Learning with Big Data I: Divide and Conquer

Chris Dyer

25 July 2013 – LxMLS
No data like more data!

(Banko and Brill, ACL 2001)
(Brants et al., EMNLP 2007)
Problem

Moore’s law (~ power/CPU)

\[ \#(t) \approx n_0 \times 2^{t/(2 \text{ years})} \]

Hard disk capacity

\[ \#(t) \approx m_0 \times 3.2^{t/(2 \text{ years})} \]

We can represent more data than we can centrally process ... and this will only get worse in the future.
Solution

• **Partition training data into subsets**
  - Core concept: *data shards*
  - Algorithms that work primarily “inside” shards and communicate infrequently

• **Alternative solutions**
  - Reduce data points by instance selection
  - Dimensionality reduction / compression

• **Related problems**
  - Large numbers of dimensions (horizontal partitioning)
  - Large numbers of related tasks
    - every Gmail user has his own opinion about what is spam
    - Stefan Riezler’s talk tomorrow

Stefan Riezler’s talk tomorrow
Outline

• MapReduce
• Design patterns for MapReduce
• Batch Learning Algorithms on MapReduce
• Distributed Online Algorithms
MapReduce

cheap commodity clusters
+ simple, distributed programming models
= data-intensive computing for all
Divide and Conquer

Partition

Combine

“Work”

\( w_1 \)

\( w_2 \)

\( w_3 \)

“worker”

“worker”

“worker”

\( r_1 \)

\( r_2 \)

\( r_3 \)

“Result”
It’s a bit more complex…

**Fundamental issues**
- scheduling, data distribution, synchronization, inter-process communication, robustness, fault tolerance, …

**Architectural issues**
- Flynn’s taxonomy (SIMD, MIMD, etc.), network typology, bisection bandwidth
- UMA vs. NUMA, cache coherence

**Different programming models**
- Message Passing
- Shared Memory

**Different programming constructs**
- mutexes, conditional variables, barriers, …
- masters/slaves, producers/consumers, work queues, …

**Common problems**
- livelock, deadlock, data starvation, priority inversion…
- dining philosophers, sleeping barbers, cigarette smokers, …

The reality: programmer shoulders the burden of managing concurrency…
Typical Problem

- Iterate over a large number of records
  - Extract something of interest from each
  - Shuffle and sort intermediate results
  - Aggregate intermediate results
  - Generate final output

**Key idea:** functional abstraction for these two operations
MapReduce

- Programmers specify two functions:
  \[ \text{map} \ (k, v) \rightarrow <k', v'> \]
  \[ \text{reduce} \ (k', v') \rightarrow <k', v'> \]
  - All values with the same key are reduced together

- Usually, programmers also specify:
  \[ \text{partition} \ (k', \text{number of partitions}) \rightarrow \text{partition for } k' \]
  - Often a simple hash of the key, e.g. hash(k') mod n
  - Allows reduce operations for different keys in parallel
  \[ \text{combine} \ (k', v') \rightarrow <k', v'> \]
  - “Mini-reducers” that run in memory after the map phase
  - Optimizes to reduce network traffic & disk writes

- Implementations:
  - Google has a proprietary implementation in C++
  - Hadoop is an open source implementation in Java
Shuffle and Sort: aggregate values by keys

map

map

map

map

reduce

reduce

reduce

\( k_1 v_1 \)

\( k_2 v_2 \)

\( k_3 v_3 \)

\( k_4 v_4 \)

\( k_5 v_5 \)

\( k_6 v_6 \)

a 1 b 2 c 3 c 6 a 5 c 2 b 7 c 9 a 1 5 b 2 7 c 2 3 6 9 r_1 s_1 r_2 s_2 r_3 s_3 \)
MapReduce Runtime

- Handles scheduling
  - Assigns workers to map and reduce tasks
- Handles “data distribution”
  - Moves the process to the data
- Handles synchronization
  - Gathers, sorts, and shuffles intermediate data
- Handles faults
  - Detects worker failures and restarts
- Everything happens on top of a distributed FS (later)
“Hello World”: Word Count

Map(String input_key, String input_value):
   // input_key: document name
   // input_value: document contents
   for each word w in input_values:
      EmitIntermediate(w, "1");

Reduce(String key, Iterator intermediate_values):
   // key: a word, same for input and output
   // intermediate_values: a list of counts
   int result = 0;
   for each v in intermediate_values:
      result += ParseInt(v);
   Emit(AsString(result));
How do we get data to the workers?

