Copyright Notice

These slides are distributed under the Creative Commons License.

DeepLearning.AI makes these slides available for educational purposes. You may not use or distribute these slides for commercial purposes. You may make copies of these slides and use or distribute them for educational purposes as long as you cite DeepLearning.AI as the source of the slides.

For the rest of the details of the license, see https://creativecommons.org/licenses/by-sa/2.0/legalcode
Basics of Neural Network Programming

Binary Classification
Binary Classification

\[ X = \begin{bmatrix} 255 \\ 231 \\ \vdots \\ 255 \\ 134 \end{bmatrix} \]

\[ n = n_X = 12288 \]

\[ X \rightarrow y \]
Notation

\((x, y) \in \mathbb{R}^{n_x}, y \in \{0, 1\}\)

\(m\) training examples: \[\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}\]

\(M = M_{\text{train}}\quad M_{\text{test}} = \#\text{test examples.}\)

\(X = \begin{bmatrix} x^{(1)} & x^{(2)} & \ldots & x^{(m)} \end{bmatrix}^{\top} \in \mathbb{R}^{m \times n_x}

\)

\(Y = \begin{bmatrix} y^{(1)} & y^{(2)} & \ldots & y^{(m)} \end{bmatrix} \in \mathbb{R}^{1 \times m}

\)

\(Y: \text{shape} = (1, m)\)

Andrew Ng
Basics of Neural Network Programming

Logistic Regression

deeplearning.ai
Logistic Regression

Given $x$, want $\hat{y} = P(y=1 \mid x)$
$x \in \mathbb{R}^n$

Parameters: $w \in \mathbb{R}^n$, $b \in \mathbb{R}$

Output $\hat{y} = \sigma(w^T x + b)$

$\sigma(z) = \frac{1}{1+e^{-z}}$

If $z$ large, $\sigma(z) \approx \frac{1}{1+0} = 1$
If $z$ large negative number
$\sigma(z) = \frac{1}{1+e^{-z}} \approx \frac{1}{1+Bignum} \approx 0$

$x_0 = 1$, $x \in \mathbb{R}^{n+1}$
$\hat{y} = \sigma(\theta^T x)$

$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix}$

Andrew Ng
Basics of Neural Network Programming

Logistic Regression

cost function
Logistic Regression cost function

\[ \hat{y}^{(i)} = \sigma(w^T x^{(i)} + b), \text{ where } \sigma(z^{(i)}) = \frac{1}{1 + e^{-z^{(i)}}} \]

Given \( \{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\} \), want \( \hat{y}^{(i)} \approx y^{(i)} \).

Loss (error) function:

\[
L(\hat{y}, y) = \frac{1}{2} (\hat{y} - y)^2
\]

If \( y = 1 \):

\[ L(\hat{y}, y) = - \log \hat{y} \leq \text{want } \log \hat{y} \text{ large, want } \hat{y} \text{ large} \]

If \( y = 0 \):

\[ L(\hat{y}, y) = - \log (1 - \hat{y}) \leq \text{want } \log (1 - \hat{y}) \text{ large, want } \hat{y} \text{ small} \]

Cost function:

\[ J(w, b) = \frac{1}{m} \sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) = \frac{1}{m} \sum_{i=1}^{m} \left[ y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log (1 - \hat{y}^{(i)}) \right] \]
Basics of Neural Network Programming

Gradient Descent
Gradient Descent

Recap: \( \hat{y} = \sigma(w^T x + b), \sigma(z) = \frac{1}{1+e^{-z}} \leftarrow \)

\[
J(w, b) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m} \sum_{i=1}^{m} y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})
\]

Want to find \( w, b \) that minimize \( J(w, b) \)
Gradient Descent

Repeat

$w_i := w_i - \alpha \frac{\partial J(w, b)}{\partial w_i}$

with learning rate $\alpha$ and $\frac{\partial J(w, b)}{\partial w}$ is the derivative of the cost function with respect to the weight $w_i$.

$J(w, b)$

$w := w - \alpha \frac{\partial J(w, b)}{\partial w}$

$b := b - \alpha \frac{\partial J(w, b)}{\partial b}$

and $\frac{\partial J(w, b)}{\partial b}$ is the derivative of the cost function with respect to the bias $b$.

