1/30 8:20:17 ***

← →

Perf. tuning

We tune sports cars - so why not tune DBs?!

Ch.11

11e
Database Systems
Design, Implementation, and Management
Coronel | Morris
Chapter 11
Database Performance Tuning and Query Optimization

'Tuning'

Database Performance-Tuning Concepts

- Goal of database performance is to execute queries as fast as possible
- **Database performance tuning**: Set of activities and procedures that reduce response time of database system
- Fine-tuning the performance of a system requires that all factors must operate at optimum level with minimal bottlenecks

Performance Tuning: Client and Server

- Client side
  - **SQL performance tuning**: Generates SQL query that returns correct answer in least amount of time
    - Using minimum amount of resources at server
- Server side
  - **DBMS performance tuning:** DBMS environment configured to respond to clients' requests as fast as possible
    - Optimum use of existing resources

Figure 11.1 - Basic DBMS Architecture

DBMS Architecture

- All data in a database are stored in **data files**
  - Data files automatically expand in predefined increments known as **extends**
- Data files are grouped in file groups or table spaces
  - **Table space** or **file group**: Logical grouping of several data files that store data with similar characteristics
- **Data cache** or **buffer cache**: Shared, reserved memory area
  - Stores most recently accessed data blocks in RAM

DBMS Architecture

- **SQL cache** or **procedure cache**: Stores most recently executed SQL statements or PL/SQL procedures
- DBMS retrieves data from permanent storage and places them in RAM. **Input/output request**: Low-level data access operation that reads or writes data to and from computer devices (note: reads fetch entire disk datablocks)
- Data cache is faster than working with data files
- Majority of performance-tuning activities focus on minimizing I/O operations

- *Listener*. The listener process listens for clients' requests and handles the processing of the SQL requests to other DBMS processes. Once a request is received, the listener passes the request to the appropriate user process.
- *User*. The DBMS creates a user process to manage each client session. Therefore, when you log on to the DBMS, you are assigned a user process. This process handles all requests you submit to the server. There are many user processes—at least one per logged-in client.
- *Scheduler*. The scheduler process organizes the concurrent execution of SQL requests. (See Chapter 10, Transaction Management and Concurrency Control.)
- *Lock manager*. This process manages all locks placed on database objects, including disk pages. (See Chapter 10.)
- *Optimizer*. The optimizer process analyzes SQL queries and finds the most efficient way to access the data. You will learn more about this process later in the chapter.

Database Query Optimization Modes

- Algorithms proposed for query optimization are based on:
  - Selection of the optimum order to achieve the fastest execution time
  - Selection of sites to be accessed to minimize communication costs
- Evaluated on the basis of:
  - Operation mode
  - Timing of its optimization
  - Type of information (used for optimization)

Classification of Operation Modes

- **Automatic query optimization**: DBMS finds the most cost-effective access path without user intervention
- **Manual query optimization**: Requires that the optimization be selected and scheduled by the end user or programmer

Based on Timing of Optimization

- **Static query optimization**: best optimization strategy is selected when the query is compiled by the DBMS
  - Takes place at compilation time
  - Best for embedded queries
- **Dynamic query optimization**: Access strategy is dynamically determined by the DBMS at run time, using the most up-to-date information about the database
  - Takes place at execution time; more processing o'head

Type of Information Used to Optimize

- **Statistically based query optimization algorithm**: Statistics are used by the DBMS to determine the best access strategy
- Statistical information is generated by DBMS through:
  - **Dynamic statistical generation mode – auto eval**
  - **Manual statistical generation mode - via user**
- **Rule-based query optimization algorithm**: based on a set of user-defined rules to determine the best query access strategy

Table 11.2 - Sample Database Statistics Measurements

DATABASE OBJECT	SAMPLE MEASUREMENTS
Tables	Number of rows, number of disk blocks used, row length, number of columns in each row, number of distinct values in each column, maximum value in each column, minimum value in each column, and columns that have indexes
Indexes	Number and name of columns in the index key, number of key values in the index, number of distinct key values in the index key, histogram of key values in an index, and number of disk pages used by the index
Environment Resources	Logical and physical disk block size, location and size of data files, and number of extends per data file

Cengage Learning © 2015

Commands to manually generate stats:

* COMPUTE STATISTICS (Oracle)
* ANALYZE TABLE (MySQL)
* UPDATE STATISTICS (SQL Server)

Select ....
From ...
Where ...
Data files
Parsing phase
SQL cache
Access plan
Execution phase
Data cache
Fetching phase
• Syntax check
• Naming check
• Access rights check
• Decompose and analyze
• Generate access plan
• Store access plan in SQL cache
• Execute I/O operations
• Add locks for transaction mgmt
• Retrieve data blocks from data files
• Place data blocks in data cache
• Generate result set

Query Processing

Parsing

- DBMS parses the SQL query and chooses the most efficient access/execution plan

Execution

- DBMS executes the SQL query using the chosen execution plan

Fetching

- DBMS fetches the data and sends the result set back to the client

SQL Parsing Phase (done by **query optimizer**)

- Query is broken down into smaller units
- Original SQL query is transformed into slightly different version of the original SQL code which is fully equivalent and more efficient
- **Query optimizer**: Analyzes SQL query and finds most efficient way to access data
- **Access plans**: DBMS-specific and translate client's SQL query into a series of complex I/O operations

SQL Parsing Phase

- If access plan already exists for query in SQL cache, DBMS reuses it
  - If not, optimizer evaluates various plans and chooses one to be placed in SQL cache for use

SQL Execution Phase

- All I/O operations indicated in the access plan are executed
  - Locks are acquired
  - Data are retrieved and placed in data cache
  - Transaction management commands are processed

Note that 'execution' here refers to executing the access plan, ie. fetching/sending data from/to the backend database – it does not refer to executing your SQL query. Query execution occurs in the next step (the 'fetching' step).

SQL Fetching Phase

- Rows of resulting query result set are returned to client
  - DBMS may use temporary table space to store temporary data
  - Database server coordinates the movement of the result set rows from the server cache to the client cache (eg will send blocks of rows to the client, wait for next request, send next block..)

Query Processing Bottlenecks

- Delay introduced in the processing of an I/O operation that slows the system
- Caused by the:
  - CPU – slow processor, rogue processes..
  - RAM – shared among running processes
  - Hard disk – disk speed, transfer rates..
  - Network – bandwidth shared among clients
  - Application code – bad user code, poor db design..

Indexes and Query Optimization

- Indexes
  - Help speed up data access
  - Facilitate searching, sorting, using aggregate functions, and join operations
  - Ordered set of values that contain the index key and pointers
  - More efficient than a full table scan

The reason for using an index is simple - provided we incur an upfront cost of creating (computing) one, runtime lookup costs using the index are vastly cheaper than doing full searches through non-indexed rows.

Eg. given this book, find all the places that discuss table updating..

A sample index

Indexes and Query Optimization

- **Data sparsity**: Number of different values a column could have; low sparsity => index might be useless
- Data structures used to implement indexes:
  - **Hash indexes**
  - **B-tree indexes**
  - **Bitmap indexes**
- DBMSs determine best type of index to use

Example of low sparsity: STU_SEX gender column in a table is either M or F, so indexing this is useless (we'd only have 2 keys, each with 1000s of records!). On the other hand, DOB is a column with relatively higher sparsity, and is worth creating an index for..

A hash index is based on an ordered list of hash values, computed from a key column, using a hashing algorithm - it is much faster to search through the hash values than search the columns. Each hash value then points to the actual (column) data.

A user query (eg. LNAME="Johnson") is converted to a hash 'key' which is then used to search through the pre-computed list of hash values and retrieve an exact match or a small set of values that are all stored

with the same key (as a result of 'hash collision').

B-tree index, bitmap index

A bitmap index can use hardware arithmetic to look up actual rows in a table.

We use it when a column can have any of 'N' values, where N is small, eg. a customer's location in the US can only one of 6 values, like so:

In the above case, we'll create 6 bit columns (because there can be one of 6 values in the actual 'CustomerLocation' table column), eg like so: Alaska Hawaii Western Mountain Central Eastern

Then, for each customer, we set the bit to 1 where the customer lives, and the rest to 0. Further, we'll 'pad' with two more 0s, to use 8 bits (a byte). Eg three rows could be

```
0 0 1 0 0 0 0 0 (lives in CA)
0 1 0 0 0 0 0 0 (lives in Hawaii)
0 0 0 0 0 1 0 0 (lives in Maine)
...
```

So in the bitmap index, the number of rows will equal the # of rows in the actual table, and the number of columns, and the column names, will equal the possible values for the table column we are indexing.

The DBMS will store each above row as a byte. When we query for just the western region customers, the DBMS will '&' each row with a '00100000' "bitmask", ie efficiently go through each row in the bitmap index to filter just the 00100000 rows, for ex.

Index Selectivity

- Measure of the likelihood that an index will be used in query processing
- Indexes are used when a subset of rows from a large table is to be selected based on a given condition
- Index cannot always be used to improve performance
- **Function-based index**: Based on a specific SQL function or expression (eg. EMP_SALARY+EMP_COMMISSION)

Index selectivity: a measure of how likely an index will be used in a query. So we strive to create indexes that have high selectivity..

Indexes are useful when:

- an indexable column occurs in a WHERE or HAVING search expression
- an indexable column appears in a GROUP BY or ORDER BY clause
- MAX or MIN is applied to an indexable column
- there is high data sparsity on an indexable column

Worth creating indexes on single columns that appear in WHERE, HAVING, ORDER BY, GROUP BY and join conditions.

Rule-based vs cost-based (statistics-based)

Optimizer Choices

- **Rule-based optimizer**: Uses preset rules and points to determine the best approach to execute a query (choose a plan that minimizes the sum of rules' points (minimum cost))
- **Cost-based optimizer**: Uses algorithms based on statistics about objects being accessed (processing cost, RAM cost, I/O cost..) to determine the best approach to execute a query

Using Hints to Affect Optimizer Choices

- Optimizer might not choose the best execution plan
  - Makes decisions based on existing statistics, which might be old
  - Might choose less-efficient decisions
- **Optimizer hints**: Special instructions for the optimizer, embedded in the SQL command text

Table 11.5 - Optimizer Hints

HINT	USAGE
ALL_ROWS	Instructs the optimizer to minimize the overall execution time—that is, to minimize the time needed to return all rows in the query result set. This hint is generally used for batch mode processes. For example: SELECT /*+ ALL_ROWS */ * FROM PRODUCT WHERE P_QOH < 10;
FIRST_ROWS	Instructs the optimizer to minimize the time needed to process the first set of rows—that is, to minimize the time needed to return only the first set of rows in the query result set. This hint is generally used for interactive mode processes. For example: SELECT /*+ FIRST_ROWS */ * FROM PRODUCT WHERE P_QOH < 10;
INDEX(name)	Forces the optimizer to use the P_QOH_NDX index to process this query. For example: SELECT /*+ INDEX(P_QOH_NDX) */ * FROM PRODUCT WHERE P_QOH < 10;

Cengage Learning © 2015

SQL Performance Tuning

- Evaluated from client perspective
  - Most current relational DBMSs perform automatic query optimization at the server end
  - Most SQL performance optimization techniques are DBMS-specific and thus rarely portable
- Majority of performance problems are related to poorly written SQL code

Conditional Expressions

- Expressed within WHERE or HAVING clauses of a SQL statement
  - Restricts the output of a query to only rows matching conditional criteria
- Guidelines to write efficient conditional expressions in SQL code
  - Use simple columns or literals as operands
  - Numeric field comparisons are faster than character, date, and NULL comparisons

Conditional Expressions

- Equality comparisons are faster than inequality comparisons
- Transform conditional expressions to use literals
- Write equality conditions first when using multiple conditional expressions
- When using multiple AND conditions, write the condition most likely to be false first
- When using multiple OR conditions, put the condition most likely to be true first
- Avoid the use of NOT logical operator

Query Formulation

- Identify what columns and computations are required
- Identify source tables
- Determine how to join tables
- Determine what selection criteria are needed
- Determine the order in which to display the output

All of the above can be summarized like so: "know what you want, then find a good way to get it" [what do we want to return/compute/generate, how to best go about it (what SQL constructs to use, on what data)].

DBMS Performance Tuning

- Managing DBMS processes in primary memory and the structures in physical storage
- DBMS performance tuning at server end focuses on setting parameters used for:
  - Data cache
  - SQL cache
  - Sort cache
  - Optimizer mode
- **In-memory database**: Store large portions of the database in primary storage

DBMS Performance Tuning

- Recommendations for physical storage of databases:
  - Use **RAID** (Redundant Array of Independent Disks) to provide a balance between performance improvement and fault tolerance
  - Minimize disk contention
  - Put high-usage tables in their own table spaces
  - Assign separate data files in separate storage volumes for indexes, system, and high-usage tables

More info..

If you'd like more detail (MySQL-related, but the overall ideas are non-DB-specific), read this doc..

