10-701 Introduction to Machine Learning

Perceptron & Neural Networks

Readings:
Bishop Ch. 4.1.7, Ch. 5
Murphy Ch. 16.5, Ch. 28
Mitchell Ch. 4

Matt Gormley
Lecture 12
October 17, 2016
Reminders

• Homework 3:
  – due 10/24/16
Outline

• Discriminative vs. Generative
• Perceptron
• Neural Networks
• Backpropagation
DISCRIMINATIVE AND GENERATIVE CLASSIFIERS
Generative vs. Discriminative

• **Generative Classifiers:**
  – Example: Naïve Bayes
  – Define a joint model of the observations \( x \) and the labels \( y \): \( p(x, y) \)
  – Learning maximizes (joint) likelihood
  – Use Bayes’ Rule to classify based on the posterior:
    \[
    p(y|x) = \frac{p(x|y)p(y)}{p(x)}
    \]

• **Discriminative Classifiers:**
  – Example: Logistic Regression
  – Directly model the conditional: \( p(y|x) \)
  – Learning maximizes conditional likelihood
Generative vs. Discriminative

Finite Sample Analysis (Ng & Jordan, 2002)

[Assume that we are learning from a finite training dataset]

If model assumptions are correct: Naive Bayes is a more efficient learner (requires fewer samples) than Logistic Regression

If model assumptions are incorrect: Logistic Regression has lower asymptotic error, and does better than Naïve Bayes
solid: NB  dashed: LR

pima (continuous)

adult (continuous)

boston (predict if > median price, continuous)

liver disorders (continuous)

sonar (continuous)

Slide courtesy of William Cohen
Naïve Bayes makes stronger assumptions about the data but needs fewer examples to estimate the parameters

Generative vs. Discriminative Learning (Parameter Estimation)

**Naïve Bayes:**
Parameters are decoupled $\rightarrow$ Closed form solution for MLE

**Logistic Regression:**
Parameters are coupled $\rightarrow$ No closed form solution – must use iterative optimization techniques instead
Naïve Bayes vs. Logistic Reg.

Learning (MAP Estimation of Parameters)

**Bernoulli Naïve Bayes:**
Parameters are probabilities → Beta prior (usually) pushes probabilities away from zero / one extremes

**Logistic Regression:**
Parameters are not probabilities → Gaussian prior encourages parameters to be close to zero

(Effectively pushes the probabilities away from zero / one extremes)
Naïve Bayes vs. Logistic Reg.

Features

**Naïve Bayes:**
Features $x$ are assumed to be conditionally independent given $y$. (i.e. Naïve Bayes Assumption)

**Logistic Regression:**
No assumptions are made about the form of the features $x$. They can be dependent and correlated in any fashion.
THE PERCEPTRON ALGORITHM
Why don’t we drop the generative model and try to learn this hyperplane directly?
Background: Hyperplanes

Hyperplane (Definition 1):
\[ \mathcal{H} = \{ \mathbf{x} : \mathbf{w}^T \mathbf{x} = b \} \]

Hyperplane (Definition 2):
\[ \mathcal{H} = \{ \mathbf{x} : \mathbf{w}^T \mathbf{x} = 0 \]
and \( x_1 = 1 \} \]

Half-spaces:
\[ \mathcal{H}^+ = \{ \mathbf{x} : \mathbf{w}^T \mathbf{x} > 0 \text{ and } x_1 = 1 \} \]
\[ \mathcal{H}^- = \{ \mathbf{x} : \mathbf{w}^T \mathbf{x} < 0 \text{ and } x_1 = 1 \} \]
Why don’t we drop the generative model and try to learn this hyperplane directly?

Directly modeling the hyperplane would use a decision function:

\[ h(x) = \text{sign}(\theta^T x) \]

for:

\[ y \in \{-1, +1\} \]
Online Learning Model

Setup:
- We receive an example \((x, y)\)
- Make a prediction \(h(x)\)
- Check for correctness \(h(x) = y?\)

Goal:
- **Minimize** the number of mistakes
Margins

Definition: The **margin** of example $x$ w.r.t. a linear separator $w$ is the distance from $x$ to the plane $w \cdot x = 0$ (or the negative if on wrong side).

Definition: The margin $\gamma_w$ of a set of examples $S$ wrt a linear separator $w$ is the smallest margin over points $x \in S$.