Andrew Ng
Basics of Neural Network Programming

Derivatives

deeplearning.ai
Intuition about derivatives

$$f(a) = 3a$$

- $a = 2$  
  $f(a) = 6$
  
  $a = 2.001$  
  $f(a) = 6.003$

- Slope (derivative) of $f(a)$ at $a = 2$ is $3$

- $a = 5$  
  $f(a) = 15$
  
  $a = 5.001$  
  $f(a) = 15.003$

- Slope at $a = 5$ is also $3$

$$\frac{df(a)}{da} = 3 = \lim_{a \to 0} \frac{f(a)}{a}$$

Andrew Ng
Basics of Neural Network Programming

More derivatives examples
Intuition about derivatives

\[ f(a) = a^2 \]

\[ \frac{d}{da} a^2 = 2a \]

\[ a = 2 \quad f(a) = 4 \]
\[ a = 2.001 \quad f(a) \approx 4.004 \]
\[ \text{slope (derivative) of } f(a) \text{ at } a = 2 \quad \frac{d}{da} f(a) = 4 \quad \text{when } a = 2 \]
\[ a = 5 \quad f(a) = 25 \]
\[ a = 5.001 \quad f(a) \approx 25.010 \]
\[ \frac{d}{da} f(a) = 10 \quad \text{when } a = 5 \]

Andrew Ng
More derivative examples

\[ f(a) = a^2 \]
\[ \frac{d}{da} f(a) = 2a \]
\[ a = 2 \quad f(a) = 4 \]
\[ a = 2.001 \quad f(a) \approx 4.004 \]

\[ f(a) = a^3 \]
\[ \frac{d}{da} f(a) = 3a^2 \]
\[ \frac{3 \times 2^2}{2 + 2^2} = 12 \]
\[ a = 2 \quad f(a) = 8 \]
\[ a = 2.001 \quad f(a) \approx 8.012 \]

\[ f(a) = \log_e(a) \]
\[ \frac{d}{da} f(a) = \frac{1}{a} \]
\[ a = 2 \quad f(a) = 0.69315 \]
\[ a = 2.001 \quad f(a) \approx 0.69365 \]

[Graph showing \( \ln(a) \) and \( \frac{d}{da} \ln(a) \)]
Basics of Neural Network Programming

Computation Graph

deeplearning.ai
Computation Graph

\[ J(a, b, c) = 3(a + bc) = 3(5 + 3 \times 2) = 33 \]

\[ u = bc \]
\[ v = a + u \]
\[ J = 3v \]
Basics of Neural Network Programming

Derivatives with a Computation Graph
Computing derivatives

\[ a = 5 \]
\[ b = 3 \]
\[ c = 2 \]

\[ u = bc \]

\[ v = a + u \]

\[ f(a) = 3a \]

\[ \frac{df}{da} = \frac{df}{dv} \]

\[ \frac{dj}{dv} = 3 \]

\[ \frac{dj}{da} = 3 = \frac{dj}{dv} \frac{dv}{da} \]

\[ \frac{dv}{da} = 1 \]

\[ d\text{Final Output} = \frac{dJ}{d\text{Var}} \\
\text{"dvar"} \]

\[ J = 3v \]

\[ v = 11 \rightarrow 11.001 \]
\[ J = 33 \rightarrow 33.003 \]

\[ a = 5 \rightarrow 5.001 \]
\[ v = 11 \rightarrow 11.001 \]
\[ J = 33 \rightarrow 33.003 \]
Computing derivatives

\[ \frac{dJ}{da} = 3 \]
\[ \frac{dJ}{db} = 6 \]
\[ \frac{dJ}{dc} = 9 \]

\[ a = 5 \]
\[ b = 3 \]
\[ c = 2 \]

\[ u = bc \]
\[ \frac{du}{da} = 3 \]
\[ \frac{du}{db} = 6 \]
\[ \frac{du}{dc} = 9 \]

\[ v = a + u \]
\[ \frac{dv}{du} = 3 \]
\[ \frac{dv}{da} = 1 \]
\[ \frac{dv}{db} = 0 \]
\[ \frac{dv}{dc} = 0 \]

\[ J = 3v \]

\[ J = 33 \]
\[ J = 33.006 \]

\[ u = 6 \rightarrow 6.001 \]
\[ v = 11 \rightarrow 11.001 \]
\[ J = 33 \rightarrow 33.003 \]

\[ b = 3 \rightarrow 3.001 \]
\[ u = b \cdot c = 6 \rightarrow 6.002 \]
\[ J = 33.006 \]

