Logistic Regression

- **Goal:** model the probability of a random variable $Y$ being 0 or 1 given experimental data.

- **Consider a generalized linear model function parameterized by $\theta$,**
  
  $$h_\theta(x) = \frac{1}{1 + e^{-\theta^\top x}}$$

- **Attempt to model the probability that $y$ is 0 or 1 with function**
  
  $$\Pr(y|x; \theta) = h_\theta(x)^y(1 - h_\theta(x))^{1-y}$$
Logistic Regression

- The likelihood assuming all the samples are independent
  
  \[ L(\theta|x) = \Pr(Y|X; \theta) = \prod_i \Pr(y_i|x_i; \theta) = \prod_i h_{\theta}(x_i)^{y_i}(1 - h_{\theta}(x_i))^{1-y_i} \]

- Maximum likelihood
  
  \[
  \max_{\theta} L(\theta|x) = \max_{\theta} \prod_i h_{\theta}(x_i)^{y_i}(1 - h_{\theta}(x_i))^{1-y_i}
  \]
Neural Network with Softmax Classifier

- Softmax Classifier: multinomial Logistic Regression, the number of classes more than 2.

- Score: Instead of linear function as the exponent, we use a nonlinear function (e.g., a neural network $s = f(x_i; W)$)
## Softmax Classifier (Multinomial Logistic Regression)

<table>
<thead>
<tr>
<th></th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>3.2</td>
</tr>
<tr>
<td>car</td>
<td>5.1</td>
</tr>
<tr>
<td>frog</td>
<td>-1.7</td>
</tr>
</tbody>
</table>

**Slide Credit:** Fei-Fei Li & Justin Johnson & Serena Yeung
Softmax Classifier (Multinomial Logistic Regression)

scores = unnormalized log probabilities of the classes.

\[ s = f(x_i; W) \]

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>3.2</td>
</tr>
<tr>
<td>car</td>
<td>5.1</td>
</tr>
<tr>
<td>frog</td>
<td>-1.7</td>
</tr>
</tbody>
</table>
Softmax Classifier (Multinomial Logistic Regression)

scores = unnormalized log probabilities of the classes.

\[ P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}} \]

where

\[ s = f(x_i; W) \]

- cat: 3.2
- car: 5.1
- frog: -1.7
Softmax Classifier (Multinomial Logistic Regression)

**Softmax Classifier** (Multinomial Logistic Regression)

\[ P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}} \quad \text{where} \quad s = f(x_i; W) \]

scores = unnormalized log probabilities of the classes.

- **cat**: 3.2
- **car**: 5.1
- **frog**: -1.7

Softmax function

Slide Credit: Fei-Fei Li & Justin Johnson & Serena Yeung
Softmax Classifier (Multinomial Logistic Regression)

scores = unnormalized log probabilities of the classes.

\[ P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}} \]

where

\[ s = f(x_i; W) \]

Want to maximize the log likelihood, or (for a loss function) to minimize the negative log likelihood of the correct class:

\[ L_i = -\log P(Y = y_i | X = x_i) \]

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>3.2</td>
</tr>
<tr>
<td>car</td>
<td>5.1</td>
</tr>
<tr>
<td>frog</td>
<td>-1.7</td>
</tr>
</tbody>
</table>

Slide Credit: Fei-Fei Li & Justin Johnson & Serena Yeung
Softmax Classifier (Multinomial Logistic Regression)

scores = unnormalized log probabilities of the classes.

\[ P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}} \]

where \( s = f(x_i; W) \)

Want to maximize the log likelihood, or (for a loss function) to minimize the negative log likelihood of the correct class:

\[ L_i = - \log P(Y = y_i | X = x_i) \]

In summary:

\[ L_i = - \log \left( \frac{e^{s_{y_i}}}{\sum_j e^{s_j}} \right) \]
Softmax Classifier (Multinomial Logistic Regression)

\[ L_i = - \log \left( \frac{e^{s_{y_i}}}{\sum_j e^{s_j}} \right) \]