1/30 8:20:44 ***

← →

Business Intelligence

("BI")

Business Intelligence (BI)

- Comprehensive, cohesive, integrated set of tools and processes
  - Captures, collects, integrates, stores, and analyzes data
- Purpose - Generate and present information to support business decision making
- Allows a business to transform:
  - Data into information
  - Information into knowledge
  - Knowledge into wisdom

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

4

Figure 13.1 - Business Intelligence Framework

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

5

Business Intelligence Benefits

- Improved decision making
- Integrating architecture
- Common user interface for data reporting and analysis
- Common data repository fosters single version of company data
- Improved organizational performance

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

11

Figure 13.3 - Evolution of BI Information Dissemination Formats

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part. 14

Transactional, operational data: individual units.

'Analytical', decision-support data: aggregated.

Table 13.5 - Contrasting Operational and Decision Support Data Characteristics

CHARACTERISTIC	OPERATIONAL DATA	DECISION SUPPORT DATA
Data currency	Current operations Real-time data	Historic data Snapshot of company data Time component (week/month/year)
Granularity	Atomic-detailed data	Summarized data
Summarization level	Low; some aggregate yields	High; many aggregation levels
Data model	Highly normalized Mostly relational DBMSs	Non-normalized Complex structures Some relational, but mostly multidimensional DBMSs
Transaction type	Mostly updates	Mostly query
Transaction volumes	High-update volumes	Periodic loads and summary calculations
Transaction speed	Updates are critical	Retrievals are critical
Query activity	Low to medium	High
Query scope	Narrow range	Broad range
Query complexity	Simple to medium	Very complex
Data volumes	Hundreds of gigabytes	Terabytes to petabytes

Cengage Learning © 2015

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

Decision Support Database Requirements

- Database schema
  - Must support complex, non-normalized data representations
  - Data must be aggregated and summarized
  - Queries must be able to extract multidimensional time slices

Decision Support Database Requirements

- Data extraction and loading
  - Allow batch and scheduled data extraction
  - Support different data sources and check for inconsistent data or data validation rules
  - Support advanced integration, aggregation, and classification
- Database size should support:
  - **Very large databases (VLDBs)**
  - Advanced storage technologies
  - Multiple-processor technologies

Table 13.8 - Characteristics of Data Warehouse Data and Operational Database Data

CHARACTERISTIC	OPERATIONAL DATABASE DATA	DATA WAREHOUSE DATA
Integrated	Similar data can have different representations or meanings. For example, Social Security numbers may be stored as ###-##-#### or as #########, and a given condition may be labeled as T/F or 0/1 or Y/N. A sales value may be shown in thousands or in millions.	Provide a unified view of all data elements with a common definition and representation for all business units.
Subject-oriented	Data are stored with a functional, or process, orientation. For example, data may be stored for invoices, payments, and credit amounts.	Data are stored with a subject orientation that facilitates multiple views of the data and decision making. For example, sales may be recorded by product, division, manager, or region.
Time-variant	Data are recorded as current transactions. For example, the sales data may be the sale of a product on a given date, such as $342.78 on 12-MAY-2014.	Data are recorded with a historical perspective in mind. Therefore, a time dimension is added to facilitate data analysis and various time comparisons.
Nonvolatile	Data updates are frequent and common. For example, an inventory amount changes with each sale. Therefore, the data environment is fluid.	Data cannot be changed. Data are added only periodically from historical systems. Once the data are properly stored, no changes are allowed. Therefore, the data environment is relatively static.

Cengage Learning © 2015

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

ETL is a classic "schema on write" process, where we define a schema (table structure) first, THEN load the data into that structure.

An ETL-related job ad:

DATA WAREHOUSE AND ANALYTICS DEVELOPERS (ETL/INFORMATICA)

Ascension Health-IS, Inc. is seeking two Data Warehouse and Analytics Developers (ETL/Informatica) in St. Louis, Missouri to code design and development on the data warehouse/ analytics Extract Transform Load (ETL) toolset, Informatica PowerCenter; support Informatica toolset; integrate and develop other technologies. Research solutions and technology; participate in testing (e.g. user acceptance testing, unit, system, regression, integration testing); develop test plans and documentation; debug code. Contact Jenna Mihm, Vice President Legal Services & Associate General Counsel, Ascension Health, 4600 Edmundson Road, St. Louis, MO 63134, 314-733-8692, Jenna.Mihm@ascensionhealth.org To apply for this position, please reference Job Number 03.

Data Marts

- Small, single-subject data warehouse subset
- Provide decision support to a small group of people
- Benefits over data warehouses
  - Lower cost and shorter implementation time
  - Technologically advanced
  - Inevitable people issues

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

Table 13.9 - Twelve Rules for a Data Warehouse

RULE NO.	DESCRIPTION
1	The data warehouse and operational environments are separated.
2	The data warehouse data are integrated.
3	The data warehouse contains historical data over a long time.
4	The data warehouse data are snapshot data captured at a given point in time.
5	The data warehouse data are subject oriented.
6	The data warehouse data are mainly read-only with periodic batch updates from operational data. No online updates are allowed.

Cengage Learning © 2015

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

Table 13.9 - Twelve Rules for a Data Warehouse

RULE NO.	DESCRIPTION
7	The data warehouse development life cycle differs from classical systems development. Data warehouse development is data-driven; the classical approach is process-driven.
8	The data warehouse contains data with several levels of detail: current detail data, old detail data, lightly summarized data, and highly summarized data.
9	The data warehouse environment is characterized by read-only transactions to very large data sets. The operational environment is characterized by numerous update transactions to a few data entities at a time.
10	The data warehouse environment has a system that traces data sources, transformations, and storage.
11	The data warehouse's metadata are a critical component of this environment. The metadata identify and define all data elements. The metadata provide the source, transformation, integration, storage, usage, relationships, and history of each data element.
12	The data warehouse contains a chargeback mechanism for resource usage that enforces optimal use of the data by end users.

Cengage Learning © 2015

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

Star Schema

- Data-modeling technique
- Maps multidimensional decision support data into a relational database
- Creates the near equivalent of multidimensional database schema from existing relational database
- Yields an easily implemented model for multidimensional data analysis

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

Components of Star Schemas

Facts

- Numeric values that represent a specific business aspect

Dimensions

- Qualifying characteristics that provide additional perspectives to a given fact

Attributes

- Used to search, filter, and classify facts
- **Slice and dice**: Ability to focus on slices of the data cube for more detailed analysis

Attribute hierarchy

- Provides a top-down data organization

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

28

FIGURE 13.8 Slice-and-dice view of sales

Cengage Learning © 2015

FIGURE 13.9
Location attribute hierarchy
The attribute hierarchy allows the end user to perform drill-down and roll-up searches.
Region
State
City
Store
Cengage Learning © 2015

FIGURE 13.10
Attribute hierarchies in multidimensional analysis
Time dimension
Year
Quarter
Month
Week
Product dimension
All products
By product type
One product
Q1 Q2 Q3 Q4
Total of product
Product A
Product B
Product C
Total of quarters
Location hierarchy
Region
State
City
Store
Cengage Learning © 2015

Star Schema Representation

- Facts and dimensions represented by physical tables in data warehouse database
- Many-to-one (M:1) relationship between fact table and each dimension table
- Fact and dimension tables
  - Related by foreign keys
  - Subject to primary and foreign key constraints

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

Star Schema Representation

- Primary key of a fact table
  - Is a composite primary key because the fact table is related to many dimension tables
  - Always formed by combining the foreign keys pointing to the related dimension tables

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

A sample star schema

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

We have the fact table (of transactions) at the center, and denormalized (all-in-one) dimension tables all around.

Here is another representation.

Note: "dimensions are qualifying characteristics that provide additional perspectives to a given fact; dimensions provide descriptive characteristics about the facts through their attributes."

Each fact (transaction) can now pictured to be located in a multi-dimensional cube where the axes are dimensions. Eg. a 3D representation of our data for the above schema would look like this:

Slicing and dicing the cube provides specific insights..

Additionally, an attribute hierarchy would provide drill-down/roll-up capability as well, eg.

FIGURE 13.9
Location attribute hierarchy
The attribute hierarchy allows the end user to perform drill-down and roll-up searches.
Region
State
City
Store
Cengage Learning © 2015

Snowflake schema

Dimensional tables can be normalized so that they have their own dimensional tables - this is done to simplify the design, but requiring navigation across the normalized chains.

Here is another representation.

Techniques Used to Optimize Data Warehouse Design

- Normalizing dimensional tables
  - **Snowflake schema**: Dimension tables can have their own dimension tables
- Maintaining multiple fact tables to represent different aggregation levels
- Denormalizing fact tables

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part. 32

There are four different ways in which we can organize (structure) a data warehouse:

- 1. star schema: fact table FKs point to a single level of dimension tables (points of a star)
- 2. snowflake schema: each dimension table can be normalized to create a 1:M chain
- 3. the fact table can be supplanted with all the columns in the star/snowflake dimensions
- 4. a separate fact table can be created for each attribute in a dimension hierarchy

To denormalize a fact table (#3 above), we simply add extra 'dimension' columns to it, and fill them with redundant data – this permits fast queries (no joins needed) at the expense of disk space (and cleanliness

of design).

Redundant fact tables!

Instead of a denormalized fact table (#3), or a fact table pointing to denormalized star dimensions (#1), or a fact table with lowest attrs pointing to a chain of rolled-up attrs, ie. snowflake schema (#2), we can create multiple fact tables, one for each level in an attr hierarchy (#4) - it is a different form of denormalization, where the redundant data is stored in physically separate tables.

A summarization...

Fact tables that we see in the middle of star/snowflake schema, are ALWAYS denormalized, with multiple repeating values in the columns that link to dimensions - eg. multiple date values, product values, location values, POS terminal # values etc (because each row in a fact table contains those columns as raw 'facts').

Dimension tables, in a star schema are ALSO denormalized - eg. location dimension, with city,state,region columns, will have repeating values for states (because many cities are in each state), and repeating region values (because many states are in each region).

Dimension tables in a snowflake schema are normalized, because we create a chain (hierarchy) of them using the star's dimension columns.

The fact table ALWAYS stays denormalized. Such a fact table is said to employ star schema, if we use star-like denormalized columns for BI - eg. to find out how much of a product we sold in a city, we'd query the fact rows for city name, and if we need it, can also do state-level analyses (because states are listed in the location dimension table).

Using a snowflake schema, doing location analysis for a product at a city level is similar to the above paragraph - we simply look for the city name, and if necessary, get extra info about the city (eg tax rate) by looking at the dimension table. BUT to do state level analysis, we need to follow the city->state link, and use the state-level dimension table ie traverse a branch of the snowflake.

To avoid traversing those branches in a snowflake, we trade off ('waste') space by creating extra 'copies' of the fact table, where a column such as city (lowest value in the hierarchy of 'location') is REPLACED instead with 'state' values, and in another copy, with 'region' values. This lets us do star-like analyses again, because a fact row directly points the state table, and in another copy, directly points to the region table - no traversing the chain necessary (at the expense of extra storage).

Which schema (star or snowflake) is used to model the warehouse, determines whether we maintain denormalized (or normalized) dimension tables [fact tables always stay denormalized]. 'For BI purposes, the idea is to take the 'single unified view' of data which is in the fact table (which contains numerous columns (think of a single Amazon purchase order item) - they can be categorized into dimensions, and in each dimension, even be hierarchically grouped - an example would be 'location'), and DERIVE additional tables, with data pre-aggregated along those (hierarchies of) dimensions. This lets us slice-and-dice (along dimensions), and zoom in/out (along just one dimension), all without expensive querying at runtime (on billions of rows), because the 'group by' calculations have been done already (that resulted in those aggregated data tables).'