Definition: The margin $\gamma$ of a set of examples $S$ is the maximum $\gamma_w$ over all linear separators $w$. 

Slide from Nina Balcan
Perceptron Algorithm

**Data:** Inputs are continuous vectors of length K. Outputs are discrete.

\[
D = \{x^{(i)}, y^{(i)}\}_{i=1}^N \quad \text{where} \quad x \in \mathbb{R}^K \quad \text{and} \quad y \in \{+1, -1\}
\]

**Prediction:** Output determined by hyperplane.

\[
\hat{y} = h_\theta(x) = \text{sign}(\theta^T x)
\]

\[
\text{sign}(a) = \begin{cases} 
  1, & \text{if } a \geq 0 \\
  -1, & \text{otherwise}
\end{cases}
\]

**Learning:** Iterative procedure:
- **while** not converged
  - **receive** next example \((x, y)\)
  - **predict** \(y' = h(x)\)
  - **if** positive mistake: **add** \(x\) to parameters
  - **if** negative mistake: **subtract** \(x\) from parameters
Perceptron Algorithm

Learning:

**Algorithm 1** Perceptron Learning Algorithm (Batch)

1: **procedure** \( \text{PERCEPTRON}(\mathcal{D} = \{(x^{(1)}, y^{(1)}), \ldots, (x^{(N)}, y^{(N)})\}) \)
2: \( \theta \leftarrow 0 \) \hfill \( \triangleright \) Initialize parameters
3: while not converged do
4: \hspace{1em} for \( i \in \{1, 2, \ldots, N\} \) do \hfill \( \triangleright \) For each example
5: \hspace{2em} \( \hat{y} \leftarrow \text{sign}(\theta^T x^{(i)}) \) \hfill \( \triangleright \) Predict
6: \hspace{2em} if \( \hat{y} \neq y^{(i)} \) then \hfill \( \triangleright \) If mistake
7: \hspace{3em} \( \theta \leftarrow \theta + y^{(i)} x^{(i)} \) \hfill \( \triangleright \) Update parameters
8: return \( \theta \)
Analysis: Perceptron

**Perceptron Mistake Bound**

**Guarantee:** If data has margin $\gamma$ and all points inside a ball of radius $R$, then Perceptron makes $\leq (R/\gamma)^2$ mistakes.

(Normalized margin: multiplying all points by 100, or dividing all points by 100, doesn’t change the number of mistakes; algo is invariant to scaling.)
Theorem 0.1 (Block (1962), Novikoff (1962)).
Given dataset: $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$.
Suppose:

1. Finite size inputs: $\|x^{(i)}\| \leq R$
2. Linearly separable data: $\exists \theta^* \text{ s.t. } \|\theta^*\| = 1$ and $y^{(i)}(\theta^* \cdot x^{(i)}) \geq \gamma, \forall i$

Then: The number of mistakes made by the Perceptron algorithm on this dataset is

$$k \leq (R/\gamma)^2$$
Proof of Perceptron Mistake Bound:

We will show that there exist constants $A$ and $B$ s.t.

$$A k \leq \|	heta^{(k+1)}\| \leq B \sqrt{k}$$
Theorem 0.1 (Block (1962), Novikoff (1962)).
Given dataset: $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^N$.
Suppose:
1. Finite size inputs: $\|x^{(i)}\| \leq R$
2. Linearly separable data: $\exists \theta^* \text{ s.t. } \|\theta^*\| = 1$ and $y^{(i)}(\theta^* \cdot x^{(i)}) \geq \gamma, \forall i$
Then: The number of mistakes made by the Perceptron algorithm on this dataset is

$$k \leq \left(\frac{R}{\gamma}\right)^2$$

Algorithm 1 Perceptron Learning Algorithm (Online)

1: procedure PERCEPTRON($\mathcal{D} = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots\}$)
2: $\theta \leftarrow 0, k = 1$ ▷ Initialize parameters
3: for $i \in \{1, 2, \ldots\}$ do ▷ For each example
4: if $y^{(i)}(\theta^{(k)} \cdot x^{(i)}) \leq 0$ then ▷ If mistake
5: $\theta^{(k+1)} \leftarrow \theta^{(k)} + y^{(i)}x^{(i)}$ ▷ Update parameters
6: $k \leftarrow k + 1$
7: return $\theta$
Analysis: Perceptron