- **cat**: 3.2
- **car**: 5.1
- **frog**: -1.7

Unnormalized log probabilities

Slide Credit: Fei-Fei Li & Justin Johnson & Serena Yeung
Softmax Classifier (Multinomial Logistic Regression)

\[ L_i = - \log\left( \frac{e^{s_{y_i}}}{\sum_j e^{s_j}} \right) \]

Slide Credit: Fei-Fei Li & Justin Johnson & Serena Yeung
Softmax Classifier (Multinomial Logistic Regression)

\[ L_i = - \log \left( \frac{e^{s_{y_i}}}{\sum_j e^{s_j}} \right) \]

Slide Credit: Fei-Fei Li & Justin Johnson & Serena Yeung
Softmax Classifier (Multinomial Logistic Regression)

\[ L_i = -\log\left(\frac{e^{s_{yi}}}{\sum_j e^{s_j}}\right) \]

unnormalized probabilities

<table>
<thead>
<tr>
<th></th>
<th>Exp</th>
<th>Normalize</th>
<th>Probabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>3.2</td>
<td></td>
<td>0.13</td>
</tr>
<tr>
<td>car</td>
<td>5.1</td>
<td></td>
<td>0.87</td>
</tr>
<tr>
<td>frog</td>
<td>-1.7</td>
<td></td>
<td>0.00</td>
</tr>
</tbody>
</table>

\[ L_i = -\log(0.13) = 0.89 \]

Slide Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

UNIVERSITY OF MARYLAND

FEARLESS IDEAS
Softmax Classifier (Multinomial Logistic Regression)

\[ L_i = -\log \left( \frac{e^{s_{yi}}}{\sum_j e^{s_{yj}}} \right) \]

Q: What is the min/max possible loss \( L_i \)?

Unnormalized probabilities

- cat: 3.2
- car: 5.1
- frog: -1.7

Unnormalized log probabilities

- cat: 24.5
- car: 164.0
- frog: 0.18

Probabilities

- cat: 0.13
- car: 0.87
- frog: 0.00

\[ L_i = -\log(0.13) = 0.89 \]

Slide Credit: Fei-Fei Li & Justin Johnson & Serena Yeung
Softmax Classifier (Multinomial Logistic Regression)

$$L_i = - \log \left( \frac{e^{s_{y_i}}}{\sum_j e^{s_j}} \right)$$

Q2: Usually at initialization $W$ is small so all $s \approx 0$. What is the loss?

<table>
<thead>
<tr>
<th></th>
<th>$s_{y_i}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>3.2</td>
</tr>
<tr>
<td>car</td>
<td>5.1</td>
</tr>
<tr>
<td>frog</td>
<td>-1.7</td>
</tr>
</tbody>
</table>

Unnormalized log probabilities:
- cat: 24.5
- car: 164.0
- frog: 0.18

Normalize:
- cat: 0.13
- car: 0.87
- frog: 0.00

$L_i = -\log(0.13) = 0.89$

Slide Credit: Fei-Fei Li & Justin Johnson & Serena Yeung
Softmax Classifier (Multinomial Logistic Regression)

Recap

- We have some dataset of \((x, y)\)
- We have a score function: \(s = f(x; W) = Wx\)
- We have a loss function:

\[
L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{sj}}\right)
\]

\[
L = \frac{1}{N} \sum_{i=1}^{N} L_i + R(W) \quad \text{Full loss}
\]

Slide Credit: Fei-Fei Li & Justin Johnson & Serena Yeung
Softmax Classifier (Multinomial Logistic Regression)

Recap

- We have some dataset of $(x,y)$
- We have a score function: $s = f(x; W) = W x$
- We have a loss function:

$$L_i = - \log \left( \frac{e^{s y_i}}{\sum_j e^{s j}} \right)$$

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i + R(W) \quad \text{Full loss}$$

How do we find the best $W$?