Data Analytics

- Encompasses a wide range of mathematical, statistical, and modeling techniques to extract knowledge from data
  - Subset of BI functionality
- Classification of tools
  - **Explanatory analytics**: Focuses on discovering and explaining data characteristics and relationships based on existing data
  - **Predictive analytics**: Focuses on predicting future outcomes with a high degree of accuracy

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

Online Analytical Processing

- Advanced data analysis environment that supports decision making, business modeling, and operations research
- Characteristics
  - Multidimensional data analysis techniques
  - Advanced database support
  - Easy-to-use end-user interfaces

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

Figure 13.19 - OLAP Architecture

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

39

Figure 13.20 - OLAP Server with Local Miniature Data Marts

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

40

ROLAP, MOLAP

Relational OLAP ('ROLAP'):

Relational Online Analytical Processing (ROLAP)

- Provides OLAP functionality using relational databases and familiar relational tools to store and analyze multidimensional data
- Extensions added to traditional RDBMS technology
  - Multidimensional data schema support within the RDBMS
  - Data access language and query performance optimized for multidimensional data
  - Support for very large databases (VLDBs)

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part. 41

Multidimensional OLAP ('MOLAP'):

Multidimensional Online Analytical Processing (MOLAP)

- Extends OLAP functionality to multidimensional database management systems (MDBMSs)
  - **MDBMS**: Uses proprietary techniques store data in matrix-like n-dimensional arrays
  - End users visualize stored data as a 3D **data cube**
    - Grow to n dimensions, becoming hypercubes
    - Held in memory in a **cube cache** to speed access
- **Sparsity**: Measures the density of the data held in the data cube

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part. 42

Table 13.12 - Relational vs. Multidimensional OLAP

CHARACTERISTIC	ROLAP	MOLAP
Schema	Uses star schema Additional dimensions can be added dynamically	Uses data cubes Multidimensional arrays, row stores, column stores Additional dimensions require re-creation of the data cube
Database size	Medium to large	Large
Architecture	Client/server Standards-based	Client/server Open or proprietary, depending on vendor
Access	Supports ad hoc requests Unlimited dimensions	Limited to predefined dimensions Proprietary access languages
Speed	Good with small data sets; average for medium-sized to large data sets	Faster for large data sets with predefined dimensions

Cengage Learning © 2015

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

BI-oriented SQL extensions

ROLLUP and CUBE are GROUP BY modifiers - they help generate subtotals for a list of specified columns (see examples that follow). Depending on granularity of the columns (eg. US_REGION vs STORE_NUMBER), these subtotals help provide a rolled-up (aggregated) or drilled-down (detailed) analysis of data.

SQL Extensions for OLAP

The ROLLUP extension

- Used with GROUP BY clause to generate aggregates by different dimensions
- Enables subtotal for each column listed except for the last one, which gets a grand total
- Order of column list important

The CUBE extension

- Used with GROUP BY clause to generate aggregates by the listed columns
- Includes the last column

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

44

ROLLUP, CUBE: usage examples

ROLLUP extension

```
SELECT column1 [, column2, ...],
aggregate_function(expression)
FROM table1 [, table2, ...]
[WHERE condition]
GROUP BY ROLLUP (column1 [, column2, ...])
[HAVING condition]
[ORDER BY column1 [, column2, ...]]
```

Oracle SQL*Plus

File Edit Search Options Help

```
SQL> SELECT   V_CODE, P_CODE, SUM(SALE_UNITS*SALE_PRICE) AS TOTSALES
  2  FROM     DVDAYSALESFACT NATURAL JOIN DVPRODUCT NATURAL JOIN DVVENDOR
  3  GROUP BY ROLLUP (V_CODE, P_CODE)
  4  ORDER BY V_CODE, P_CODE;

   V_CODE P_CODE       TOTSALES
--------- ---------- ----------
    21225 23109-HB         99.5
    21225 PVC23DRT       199.58
    21225 SM-18277        41.94
    21225                341.02
    21344 13-Q2/P2       239.84
    21344 54778-2T        59.88
    21344                299.72
    23119 1546-QQ2        79.9
    23119                 79.9
    24288 2232/QTY       219.84
    24288 89-WRE-Q       513.98
    24288                733.82
    25595 2238/QPD         77.9
    25595 WR3/TT3        719.7
    25595                797.6
                        2252.06

16 rows selected.
```

Subtotals by V_CODE

Grand total for all P_CODE values

Cengage Learning © 2015

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part. 45

"Subtotal for each vendor – all products; sum of (the only set of) subtotals".

CUBE extension

```
SELECT column1 [, column2, ...],
aggregate_function(expression)
FROM table1 [, table2, ...]
[WHERE condition]
GROUP BY CUBE (column1 [, column2, ...])
[HAVING condition]
[ORDER BY column1 [, column2, ...]]
```

©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part. 46

"Subtotal for each month – all products; subtotal for each product – all months; sum of (either set of) subtotals".

Data Lakes

A 'traditional' data warehouse is an ETL-based, historical record of transactions - very RDB-like (schema-on-write).

A 'modern' alternative is a 'data lake', which offers a more continuous form of analytics, driven by the rise of unstructured (semi-structured, really) data, streaming, cloud storage, etc. In a data lake, data is NOT ETLd, rather, it is stored in its 'raw' ("natural") form [even incomplete, untransformed...] - it is 'schema on read', where we create a schema AFTER storing (raw) data in a DB.

Also, look up 'lakehouse', 'reverse ETL'...

1/30 8:22:22 ***

← →

Spatial DBs

Objectives/TOC

- spatial DBs: definition, characteristics, need, creation..
- spatial datatypes
- spatial operators
- spatial indices
- implementations
- miscellany

What is a spatial database?

"A spatial database is a database that is optimized to store and query data related to objects in space, including points, lines and polygons."

In other words, it includes objects that have a SPATIAL location (and extent). A chief category of spatial data is geospatial data - derived from the geography of our earth.

Characteristics of geographic data:

- has location
- has size
- is auto-correlated
- scale dependent
- might be temporally dependent too

Geographic data is NOT 'business as usual'!

Entity view vs field view

In spatial data analysis, we distinguish between two conceptions of space:

- entity view: space as an area filled with a set of discrete objects
- field view: space as an area covered with essentially continuous surfaces

For our purposes, we will adopt the 'entity' view, where space is populated by discrete objects (roads, buildings, rivers..).

Components

So a spatial DB is a collection of the following, specifically built to handle spatial data:

- types
- operators
- indices

Soon, we will explore what types, operators and indices mean.

Examples of spatial data

CAD data:

Agricultural data:

3D data:

- Three-dimensional data examples
  - Weather
  - Cartesian coordinates (3-D)
  - Topological
  - Satellite images

What can be plotted on to a map?

- crime data
- spread of disease, risk of disease [look at this too]
- drug overdoses - over time
- census data
- income distribution, home prices
- locations of Starbucks (!)
- (real-time) traffic
- agricultural land use, deforestation

Who creates/uses spatial data?

- **Army Field Commander**: Has there been any significant enemy troop movement since last night?
- **Insurance Risk Manager**: Which homes are most likely to be affected in the next great flood on the Mississippi?
- **Medical Doctor**: Based on this patient's MRI, have we treated somebody with a similar condition ?
- **Molecular Biologist**:Is the topology of the amino acid biosynthesis gene in the genome found in any other sequence feature map in the database ?
- **Astronomer**:Find all blue galaxies within 2 arcmin of quasars.

Various government agencies routinely coordinate spatial data collection and use, operating in effect, a national spatial data infrastructure (NSDI) - these include federal, state and local agencies. At the federal level, participating agencies include:

- Department of Commerce
  - Bureau of the Census
  - NIST
  - NOAA
- Department of Defense
  - Army Corps of Engineers
  - Defense Mapping Agency
- Department of the Interior
  - Bureau of Land Management
  - Fish and Wildlife Service
  - U.S Geological Survey [earthquakes, map projections]
- Department of Agriculture
  - Agricultural Stabilization and Conservation Service
  - Economic Research Service
  - Forest Service
  - National Agriculture Statistical Service
  - Soil Conservation Service
- Department of Transportation
  - Federal Highway Administration
- Environmental Protection Agency
- NASA

As you can see, spatial data is a SERIOUS resource, vital to US' national interests.

Where does spatial data come from?

Spatial data is created in a variety of ways:

- CAD: user creation
- CAD: reverse engineering
- maps: cartography (surveying, plotting)
- maps: satellite imagery
- maps: 'copter, drone imagery
- maps: driving around
- maps: walking around

What to store?

All spatial data can be described via the following entities/types:

- points/vertices/nodes
- polylines/arcs/linestrings
- polygons/regions
- pixels/raster

Focus - Unnamed Project - Unnamed Map
File Edit View Layer Analysis Tools Help
Maps Files
Visualization Tools
Flicker Swipe Blend Loop Cycle
Direction: Horizontal Vertical
Position: 50 %
Auto Mode
Speed: 0.5 frame/sec
Actual speed: 0.5
Control:
Step size: 1 %
Apply and Close
Close
Scale: 94,900
x1.0
301694.362E 3951121.753N
G:107

Points, lines, polys => models and non-spatial attrs

Once we have spatial data (points, lines, polygons), we can:

- 'model' features such as lakes, soil type, highways, buildings etc, using the geometric primitives as underlying types
- add 'extra', non-spatial attributes/features to the underlying spatial data

Source: GAO.

Look at this map, overlaid with scary data..

SDBMS architecture

GIS vs SDBMS

GIS is a specific application architecture built on top of a [more general purpose] SDBMS.

GIS typically tend to be used for:

Search	Thematic search, search by region, (re-)classification
Location analysis	Buffer, corridor, overlay
Terrain analysis	Slope/aspect, catchment, drainage network
Flow analysis	Connectivity, shortest path
Distribution	Change detection, proximity, nearest neighbor
Spatial analysis/Statistics	Pattern, centrality, autocorrelation, indices of similarity, topology: hole description
Measurements	Distance, perimeter, shape, adjacency, direction

Spatial relationships

In 1D (and higher), spatial relationships can be expressed using 'intersects', 'crosses', 'within', 'touches' (these are T/F predicates).

Here is a sampling of spatial relationships in 2D:

Another diagram showing the [binary] operations:

Figure 2. Examples of the binary topological relationships (a) disjoint, (b) meet, (c) overlap, (d) covers/covered_by, (e) inside/contains, and (f) equal [24].

Minimum Bounding Rectangles (MBRs) are what are used to compute the results of operations shown above:

Spatial relations – categories

Spatial relationships can be:

- topology-based [using defns of boundary, interior, exterior]
- metric-based [distance/Euclidian, angle measures]
- direction-based
- network-based [eg. shortest path]

Topological relationships could be further grouped like so:

- proximity
- overlap
- containment

How can we put these relations to use?

We can perform the following, on spatial data:

- spatial measurements: find the distance between points, find polygon area..
- spatial functions: find nearest neighbors..
- spatial predicates: test for proximity, containment..

Spatial Data Entity Creation

- Form an entity to hold county names, states, populations, and geographies

```
CREATE TABLE County(
    Name      varchar(30),
    State     varchar(30),
    Pop       Integer,
    Shape     Polygon);
```

Spatial Data Entity Creation (Cont.)

- Form an entity to hold river names, sources, lengths, and geographies

```
CREATE TABLE River(
    Name      varchar(30),
    Source    varchar(30),
    Distance  Integer,
    Shape     LineString);
```


Example Spatial Query

- Find all the counties that border on Contra Costa county

```
SELECT    C1.Name
FROM      County C1, County C2
WHERE     Touch(C1.Shape, C2.Shape) = 1
          AND C2.Name = 'Contra Costa';
```

Example Spatial Query (Cont.)

- Find all the counties through which the Merced river runs

```
SELECT    C.Name, R.Name
FROM      County C, River R
WHERE     Intersect(C.Shape, R.Shape) = 1
          AND R.Name = 'Merced';
```

Spatial operators, functions

Spatial Functions

- Returns a geometry
  - Union
  - Difference
  - Intersect
  - XOR
  - Buffer
  - CenterPoint
  - ConvexHull
- Returns a number
  - LENGTH
  - AREA
  - Distance

Spatial Operators

- Full range of spatial operators
  - Implemented as functional extensions in SQL
  - Topological Operators
    - Inside Contains
    - Touch Disjoint
    - Covers Covered By
    - Equal Overlap Boundary
  - Distance Operators
    - Within Distance
    - Nearest Neighbor

```
#query

+ equals(another :Geometry) : Boolean
+ disjoint(another :Geometry) : Boolean
+ intersects(another :Geometry) : Boolean
+ touches(another :Geometry) : Boolean
+ crosses(another :Geometry) : Boolean
+ within(another :Geometry) : Boolean
+ contains(another :Geometry) : Boolean
...

#analysis

+ distance(another : Geometry) : Distance
+ buffer(another : Distance) : Geometry
+ convexHull() : Geometry
...
```

This doc [from 'FME Knowledge Center'; thanks to Minaxi Singla for the link] provides more info on the spatial operators.

Oracle Spatial

Oracle offers a 'Spatial' library for spatial queries – this includes UDTs and custom functions to process them.