What’s the problem here?
Distributed File System

- Don’t move data to workers… Move workers to the data!
  - Store data on the local disks for nodes in the cluster
  - Start up the workers on the node that has the data local
- Why?
  - Not enough RAM to hold all the data in memory
  - Disk access is slow, disk throughput is good
- A distributed file system is the answer
  - GFS (Google File System)
  - HDFS for Hadoop (= GFS clone)
GFS: Assumptions

- Commodity hardware over “exotic” hardware
- High component failure rates
  - Inexpensive commodity components fail all the time
- “Modest” number of HUGE files
- Files are write-once, mostly appended to
  - Perhaps concurrently
- Large streaming reads over random access
- High sustained throughput over low latency
GFS: Design Decisions

- Files stored as chunks
  - Fixed size (64MB)
- Reliability through replication
  - Each chunk replicated across 3+ chunkservers
- Single master to coordinate access, keep metadata
  - Simple centralized management
- No data caching
  - Little benefit due to large data sets, streaming reads
- Simplify the API
  - Push some of the issues onto the client
Redrawn from Ghemawat et al. (SOSP 2003)
Master’s Responsibilities

- Metadata storage
- Namespace management/locking
- Periodic communication with chunkservers
- Chunk creation, replication, rebalancing
- Garbage collection
Questions?
MapReduce “killer app”: Graph Algorithms
Graph Algorithms: Topics

• Introduction to graph algorithms and graph representations
• Single Source Shortest Path (SSSP) problem
  o Refresher: Dijkstra’s algorithm
  o Breadth-First Search with MapReduce
• PageRank
What's a graph?

- $G = (V,E)$, where
  - $V$ represents the set of vertices (nodes)
  - $E$ represents the set of edges (links)
  - Both vertices and edges may contain additional information

- Different types of graphs:
  - Directed vs. undirected edges
  - Presence or absence of cycles
  - ...

Some Graph Problems

- Finding shortest paths
  - Routing Internet traffic and UPS trucks
- Finding minimum spanning trees
  - Telco laying down fiber
- Finding Max Flow
  - Airline scheduling
- Identify “special” nodes and communities
  - Breaking up terrorist cells, spread of swine/avian/… flu
- Bipartite matching
  - Monster.com, Match.com
- And of course... PageRank
Representing Graphs

• $G = (V, E)$
  o A poor representation for computational purposes

• Two common representations
  o Adjacency matrix
  o Adjacency list
Adjacency Matrices

Represent a graph as an $n \times n$ square matrix $M$

- $n = |V|$
- $M_{ij} = 1$ means a link from node $i$ to $j$

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
\hline
1 & 0 & 1 & 0 & 1 \\
2 & 1 & 0 & 1 & 1 \\
3 & 1 & 0 & 0 & 0 \\
4 & 1 & 0 & 1 & 0 \\
\end{array}
\]
Adjacency Lists

Take adjacency matrices... and throw away all the zeros

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

1: 2, 4
2: 1, 3, 4
3: 1
4: 1, 3
Adjacency Lists: Critique

**Advantages:**
- Much more compact representation
- Easy to compute over outlinks
- Graph structure can be broken up and distributed

**Disadvantages:**
- Much more difficult to compute over inlinks
Single Source Shortest Path

- **Problem**: find shortest path from a source node to one or more target nodes
- First, a refresher: Dijkstra’s Algorithm
Dijkstra’s Algorithm Example

Example from CLR
Dijkstra’s Algorithm

Example

Example from CLR
Dijkstra’s Algorithm
Example

Example from CLR
Dijkstra’s Algorithm

Example from CLR
Dijkstra’s Algorithm
Example from CLR
Dijkstra’s Algorithm Example

Example from CLR
Single Source Shortest Path

- **Problem:** find shortest path from a source node to one or more target nodes
- Single processor machine: Dijkstra’s Algorithm
- MapReduce: parallel Breadth-First Search (BFS)
Finding the Shortest Path