Slide Credit: Fei-Fei Li & Justin Johnson & Serena Yeung
Multi-Layer: Backpropagation

\[
\frac{\partial E}{\partial z_j} = \frac{\partial E}{\partial \hat{y}_j} \frac{d\hat{y}_j}{dz_j}
\]

\[
\frac{\partial E}{\partial \hat{y}_i} = \sum_j \frac{\partial E}{\partial z_j} \frac{d\hat{y}_i}{d\hat{y}_j} = \sum_j w_{ij} \frac{\partial E}{\partial z_j} = \sum_j w_{ij} \frac{\partial E}{\partial \hat{y}_j} \frac{d\hat{y}_j}{dz_j}
\]

\[
\frac{\partial E}{\partial w_{ki}} = \sum_n \frac{\partial E}{\partial \hat{y}_i^n} \frac{d\hat{y}_i^n}{dz_i^n} \frac{dz_i^n}{\partial w_{ki}} = \sum_n \frac{d\hat{y}_i^n}{dz_i^n} \frac{\partial z_i^n}{\partial w_{ki}} \sum_j w_{ij} \frac{\partial E}{\partial \hat{y}_j^n} \frac{d\hat{y}_j^n}{dz_j^n}
\]

Slide credit: Bohyung Han
Back Propagation Revisited

- On board (gradient w.r.t elements): Please take notes!
- Vector format of the backpropagation on slides
Neural Network: Definition

Consider a neural network defined as following:

\[ \text{loss} = L(Y, \tilde{Y}) \]
\[ \tilde{Y} = W_n \sigma (W_{n-1} \sigma (\cdots \sigma (W_1 X + b_1 1^T) \cdots ) + b_{n-1} 1^T) + b_n 1^T \]
Neural Network: Forward Pass

Forward pass  The forward pass of the neural network can be done as the following. We first initialize $D_1$:

$$D_1 = X$$

Then we can iteratively calculate:

$$\begin{cases} 
E_k = W_{k-1}D_{k-1} + b_{k-1}1^T \\
D_k = \sigma(E_k) = \sigma(W_{k-1}D_{k-1} + b_{k-1}1^T) \quad \forall k = 2, \cdots, n 
\end{cases}$$

Finally,

$$\tilde{Y} = W_nD_n + b_n1^T$$
Neural Network: Back Prop I

Fact  We will use the following results in deduction for backpropagation.

$$\frac{\partial f(AX)}{\partial X} = A^T \nabla f(Y)_{|Y=AX}$$

$$\frac{\partial f(AX)}{\partial A} = \nabla f(Y)_{|Y=AX} X^T$$

Notations  We use the following notation. \( \ast \) means elementwise multiplication. \( d\sigma(X) \) for \( X \in \mathbb{R}^{a \times b} \) is also a matrix in \( \mathbb{R}^{a \times b} \). The elements on the \( i \)-th row and \( j \)-th column \( d\sigma(X)_{ij} \) is defined as \( \frac{\partial \sigma(X_{ij})}{\partial X_{ij}} \):
Neural Network: Back Prop II

Backprop \hspace{1em} We start BP with following initialization:

\[
\begin{align*}
G_n &= \frac{\partial L}{\partial Y} \\
\frac{\partial L}{\partial W_n} &= G_n D_n^T \\
\frac{\partial L}{\partial b_n} &= G_n 1 \\
\frac{\partial L}{\partial D_n} &= W_n^T G_n
\end{align*}
\]

(2)

We know that \(\frac{\partial L}{\partial D_k} = W_k^T G_k\), so we can iteratively calculate the following:

\[
\begin{align*}
G_k &= \frac{\partial L}{\partial E_{k+1}} = d\sigma(E_{k+1}) \ast \frac{\partial L}{\partial D_{k+1}} = d\sigma(W_k D_k + b_k 1^T) \ast (W_{k+1}^T G_{k+1}) \\
\frac{\partial L}{\partial W_k} &= G_k D_k^T \\
\frac{\partial L}{\partial b_k} &= G_k 1
\end{align*}
\]