SDO_GEOMETRY Object

- SDO_GEOMETRY Object

```
SDO_GTYPE          NUMBER
SDO_SRID           NUMBER
SDO_POINT          SDO_POINT_TYPE
SDO_ELEM_INFO      SDO_ELEM_INFO_ARRAY
SDO_ORDINATES      SDO_ORDINATE_ARRAY
```

- Example

```
SQL> CREATE TABLE states (
  2     state       VARCHAR2(30),
  3     totpop      NUMBER(9),
  4     geom        SDO_GEOMETRY);
```

SDO_GEOMETRY Object

- SDO_GTYPE – Defines the type of geometry stored in the object

GTYPE	Explanation
1 POINT	Geometry contains one point
2 LINESTRING	Geometry contains one line string
3 POLYGON	Geometry contains one polygon
4 HETEROGENEOUS COLLECTION	Geometry is a collection of elements of different types: points, lines, polygons
5 MULTIPOINT	Geometry has multiple points
6 MULTILINESTRING	Geometry has multiple line strings
7 MULTIPOLYGON	Geometry has multiple polygons

SDO_GTYPE

SDO_GTYPE	Four digit GTYPEs - Include dimensionality		
	2D	3D	4D
1 POINT	2001	3001	4001
2 LINESTRING	2002	3002	4002
3 POLYGON	2003	3003	4003
4 COLLECTION	2004	3004	4004
5 MULTIPOINT	2005	3005	4005
6 MULTILINESTRING	2006	3006	4006
7 MULTIPOLYGON	2007	3007	4007

Constructing Geometries

```
SQL> INSERT INTO LINES VALUES (
  2>     attribute_1, …. attribute_n,
  3>     SDO_GEOMETRY (
  4>       2002, null, null,
  5>       SDO_ELEM_INFO_ARRAY (1,2,1),
  6>       SDO_ORDINATE_ARRAY (
  7>         10,10, 20,25, 30,10, 40,10))
  8>     );
```

Spatial Operators

- Operators
  - SDO_FILTER
    - Performs a primary filter only
  - SDO_RELATE and SDO_<relationship>
    - Performs a primary and secondary filter
  - SDO_WITHIN_DISTANCE
    - Generates a buffer around a geometry and performs a primary and optionally a secondary filter
  - SDO_NN
    - Returns nearest neighbors

SDO_FILTER Example

- Find all the cities in a selected rectangular area
- Result is approximate

```
SELECT c.city, c.pop90
FROM proj_cities c
WHERE sdo_filter (
        c.location,
        sdo_geometry (2003, 32775, null,
          sdo_elem_info_array (1,1003,3),
          sdo_ordinate_array (1720300,1805461,
                              1831559, 2207250))
        ) = 'TRUE';
```

Hint 1: All Spatial operators return TRUE or FALSE. When writing spatial queries always test with = 'TRUE', never <> 'FALSE' or = 'true'.

SDO_RELATE Example

- Find all counties in the state of New Hampshire

```
SELECT c.county, c.state_abrv
FROM geod_counties c,
     geod_states s
WHERE s.state = 'New Hampshire'
 AND sdo_relate (c.geom,
                 s.geom,
                 'mask=INSIDE+COVEREDBY')
                    ='TRUE';
```

Note: For optimal performance, don't forget to index `GEOD_STATES(state)`

Relationship Operators Example

- Find all the counties around Passaic county in New Jersey:

```
SELECT /*+ ordered */ a.county
FROM geod_counties b,
     geod_counties a
WHERE b.county = 'Passaic'
    AND b.state = 'New Jersey'
    AND SDO_TOUCH(a.geom,b.geom) = 'TRUE';
```

- Previously:

```
. . .
AND SDO_RELATE(a.geom,b.geom,
     'MASK=TOUCH') = 'TRUE';
```


SDO_NN Example

- Find the five cities nearest to Interstate I170, ordered by distance

```
SELECT /*+ ordered */
       c.city, c.state_abrv,
       sdo_nn_distance (1) distance_in_miles
FROM geod_interstates i,
     geod_cities c
WHERE i.highway = 'I170'
  AND sdo_nn(c.location, i.geom,
             'sdo_num_res=5 unit=mile', 1)= 'TRUE'
ORDER by distance_in_miles;
```

- Note: Make sure you have an index on `GEOD_INTERSTATES(HIGHWAY)`.

SDO_WITHIN_DISTANCE Examples

- Find all cities within a distance from an interstate

```
SELECT /*+ ordered */ c.city
FROM geod_interstates i, geod_cities c
WHERE i.highway = 'I170'
   AND sdo_within_distance (
        c.location, i.geom,
       'distance=15 unit=mile') = 'TRUE';
```

- Find interstates within a distance from a city

```
SELECT /*+ ordered */ i.highway
FROM geod_cities c, geod_interstates i
WHERE c.city = 'Tampa'
   AND sdo_within_distance (
        i.geom, c.location,
       'distance=15 unit=mile') = 'TRUE';
```

Spatial Indexing

- Used to optimize spatial query performance
- R-tree Indexing
  - Based on minimum bounding rectangles (MBRs) for 2D data or minimum bounding volumes (MBVs) for 3D data
  - Indexes two, three, or four dimensions
- Provides an exclusive and exhaustive coverage of spatial objects
- Indexes all elements within a geometry including points, lines, and polygons

Optimized Query Model

Postgres PostGIS

Types of queries - PostGIS

The function names for queries differ across geodatabases. The following list contains commonly used functions built into PostGIS, a free geodatabase which is a PostgreSQL extension (the term 'geometry' refers to a point, line, box or other two or three dimensional shape):

Types of queries - PostGIS (Cont.)

1. Distance(geometry, geometry) : number
2. Equals(geometry, geometry) : boolean
3. Disjoint(geometry, geometry) : boolean
4. Intersects(geometry, geometry) : boolean
5. Touches(geometry, geometry) : boolean
6. Crosses(geometry, geometry) : boolean

Types of queries - PostGIS (Cont.)

7. Overlaps(geometry, geometry) : boolean
8. Contains(geometry, geometry) : boolean
9. Intersects(geometry, geometry) : boolean
10. Length(geometry) : number
11. Area(geometry) : number
12. Centroid(geometry) : geometry

Here is an example - table creation, and polygon insertion:

```
Create Table County (
name VARCHAR(30),
shape geometry);
CREATE TABLE


Insert into County values ('Lynn', ST_Polygon(ST_GeomFromText('LINESTRING(75.15 29.53 1,77 29 1,77.6 29.5 1,
 75.15 29.53 1)'),4326));
INSERT 0 1

SELECT *
FROM County;
 name |
          shape
------+-----------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------
-------------------
 Lynn | 01030000A0E6100000010000000400000009A99999999C9524048E17A14AE873D4000000000000000F03F0000000000405340000
0000000003D4000000000000000F03F6666666666666534000000000000803D40000000000000F03F9A99999999C9524048E17A14AE873D
40000000000000F03F
(1 row)


Saty@Satys_USC_PC ~
$
```

To do the above, here are the steps on a PC (similar steps on a Mac):

- install Postgres (v.9.5, not 9.6 beta!)
- bring up 'Application Stack Builder' (an add-on that gets installed when Postgres v9.5 is installed), from the available installation options that come up, pick Spatial Extensions -> 'PostGIS 2.2 for Postgres 9.5', install
- bring up a shell (I use 'cygwin'); note - if you want to use cygwin, be sure to use the shell that comes up when you run cygwin.bat, *not* the 'mintty' shell that you get when you double-click on the cygwin icon; Mac users would use the built-in shell
- 9.5/bin/initdb (on a Mac the path would be different)
- 9.5/bin/pg_ctl start - this starts the Postgres server
- 9.5/bin/createdb mydb - a new db for us to create tables in
- 9.5/bin/psql.exe -d mydb -c "CREATE EXTENSION postgis;" - this adds spatial types to our db; note: 'psql' is the program that lets us communicate with the db server, via the shell
- 9.5/bin/psql.exe -d mydb -a -f county.sql - this is how you can execute SQL commands that you store in a .sql file
- edit the .sql file (eg add more data [including spatial data], create new tables, write SQL queries [including spatial ones]..), run the file (as shown above), edit, run.....
- 9.5/bin/pg_ctl stop - optionally you can stop the server and restart it later
- ...

You can learn a lot about spatial queries from this page.

Creating spatial indexes

As (more so than) with non-spatial data, the creation and use of spatial indexes VASTLY speed up processing!

Can B Trees index spatial data?

In short, YES, if we pair it up with a 'z curve' indexing scheme (using a space-filling curve):

Organizing spatial data with space filling curves

- Issue:
  - Sorting is not naturally defined on spatial data
  - Many efficient search methods are based on sorting datasets
- Space filling curves
  - Impose an ordering on the locations in a multi-dimensional space
    - Examples: row-order (Fig. 1.11(a), z-order (Fig 1.11(b))
  - Allow use of traditional efficient search methods on spatial data

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

(a)

7	8	14	16
5	6	13	15
2	4	10	12
1	3	9	11

(b)

The idea is to quantize every (x,y) location into a recursively-divided 'quadtree' cell, and use the cell's binary (x,y) location to create a (binary) 'z' key, which is ordered along the unit (0..1) interval - in other words, 2D (x,y) points get mapped (indexed) to ordered 1D 'z' locations.

But, this is of academic interest mostly, not commonly practiced in industry - Apple's FoundationDB is an exception.

R trees

R trees use MBRs to create a hierarchy of bounds.

Variations, FYI: R+ tree, R* tree, Buddy trees, Packed R trees..

k-d trees, K-D-B trees

k-d tree

Alternate: K-D-B tree:

(a) Planar partition

(b) A hierarchical K-D-B-tree structure

Quadtrees (and octrees)

Here we recursively and adaptively subdivide space [subdivisions happen only where necessary].

A representation of how a quadtree divides an indexed area. Source: Wikipedia

Each node is either a leaf node, with indexed points or null, or an internal (non-leaf) node that has exactly 4 children. The hierarchy of such nodes forms the quadtree.

Indexing evolution

Indexing schemes continue to evolve.

Query processing: filter, refine

Query Processing

- Efficient algorithms to answer spatial queries
- Common Strategy - filter and refine
  - Filter Step:Query Region overlaps with MBRs of B,C and D
  - Refine Step: Query Region overlaps with B and C

Fig 1.8

Visualizing spatial data

A variety of non-spatial attrs can be mapped on to spatial data, providing an intuitive grasp of patterns, trends and abnormalities. Following are some examples.

Dot map:

Here's another one.

Proportional symbol map:

Diagram map:

Another diagram map:

Also possible to plot multivariate data this way.

Choropleth maps (plotting of a variable of interest, to cover an entire region of a map):

So who (else) has spatial extensions?

Everyone!

Thanks to SQL's facility for custom datatype ('UDT') and function creation ('functional extension'), "spatial" has been implemented for every major DB out there:

- Oracle: Locator, Spatial, SDO
- Postgres: PostGIS
- DB2: Spatial Datablade
- Informix: Geodetic Datablade
- SQL Server: Geometric and Geodetic Geography types
- MySQL: spatial library comes 'built in'
- SQLite: SpatiaLite
- ..

Google KML

Google's KML format is used to encode spatial data for Google Earth, etc. Here is a page on importing other geospatial dataset formats into Google Earth.

OpenLayers

OpenLayers is an open GIS platform.

ESRI: Arc*

ESRI is the home of the powerful, flexible family of ArcGIS products - and they are local!

QGIS etc.

There is a variety of inexpensive/open source mapping platforms, competing with more pricey commercial offerings (from ESRI etc). Here are several:

- QGIS
- MapBox
- Carto
- GIS Cloud

1/25 8:22:53 ***

← →

NoSQL

WHAT? Why NoSQL? Why not SQL? Is SQL going away??

SQL is dead! Long live SQL!

Relational DBs held sway for almost 3 DECADES: 80s, 90s, 00s.

What started to change was this - the Internet, coupled with storage and processing revolution has enabled:

- BigUsers - LOTS of people using databases
- BigData - LOTS of data being generated
- BigVariety - there is a huge VARIETY in the types of data being stored and searched
- BigFlux - there is rapid ADDITION, and CHANGE, in a lot of data being handled

The last three items above, constitute the 3 Vs of Big Data: volume, variety, velocity

Big Data comes from a variety of sources:

- people: web browsing 'clickstreams'
- people: purchasing etc. habits - shopping, dining..
- people: social media [communications]
- people: surveys, polls, census, court docs, employment histories, credit ratings..
- people etc: genomic data
- people: entertainment/education..
- people + devices: medical, fitness etc. data
- people + devices: transportation [cameras, sensors, GPS..]
- devices: scientific instruments
- devices: sensors ("IoT")

Big Data can lead to, or result from, 'datafication'.

Wikipedia: Datafication is a modern technological trend turning many aspects of our life into computerised data and transforming this information into new forms of value. Examples of datafication as applied to social and communication media are how Twitter datafies stray thoughts or datafication of HR by LinkedIn and others.

In other words, it is the notion that people, our built envrironment (eg. number of freeways in the US), etc. can lead to data generation.

"Once we datafy things, we can transform their purpose and turn the information into new forms of value."

RDBMSs are simply unsuitable for the above (for handing Big Data)! A different kind of NON-RELATIONAL database scheme was needed - enter 'NoSQL' DBs.

Again - what is happening is this:

- lots of new data, new types of data, are being rapidly generated
- developers are finding it hard to 'shoehorn' all this data into a relational model
- also hard to scale up to fit more data, more users
- and, hard to keep up performance too

So we need the following [ie. 'one-size-fits-all' was wrong (Stonebraker)]:

- avoidance of complexity [eg. no need to worry about immediate consistency]
- high throughput
- easy, quick scalability
- ability to run on commodity hardware
- avoidance of need for O-R mapping
- avoidance of complexity associated with cluster setup
- ability to use DB APIs for various programming languages, that mirror the languages' own structures

Need a **flexible, efficient, available, scalable** solution/DB design! THAT is what NoSQL provides - **high performance, high availability at a large scale**.