Whiteboard:
Proof of Perceptron Mistake Bound
Analysis: Perceptron

Proof of Perceptron Mistake Bound:
Part 1: for some $A$, $A \kappa \leq \|\theta^{(k+1)}\|$

$$\theta^{(k+1)} \cdot \theta^* = (\theta^{(k)} + y^{(i)}x^{(i)})\theta^*$$
by Perceptron algorithm update

$$= \theta^{(k)} \cdot \theta^* + y^{(i)}(\theta^* \cdot x^{(i)})$$

$$\geq \theta^{(k)} \cdot \theta^* + \gamma$$
by assumption

$$\Rightarrow \theta^{(k+1)} \cdot \theta^* \geq k\gamma$$
by induction on $k$ since $\theta^{(1)} = 0$

$$\Rightarrow \|\theta^{(k+1)}\| \geq k\gamma$$
since $\|w\| \times \|u\| \geq w \cdot u$ and $\|\theta^*\| = 1$

Cauchy-Schwartz inequality
Proof of Perceptron Mistake Bound:
Part 2: for some B, \( \|\theta^{(k+1)}\| \leq B\sqrt{k} \)

\[
\|\theta^{(k+1)}\|^2 = \|\theta^{(k)} + y(i)x(i)\|^2
\]

by Perceptron algorithm update

\[
= \|\theta^{(k)}\|^2 + (y(i))^2\|x(i)\|^2 + 2y(i)(\theta^{(k)} \cdot x(i))
\]

\[
\leq \|\theta^{(k)}\|^2 + (y(i))^2\|x(i)\|^2
\]

since \( k \)th mistake \( \Rightarrow y(i)(\theta^{(k)} \cdot x(i)) \leq 0 \)

\[
= \|\theta^{(k)}\|^2 + R^2
\]

since \( (y(i))^2\|x(i)\|^2 = \|x(i)\|^2 = R^2 \) by assumption and \( (y(i))^2 = 1 \)

\[
\Rightarrow \|\theta^{(k+1)}\|^2 \leq kR^2
\]

by induction on \( k \) since \( (\theta^{(1)})^2 = 0 \)

\[
\Rightarrow \|\theta^{(k+1)}\| \leq \sqrt{k}R
\]
Proof of Perceptron Mistake Bound:
Part 3: Combining the bounds finishes the proof.

\[ k \gamma \leq \| \theta^{(k+1)} \| \leq \sqrt{k} R \]

\[ \Rightarrow k \leq \left( \frac{R}{\gamma} \right)^2 \]

The total number of mistakes must be less than this
Extensions of Perceptron

• **Kernel Perceptron**
  – Choose a kernel $K(x', x)$
  – Apply the **kernel trick** to Perceptron
  – Resulting algorithm is **still very simple**

• **Structured Perceptron**
  – Basic idea can also be applied when $y$ ranges over an exponentially large set
  – Mistake bound **does not** depend on the size of that set
Summary: Perceptron

• Perceptron is a simple **linear classifier**
• **Simple learning algorithm**: when a mistake is made, add / subtract the features
• For linearly separable and inseparable data, we can **bound the number of mistakes** (geometric argument)
• Extensions support nonlinear separators and structured prediction
RECALL: LOGISTIC REGRESSION
Using gradient ascent for linear classifiers

Key idea behind today’s lecture:

1. Define a linear classifier (logistic regression)
2. Define an objective function (likelihood)
3. Optimize it with gradient descent to learn parameters
4. Predict the class with highest probability under the model

Recall…
Using gradient ascent for linear classifiers

This decision function isn’t differentiable:

\[ h(x) = \text{sign}(\theta^T x) \]

Use a differentiable function instead:

\[ p_\theta(y = 1|x) = \frac{1}{1 + \exp(-\theta^T x)} \]

Recall…

[Graph showing the decision boundaries and logistic function]
Using gradient ascent for linear classifiers

This decision function isn’t differentiable:
\[ h(x) = \text{sign}(\theta^T x) \]

Use a differentiable function instead:
\[ p_\theta(y = 1|x) = \frac{1}{1 + \exp(-\theta^T x)} \]

Recall…

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

**Data:** Inputs are continuous vectors of length K. Outputs are discrete.

\[ D = \{ x^{(i)}, y^{(i)} \}_{i=1}^{N} \text{ where } x \in \mathbb{R}^{K} \text{ and } y \in \{0, 1\} \]