\[ v = 11.002 \]
\[ J = 33 \]
Basics of Neural Network Programming

Logistic Regression

Gradient descent

deeplearning.ai
Logistic regression recap

\[ z = w^T x + b \]
\[ \hat{y} = a = \sigma(z) \]
\[ \mathcal{L}(a, y) = -(y \log(a) + (1 - y) \log(1 - a)) \]
Logistic regression derivatives

\[ z = w_1 x_1 + w_2 x_2 + b \]

\[ a = \sigma(z) \]

\[ \mathcal{L}(a, y) \]

\[ \frac{\partial}{\partial w_1} = x_1 \cdot dz \]

\[ \frac{\partial}{\partial w_2} = x_2 \cdot dz \]

\[ \frac{\partial}{\partial b} = dz \]

\[ w_1 := w_1 - \alpha \cdot \frac{\partial \mathcal{L}}{\partial w_1} \]

\[ w_2 := w_2 - \alpha \cdot \frac{\partial \mathcal{L}}{\partial w_2} \]

\[ b := b - \alpha \cdot \frac{\partial \mathcal{L}}{\partial b} \]
Basics of Neural Network Programming

Gradient descent on $m$ examples
Logistic regression on \( m \) examples

\[
J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \ell(a_i, y_i)
\]

\[
\Rightarrow a_i = \hat{y}_i = \sigma(z_i) = \sigma(\theta^T x_i + b)
\]

\[
\frac{\partial}{\partial \theta_j} J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \frac{\partial}{\partial \theta_j} \ell(a_i, y_i)
\]

\[
\theta_j = (x_i^T, y_i)
\]
Logistic regression on $m$ examples

$J = 0; \frac{\partial J}{\partial w_1} = 0; \frac{\partial J}{\partial w_2} = 0; \frac{\partial J}{\partial b} = 0$

$\rightarrow \text{For } i = 1 \text{ to } m$

$z^{(i)} = \omega^T x^{(i)} + b$

$a^{(i)} = \sigma(z^{(i)})$

$J_t = -[y^{(i)} \log a^{(i)} + (1 - y^{(i)}) \log (1 - a^{(i)})]$

$\frac{\partial z^{(i)}}{\partial a^{(i)}} = a^{(i)} - y^{(i)}$

$\frac{\partial J}{\partial w_1} = x^{(i)} z^{(i)}$

$\frac{\partial J}{\partial w_2} = x^{(i)} z^{(i)}$

$\frac{\partial J}{\partial b} = z^{(i)}$

$J/t = m \leq \frac{\partial J}{\partial w_1} = m; \frac{\partial J}{\partial w_2} = m; \frac{\partial J}{\partial b} = m.$

$\frac{\partial w_1}{\partial J} = \frac{2J}{\partial w_1},$

$w_1 := w_1 - \alpha \frac{\partial w_1}{\partial J},$

$w_2 := w_2 - \alpha \frac{\partial w_2}{\partial J},$

$b := b - \alpha \frac{\partial b}{\partial J}.$

$\text{Vectorization}$
Basics of Neural Network Programming

Vectorization

deeplearning.ai
What is vectorization?

\[ z = \mathbf{w}^T \mathbf{x} + b \]

Non-vectorized:

\[ z = 0 \]

for \( i \) in range \((n-x)\):

\[ z += \mathbf{w}[i,j] \times x[i,j] \]

\[ z += b \]

Vectorized:

\[ z = \text{np.dot} (\mathbf{w}, \mathbf{x}) + b \]

\[ \mathbf{w} \in \mathbb{R}^{n \times x} \]

\[ \mathbf{x} \in \mathbb{R}^{x} \]

\[ \mathbf{w}^T \mathbf{x} \]

\[ \rightarrow \text{GPU} \]

SIMD - single instruction

\[ \rightarrow \text{CPU} \]

multiple data.
Basics of Neural Network Programming
More vectorization examples
Neural network programming guideline

Whenever possible, avoid explicit for-loops.

\[ u = A v \]

\[ u_i = \sum_j A_{ij} v_j \]

\[ u = \text{np.zeros}((n, 1)) \]

```python
for i in ...:
    for j in ...:
```

\[ u = \text{np.dot}(A, v) \]
Vectors and matrix valued functions

Say you need to apply the exponential operation on every element of a matrix/vector.