The term NoSQL was used as early as.. 1998!

The term started to become reused in 2009, by Last.fm's developer, and subsequently by a Rackspace employee blogger who popularized it.

Today, NoSQL refers to an umbrella of technologies that are all non-relational-DB-oriented.

So - what does 'No' stand for? NoSQL DB means/meant:

- Non relational, non SQL
- NO SQL
- NotOnly SQL
- ..

NoSQL DBs

A NoSQL DB is:

- schema-less: no tables, no relations!
- flexible: easy to add new types of data - which are semi-structured (aka unstructured) [as opposed to being structured]
- (data) scalable: specifically, ability to 'scale out', ie. do 'horizontal scaling' - both terms means that we can simply add more nodes (eg. servers) to an existing cluster, to accommodate more users, or to add more data to existing users.
- fast: easy to process large (massive) volumes of data

JSON (or XML) for storing an ENTIRE db!!

db == JSON []array of {}objects, where each object ('row') is a set of a key-value pairs.

http://www.data.gov has almost 200,000+ datasets, 50,000+ of which are in JSON.

Examples: https://api.publicapis.org/random, https://health.data.ny.gov/api/views/es3k-2aus/rows.json

JSON viewer: https://bytes.usc.edu/~saty/tools/jsoned or http://jsonviewer.stack.hu/ etc.

The simplicity of the JSON scheme also makes it possible/easy to add extra data (eg metadata) to the files.

JSON can also be returned by webservers, which can even call a client function with the returned data - this is called JSON-P.

JSON can used to describe structured data in a specific format - this is called JSON-LD.

It is very straightforward to represent table data as JSON:

Note - we also have the choice of using a 'related model' (where a JSON value is contained in a related (pointed-to) structure), instead of a 'nested model' (representing joined data):

Big idea: since a JSON value can be a string, a number, a boolean, an array, or another object [a key is ALWAYS a string], **this makes for a powerful representation!**

Here is a JSON file, not 'pretty printed' - we can always format such data, using an online formatter.

Here are MANY more JSON files!!

An XML db example

Compared to JSON, XML is more verbose (on account of opening and closing tags), but is a popular alternative format for creating DBs. Here is an example file (in addition to viewing the data here in the browser, you can use a viewer such as this one to view the XML file, or you can save locally and open with a text editor).

BASE, not ACID

The NoSQL DBs are characterized by their 'BASE' property, in contrast with RDBMS' 'ACID' property.

BASE stands for BAsic availability (db is up most of the time), Soft state (of consistency - consistency is not guaranteed while data is written to a node, or between replicas), Eventual consistency (at a later point in time, by push or pull, data will become consistent across nodes and replicas).

ACID	BASE
Strong consistency	Weak consistency – stale data OK
Isolation	Availability first
Focus on "commit"	Best effort
Nested transactions	Approximate answers OK
Availability?	Aggressive (optimistic)
Conservative (pessimistic)	Simpler!
Difficult evolution (e.g. schema)	Faster
	Easier evolution

A NoSQL database does not have an explicit schema that describes the relationships between its data items - it is said to be 'schema-less'

But, calling this situation 'schema-less' is not quite appropriate - rather, the schema is **implicit** - the schema resides in the application code that reads [and writes] data - so, 'scheme-on-read' is a better way to describe how we handle such data [in comparison, RDBMS is 'schema-on-write'].

To put it differently, the DB itself "doesn't care" about what it is storing, it is the application code that imparts meaning to the data. As a result, changing the data model (eg. adding or deleting an attribute) is trivial - just write and run (application) code to make the change in the DB!

In a schema-less environment, developers use intuitive data structures (well supported by underlying host languages) to do data manipulation (including querying and updating).

Eg. JSON is very easy/intuitive to grasp, comprising of just **six** underlying datatypes (number, string, boolean, array, object, null). It is also quite easy to parse.

Types of NoSQL databases (based on underlying data model)

There are [only] FOUR types of NoSQL DBs, based on their underlying data representation:

- key-value store: DynamoDB (Amazon), Project Voldemort, Redis, Tokyo Cabinet/Tyrant..
- column-family store: Cassandra, HBase..
- document store: MongoDB, CouchDB, MarkLogic..
- graph store: Neo4j, HyperGraphDB, Seasame..

Let's take a look at the above in more detail..

HOW MANY of these (NoSQL DB implementations) are there? :)

Martin Fowler (of 'Code Refactoring' fame) has an overview article on post-RDBMS alternatives, at http://martinfowler.com.

Polyglot persistence means the use of different storage technologies (ie. NoSQL DBs) to store different parts (data) of a single application. These individual data stores are then tied together by the application, using DB APIs or web services APIs.

In other words, a single application can "speak" several different storage technologies, each one leveraging its strength (eg. a key-value DB can be used for storing transient user session data). Here is a visual, from Martin's article linked above:

Key-value DBs

"A key-value store, or key-value database, is a data storage paradigm designed for storing, retrieving, and managing associative arrays, a data structure more commonly known today as a dictionary or hash."

So the whole db is a dictionary, which has records ("rows") which have fields (columns).

Because there is no schema, the records all don't have to contain identical fields!

The 'key' in a key-value DB, is comparable to a PK in a relation (table); the 'value' is an aggregate of all the dependent (non-PK) columns.

Querying occurs only on keys. When querying on a key, the entire value (aggregate) (for matching keys) is returned. Likewise, the entire value needs to be updated when necessary (no per-field updating).

Rapid querying of keys is possible, on account of k-v stores employing disk-based Log Structured Merge Trees (LSMs) for search - more here.

Memcached [aka memcached - all lowercase] is a high-performance, in-memory, data caching system, with a VERY simple API:

- store (SET) a value, given a key (eg. $memcache->set(\text{key},val)$)
- retrieve (GET) the value, given the key (eg. $val = \text{memcache->get}(key)$)

The value that is stored does not have to be atomic, it can even be an associative array (k-v pairs, aka dictionary, or hash) - that said, stored values are usually rather small (~1M). Eg. the following PHP snippet shows how an array of 6 different elements [int, float.. object] is stored in memcached, as 6 k-v pairs [note that we could have stored the entire array as a single value, but that would make it inefficient to access an individual element such as the boolean]:

```
$checks = array(
    123,
    4542.32,
    'a string',
    true,
    array(123, 'string'),
    (object)array('key1' => 'value1'),
);
foreach ($checks as $i => $value) {
    print "Checking WRITE with Memcache\n";
    $key = 'cachetest' . $i;
    $memcache->set($key, $value);
    usleep(100);
    $val = $memcache->get($key);
    $valD = $memcacheD->get($key);
    if ($val !== $valD) {
        print "Not compatible!";
        var_dump(compact('val', 'valD'));
    }
```

Memcached is commonly used with a 'backend' SQL store (relational DB) - a COPY of the frequently accessed data is held in memcached, for fast retrieval and update.

Quite a few languages are supported: C/C++, Java, JavaScript, Python, PHP, Ruby, Perl, .NET, Erlang, Lua, Lisp, Ocaml..

If you have access to a Linux server (including being able to run one locally on your machine), you can experiment with memcached like so.

Redis is an extremely popular k-v database, used by a variety of (social media etc.) companies to hold vast amounts of data.

For values, rather than just atomic datatypes (number, string, boolean), Redis offers richer types such as list, set, dictionary.

Dynamo is what is internally used at Amazon (as distinct from S3, offered to cloud services users).

From the Dynamo paper that they published:

- the infrastructure is made up by tens of thousands of servers and network components located in many datacenters around the world.
- commodity hardware is used.
- component failure is the 'standard mode of operation'.
- 'Amazon uses a highly decentralized, loosely coupled, service oriented architecture consisting of hundreds of services'.

"Dynamo is used to manage the state of services that have very high reliability requirements and need tight control over the tradeoffs between availability, consistency, cost-effectiveness and performance."

In Dynamo, values are stored as BLOBS (opaque bytes of binary data). Operations are limited to a single k/v pair at a time.

Only two API ops are supported:

- get(key) - returns a list of objects and a context
- put(key, context, object) - no return value

In the above, 'context' is used to store metadata about the BLOB values, eg. version numbers (this is used during 'eventual consistency' DB updating).

Keys are hashed using the MD5 hashing algorithm, to result in appropriate storage locations (nodes).

To ensure availability and durability, each k/v pair is replicated N times (N=3, commonly) and stored in N adjacent nodes starting from the key's hash's node.

Here is a short tutorial on Dynamo - please go through it (just read it), for usage examples of the simple API [in section 3.3: putItem(), getItem(), deleteItem()].

Summary - k/v DBs are lightweight (simple), schema-less, transaction-less.

Column family DBs

After k/v DBs, column (aka column family) DBs are the next form of data storage.

Examples: BigTable(Google), BigQuery(Google), HBase(Apache), Cassandra(Apache), SimpleDB(Amazon), RedShift(Amazon).

Rather than dealing with rows of data, we deal with columns of them. So such databases are good for aggregate queries (eg. average age of employees in a company), and queries involving just a subset of all columns (eg. retrieve a student's academic info [but not personal info]).

Column family DBs - terminology

As the name signifies, data is stored as groups of columns (column family), as opposed to an RDBMS where data is stored as rows.

In a column family DB, data is stored using 'row keys' (each row (picturing the data as a classic relational table) is assigned a unique key, analogous to a PK).

Column: consists of a name (key) and a value; there is such a name:value pair for each row key, giving us a single column's worth of data for all rows (eg. GPAs of all students at USC).

Column family: contains columns of related data (eg. for a 'Users' DB, the columns might be Name, Age, DOB, Address, Email), for all rows. A column family would have many rows of data, where for each row, there would be multiple columns and values. Eg:

```
Cath:{
         firstname:'Cath',
         lastname:'Yoon'
     }
,
Terry:{
          firstname:'Terry',
          lastname:'Cho'
      }
```

Supercolumn: is a named grouping of columns; consists of a key-value pair, where the key (which would be the supercolumn's name) is used to index values consisting of raw column data, itself in the form of key-value pairs [in other words, the value looks like column family data]. Supercolumns are how we group related columns. Eg. here is a supercolumn called username, used to group the firstname and lastname columns (rather confusingly, shown just for a single, implied, rowkey, of a column family):

```
username:{firstname:'Cath', lastname:'Yoon'}
```

In other words, a supercolumn is a k:v pair where the key is the name of the supercolumn, and value would be column family data.

Supercolumn family: a collection of supercolumns. As noted above, a supercolumn has a name (key), and contains as its value, a group of actual column names and their values (ie a column family). So a supercolumn family can be thought of as a 'namespace' for a group of supercolumns (it is also a k:v pair). **Note that a supercolumn family is ALSO equivalent to a table, but with extra info (which would be the names of its supercolumns).** Eg. here is a supercolumn family (called UserList, with the "extra info" being supercolumn names 'username', 'address', 'account'):


```
UserList={
   Cath:{
     username:{firstname:'Cath',lastname:'Yoon'}
     address:{city:'Seoul',postcode:'1234'}
   },
   Terry:{
     username:{firstname:'Terry',lastname:'Cho'}
     account:{bank:'hana',accountID:'5678'}
   }
 }
```

In the above, 'UserList' is the supercolumn family name, pointing to supercolumns called 'username' and 'address'.

Pictorially, the data for 'Cath' above (for ex) would look like this:

Supercolumn families are meant for 'inverted indexing' [which is an index of terms that point to possibly multiple documents where they occur; in contrast, in a regular document index, we simply store the locations (eg. page numbers) of terms occurring in that document], we don't always *need* to use them in a column family DB [they are optional].

Here is another example of a column family (two column families, actually - Musician, Band) - note that ColumnFamily1 (Musician) has 'jagged' (uneven) data:

```
Musician:                        ColumnFamily 1
    bootsy:                      RowKey
        email: bootsy@pfunk.com,  ColumnName:Value
        instrument: bass         ColumnName:Value
    george:                      RowKey
        email: george@pfunk.com   ColumnName:Value

Band:                            ColumnFamily 2
    george:                      RowKey
        pfunk: 1968-2010         ColumnName:Value
```

Summary: in a column family, row keys directly contain (as values) column data, similar to a rectangular table; in contrast, in a supercolumn family, row keys contain (as values), k:v pairs, with supercolumn ("column group") keys, and 'column_name:column_value' values [so there is an extra level of indirection, provided by the supercolumn names]. Column family -> column data; supercolumn family -> supercolumns -> column data.

Column family DBs - storing tweets

Here is how we'd store tweets, using a column family DB [example from https://ayende.com/blog/4500/that-no-sql-thing-column-family-databases]. We create two column families - Users, Tweets. We also create a supercolumn family - UsersTweets.