**Model:** Logistic function applied to dot product of parameters with input vector.

\[ p_{\theta}(y = 1|x) = \frac{1}{1 + \exp(-\theta^T x)} \]

**Learning:** finds the parameters that minimize some objective function.

\[ \theta^* = \arg\min_{\theta} J(\theta) \]

**Prediction:** Output is the most probable class.

\[ \hat{y} = \arg\max_{y \in \{0,1\}} p_{\theta}(y|x) \]

Recall…
NEURAL NETWORKS
Learning highly non-linear functions

\( f: X \rightarrow Y \)

- \( f \) might be non-linear function
- \( X \) (vector of) continuous and/or discrete vars
- \( Y \) (vector of) continuous and/or discrete vars

The XOR gate

Speech recognition
Perceptron and Neural Nets

- From biological neuron to artificial neuron (perceptron)

- Activation function

\[ X = \sum_{i=1}^{n} x_i w_i \]

\[ Y = \begin{cases} +1, & \text{if } X \geq \omega_0 \\ -1, & \text{if } X < \omega_0 \end{cases} \]

- Artificial neuron networks
  - supervised learning
  - gradient descent

© Eric Xing @ CMU, 2006-2011
Connectionist Models

- Consider humans:
  - Neuron switching time
    ~ 0.001 second
  - Number of neurons
    ~ $10^{10}$
  - Connections per neuron
    ~ $10^{4-5}$
  - Scene recognition time
    ~ 0.1 second
  - 100 inference steps doesn't seem like enough
    $\rightarrow$ much parallel computation

- Properties of artificial neural nets (ANN)
  - Many neuron-like threshold switching units
  - Many weighted interconnections among units
  - Highly parallel, distributed processes
Why is everyone talking about Deep Learning?

• Because a lot of money is invested in it…
  – DeepMind: Acquired by Google for $400 million
  – DNNResearch: Three person startup (including Geoff Hinton) acquired by Google for unknown price tag
  – Enlitic, Ersatz, MetaMind, Nervana, Skylab: Deep Learning startups commanding millions of VC dollars
• Because it made the front page of the New York Times
Deep learning:
- Has won numerous pattern recognition competitions
- Does so with minimal feature engineering

This wasn’t always the case!
Since 1980s: Form of models hasn’t changed much, but lots of new tricks...
- More hidden units
- Better (online) optimization
- New nonlinear functions (ReLUs)
- Faster computers (CPUs and GPUs)
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
Background

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

A Recipe for Machine Learning

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 the:
   – Decision function
     \[ \hat{y} = f_{\theta}(x_i) \]
   – Loss function
     \[ \ell(\hat{y}, y_i) \in \mathbb{R} \]

3. Define goal:

4. Train with SGD:
   \[ \theta^{(t+1)} = \theta^{(t)} - \eta_t \nabla \ell(f_{\theta}(x_i), y_i) \]

**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!
1. Given training data:
2. Define goal:
3. Choose each of these:
   - Decision function
   - Loss function
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
\[ y = h_{\theta}(\mathbf{x}) = \sigma(\theta^T \mathbf{x}) \]

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

Output

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

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

Input

\[ \begin{align*}
\theta_1 & \rightarrow x_1 \\
\theta_2 & \rightarrow x_2 \\
\theta_3 & \rightarrow x_3 \\
\theta_M & \rightarrow x_M
\end{align*} \]
Logistic Regression

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

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

Decision Functions

Output

Input

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

\[ x_1 \quad x_2 \quad x_3 \]

Face

Face

Not a face
\[ y = h_\theta(x) = \sigma(\theta^T x) \]\n
where \( \sigma(a) = \frac{1}{1 + \exp(-a)} \).
Logistic Regression

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

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

\[ y = \begin{pmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \vdots \\ \theta_M \end{pmatrix} \]

Output

Input

\[ x_1 \quad x_2 \quad x_3 \quad \cdots \quad x_M \]
“Combined logistic models”

<table>
<thead>
<tr>
<th>Independent variables</th>
<th>Weights</th>
<th>Hidden Layer</th>
<th>Weights</th>
<th>Dependent variable</th>
<th>Prediction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Age</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gender</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stage</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

Weights:
- Age: 0.6
- Gender: 0.1
- Stage: 0.7

Output: 0.6

“Probability of being Alive”
Inputs

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

Weights

- 34
- 2
- .3
- .2

Hidden Layer

Output

- "Probability of being Alive" = 0.6

Independent variables

Dependent variable

Prediction
**Independent variables**

- **Age**: 34
- **Gender**: 1
- **Stage**: 4

**Weights**

- Input to Hidden Layer:
  - Age: 0.6
  - Gender: 0.1
  - Stage: 0.3

**Hidden Layer**

- Hidden Layer Output: 0.5

**Weights**

- Output to Prediction:
  - Hidden Layer: 0.8

**Prediction**

- Probability of being Alive: 0.6

- Equation: \( \sum \cdot \)
Not really, no target for hidden units...