\[ v = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix} \Rightarrow u = \begin{bmatrix} e^{v_1} \\ \vdots \\ e^{v_n} \end{bmatrix} \]

\[ u = \text{np.zeros}((n,1)) \]

for \( i \) in range(n):

\[ u[i] = \text{math.exp}(v[i]) \]

import numpy as np

\[ u = \text{np.exp}(u) \]

\[ \text{np.log}(u) \]

\[ \text{np.abs}(u) \]

\[ \text{np.maximum}(u, 0) \]

\[ v^n \]

\[ 1/v \]
Logistic regression derivatives

\[ J = 0, \; dw_1 = 0, \; dw_2 = 0, \; db = 0 \]

\[ \text{for } i = 1 \text{ to } n:\]
\[ z^{(i)} = w^T x^{(i)} + b \]
\[ a^{(i)} = \sigma(z^{(i)}) \]
\[ J += -[y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})] \]
\[ dz^{(i)} = a^{(i)}(1 - a^{(i)}) \]
\[ dw_1 += x^{(i)} dz^{(i)} \]
\[ dw_2 += x^{(i)} dz^{(i)} \]
\[ db += dz^{(i)} \]

\[ J = J/m, \; dw_1 = dw_1/m, \; dw_2 = dw_2/m, \; db = db/m \]

\[ dw /= m. \]
Basics of Neural Network Programming

Vectorizing Logistic Regression
Vectorizing Logistic Regression

\[
\begin{align*}
  z^{(1)} &= w^T x^{(1)} + b \\
  a^{(1)} &= \sigma(z^{(1)})
\end{align*}
\]

\[
\begin{align*}
  z^{(2)} &= w^T x^{(2)} + b \\
  a^{(2)} &= \sigma(z^{(2)})
\end{align*}
\]

\[
\begin{align*}
  z^{(3)} &= w^T x^{(3)} + b \\
  a^{(3)} &= \sigma(z^{(3)})
\end{align*}
\]
Basics of Neural Network Programming

Vectorizing Logistic Regression’s Gradient Computation
Vectorizing Logistic Regression

\[
\begin{align*}
    d_z^{(i)} &= a^{(i)} - y^{(i)} \\
    d_z^{(i)} &= a^{(i)} - y^{(i)} \\
    \vdots \\
    d_z &= \begin{bmatrix} d_z^{(1)} & d_z^{(2)} & \ldots & d_z^{(m)} \end{bmatrix}_{1 \times m} \\
    A &= \begin{bmatrix} a^{(1)} & \ldots & a^{(m)} \end{bmatrix} \\
    Y &= \begin{bmatrix} y^{(1)} & \ldots & y^{(m)} \end{bmatrix} \\
    d_z &= A - Y = \begin{bmatrix} a^{(1)} - y^{(1)} & a^{(2)} - y^{(2)} & \ldots \end{bmatrix} \\
    d_w &= 0 \\
    d_w &= \frac{1}{m} \sum_{i=1}^{m} d_z^{(i)} \\
    dw &= \frac{1}{m} X d_z \\
    \vdots \\
    dw &= \frac{1}{m} \begin{bmatrix} x^{(1)} d_z^{(1)} & \ldots & x^{(m)} d_z^{(m)} \end{bmatrix}_{n \times 1} \\
    \text{db} &= 0 \\
    \text{db} &= \frac{1}{m} \sum_{i=1}^{m} d_z^{(i)} \\
    db &= \begin{bmatrix} d_z^{(1)} & \ldots & d_z^{(m)} \end{bmatrix}_{1 \times m} \\
    db &= \frac{1}{m} \begin{bmatrix} x^{(1)} d_z^{(1)} & \ldots & x^{(m)} d_z^{(m)} \end{bmatrix}_{n \times 1} \\
\end{align*}
\]
Implementing Logistic Regression

\[ J = 0, \ dw_1 = 0, \ dw_2 = 0, \ db = 0 \]

for i = 1 to m:
\[ z^{(i)} = w^T x^{(i)} + b \]
\[ a^{(i)} = \sigma(z^{(i)}) \]
\[ J += -[y^{(i)} \log a^{(i)} + (1 - y^{(i)}) \log(1 - a^{(i))}] \]
\[ dz^{(i)} = a^{(i)} - y^{(i)} \]
\[
\begin{align*}
\begin{cases}
  dw_1 += x_1^{(i)} dz^{(i)} \\
  dw_2 += x_2^{(i)} dz^{(i)}
\end{cases}
\end{align*}
\]
\[ db += dz^{(i)} \]
\[ J = J/m, \ dw_1 = dw_1/m, \ dw_2 = dw_2/m \]
\[ db = db/m \]

for iter in range(1000):
\[
\begin{align*}
  z &= w^T x + b \\
  A &= \sigma(z) \\
  dz &= A - y \\
  dw &= \frac{1}{m} X \cdot dz^T \\
  db &= \frac{1}{m} \text{np.sum}(dz) \\
  w &= w - \alpha dw \\
  b &= b - \alpha db
\end{align*}
\]
Basics of Neural Network Programming