Here are Users and Tweets:

Key	@ayende
Columns	

Location	Israel
Name	Ayende Rahine
Profession	Wizard

Key	Tweets/00000000-0000-0000-0000-000000000001
Data	

Application	TweetDeck
Private	true
Text	Err, is this on?

Key	Tweets/00000000-0000-0000-0000-000000000002
Data	

App	TweetDeck
Public	true
Text	Well, I guess this is my mandatory hello world
Version	1.2

And here is the UsersTweets supercolumn family (with Timeline as the supercolumn name, which groups two regular columns [those names are not shown, just their values are], 'guid' and 'tweetid'):

Key	@ayende
Data	

Timeline	
Timeline/00000000-0000-0000-0000-000000000003	Tweets/00000000-0000-0000-0000-000000000001
Timeline/00000000-0000-0000-0000-000000000004	Tweets/00000000-0000-0000-0000-000000000002

To query the above, we'd do this:

```
var tweetIds =
      cfdb.UsersTweets.Get('@ayende')
              .Fetch('timeline')
              .Take(25)
              .OrderByDescending()
              .Select(x=>x.Value);

var tweets = cfdb.Tweets.Get(tweetIds);
```

Note that we execute queries at two levels - from the UsersTweets super column family we get tweetIds as VALUES, which we then use as KEYS in the Tweets family.

Column family DBs - BigTable, Cassandra

BigTable - "a distributed storage system for managing structured data that is designed to scale to a very large size: petabytes of data across thousands of commodity servers".

Used in Google Earth, Google Analytics, Google Docs, etc.

"BigTable has achieved several goals: wide applicability, scalability, high performance, and high availability."

The storage data structure is "a sparse, distributed, persistent multidimensional sorted map". Example:

Values are addressed by (row-key, column-key, timestamp).

In the above, we have two column families: content (one column), anchor (two columns).

Hypertable

Written in C++, based on HDFS and distributed lock managing.

Can have column-families with an arbitrary number of distinct columns.

Tables are partitioned by ranges of row keys (like in BigTable) and the resulting partitions get replicated between servers.

Features HQL, a C++ native API and the 'Thrift' API.

HBase

BigTable clone written in Java, as part of the Hadoop implementation.

An HBase db can be the source or target for a MapReduce task running on Hadoop.

Has a native Java API, Thrift API and REST web services.

Cassandra was originally developed by Facebook and open-sourced in 2008.

Also used by Twitter, Digg, Rackspace.

Data model is as follows:

Rows - which are identified by a string-key of arbitrary length. Operations on rows are atomic per replica no matter how many columns are being read or written.

Column Families - which can occur in arbitrary number per row.

Columns have a name and store a number of values per row which are identified by a timestamp (like in Bigtable). Each row in a table can have a different number of columns, so a table cannot be thought of as a rectangle. Client applications may specify the ordering of columns within a column family and supercolumn which can either be by name or by timestamp.

Supercolumns have a name and an arbitrary number of columns associated with them. Again, the number of columns per super-column may differ per row.

Cassandra's core API is very simple:

- get()
- insert()
- delete()

Thrift API, plus language bindings for Ruby, Python, C#, Java.

Cassandra's query language is "CQL" - somewhat like SQL, but no WHERE, JOIN, GROUP BY, ORDER BY; also, results are returned as JSON. UPDATE: "these days", Cassandra DOES implement full SQL-like functionality.

```
CREATE KEYSPACE animalkeyspace
WITH REPLICATION = { 'class' : 'SimpleStrategy' ,
 'replication_factor' : 1 };

use animalkeyspace;

CREATE TABLE Monkey (
  identifier uuid,
  species text,
  nickname text,
  population int,
  PRIMARY KEY ((identifier), species));

INSERT INTO monkey (identifier, species, nickname, population)
VALUES ( 5132b130-ae79-11e4-ab27-0800200c9a66,
'Capuchin monkey', 'cute', 100000);

Select * from monkey;

./sstable2json $YourDataDirectory/data/animalkeyspace/monkey/animalk
eyspace-monkey-jb-1-Data.db

[
  {
    // The row/partition key
    "key": "5132b130ae7911e4ab270800200c9a66",
    // All Cassandra internal columns
    "columns": [
      [
        // The Cluster key. Note the cluster key does not have any d
ata associated with it. The key and the data are same
        "Capuchin monkey:",
        "",
        // Time stamp which records when this internal column was cr
```



```
eated.
        1423586894518000
      ],
      [
        // Header for the nickname internal column. Note the cluster
key is always prefixed for every additional internal column.
        // Actual data
        "Capuchin monkey:nickname",
        "cute",
        1423586894518000
      ],
      [
        // Header for the population internal column
        "Capuchin monkey:population",
        // Actual Data
        "100000",
        1423586894518000
      ]
    ]
  }
]
```

Document DBs: terminology

A document DB is a 'collection' of 'documents' (analogy with an RDMS: documents are equivalent to rows, and a collection is equivalent to a table).

The basic unit of storage in a document DB is, well, a document - this can be JSON, XML, etc. There can be an arbitrary number of fields (columns and values, ie. k/v pairs) in each document.

A document DB can be considered to be a more sophisticated version of a k-v store.

In a document DB, a key is paired with a document (which is its 'value'), where the document itself can contain multiple k/v pairs, key-array pairs, or even key-document pairs (ie nested documents).

Here are leading document-oriented DBs:

- Couchbase
- CouchDB ['Cluster of unreliable commodity hardware' :) :)]
- MongoDB
- OrientDB

You might enjoy this Couchbase application note :)

Who uses document DBs? LOTS of organizations, including eBay, ADP, MetLife, MTV, BuzzFeed..

Because entire documents are stored, there is no need to perform (expensive) JOIN operations.. Also, queries can be parallelized using MapReduce.

Document DB: example representation

Consider the following diagram, which shows how blog posts could be organized:

The corresponding document store would look like this:

Note that document contents are saved as BSON (Binary JSON) in order to save disk space. As a reminder, values can be atomic data, arrays or objects; fields can vary from document to document (no set schema).

Document DBs: querying

Queries are non-SQL, of course, and offer richness and flexibility. For example, MongoDB offers the following query types:

- simple k/v queries - can return a value for any field in the stored documents; usually, we do a simple PK search (given the doc's key, retrieve the entire doc)
- range queries: return values based on >, <=, BETWEEN..
- geo-spatial queries
- text queries - full text search, using AND, OR, NOT
- aggregation framework: column-oriented operations such as count, min, max, avg
- MapReduce queries - get executed via MapReduce

The following query snippets are for MongoDB.

```
// create a collection ("table")
db. createCollection (, {< configuration parameters >})


// a sample doc
{
title : " MongoDB ",
last_editor : "172.5.123.91" ,
last_modified : new Date ("9/23/2010") ,
body : " MongoDB is a..." ,
categories : [" Database ", " NoSQL ", " Document Database "] ,
reviewed : false
}

// add a doc into a coll
db.< collection >. insert ( { title : " MongoDB ", last_editor : ... } )
;

// retrieve
db.< collection >. find ( { categories : [ " NoSQL ", " Document Databas
es " ] } );

db.< collection >. find ( { title : " MongoDB " );

db.< collection >. find ( { where:" this .a ==1" } );

// array size comparison [num categories == 2]
{ categories : { size:2} }

// results processing
db.< collection >. find ( ... ). sort ({< field >: <1| -1 >}). limit ().
skip ();
```

```
// potentially DANGEROUS!
db. eval ( function (< formal parameters >) { ... }, );

// MapReduce
db.< collection >. mapreduce ( map : < map - function >,
reduce : < reduce - function >,
query : < selection criteria >,
sort : < sorting specificiation >,
limit : < number of objects to process >,
out: < output - collection name >,
outType : <" normal "|" merge "|" reduce ">,
keeptemp : < true |false >,
finalize : < finalize function >,
scope : < object with variables to put in global namespace >,
verbose : < true |false >
);
```

Here are the three types of NoSQL DBs we just looked at (k-v, document, column) in terms of the CAP theorem.

Graph DBs

A graph is a data structure comprised of vertices and edges:

A graph database uses (contains) graph entities such as nodes (vertices), relations (edges), and properties (k-v pairs) on vertices and edges, to store data.

A graph DB is said to be 'index free', since each node directly stores pointers to its adjacent nodes.

In a graph db, the focus is on relationships between 'linked data'.

A rich variety of graphs can be stored and manipulated - directed graphs, trees, weighted graphs, hypergraphs, etc.

Note that in addition to internal names/IDs, nodes and edges can have properties (attributes and their values, ie. k/v pairs).

It is useful to ponder how you would specify a graph in a certain format - a pre-determined format would make it possible for an application to read and write graphs.

- how would you describe a graph (named nodes, connected by not-named edges), using JSON? There are at least two ways...
- look up 'dot' for specifying graphs - eg. read this and this
- look up the .gdf and GraphML formats for describing graphs - eg. read this

Graph DBs: implementations

These are popular:

- FlockDB (from Twitter)
- Neo4J
- HyperGraphDB
- InfiniteGraph
- InfoGrid (makers of MeshBase and NetMeshBase)
- OrientDB [wasn't this also listed as a document DB?!]
- Giraph (from Apache)
- GraphLab

Uses: social networks, recommendation engines..

Gartner: "Graph analysis is possibly the single most effective competitive differentiator for organizations pursuing data-driven operations and decisions, after the design of data capture."

GraphDBs: comparisons with relational modeling

Compared to a relational scheme, a graph offers a compact, normalized, intuitive way of expressing connections/relationships.

Here is a relational way to express employee-dept relations, and the corresponding graph-based way:

Each row in a table becomes a node, and columns (and their values), node properties.

Here is an E-R diagram, and a graph version:

As you can see, the graphs help model relationships in the form of connections between nodes.

Graph DBs: querying

It is pretty straightforward to create nodes and edges, and attach properties to them; after that, it is equally simply to do queries on the resulting database.

In Neo4J, we use Cypher, a declarative graph query language - modeled after SQL, augmented with graph-specific functionality.

In the following, we compare a SQL query, and its Cypher equivalent:

```
SELECT name FROM Person
LEFT JOIN Person_Department
  ON Person.Id = Person_Department.PersonId
LEFT JOIN Department
  ON Department.Id = Person_Department.DepartmentId
WHERE Department.name = "IT Department"

MATCH (p:Person)<-[:EMPLOYEE]-(d:Department)
WHERE d.name = "IT Department"
RETURN p.name
```

Using the Neo4J JDBC driver, a query to return "John's dept" can be embedded in Java:

```
Connection con = DriverManager.getConnection("jdbc:neo4j://localhost:747
4/");

String query =
    "MATCH (:Person {name:{1}})-[:EMPLOYEE]-(d:Department) RETURN d.name
as dept";
try (PreparedStatement stmt = con.prepareStatement(QUERY)) {
    stmt.setString(1,"John");
    ResultSet rs = stmt.executeQuery();
    while(rs.next()) {
        String department = rs.getString("dept");
        ....
    }
}
```

GraphDBs: TinkerPop (Gremlin)

TinkerPop is a very interesting graph traversal language (actually, an entire graph computing framework). Here is how to get started on exploring it.

Aside: here is an interesting use case, for a graph DB [notes here].

What is a triple store?

A triple store (or triplestore, or RDF) database stores triples of (subject,predicate,object) [or equivalently, (subject,attribute(property),value)]. A triple defines a **directed binary relation**, via its predicate/attribute/property. In relational form, we express this as predicate(subject,object).

Subject: what we are describing.

Predicate: a property of the subject.

Object: the predicate's (property's) value.

In a triple, the predicate is given equal status to subject and object [upcoming examples will make this clear].

As an aside, if a fourth attribute (context) is also stored, then we'd call the DB a quad store, or 'named graph'. There are even 'quints' (with an extra 'name' or 'ID' attribute).

We issue 'semantic queries' to search a triple store DB.

Note that a triple store DB is a restricted form of a graph DB.

Here is an example of a triple store - it is a flat (non-hierarchical) list (bag) of triplets, specified as subject (node), predicate (relationship), object (another node). The column on the left shows node IDs, that's not part of the triple.

```
<triple 32: "person2" "type" "person">
<triple 33: "person2" "first-name" "Rose">
<triple 34: "person2" "middle-initial" "Elizabeth">
<triple 35: "person2" "last-name" "Fitzgerald">
<triple 36: "person2" "suffix" "none">
<triple 37: "person2" "alma-mater" "Sacred-Heart-Convent">
<triple 38: "person2" "birth-year" "1890">
<triple 39: "person2" "death-year" "1995">
<triple 40: "person2" "sex" "female">
<triple 41: "person2" "spouse" "person1">
<triple 58: "person2" "has-child" "person17">
<triple 56: "person2" "has-child" "person15">
<triple 54: "person2" "has-child" "person13">
<triple 52: "person2" "has-child" "person11">
<triple 50: "person2" "has-child" "person9">
<triple 48: "person2" "has-child" "person7">
<triple 46: "person2" "has-child" "person6">
<triple 44: "person2" "has-child" "person4">
<triple 42: "person2" "has-child" "person3">
<triple 60: "person2" "profession" "home-maker">
```

The beauty is that such a triplet list can be grown 'endlessly', eventually connecting EVERYTHING to EVERYTHING ELSE! There is no schema to modify - just keep adding triplets to the DB!