• First, consider equal edge weights
• Solution to the problem can be defined inductively
• Here’s the intuition:
  o DistanceTo(startNode) = 0
  o For all nodes \( n \) directly reachable from startNode, DistanceTo\( (n) = 1 \)
  o For all nodes \( n \) reachable from some other set of nodes \( S \), DistanceTo\( (n) = 1 + \min(\text{DistanceTo}(m), m \in S) \)
From Intuition to Algorithm

• A map task receives
  o Key: node $n$
  o Value: $D$ (distance from start), points-to (list of nodes reachable from $n$)

• $\forall p \in$ points-to: emit $(p, D+1)$

• The reduce task gathers possible distances to a given $p$ and selects the minimum one
Multiple Iterations Needed

- This MapReduce task advances the “known frontier” by one hop
  - Subsequent iterations include more reachable nodes as frontier advances
  - Multiple iterations are needed to explore entire graph
  - Feed output back into the same MapReduce task

- Preserving graph structure:
  - Problem: Where did the points-to list go?
  - Solution: Mapper emits \((n, \text{points-to})\) as well
Visualizing Parallel BFS
Termination

• Does the algorithm ever terminate?
  o Eventually, all nodes will be discovered, all edges will be considered (in a connected graph)

• When do we stop?
Weighted Edges

- Now add positive weights to the edges
- Simple change: points-to list in map task includes a weight $w$ for each pointed-to node
  - emit $(p, D+w_p)$ instead of $(p, D+1)$ for each node $p$
- Does this ever terminate?
  - Yes! Eventually, no better distances will be found. When distance is the same, we stop
  - Mapper should emit $(n, D)$ to ensure that “current distance” is carried into the reducer
Comparison to Dijkstra

• Dijkstra's algorithm is more efficient
  o At any step it only pursues edges from the minimum-cost path inside the frontier

• MapReduce explores all paths in parallel
  o Divide and conquer
  o Throw more hardware at the problem
General Approach

• MapReduce is adept at manipulating graphs
  o Store graphs as adjacency lists

• Graph algorithms with for MapReduce:
  o Each map task receives a node and its outlinks
  o Map task compute some function of the link structure, emits value with target as the key
  o Reduce task collects keys (target nodes) and aggregates

• Iterate multiple MapReduce cycles until some termination condition
  o Remember to “pass” graph structure from one iteration to next
Random Walks Over the Web

• Model:
  o User starts at a random Web page
  o User randomly clicks on links, surfing from page to page

• PageRank = the amount of time that will be spent on any given page
PageRank: Defined

Given page $x$ with in-bound links $t_1, \ldots, t_n$, where

- $C(t)$ is the out-degree of $t$
- $\alpha$ is probability of random jump
- $N$ is the total number of nodes in the graph

$$PR(x) = \alpha \left( \frac{1}{N} \right) + (1 - \alpha) \sum_{i=1}^{n} \frac{PR(t_i)}{C(t_i)}$$
Computing PageRank

• Properties of PageRank
  o Can be computed iteratively
  o Effects at each iteration is local

• Sketch of algorithm:
  o Start with seed $PR_i$ values
  o Each page distributes $PR_i$ "credit" to all pages it links to
  o Each target page adds up "credit" from multiple in-bound links to compute $PR_{i+1}$
  o Iterate until values converge
PageRank in MapReduce

**Map:** distribute PageRank “credit” to link targets

**Reduce:** gather up PageRank “credit” from multiple sources to compute new PageRank value

Iterate until convergence
PageRank: Issues

- Is PageRank guaranteed to converge? How quickly?
- What is the “correct” value of \( \alpha \), and how sensitive is the algorithm to it?
- What about dangling links?
- How do you know when to stop?
Graph algorithms in MapReduce

• General approach
  o Store graphs as adjacency lists (node, points-to, points-to …)
  o Mappers receive (node, points-to*) tuples
  o Map task computes some function of the link structure
  o Output key is usually the target node in the adjacency list representation
  o Mapper typically outputs the graph structure as well

• Iterate multiple MapReduce cycles until some convergence criterion is met
Questions?
MapReduce Algorithm Design
Managing Dependencies

• Remember: Mappers run in isolation
  o You have no idea in what order the mappers run
  o You have no idea on what node the mappers run
  o You have no idea when each mapper finishes