(3)
Revival in the 1980’s

- Backpropagation discovered in 1970’s but popularized in 1986

- MLP is a universal approximator
  - Can approximate any non-linear function in theory, given enough neurons, data

- Generated lots of excitement and applications

Neural Networks Applied to Vision

LeNet – vision application


USPS digit recognition, later check reading

Convolution, pooling (“weight sharing”), fully connected layers

Unsupervised Neural Networks

**Autoencoders**

- Encode then decode the same input
- No supervision needed

![Autoencoder Diagram](image)


**(Restricted) Boltzmann Machines (RBMs)**

- Stochastic networks that can learn representations
- Restricted version: neurons must form bipartite graph

![Restricted Boltzmann Machine Diagram](image)


Recurrent Neural Networks

Networks with loops
- The output of a layer is used as input for the same (or lower) layer
- Can model dynamics (e.g. in space or time)

Loops are unrolled
- Now a standard feed-forward network with many layers
- Suffers from vanishing gradient problem
- In theory, can learn long term memory, in practice not (Bengio et al, 1994)

Image credit: Chritopher Olah’s blog http://colah.github.io/posts/2015-08-Understanding-LSTMs/
Long Short Term Memory (LSTM)

- A type of RNN explicitly designed not to have the vanishing or exploding gradient problem
- Models long-term dependencies
- Memory is propagated and accessed by gates
- Used for speech recognition, language modeling ...

Issues in Deep Neural Networks

Large amount of training time
- There are sometimes a lot of training data
- Many iterations (epochs) are typically required for optimization
- Computing gradients in each iteration takes too much time

Overfitting
- Learned function fits training data well, but performs poorly on new data (high capacity model, not enough training data)

Vanishing gradient problem

\[
\frac{\partial E}{\partial w_{ki}} = \sum_n \frac{\partial E}{\partial \hat{y}_i^n} \frac{d\hat{y}_i^n}{dz_i^n} \frac{\partial z_i^n}{\partial w_{ki}} = \sum_n \frac{\partial z_i^n}{\partial w_{ki}} \frac{d\hat{y}_i^n}{dz_i^n} \sum_j w_{ij} \frac{d\hat{y}_j^n}{dz_j^n} \frac{\partial E}{\partial \hat{y}_j^n}
\]

Gradients in the lower layers are typically extremely small
- Optimizing multi-layer neural networks takes huge amount of time

Slide credit: adapted from Bohyung Han
New “winter” and revival in early 2000’s

New “winter” in the early 2000’s due to
• problems with training NNs
• Support Vector Machines (SVMs), Random Forests (RF) – easy to train, nice theory

Revival again by 2011-2012
• Name change (“neural networks” -> “deep learning”)
• + Algorithmic developments
  ➢ unsupervised layer-wise pre-training
• ReLU, dropout, layer normalization
• + Big data + GPU computing =
• Large outperformance on many datasets (Vision: ILSVRC’12)

Big Data

ImageNet Large Scale Visual Recognition Challenge

- 1000 categories w/ 1000 images per category
- 1.2 million training images, 50,000 validation, 150,000 testing

AlexNet Architecture

60 million parameters!

Various tricks

- ReLU nonlinearity
- Overlapping pooling
- Local response normalization
- Dropout – set hidden neuron output to 0 with probability .5
- Data augmentation
- Training on GPUs

GPU Computing

- **Big data and big models** require lots of computational power

- **GPUs**
  - thousands of cores for parallel operations
  - multiple GPUs
  - still took about 5-6 days to train AlexNet on two NVIDIA GTX 580 3GB GPUs (much faster today)
Image Classification Performance

![Image Classification Top-5 Errors (%)](image)


Slide credit: Bohyung Han
Questions?

References (& great tutorials):
http://colah.github.io/posts/2015-08-Understanding-LSTMs/