Triple store database implementations

- AllegroGraph
- MarkLogic
- SparkleDB
- Stardog (http://stardog.com/)

Triplet store DBs: querying

Querying a triplet store can be done in one of several RDF (what is THAT?) query languages, eg. RDQL, SPARQL, RQL, SeRQL, Versa.. Of these, SPARQL is currently the most popular.

The output of a triple store query is called a 'graph'.

Queries tend to span disparate data, perform complex rule-based logic processing or inference chaining (AI-like).

Eg. given

```
:human rdfs:subClassOf :mammal
:man rdfs:subClassOf :human
```

an RDF database can infer

```
:man rdfs:subClassOf :mammal
```

If you want LARGE triple store datasets to play with, look at DBpedia, which is all of Wikipedia in RDF form!

Triple store DBs: equivalent to a form of RDF databases!

RDF (Resource Description Framework) is a metadata data model - a set of specifications (from W3C), for modeling information, for use in KM applications and the semantic web.

Specifically, one of RDF's serialization formats is N-Triples, which is *exactly* what store in our triple store DBs - subject,predicate,object. In that sense, every triple store DB is an RDF DB as well (but not the other way around).

Here is a small graph that shows a single transaction at a barbershop:

Here is the N-Triples representation for the above graph:

```
# barbershop.nt
# N-triples representation
<mailto://jim@FAKEDOMAIN.com> <http://www.example.com/barbershop/Properties#receives> _:service
<mailto://jim@FAKEDOMAIN.com> <http://www.example.com/barbershop/Properties#name> "Jim"
_:service <http://www.example.com/barbershop/Properties#cost> "10"
_:service <http://www.example.com/barbershop/Properties#date> "2010-07-31"
_:service <http://www.example.com/barbershop/Properties#providedBy> <http://www.example.com/barbersh
op/barbers#Ted>
<http://www.example.com/barbershop/Barbers#Ted> <http://www.example.com/barbershop/Properties#name>
"Ted"
```

If you want to play with RDF, try RDF4J [formerly, 'OpenRDF Sesame'].

Triple store DBs: architecture

These databases are set up to run in one of three modes:

- in-memory: triples are stored in main memory
- native store: persistence provided by the DB vendors, eg. AllegroGraph
- non-native store: uses third-party service, eg. Jena SDB uses MySQL as the store

1/20 8:23:21 ***

← →

"Map(Shuffle)Reduce"

[(big) data processing 'at scale']

Comp. machinery to process large volumes of data

Modern databases store data as key/value pairs, resulting in explosive growth when it comes to number of rows and file sizes.

Traditional, sequential, single machine oriented access does NOT work at all - what is needed is a massively parallel way to process the data.

Note that we are talking about SIMD form of parallelism.

Aside: functional constructs

Before we look at 'MapReduce' etc., let us take a side trip and look at some functional programming features.

In functional programming, functions are first-class objects that can be passed into a function as arguments; a function can also be returned from a function as output. JavaScript, Python etc. are languages that provide functional programming facilities.

In Python, the built-in functions map(), filter() and reduce(), all accept a function as input.

lambda

A lambda operator lets us define anonymous, JIT functions, eg.

```
>>> f = lambda x, y : x + y # f is a var that receives an anonymous fn assignment

>>> f(1,1)
2
```

We can use lambda to define a function inside another function's parameter list, as shown below.

map

```
r = map(func, seq)
```

map() applies the function func, to a sequence (eg. a list) seq.

```
>>> Celsius = [39.2, 36.5, 37.3, 37.8]

>>> Fahrenheit = map(lambda x: (float(9)/5)*x + 32, Celsius)
>>> print Fahrenheit
[102.56, 97.700000000000003, 99.140000000000001, 100.03999999999999]

>>> Centigrade = map(lambda x: (float(5)/9)*(x-32), Fahrenheit)
>>> print Centigrade
[39.200000000000003, 36.5, 37.300000000000004, 37.799999999999997]
```

filter

```
r = filter(func, l)
```

filter() filters (outputs) all the elements of a list l, for which the passed-in Boolean-valued function func returns True.


```
>>> fib = [0,1,1,2,3,5,8,13,21,34,55]

>>> # for an odd number x, x%2 is 1, ie. True; so the following filters odd numbers
>>> result = filter(lambda x: x % 2, fib)
>>> print result
[1, 1, 3, 5, 13, 21, 55] # odd nums

>>> # to filter out even numbers, make it true that the mod has 0 remainder
>>> result = filter(lambda x: x % 2 == 0, fib)
>>> print result
[0, 2, 8, 34] # only evens
```

reduce

reduce() applies func repeatedly to the elements of l, to generate a single value and output it.

```
r = reduce(func, l)
```

```
>>> reduce(lambda x,y: x+y, [47,11,42,13])
113
```

In the above, reduce() repeatedly calls the lambda function with the 'current' result ('x'), along with the next element in the list ('y'); the starter value of the current result is initialized to the first element of the incoming list (47, in our example), so the function is called with (47,11), (58,42), (100,13).

Here are the above examples, in PyScript ("Python in the browser").

Out of map(), filter() and reduce() triplet functions, map() and reduce() together in particular have become a **potent data processing combination** (aka "MapReduce"!) in recent years, when applied in a **parallel** fashion to Big Data.

MapReduce

MapReduce is a programming paradigm invented at Google, one which has become **wildly popular** since it is designed to be applied to Big Data in NoSQL DBs, in data and disk parallel fashion - resulting in ****dramatic**** processing gains.

MapReduce works like this:

- 0. [big] data is split into file segments, held in a compute cluster made up of nodes (aka partitions)
- 1. a mapper task is run in parallel on all the segments (ie. in each node/partition, in each of its segments); each mapper produces output in the form of multiple (key,value) pairs
- 2. key/value output pairs from all mappers are forwarded to a shuffler, which consolidates each key's values into a list (and associates it with that key)
- 3. the shuffler forwards keys and their value lists, to multiple reducer tasks; each reducer processes incoming key-value lists, and emits a single value for each key

Optionally, before forwarding to shufflers, a 'combiner' operation in each node can be set up to perform a local per-key reduction - if specified, this would be 'step 1.5', in the above workflow.

The cluster user (programmer) only needs to supply a mapper task and a reducer task, the rest is automatically handled!

The following diagrams illustrate this elegant, embarrassingly simple idea.

Summary: "MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key."

Since MapReduce involves accessing (reading, writing) distributed (in clusters) data in parallel, there needs to be a high-performance, distributed file system that goes along with it - Google created GFS to fill this need.

GFS abstracts details of network file access so that remote reads/writes and local reads/writes are handled (in code) identically.

GFS differs from other distributed file systems such as (Sun's) NFS, in that the file system is implemented as a process in each machine's OS; striping is used to split each file and store the resulting chunks on several 'chunkservers', details of which are handled by a single master.

Hadoop: backstory

Doug Cutting and Mike Cafarella started 'Nutch' (a search engine project) in 2002.

Nutch had severe scalability problems.

Even as Doug and Mike were struggling, Google published a paper on GFS, and a year later, another on MapReduce [links to the papers are in previous slides, you can find them here as well]; Doug and Mike gave up on their own work-in-progress, and implemented BOTH! Result: "Hadoop"..

Why "Hadoop"? Doug's son's toy elephant's name was "Hadoop" :) Read more here.

Here he is!

And in 3D too?!

Hadoop is modeled after the MapReduce paradigm, and is utilized identically (by having users run mappers and reducers on (big) data).

HDFS is modeled after Google's GFS, but with some important differences [read the paper to find out if you are interested] - as for similarities, there is a single master NameNode, and multiple DataNodes.

MR example[s]

WordCount is the 'Hello World' of Hadoop [counts the number of occurrences of each word in a given input set].

Following is the Java code for WordCount - note that it has both the mapper and reducer specified in it:

```
import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {

  public static class TokenizerMapper
      extends Mapper{

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(Object key, Text value, Context context
                ) throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
      }
    }
  }

  public static class IntSumReducer
      extends Reducer {
    private IntWritable result = new IntWritable();
```



```
    public void reduce(Text key, Iterable values,
                Context context
                ) throws IOException, InterruptedException {
      int sum = 0;
      for (IntWritable val : values) {
        sum += val.get();
      }
      result.set(sum);
      context.write(key, result);
    }
  }

  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

Below are the results of running this on a small two-file dataset.

```
Input:
$ bin/hadoop fs -cat /user/joe/wordcount/input/file01
Hello World Bye World
$ bin/hadoop fs -cat /user/joe/wordcount/input/file02
Hello Hadoop Goodbye Hadoop

Run:
$ bin/hadoop jar wc.jar  WordCount  /user/joe/wordcount/input   /user/joe/wordcount/output

Output:
$ bin/hadoop fs -cat /user/joe/wordcount/output/part-r-00000`
Bye 1
Goodbye 1
Hadoop 2
Hello 2
World 2
```

You can get more details here.

```
#mapper:
#!/usr/bin/env python

import sys

# input comes from STDIN (standard input)
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()
    # split the line into words
    words = line.split()
    # increase counters
    for word in words:
        # write the results to STDOUT (standard output);
        # what we output here will be the input for the
        # Reduce step, i.e. the input for reducer.py
        #
        # tab-delimited; the trivial word count is 1
        print '%s\t%s' % (word, 1)
```



```
# reducer:
#!/usr/bin/env python

from operator import itemgetter
import sys

current_word = None
current_count = 0
word = None

# input comes from STDIN
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()

    # parse the input we got from mapper.py
    word, count = line.split('\t', 1)

    # convert count (currently a string) to int
    try:
        count = int(count)
    except ValueError:
        # count was not a number, so silently
        # ignore/discard this line
        continue

    # this IF-switch only works because Hadoop sorts map output
    # by key (here: word) before it is passed to the reducer
    if current_word == word:
        current_count += count
    else:
        if current_word:
            # write result to STDOUT
            print '%s\t%s' % (current_word, current_count)
        current_count = count
        current_word = word

# do not forget to output the last word if needed!
if current_word == word:
    print '%s\t%s' % (current_word, current_count)
```

The Hadoop 'ecosystem'

Because the core Hadoop system has been so popular, a whole bunch of associated projects have resulted, leading to a thriving 'ecosystem'. - don't feel overwhelmed though, you don't need to learn to use all these all at once!

Another view:

The following databases are most commonly used inside a Hadoop cluster:

- MongoDB
- Cassandra
- HBase
- Hive
- Spark
- Blur
- Accumulo
- Memcached

- Solr
- Giraph

Hadoop: Hive

Hive provides a SQL-like scripting language called HQL. "Better than SQL" - eg. no need to create relational table schemas and populate with data.

"Hive translates most queries to MapReduce jobs, thereby exploiting the scalability of Hadoop, while presenting a familiar SQL abstraction."

Below is the WordCount task expressed in HiveQL - just 8 lines, compared to the standard MR Java code (shown earlier) of 53 lines!