• Tools for synchronization:
  o Ability to hold state in reducer across multiple key-value pairs
  o Sorting function for keys
  o Partitioner
  o Cleverly-constructed data structures
Motivating Example

- Term co-occurrence matrix for a text collection
  - $M = N \times N$ matrix ($N =$ vocabulary size)
  - $M_{ij}$: number of times $i$ and $j$ co-occur in some context
    (for concreteness, let’s say context = sentence)

- Why?
  - Distributional profiles as a way of measuring semantic distance
  - Semantic distance useful for many language processing tasks

“You shall know a word by the company it keeps” (Firth, 1957)

e.g., Mohammad and Hirst (EMNLP, 2006)
MapReduce: Large Counting Problems

- Term co-occurrence matrix for a text collection = specific instance of a large counting problem
  - A large event space (number of terms)
  - A large number of events (the collection itself)
  - Goal: keep track of interesting statistics about the events

- Basic approach
  - Mappers generate partial counts
  - Reducers aggregate partial counts

How do we aggregate partial counts efficiently?
First Try: “Pairs”

- Each mapper takes a sentence:
  - Generate all co-occurring term pairs
  - For all pairs, emit \((a, b) \rightarrow count\)

- Reducers sums up counts associated with these pairs
- Use combiners!

Note: in these slides, we donate a key-value pair as \(k \rightarrow v\)
“Pairs” Analysis

• Advantages
  o Easy to implement, easy to understand

• Disadvantages
  o Lots of pairs to sort and shuffle around (upper bound?)
Another Try: “Stripes”

- Idea: group together pairs into an associative array

\[
\begin{align*}
(a, b) & \rightarrow 1 \\
(a, c) & \rightarrow 2 \\
(a, d) & \rightarrow 5 \\
(a, e) & \rightarrow 3 \\
(a, f) & \rightarrow 2
\end{align*}
\]

- Each mapper takes a sentence:
  - Generate all co-occurring term pairs
  - For each term, emit \( a \rightarrow \{ b: \text{count}_b, c: \text{count}_c, d: \text{count}_d \ldots \} \)

- Reducers perform element-wise sum of associative arrays

\[
\begin{align*}
\text{(a, b) } & \rightarrow \{ b: 1, d: 5, e: 3 \} \\
\text{(a, c) } & \rightarrow \{ b: 1, c: 2, d: 5, e: 3, f: 2 \}
\end{align*}
\]

\[
\begin{align*}
\text{a } & \rightarrow \{ b: 2, c: 2, d: 7, e: 3, f: 2 \}
\end{align*}
\]
“Stripes” Analysis

• Advantages
  o Far less sorting and shuffling of key-value pairs
  o Can make better use of combiners

• Disadvantages
  o More difficult to implement
  o Underlying object is more heavyweight
  o Fundamental limitation in terms of size of event space
Efficiency comparison of approaches to computing word co-occurrence matrices

Cluster size: 38 cores
Data Source: Associated Press Worldstream (APW) of the English Gigaword Corpus (v3), which contains 2.27 million documents (1.8 GB compressed, 5.7 GB uncompressed)
Relative frequency estimates

- How do we compute relative frequencies from counts?

\[ P(B | A) = \frac{\text{count}(A, B)}{\text{count}(A)} = \frac{\text{count}(A, B)}{\sum_{B'} \text{count}(A, B')} \]

- Why do we want to do this?
- How do we do this with MapReduce?
P(B | A): “Pairs”

Reducer holds this value in memory

\[(a, *) \rightarrow 32\]

\[(a, b_1) \rightarrow 3\]
\[(a, b_2) \rightarrow 12\]
\[(a, b_3) \rightarrow 7\]
\[(a, b_4) \rightarrow 1\]

\[
\begin{align*}
(a, b_1) & \rightarrow 3 / 32 \\
(a, b_2) & \rightarrow 12 / 32 \\
(a, b_3) & \rightarrow 7 / 32 \\
(a, b_4) & \rightarrow 1 / 32 \\
\end{align*}
\]

• For this to work:
  o Must emit extra \((a, *)\) for every \(b_n\) in mapper
  o Must make sure all \(a\)'s get sent to same reducer (use partitioner)
  o Must make sure \((a, *)\) comes first (define sort order)
  o Must hold state in reducer across different key-value pairs
$P(B \mid A)$: “Stripes”