**Independent variables**
- **Age**: 34
- **Gender**: 2
- **Stage**: 4

**Weights**
- **Age**: 0.6
- **Gender**: 0.1
- **Stage**: 0.3

**Hidden Layer**
- Summation

**Dependent variable**
- **Weights**:
  - Stage to hidden layer: 0.5
  - Gender to hidden layer: 0.8

**Prediction**
- Probability of being alive: 0.6

“Probability of being alive”
**Jargon Pseudo-Correspondence**

- Independent variable = input variable
- Dependent variable = output variable
- Coefficients = “weights”
- Estimates = “targets”

**Logistic Regression Model (the sigmoid unit)**

```
  Inputs                      Output
  Age  34                      0.6
  Gender  1                    “Probability of beingAlive”
  Stage  4                      Σ

Independent variables: x1, x2, x3
Coefficients: a, b, c
Dependent variable: p Prediction
```
Neural Network

Decision Functions

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

Hidden Layer: $a_1, a_2, \ldots, a_D$

Output: $y$
Decision Functions

Neural Network

Given $x_i$, $\forall i$

$\text{(B) Hidden (linear)}$

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

$\text{(C) Hidden (sigmoid)}$

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

$\text{(D) Output (linear)}$

$$b = \sum_{j=0}^{D} \beta_j z_j$$

$\text{(E) Output (sigmoid)}$

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

Input

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

Hidden Layer

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

Output

$\mathbf{y}$
Building a Neural Net

Output

Features

\[ y \]

\[ x_1 \quad x_2 \quad \ldots \quad x_M \]
Building a Neural Net

Output

Hidden Layer

Input

$D = M$
Building a Neural Net

Output

Hidden Layer

Input

$D = M$
Building a Neural Net

Output

Hidden Layer

Input

\[ D = M \]
Building a Neural Net

\[ y \]

Output

Hidden Layer

\[ a_1 \]
\[ a_2 \]
\[ \ldots \]
\[ a_D \]

\[ D < M \]

Input

\[ x_1 \]
\[ x_2 \]
\[ x_3 \]
\[ \ldots \]
\[ x_M \]
• 0 hidden layers: linear classifier
  – Hyperplanes

Example from Eric Postma via Jason Eisner
• 1 hidden layer
  – Boundary of convex region (open or closed)

Example from Eric Postma via Jason Eisner
Decision Boundary

- 2 hidden layers
  - Combinations of convex regions

Example from Eric Postma via Jason Eisner
Decision Functions

Multi-Class Output

Output

Hidden Layer

Input
Deeper Networks

Next lecture:

Decision Functions
Deeper Networks

Next lecture:

Decision Functions
Next lecture: Making the neural networks deeper
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)
Decision Functions

Different Levels of Abstraction

Example from Honglak Lee (NIPS 2010)
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
**Activation Functions**

Neural Network with sigmoid activation functions

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

- **Hidden Layer**
  - $z_1, z_2, \ldots, z_D$
  - $a_j = \sum_{i=0}^{M} \alpha_{ji} x_i, \ \forall j$
  - $z_j = \frac{1}{1+\exp(-a_j)}, \ \forall j$

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

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

- **Output**
  - $y$

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

---

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

(B) Hidden (linear)

(C) Hidden (sigmoid)

(D) Output (linear)

(E) Output (sigmoid)

(F) Loss
Activation Functions

Neural Network with arbitrary nonlinear activation functions

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

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

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

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

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

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

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

Sigmoid / Logistic Function

\[
\text{logistic}(u) = \frac{1}{1 + e^{-u}}
\]
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 & Bengio (2010)
Activation Functions

- A new change: modifying the nonlinearity – ReLU often used in vision tasks

\[ \max(0, w \cdot x + b) \]

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

- **Regression:**
  - Use the same objective as Linear Regression
  - Quadratic loss (i.e. mean squared error)

- **Classification:**
  - Use the same objective as Logistic Regression
  - Cross-entropy (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>Objective</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>
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 \( x_i, \quad \forall i \)
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)
Background

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)
   \[ \theta^{(t+1)} = \theta^{(t)} - \eta_t \nabla \ell(f_\theta(x_i), y_i) \]
Objective Functions

Matching Quiz: Suppose you are given a neural net with a single output, y, and one hidden layer.

