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

```
Input: $x_1, x_2, \ldots, x_M$
Hidden Layer: $z_1, z_2, \ldots, z_D$
Output: $y$
```

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

Output

Hidden Layer

Input

What method(s) is this setting similar to?

D < M
What method(s) is this setting similar to?

Q: How many hidden units, $D$, should we use?
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 $h_\theta(x)$ 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)
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

(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

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

Given $x_i$,

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

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

Backward

<table>
<thead>
<tr>
<th>$\frac{dJ}{dy}$</th>
<th>$y - y^*$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\frac{dJ}{dy}$</td>
<td>$y^* \frac{1}{y} + (1 - y^*) \frac{1}{y - 1}$</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:

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

- **Input**: \( x_1, x_2, \ldots, x_M \)
- **Hidden (linear)**
  \[ a_j = \sum_{i=0}^{M} \alpha_{j,i} x_i, \forall j \]
- **Hidden (nonlinear)**
  \[ z_j = \sigma(a_j), \forall j \]
- **Output (linear)**
  \[ b_k = \sum_{j=0}^{D} \beta_{k,j} z_j \forall k \]
- **Output (softmax)**
  \[ y_k = \frac{\exp(b_k)}{\sum_{l=1}^{K} \exp(b_l)} \]
- **Loss**
  \[ J = \sum_{k=1}^{K} y_k^* \log(y_k) \]

Given \( x_i, \forall i \)
BACKPROPAGATION
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)
   \[ \theta^{(t+1)} = \theta^{(t)} - \eta_t \nabla \ell(f_{\theta}(\mathbf{x}_i), y_i) \]
• **Question 1:**
  When can we compute the gradients of the parameters of 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 $f(x)$

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 $f(x)$

Given $f : \mathbb{R}^A \to \mathbb{R}^B$, $f(x)$
Compute $\frac{\partial f(x)_i}{\partial x_j} \forall i, j$
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
Chain Rule Quiz #1:
Suppose $x = 2$ and $z = 3$, what are $\frac{dy}{dx}$ and $\frac{dy}{dz}$ for the function below?

$$y = \exp(xz) + \frac{xz}{\log(x)} + \frac{\sin(\log(x))}{\exp(xz)}$$
Calculus 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: \( \sigma(a) = \frac{1}{1 + \exp(-a)} \)

What is \( \frac{\partial y}{\partial \beta_j} \) and \( \frac{\partial y}{\partial \alpha^{(i)}_j} \) for all \( i, j \).
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:

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