$a \rightarrow \{b_1:3, b_2:12, b_3:7, b_4:1, \ldots \}$

- Easy!
  - One pass to compute $(a, \ast)$
  - Another pass to directly compute $f(B \mid A)$
Synchronization in MapReduce

• **Approach 1: turn synchronization into an ordering problem**
  - Sort keys into correct order of computation
  - Partition key space so that each reducer gets the appropriate set of partial results
  - Hold state in reducer across multiple key-value pairs to perform computation
    - Illustrated by the “pairs” approach

• **Approach 2: construct data structures that “bring the pieces together”**
  - Each reducer receives all the data it needs to complete the computation
    - Illustrated by the “stripes” approach
Issues and Tradeoffs

• Number of key-value pairs
  o Object creation overhead
  o Time for sorting and shuffling pairs across the network
  o In Hadoop, every object emitted from a mapper is written to disk

• Size of each key-value pair
  o De/serialization overhead

• Combiners make a big difference!
  o RAM vs. disk and network
  o Arrange data to maximize opportunities to aggregate partial results
Questions?
Batch Learning Algorithms in MR
Batch Learning Algorithms

• Expectation maximization
  o Gaussian mixtures / k-means
  o Forward-backward learning for HMMs

• Gradient-ascent based learning
  o Computing (gradient, objective) using MapReduce
  o Optimization questions
[iy]
[uw]
[a:]
But what if the data look like this?
How do we learn with no labels??
MR Assumptions

- **E-step (expensive)**
  - Compute posterior distribution over latent variable
  - In the case of k-means, that’s the class label, given the data and the parameters

- **M-step**
  - Solving an optimization problem given the posteriors over latent variables
  - K-means, HMMs, PCFGs: analytic solutions
  - Not always the case

- When is this assumption appropriate?
Compute the posterior distribution over the latent variables y and the current parameters $\theta^{(t)}$

$$\mathbb{E}[y] = p(y \mid x, \theta^{(t)})$$

M step (reducer)

$$\theta^{(t+1)} \leftarrow \arg \max_{\theta} \mathcal{L}(x, \mathbb{E}[y])$$
Compute the posterior distribution over the latent variables $y$ and the current parameters $\theta^{(t)}$.

$$\mathbb{E}[y] = p(y \mid x, \theta^{(t)})$$

**E step (mappers)**

Cluster labels

Data points

**M step (reducer)**

$$\theta^{(t+1)} \leftarrow \arg \max_{\theta} \mathcal{L}(x, \mathbb{E}[y])$$
EM Algorithms in MapReduce

E step (mappers)
Compute the posterior distribution over the latent variables $y$ and the current parameters $\theta^{(t)}$.

\[ \mathbb{E}[y] = p(y | x, \theta^{(t)}) \]

M step (reducer)

\[ \theta^{(t+1)} \leftarrow \text{arg max}_{\theta} \mathcal{L}(x, \mathbb{E}[y]) \]
EM Algorithms in MapReduce

E step

Compute the expected log likelihood with respect to the conditional distribution of the latent variables with respect to the observed data.

\[
E[y] = p(y \mid x, \theta^{(t)})
\]

Expectations are just sums of function evaluation over an event times that event’s probability: *perfect for MapReduce!*

Mappers compute model likelihood given small pieces of the training data (scale EM to large data sets!)
EM Algorithms in MapReduce

M step

\[ \theta^{(t+1)} \leftarrow \arg \max_{\theta} \mathcal{L}(x, \mathbb{E}[y]) \]

The solution to this problem depends on the parameterization used. For HMMs, PCFGs with multinomial parameterizations, this is just computing the relative frequency.