1) Minimizing sum of squared errors...
2) Minimizing sum of squared errors plus squared Euclidean norm of weights...
3) Minimizing cross-entropy...
4) Minimizing hinge loss...
5) ... MLE estimates of weights assuming target follows a Bernoulli with parameter given by the output value
6) ... MAP estimates of weights assuming weight priors are zero mean Gaussian
7) ... estimates with a large margin on the training data
8) ... MLE estimates of weights assuming zero mean Gaussian noise on the output value

A. 1=5, 2=7, 3=6, 4=8
B. 1=5, 2=7, 3=8, 4=6
C. 1=7, 2=5, 3=5, 4=7
D. 1=7, 2=5, 3=6, 4=8
E. 1=8, 2=6, 3=5, 4=7
F. 1=8, 2=6, 3=8, 4=6
BACKPROPAGATION
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)
   \[ \theta^{(t+1)} = \theta^{(t)} - \eta_t \nabla \ell(f_{\theta}(x_i), y_i) \]
• **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?
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.
Training

Chain Rule

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:
1. **Instantiate the computation as a directed acyclic graph**, where each intermediate quantity is a node.
2. At each node, store (a) the quantity computed in the forward pass and (b) the **partial derivative** of the goal with respect to that node’s intermediate quantity.
3. **Initialize** all partial derivatives to 0.
4. Visit each node in **reverse topological order**. At each node, add its contribution to the partial derivatives of its parents.

This algorithm is also called **automatic differentiation in the reverse-mode**.
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

\[
J = \cos(u) \\
u = u_1 + u_2 \\
u_1 = \sin(t) \\
u_2 = 3t \\
t = x^2
\]
**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.

<table>
<thead>
<tr>
<th>Forward</th>
<th>Backward</th>
</tr>
</thead>
<tbody>
<tr>
<td>$J = \cos(u)$</td>
<td>$\frac{dJ}{du} += -\sin(u)$</td>
</tr>
<tr>
<td>$u = u_1 + u_2$</td>
<td>$\frac{dJ}{du_1} += \frac{dJ}{du} \frac{du}{du_1}$, $\frac{du}{du_1} = 1$</td>
</tr>
<tr>
<td>$u_1 = \sin(t)$</td>
<td>$\frac{dJ}{du_1} += \frac{dJ}{du_1} \frac{du}{dt}$, $\frac{du_1}{dt} = \cos(t)$</td>
</tr>
<tr>
<td>$u_2 = 3t$</td>
<td>$\frac{dJ}{du_2} += \frac{dJ}{du_2} \frac{du_2}{dt}$, $\frac{du_2}{dt} = 3$</td>
</tr>
<tr>
<td>$t = x^2$</td>
<td>$\frac{dJ}{dt} += \frac{dJ}{dt} \frac{dt}{dx}$, $\frac{dt}{dx} = 2x$</td>
</tr>
</tbody>
</table>
Training

Backpropagation

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

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

\[ \frac{dJ}{da} = \frac{dJ}{dy} \frac{dy}{da}, \quad \frac{dy}{da} = \frac{\exp(-a)}{(\exp(-a) + 1)^2} \]

\[ \frac{dJ}{d\theta_j} = \frac{dJ}{da} \frac{da}{d\theta_j}, \quad \frac{da}{d\theta_j} = x_j \]

\[ \frac{dJ}{dx_j} = \frac{dJ}{da} \frac{da}{dx_j}, \quad \frac{da}{dx_j} = \theta_j \]
Backpropagation

\[ J = \frac{1}{2} (y - y^d)^2 \]

Output (sigmoid)
\[ y = \frac{1}{1 + \exp(-b)} \]

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

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

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

Input
Given \( x_i, \forall i \)

Training
Training

Backpropagation

(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 \( x_i, \, \forall i \)
Training