Broadcasting in Python

deeplearning.ai
**Broadcasting example**

**Calories from Carbs, Proteins, Fats in 100g of different foods:**

<table>
<thead>
<tr>
<th></th>
<th>Apples</th>
<th>Beef</th>
<th>Eggs</th>
<th>Potatoes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Carb</td>
<td>56.0</td>
<td>0.0</td>
<td>4.4</td>
<td>68.0</td>
</tr>
<tr>
<td>Protein</td>
<td>1.2</td>
<td>104.0</td>
<td>52.0</td>
<td>8.0</td>
</tr>
<tr>
<td>Fat</td>
<td>1.8</td>
<td>135.0</td>
<td>99.0</td>
<td>0.9</td>
</tr>
</tbody>
</table>

Calculate % of calories from Carb, Protein, Fat. Can you do this without explicit for-loop?

```
cal = A.sum(axis = 0)  # broadcasting
percentage = 100*A/(cal.reshape(1,4)) / (1,4)
```
Broadcasting example

\[
\begin{bmatrix}
1 \\
2 \\
3 \\
4
\end{bmatrix}
+ \begin{bmatrix}
100 \\
100 \\
100 \\
100
\end{bmatrix}
= 100
\]

\[
\begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6
\end{bmatrix}
\begin{pmatrix}
(m, n) \\
(1, n) \\
(m, 1) \\
(m, n)
\end{pmatrix}
\begin{bmatrix}
100 & 200 & 300 \\
100 & 200 & 300
\end{bmatrix}
= \begin{bmatrix}
1000 & 1000 \\
2000 & 2000
\end{bmatrix}
\]
General Principle

\[ (m, n) + (1, n) \Rightarrow (m, n) \]

\[ (m, 1) + \mathbb{R} \]

\[
\begin{bmatrix}
\frac{1}{2} \\
\frac{1}{2}
\end{bmatrix} + 100 = \begin{bmatrix}
\frac{101}{2} \\
\frac{102}{2}
\end{bmatrix}
\]

\[
\begin{bmatrix}
1 & 2 & 3
\end{bmatrix} + 100 = \begin{bmatrix}
101 & 102 & 103
\end{bmatrix}
\]

Matlab/Octave: \texttt{bsxfun}
Basics of Neural Network Programming

Explanation of logistic regression cost function (Optional)
Logistic regression cost function

\[ \hat{y} = \sigma(\omega^T \mathbf{x} + b) \quad \text{where} \quad \sigma(z) = \frac{1}{1 + e^{-z}} \]

Interpret \( \hat{y} = \mathbb{P}(y=1 \mid \mathbf{x}) \)

If \( y=1 \) : \quad \mathbb{P}(y=1 \mid \mathbf{x}) = \hat{y}

If \( y=0 \) : \quad \mathbb{P}(y=0 \mid \mathbf{x}) = 1 - \hat{y} \]
Logistic regression cost function

\[
p(y|x) = \begin{cases} 
  \hat{y}^y (1-\hat{y})^{(1-y)} & \text{if } y = 1 \\
  \hat{y}^0 (1-\hat{y})^{(1-y)} & \text{if } y = 0 
\end{cases}
\]

\[
\log p(y|x) = \begin{cases} 
  y \log \hat{y} + (1-y) \log(1-\hat{y}) & \text{if } y = 1 \\
  0 & \text{if } y = 0 
\end{cases}
\]
Cost on $m$ examples

\[ \log p(\text{labels in training set}) = \log \prod_{i=1}^{m} p(y(i) | x(i)) \leftarrow \]

\[\log p(\ldots) = \sum_{i=1}^{m} \log p(y(i) | x(i)) \]

\[ - L(y(i), y(i)) \]

\[ = - \sum_{i=1}^{m} L(y(i), y(i)) \]

Cost: \[ J(w, b) = \frac{1}{m} \sum_{i=1}^{m} L(y(i), y(i)) \]

(Minimize)