• Easy!
  o One pass to compute \((a, *)\)
  o Another pass to directly compute \(f(B \mid A)\)

\[ a \rightarrow \{b_1:3, b_2:12, b_3:7, b_4:1, \ldots \} \]
Challenges

• Each iteration of EM is one MapReduce job
• Mappers require the current model parameters
  o Certain models may be very large
  o Optimization: any particular piece of the training data probably depends on only a small subset of these parameters
• Reducers may aggregate data from many mappers
  o Optimization: Make smart use of combiners!
Log-linear Models

- NLP's favorite discriminative model:

\[ p(y|x) = \frac{1}{Z(x)} \exp \sum_i \lambda_i h_i(x, y) \]

- Applied successfully to classification, POS tagging, parsing, MT, word segmentation, named entity recognition, LM...
  - Make use of millions of features (h_i’s)
  - Features may overlap
  - Global optimum easily reachable, assuming no latent variables
Exponential Models in MapReduce

• Training is usually done to maximize likelihood (minimize negative llh), using first-order methods
  o Need an objective and gradient with respect to the parameterizes that we want to optimize

\[
\mathcal{L}(\theta) = -\sum_{(x,y)} \log p(y|x; \theta)
\]

\[
\frac{\partial \mathcal{L}}{\partial \theta_i} = \sum_{(x,y)} \mathbb{E}_{p(y'|x; \theta)} [h_i(x, y')] - h_i(x, y)
\]
Exponential Models in MapReduce

How do we compute these in MapReduce?

\[ \mathcal{L}(\theta) = -\sum_{\langle x, y \rangle} \log p(y|x; \theta) \]

\[ \frac{\partial \mathcal{L}}{\partial \theta_i} = \sum_{\langle x, y \rangle} \mathbb{E}_{p(y'|x; \theta)} [h_i(x, y')] - h_i(x, y) \]

As seen with EM: expectations map nicely onto the MR paradigm.

Each mapper computes two quantities: the LLH of a training instance \( \langle x, y \rangle \) under the current model and the contribution to the gradient.
Exponential Models in MapReduce

• What about reducers?

\[ \mathcal{L}(\theta) = - \sum_{\langle x, y \rangle} \log p(y|x; \theta) \]

\[ \frac{\partial \mathcal{L}}{\partial \theta_i} = \sum_{\langle x, y \rangle} E_{p(y'|x; \theta)} \left[ h_i(x, y') \right] - h_i(x, y) \]

The objective is a single value – make sure to use a combiner!

The gradient is as large as the feature space – but may be quite sparse. Make use of sparse vector representations!
Exponential Models in MapReduce

• After one MR pair, we have an objective and gradient
• Run some optimization algorithm
  o LBFGS, gradient descent, etc...
• Check for convergence
• If not, re-run MR to compute a new objective and gradient
Challenges

- Each iteration of training is one MapReduce job
- Mappers require the current model parameters
- Reducers may aggregate data from many mappers
- Optimization algorithm (LBFGS for example) may require the full gradient
  - This is okay for millions of features
  - What about billions?
  - ...or trillions?
Case study: statistical machine translation
Statistical Machine Translation

- Conceptually simple:
  (translation from foreign f into English e)
  \[ \hat{e} = \arg \max_e P(f \mid e)P(e) \]

- Difficult in practice!

- Phrase-Based Machine Translation (PBMT):
  - Break up source sentence into little pieces (phrases)
  - Translate each phrase individually

Dyer et al. (Third ACL Workshop on MT, 2008)
Maria no dio una bofetada a la bruja verde

Example from Koehn (2006)
The Data Bottleneck

![Graph showing the relationship between training time and translation quality with corpus size](image)

- **Time (seconds)**: 15 min, 30 min, 45 min, 1.5 hrs, 3 hrs, 6 hrs, 12 hrs, 1 day, 2 days
- **Corpus size (sentences)**: 10,000, 100,000, 1e+06, 1e+07
- **Translation quality (BLEU)**: 0.3, 0.35, 0.4, 0.45, 0.5, 0.55, 0.6

The graph illustrates how training time and translation quality increase as the corpus size grows.
Maria no daba una bofetada a la bruja verde
Mary did not slap the green witch

There are MapReduce Implementations of these two components!
Alignment with HMMs

Mary deu um tapa a bruxa verde

1  2  3  4  5  6  7

Mary  slapped  the  green  witch

Vogel et al. (1996)
Alignment with HMMs

Mary deu um tapa a bruxa verde

1  2  3  4  5  6  7

1 → 4 → 5 → 7 → 6

Mary   slapped   the   green   witch
Alignment with HMMs

• Emission parameters: translation probabilities
• States: words in source sentence
• Transition probabilities: probability of jumping +1, +2, +3, -1, -2, etc.

