Neural Networks
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
Q&A
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:
\[ \{ \mathbf{x}_i, y_i \}_{i=1}^{N} \]

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)
\[ \theta^{(t+1)} = \theta^{(t)} - \eta_t \nabla \ell(f_{\theta}(\mathbf{x}_i), y_i) \]
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!

\[ \theta^{(t)} = \theta^{(t)} - \eta_t \nabla \ell(f_\theta(x_i), y_i) \]
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

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

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

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

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

Decision Functions

\[ \theta_1 \quad \theta_2 \quad \theta_3 \]

Output

Input

Face

Face

Not a face
$$y = h_\theta(x) = \sigma(\theta^T x)$$

In-Class Example

Logistic Regression

Decision Functions
Perceptron

Decision Functions

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

where \( \sigma(a) = \text{sign}(a) \)
Neural Network

Decision Functions

Output

Hidden Layer

Input

\[ x_1 \quad x_2 \quad x_3 \quad \cdots \quad x_M \]

\[ z_1 \quad z_2 \quad \cdots \quad z_D \]

\[ y \]
Neural Network Model

Independent variables

Weights

Hidden Layer

Weights

Dependent variable

Prediction

© Eric Xing @ CMU, 2006-2011
“Combined logistic models”

Inputs

<table>
<thead>
<tr>
<th>Age</th>
<th>Gender</th>
<th>Stage</th>
</tr>
</thead>
<tbody>
<tr>
<td>34</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

Weights

- Independent variables
- Weights
- Hidden Layer
- Weights

Output

“Probability of being Alive”

Predicted probability: 0.6

© Eric Xing @ CMU, 2006-2011
Inputs

- Age: 34
- Gender: 2
- Stage: 4

Weights

- Inputs to Hidden Layer: 0.2, 0.3, 0.2
- Hidden Layer to Output: 0.5, 0.8

Output

- “Probability of being Alive”: 0.6

Independent variables

- Age
- Gender
- Stage

Weights

- Hidden Layer

Dependent variable

Prediction

© Eric Xing @ CMU, 2006-2011
Inputs

Age
34

Gender
1

Stage
4

Independent variables

Weights

Hidden Layer

Weights

Dependent variable

Prediction

Output

0.6

“Probability of being Alive”
Not really, no target for hidden units...

Independent variables

Age
Gender
Stage

Weights

Hidden Layer

Dependent variable
Prediction

© Eric Xing @ CMU, 2006-2011
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
Neural Networks

Chalkboard

– Example: Neural Network w/1 Hidden Layer
– Example: Neural Network w/2 Hidden Layers
– Example: Feed Forward Neural Network
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)
\[ y = \frac{1}{1+\exp(-b)} \]
Question:
Suppose you are training a one-hidden layer neural network with sigmoid activations for binary classification.

True or False: There is a unique set of parameters that maximize the likelihood of the dataset above.

Answer:
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
Building a Neural Net

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

![Diagram of a neural network with input, hidden layer, and output nodes.](image)

- **Input**: $x_1, x_2, \ldots, x_M$
- **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?

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?

$D < M$
Building a Neural Net

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

Output

Hidden Layer

Input

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?
Q: How many layers should we use?

Deeper Networks
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.
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

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
$$J = \frac{1}{2}(y - y^*)^2$$
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 \( x_i, \ \forall i \)
Activation Functions

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

Sigmoid / Logistic 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 & Bento (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

Slide from William Cohen
Neural Network for **Classification**

- **Input** \( x_1, x_2, x_3, \ldots, x_M \)
- **Hidden Layer** \( z_1, z_2, \ldots, z_D \)
- **Output** \( y \)

**Equations**

**A** Input

\[ x_i, \ \forall i \]

**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

(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)
\[ y = \sum_{j=0}^{D} \beta_j z_j \]
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

<table>
<thead>
<tr>
<th></th>
<th>Forward</th>
<th>Backward</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quadratic</td>
<td>$J = \frac{1}{2}(y - y^*)^2$</td>
<td>$\frac{dJ}{dy} = y - y^*$</td>
</tr>
<tr>
<td>Cross Entropy</td>
<td>$J = y^* \log(y) + (1 - y^*) \log(1 - y)$</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 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
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)
\[ b_k = \sum_{j=0}^{D} \beta_{kj} z_j \quad \forall k \]

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

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

(A) Input
Given \( x_i, \quad \forall i \)
Question A: On which of the datasets below could a one-hidden layer neural network achieve zero classification error? Select all that apply.

A) + + + +
B) + + + +
C) + + + +
D) + + + +

Question B: On which of the datasets below could a one-hidden layer neural network for regression achieve nearly zero MSE? Select all that apply.

A) dots
B) dots
C) dots
D) dots
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
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
Example #2: One Pocket

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

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

Logistic Regression
Example #4: Two Pockets

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