2: rectified linear unit
  Soft version: \( \log(\exp(x)+1) \)
  Doesn’t saturate (at one end)
  Sparsifies outputs
  Helps with vanishing gradient
Objective Functions for NNs

1. Quadratic Loss:
   – the same objective as Linear Regression
   – i.e. mean squared error

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

Forward

<table>
<thead>
<tr>
<th>Quadratic</th>
<th>Cross Entropy</th>
</tr>
</thead>
<tbody>
<tr>
<td>$J = \frac{1}{2}(y - y^*)^2$</td>
<td>$J = y^* \log(y) + (1 - y^*) \log(1 - y)$</td>
</tr>
</tbody>
</table>

Backward

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$\frac{dJ}{dy} = y - y^*$</td>
<td>$\frac{dJ}{dy} = y^* \frac{1}{y} + (1 - y^*) \frac{1}{y - 1}$</td>
</tr>
</tbody>
</table>
Objective Functions for NNs

Cross-entropy vs. Quadratic loss

Figure 3: 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
Multi-Class Output

Softmax:

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

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

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

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

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

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

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