• Alternative parameterization of state probabilities:
  o Uniform (“Model 1”)
  o Independent of previous alignment decision, dependent only on global position in sentence (“Model 2”)
  o Many other models...

• This is still state-of-the-art in Machine Translation
• How many parameters are there?
HMM Alignment: Giza

![Graph showing the relationship between HMM alignment time and corpus size for a single-core commodity server. The graph indicates that as the corpus size increases, the time required for alignment also increases.](image)
HMM Alignment: MapReduce

![Graph showing time vs. corpus size for HMM alignment using Giza toolkit and MapReduce implementation. The graph compares the performance of single-core commodity server and 38-processor cluster for different corpus sizes.]
HMM Alignment: MapReduce

![Graph showing comparison between HMM alignment (hypothetical, optimally-parallelized) and HMM alignment (MapReduce implementation)]

- 38 processor cluster
- 1/38 Single-core commodity server
Online Learning Algorithms in MR
Online Learning Algorithms

- **MapReduce** is a batch processing system
  - Associativity used to factor large computations into subproblems that can be solved
  - Once job is running, processes are independent

- **Online algorithms**
  - Great deal of research currently in ML
  - Theory: strong convergence, mistake guarantees
  - Practice: rapid convergence
  - Good (empirical) performance on nonconvex problems

- **Problems**
  - Parameter updates made **sequentially** after each training instance
  - Theoretical analysis rely on sequential model of computation
  - How do we parallelize this?
Sequential perceptron algorithm.

1: algorithm PERCEPTRON($w^{(0)}$, $\mathcal{T} = \{\langle x_i, y_i \rangle\}_{i=1}^{\ell}, N$)
2: $k \leftarrow 0$
3: for $i \leftarrow 1 \ldots N$ do
4:     for $t \leftarrow 1 \ldots \ell$ do
5:         $y' \leftarrow \arg \max_{y \in \mathcal{Y}(x_t)} [w^{(k)} \cdot \Phi(x_t, y)]$
6:         if $y' \neq y_t$ then
7:             $w^{(k+1)} \leftarrow w^{(k)} + \Phi(x_t, y_t) - \Phi(x_t, y')$
8:             $k \leftarrow k + 1$
9: return $w^{(k)}$

$\triangleright$ Run $N$ iterations of perceptron training
$\triangleright$ $k$ is the mistake counter
$\triangleright$ Viterbi or other max inference algorithm

Collins (2002), Rosenblatt (1957)
Review: Perceptron

Sequential perceptron algorithm.
1: algorithm PERCEPTRON($w^{(0)}$, $\mathcal{T} = \{(x_i, y_i)\}_{i=1}^{\ell}, N$)
2: $k \leftarrow 0$
3: for $i \leftarrow 1 \ldots N$ do
4: for $t \leftarrow 1 \ldots \ell$ do
5: $y' \leftarrow \arg\max_{y \in \mathcal{Y}(x_t)} [w^{(k)} \cdot \Phi(x_t, y)]$
6: if $y' \neq y_t$ then
7: $w^{(k+1)} \leftarrow w^{(k)} + \Phi(x_t, y_t) - \Phi(x_t, y')$
8: $k \leftarrow k + 1$
9: return $w^{(k)}$

Sequential perceptron algorithm with averaging.
1: algorithm AVGPERCEPTRON($w^{(0)}$, $\mathcal{T} = \{(x_i, y_i)\}_{i=1}^{\ell}, N$)
2: $v \leftarrow 0$
3: $k \leftarrow 0$
4: for $i \leftarrow 1 \ldots N$ do
5: for $t \leftarrow 1 \ldots \ell$ do
6: $v \leftarrow v + w^{(k)}$
7: $y' \leftarrow \arg\max_{y \in \mathcal{Y}(x_t)} [w^{(k)} \cdot \Phi(x_t, y)]$
8: if $y' \neq y_t$ then
9: $w^{(k+1)} \leftarrow w^{(k)} + \Phi(x_t, y_t) - \Phi(x_t, y')$
10: $k \leftarrow k + 1$
11: return $\frac{v}{N\ell}$

Collins (2002), Rosenblatt (1957)
Assume separability
Algorithm 1: Parameter Mixing

• Inspired by the averaged perceptron, let’s run $P$ perceptrons on shards of the data
• Return the average parameters