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

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

\[ \frac{dJ}{db} = \frac{dJ}{dy} \frac{dy}{db} \frac{db}{d\beta_j} \frac{d\beta_j}{dz_j} \frac{dz_j}{da_j} \frac{da_j}{dx_i} = \frac{\exp(-b)}{(\exp(-b) + 1)^2} \]

\[ \frac{dJ}{d\beta_j} = \frac{dJ}{db} \frac{db}{d\beta_j} \frac{d\beta_j}{dz_j} \frac{dz_j}{da_j} \frac{da_j}{dx_i} = z_j \]

\[ \frac{dJ}{dz_j} = \frac{dJ}{db} \frac{db}{dz_j} \frac{dz_j}{da_j} \frac{da_j}{dx_i} = \beta_j \]

\[ \frac{dJ}{da_j} = \frac{dJ}{dz_j} \frac{dz_j}{da_j} \frac{da_j}{dx_i} = \frac{\exp(-a_j)}{(\exp(-a_j) + 1)^2} \]

\[ \frac{dJ}{d\alpha_{ji}} = \frac{dJ}{da_j} \frac{da_j}{d\alpha_{ji}} \frac{d\alpha_{ji}}{dx_i} = x_i \]

\[ \frac{dJ}{dx_i} = \frac{dJ}{da_j} \frac{da_j}{dx_i} \frac{dx_i}{\alpha_{ji}} \sum_{j=0}^{D} \alpha_{ji} \]
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:**

1. **Instantiate the computation as a directed acyclic graph**, where each intermediate quantity is a node.
2. At each node, store (a) the quantity computed in the forward pass and (b) the **partial derivative** of the goal with respect to that node’s intermediate quantity.
3. **Initialize** all partial derivatives to 0.
4. Visit each node in **reverse topological order**. At each node, add its contribution to the partial derivatives of its parents.

This algorithm is also called **automatic differentiation in the reverse-mode**.
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:**
1. **Instantiate the computation as a directed acyclic graph**, where each node represents a Tensor.
2. At each node, store (a) the quantity computed in the forward pass and (b) the **partial derivatives** of the goal with respect to that node’s Tensor.
3. **Initialize** all partial derivatives to 0.
4. Visit each node in **reverse topological order**. At each node, add its contribution to the partial derivatives of its parents.

This algorithm is also called **automatic differentiation in the reverse-mode**.
### Case 2: Training vs. Backpropagation

<table>
<thead>
<tr>
<th>Module</th>
<th>Forward</th>
<th>Backward</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module 1</td>
<td>$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i$</td>
<td>$\frac{dJ}{d\alpha_{ji}} = \frac{dJ}{da_j} \frac{da_j}{d\alpha_{ji}}, \frac{da_j}{d\alpha_{ji}} = x_i$</td>
</tr>
<tr>
<td>Module 2</td>
<td>$z_j = \frac{1}{1 + \exp(-a_j)}$</td>
<td>$\frac{dJ}{da_j} = \frac{dJ}{dz_j} \frac{dz_j}{da_j}, \frac{dz_j}{da_j} = \beta_j$</td>
</tr>
<tr>
<td>Module 3</td>
<td>$b = \sum_{j=0}^{D} \beta_j z_j$</td>
<td>$\frac{dJ}{db} = \frac{dJ}{dy} \frac{dy}{db}$, $\frac{dy}{db} = \frac{\exp(-b)}{(\exp(-b) + 1)^2}$</td>
</tr>
<tr>
<td>Module 4</td>
<td>$y = \frac{1}{1 + \exp(-b)}$</td>
<td>$\frac{dJ}{db} = \frac{dJ}{dy} \frac{dy}{db}$, $\frac{dy}{db} = \frac{\exp(-b)}{(\exp(-b) + 1)^2}$</td>
</tr>
<tr>
<td>Module 5</td>
<td>$J = y^* \log y + (1 - y^*) \log(1 - y)$</td>
<td>$\frac{dJ}{dy} = \frac{y^<em>}{y} + \frac{(1 - y^</em>)}{y - 1}$</td>
</tr>
</tbody>
</table>
1. Given training data:
   \[ \{x_i, y_i\}_{i=1}^N \]

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

3. Define goal:

4. Train with SGD:
   \( \theta^{(t+1)} = \theta^{(t)} - \eta_t \nabla \ell(f_\theta(x_i), y_i) \)

**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!
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