Perceptron algorithm with parameter mixing

1: algorithm PARAMMIX($\mathcal{T} = \{(x_i, y_i)\}_{i=1}^{\xi}, N, P$)
2: Shard $\mathcal{T}$ into $P$ shards $\mathcal{T}_P = \{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_P\}$
3: for $p \leftarrow 1 \ldots P$ do
4:     $w^{(p)} \leftarrow$ PERCEPTRON($0, \mathcal{T}_p, N$)
5: return $\frac{\sum_{p=1}^{P} w^{(p)}}{P}$

$\triangleright$ Run perceptron on $P$ shards and mix
$\triangleright$ Execute loop in parallel

Return average parameters from $P$ runs

McDonald, Hall, & Mann (NAACL, 2010)
## Attempt 1: Parameter Mixing

- Unfortunately...
- Theorem (proof by counterexample). For any training set $T$ separable by margin $\gamma$, **ParamMix does not necessarily return a separating hyperplane.**

<table>
<thead>
<tr>
<th></th>
<th>Perceptron</th>
<th>Avg. Perceptron</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serial</td>
<td>85.8</td>
<td>88.8</td>
</tr>
<tr>
<td>1/P - serial</td>
<td>75.3</td>
<td>76.6</td>
</tr>
<tr>
<td>ParamMix</td>
<td>81.5</td>
<td>81.6</td>
</tr>
</tbody>
</table>

Researchers at **Carnegie Mellon University** in **Pittsburgh, PA**, have put together an **iOS app, DrawAFriend, ...**
Algorithm 2: Iterative Mixing

- Rather than mixing parameters just once, mix after each epoch (pass through a shard)
- Redistribute parameters and use them as a starting point on next epoch

Perceptron algorithm with iterative parameter mixing

```
1: algorithm ITERPARAMMIX(\mathcal{T} = \{\langle x_i, y_i \rangle \}_{i=1}^\xi, N, P) \quad \triangleright\text{Sharded perceptron with iterative mixing}
2: \quad \text{Shard } \mathcal{T} \text{ into } P \text{ shards } \mathcal{T}_P = \{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_P\}
3: \quad w^{(0)} \leftarrow 0
4: \quad \text{for } i \leftarrow 1 \ldots N \text{ do}
5: \quad \quad \text{for } p \leftarrow 1 \ldots P \text{ do}
6: \quad \quad \quad v^{(p)} \leftarrow \text{PERCEPTRON}(w^{(i-1)}, \mathcal{T}_p, 1) \quad \triangleright\text{Execute loop in parallel}
7: \quad \quad w^{(i)} \leftarrow \sum_p \frac{v^{(p)}}{P} \quad \triangleright\text{Do one epoch of perceptron training}
8: \quad \text{return } \sum_i \frac{w^{(i)}}{NP}
```
Algorithm 2: Analysis

- Theorem. The weighted number of mistakes has is bound (like the sequential perceptron) by
  - The worst-case number of epochs (through the full data) is the same as the sequential perceptron
  - With non-uniform mixing, it is possible to obtain a speedup of $P$

<table>
<thead>
<tr>
<th></th>
<th>Perceptron</th>
<th>Avg. Perceptron</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serial</td>
<td>85.8</td>
<td>88.8</td>
</tr>
<tr>
<td>1/P - serial</td>
<td>75.3</td>
<td>76.6</td>
</tr>
<tr>
<td>ParamMix</td>
<td>81.5</td>
<td>81.6</td>
</tr>
<tr>
<td>IterParamMix</td>
<td>87.9</td>
<td>88.1</td>
</tr>
</tbody>
</table>
Algorithm 2: Empirical
Summary

• Two approaches to learning
  o Batch: process the entire dataset and make a single update
  o Online: sequential updates

• Online learning is hard to realize in a distributed (parallelized) environment
  o Parameter mixing approaches offer a theoretically sound solution with practical performance guarantees
  o Generalizations to log-loss and MIRA loss have also been explored
  o Conceptually some workers have “stale parameters” that eventually (after a number of synchronizations) reflect the true state of the learner

• Other challenges (next couple of days)
  o Large numbers of parameters (randomized representations)
  o Large numbers of related tasks (multitask learning)
Obrigado!