```
CREATE TABLE docs (line STRING);
LOAD DATA INPATH 'docs' OVERWRITE INTO TABLE docs;
CREATE TABLE word_counts AS
SELECT word, count(1) AS count FROM
  (SELECT explode(split(line, '\s')) AS word FROM docs) w
GROUP BY word
ORDER BY word;
```

Facebook was a major early contributor.

Hadoop: Pig

"Pig provides an engine for executing data flows in parallel on Hadoop. It includes a language, Pig Latin, for expressing these data flows. Pig Latin includes operators for many of the traditional data operations (join, sort, filter, etc.), as well as the ability for users to develop their own functions for reading, processing, and writing data."

Pig Latin scripts are compiled into MR jobs that are then run on the cluster.

Here is a Pig Latin script for doing word counting [only 5 lines of code!], on the first stanza of Mary Had A Little Lamb:

```
-- Load input from the file named Mary, and call the single
-- field in the record 'line'.
input = load 'mary' as (line);
-- TOKENIZE splits the line into a field for each word.
-- flatten will take the collection of records returned by
-- TOKENIZE and produce a separate record for each one, calling the single
-- field in the record word.
words = foreach input generate flatten(TOKENIZE(line)) as word;
-- Now group them together by each word.
grpd = group words by word;
-- Count them.
cntd = foreach grpd generate group, COUNT(words);
-- Print out the results.
dump cntd;
```

Note that with Pig Latin, there is no explicit spec of mapping and reducing phases - the Pig=>MR compiler figures this out by analyzing the specified dataflow.

For completeness, here is how we picture the underlying MR job running:

"Pig Latin is a dataflow language. This means it allows users to describe how data from one or more inputs should be read, processed, and then stored to one or more outputs in parallel. These data flows can be simple linear flows like the word count example given previously. They can also be complex workflows that include points where multiple inputs are joined, and where data is split into multiple streams to be processed by different operators. To be mathematically precise, a Pig Latin script describes a directed acyclic graph (DAG), where the edges are data flows and the nodes are operators that process the data."

Yahoo! Research was a major developer of Pig.

Hadoop: Musketeer

Currently, front end workflows (eg. written using Hive) are *coupled* with back-end engines (such as Hadoop), making them less usable than if these could be decoupled.

Musketeer is an experimental approach to do the decoupling. Three benefits:

1. Users write their workflow once, in a way they choose, but can easily execute it on alternative systems;

2. Multiple sub-components of a workflow can be executed on different back-end systems; and

3. Existing workflows can easily be ported to new systems.

MRv2: YARN

YARN is "MapReduce v2".

The first version of MR/Hadoop was 'batch oriented', meaning that static, distributed data was processed via mapping, shuffling and reducing steps.

YARN (Yet Another Resource Negotiator) on the other hand makes non-MR applications (eg. graph processing, iterative modeling) possible (but is fully backwards compatible with v.1.0, ie. can run MapReduce jobs), and offers better scalability and cluster utilization (compared to MRv1). It also makes it possible to create (near) real-time applications.

MRv1:

MRv2, ie. YARN:

Cloud infrastructure

Amazon's EC2/S3, Microsoft's Azure, Google's GCP, etc. offer a 'cloud platform' on which to build apps and services.

Such cloud services offer 'elastic scaling' (of resources, ie. computing power, storage), guaranteed uptime, speedy access, etc.

Also, Google has plans to offer exascale computing capability in the future, to analyze Big(ger) Data.

One hassle-free (somewhat!) way to set up a Hadoop compute cluster is to do so inside EC2 or Azure or GCP.

Note - setting these up can be quite tedious (and is thankfully a one shot thing!). Here is how to do this in GCP.

VM infrastructure

A virtual machine (VM) is a piece of software that runs on a host machine, to enable creating self-contained 'virtual' machines inside the host - these virtual machines can then serve as platforms on which to run programs and services.

Virtualization software allows applications that previously ran on separate computers to run on one server machine.

So, another way (not cloud based) to experiment with Hadoop is to download implementations meant for virtual machines, and load them into the VMs.

Here are a couple you can try [these are the most used ones, compared to the ones listed below]: HortonWorks' Hadoop Sandbox and MapR's MapR Sandbox.

In addition, you can also experiment with Oracle's 'Big Data Lite' VM, Cloudera's CDH VM, IBM's BigInsights QSE VM and Talend's Big Data Sandbox [here is the download info] VM - this one packages an existing VM along with a custom platform and sample data.

There is also a Blaze VM for performing 'streaming analytics', ie. for analyzing live ("streaming") data in real time. This is Forrester Research's definition of streaming analytics: 'software that provides analytical operators to orchestrate data flow, calculate analytics, and detect patterns on event data from multiple, disparate live data sources to allow developers to build applications that sense, think, and act in real time.' In other words, streaming data (from sensors, databases, applications, people...) is continuously analyzed via streaming queries [with the sqlstream products, this is done via SQL operators], leading to insights in real time.

FYI - you might enjoy reading this note about VMWare's virtualizing Hadoop.

Beyond MR: Spark

Spark [developed AMPLab at Cal [UC Berkeley] in 2009, and open sourced] makes Big Data real-time and interactive - it is an **in-memory data processing engine** (so it is FAST), specifically meant for iterative processing of data. It is considered an alternative to MapReduce, and runs on top of HDFS.

Better efficiency: general execution graphs, in-memory data storage.

Being used at Yahoo!, Intel, Adobe, Quantifind, Conviva, Ooyala, Bizo, etc.

Query lang is SparkSQL (used to be called Shark; Shark itself was an alternative to Hive).

MR could not deal with complex (multi-pass) processing, interactive (ad-hoc) queries or real-time (stream) processing. Spark addresses all these.

A Spark application consists of a **'driver'** that converts high level queries into tasks, which **'executors'** run in parallel.

Big idea: resilient distributed datasets (RDDs)

- distributed collections of objects that can be cached in memory across cluster
- manipulated through parallel operators
- automatically recomputed on failure

Impressive performance during iterated computations!

Example: Logistic Regression

Goal: find best line separating two sets of points

Example: Logistic Regression

```
val data = spark.textFile(...).map(readPoint).cache()

var w = Vector.random(D)

for (i <- 1 to ITERATIONS) {
  val gradient = data.map(p =>
    (1 / (1 + exp(-p.y*(w dot p.x))) - 1) * p.y * p.x
  ).reduce(_ + _)
  w -= gradient
}

println("Final w: " + w)
```

APIs exist in Python, Java, etc..

```
# Python:
lines = sc.textFile(...) lines.filter(lambda x: "ERROR" in x).count()

// Java:
JavaRDD lines = sc.textFile(...); lines.filter(new Function() { Boolean call(String s) { return s.c
ontains("error"); } }).count();
```

Spark's modular architecture has been instrumental in enabling the following add-on functionalities:

- Spark Streaming
- Spark SQL
- Spark MLib
- Spark GraphX

Here is another view of the various Spark modules:

Apache Spark is:

- **Spark Core**
- **Spark Streaming**
  - Microbatch
  - Stateful stream processing
  - DStream
  - Socket Stream
  - File Stream
- **Spark SQL**
  - DataFrame
  - DataFrame API
  - Supported Data Formats and Sources
  - Plan Optimization and Execution
  - Rules-Based Optimization
- **MLlib**
  - Algorithms
  - Key Features
  - Pipeline
- **GraphX**
  - PropertyGraph
  - GraphViews
  - TripletView
  - Subgraph
  - Distributed Graph Representation

This is a standard MR (Java) vs Spark (Python) comparison, of WordCount:

```java
package org.apache.hadoop.examples;

import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.FileSplit;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;

public class WordCount {

  public static class TokenizerMapper extends
    Mapper<Object, Text, Text, IntWritable> {

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(Object key, Text value, Context context)
      throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
      }
    }
```

```java
  public static class IntSumReducer extends
    Reducer<Text, IntWritable, Text, IntWritable> {

    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values,
    Context context) throws IOException, InterruptedException {
      int sum = 0;
      for (IntWritable val : values) {
        sum += val.get();
      }
      result.set(sum);
      context.write(key, result);
    }
  }

  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    String[] otherArgs = new GenericOptionsParser(conf, args)
      .getRemainingArgs();

    Job job = new Job(conf, "word count");

    job.setJarByClass(WordCount.class);

    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
    FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
```

Spark WordCount Example Using pyspark

```python
logFile = "hdfs://localhost:9000/user/bigdatavm/input"

sc = SparkContext("spark://bigdata-vm:7077", "WordCount")

textFile = sc.textFile(logFile)

wordCounts = textFile.flatMap(lambda line:
line.split()).map(lambda word: (word,
1)).reduceByKey(lambda a, b: a+b)
wordCounts.saveAsTextFile("hdfs://localhost:9000/user/bigd
atavm/output")
```

"Beyond" Spark, there's Dask (esp. for data science work):

Data scientists are always looking for strategic ways to inject efficiency into training and deploying models. Recently, a new generation of distributed training engines has surfaced that delivers on that goal by providing tremendous speed and performance gains over Apache Spark.

One approach that's gaining attention is Dask, a distributed training framework built in Python.[16] Dask is designed to enable data scientists to improve model accuracy faster. Data scientists can do everything in Python end to end, which means they no longer need to convert their code to execute in Spark. The result is reduced complexity and increased efficiency.

Another open source Python framework is RAPIDS, which is built on top of Dask.[17] RAPIDS optimizes compute time and speed by providing data pipelines and executing data science code entirely on graphics processing units (GPUs) rather than CPUs. Saturn Cloud recently compared RAPIDS to Spark and discovered that model training with RAPIDS took one second on a 20-node GPU cluster, while Spark took 37 minutes on a similarly priced 20-node CPU cluster. Saturn Cloud concluded that RAPIDS enables 2,000x faster processing using GPUs while costing a fraction of the price.[18]

The impact of these distributed training frameworks is already being seen in the real world. Walmart uses RAPIDS with Dask and XGBoost (an ML algorithm) for its data analytics and ML, and NVIDIA reports that Walmart has found that "one GPU server requires only four percent of the time needed to run the same forecasting models vs a 20-node CPU server."[19] That translates to Walmart running models in four hours that previously took several weeks using CPUs.

While organizations are thinking strategically about training frameworks, some have run into barriers in the past. Today, new technologies are unlocking what's possible and demonstrating how much faster things can be when everything is done directly with Python. By eliminating the need to convert models into Spark, organizations are reducing complexity and increasing efficiency. And it's easy to try different distributed training frameworks on Snowflake's platform to find what works best.

Beyond MR: Flink

Similar to MR, Flink is a parallel data processing platform.

"Apache Flink's programming model is based on concepts of the MapReduce programming model but generalizes it in several ways. Flink offers Map and Reduce functions but also additional transformations like Join, CoGroup, Filter, and Iterations. These transformations can be assembled in arbitrary data flows including multiple sources, sinks, and branching and merging flows. Flink's data model is more generic than MapReduce's key-value pair model and allows to use any Java (or Scala) data types. Keys can be defined on these data types in a flexible manner.

Consequently, Flink's programming model is a super set of the MapReduce programming model. It allows to define many programs in a much more convenient and concise way. I also want to point out that it is possible to embed unmodified Hadoop functions (Input/OutputFormats, Mapper, Reducers) in Flink programs and execute them jointly with native Flink functions."

Beyond MR: Storm

"Apache Storm is a free and open source distributed realtime computation system. Storm makes it easy to reliably process unbounded streams of data, doing for realtime processing what Hadoop did for batch processing. Storm is simple, can be used with any programming language, and is a lot of fun to use!

Storm has many use cases: realtime analytics, online machine learning, continuous computation, distributed RPC, ETL, and more. Storm is fast: a benchmark clocked it at over a million tuples processed per second per node. It is scalable, fault-tolerant, guarantees your data will be processed, and is easy to set up and operate."

Complementing Storm, Kafka is a distributed pub-sub real-time messaging system that provides strong durability and fault tolerance guarantees. Kafka nodes are called brokers, and are used by producers/publishers (who write data, into 'topics'), and consumers/subscribers (who read data off topics).

Storm does stream processing. A stream is a sequence of tuples. Streams are originated at spouts, that read data (from live sources, files...), and are passed on to bolts for processing (streams in, streams out).

Beyond MR: Samza

Apache Samza (developed at LinkedIn in 2013, then open sourced in 2014) is a processing engine that handles batch data and streaming data in a unified manner (ie. identically).

Both streams and batch data are stored in partitions, which are then read by 'tasks' which comprise the application.

Samza SQL can be used to create pipelined jobs that can utilize stream or batch data, eg.

Released as a part of Samza 1.0, Samza SQL empowers engineers to create streaming pipelines without a line of Java code. Engineers can now declaratively specify what they need and not worry about details like capacity provisioning, resource management, or operability. For instance, here's the definition of a pipeline that reads from a Kafka topic of member profiles, filters those who are product managers at LinkedIn, and writes the results to a new topic.

```
insert into kafka.ProductManagers
        (select name, school from kafka.user-profiles where company = 'LinkedIn' and
         title = `ProductManager')
```

Samza SQL also supports implementing custom user logic by specifying user-defined functions (UDFs) in Java. Our support for SQL leverages Apache Calcite for its implementation and builds on the foundations offered by Samza's core engine.

Spark vs Storm vs Flink vs Samza

On the surface of it, they all appear to be identical - in a way, they are - they offer distributed processing of Big Data, without requiring explict mapper/reducer specifications.

Spark, Flink and Samza, can handle batch as well as streaming data.

Storm does stream processing.

So to summarize, we have Hadoop+Yarn for batch processing, Spark, Flink and Samza for batch+stream processing, Storm for stream processing, and Kafka for handling messages. Here is one way how it could all fit together.

BSP: MR alternative

The Bulk Synchronous Parallel (BSP) model is an alternative to MR.

A BSP computation is executed on a set of processors which are connected in a communication network but work independently by themselves. The BSP computation consists of a sequence of iterations, called supersteps. In each superstep, three actions occur:

- (i) concurrent computation performed LOCALLY by a set of processors. Each processor has its own local memory and uses local variables to independently complete its computations. This is the asynchronous part.
- (ii) communication, during which processors send and receive messages (exchange/access data).
- (iii) synchronization which is achieved by setting a barrier - when a processor completes its part of computation and communication, it reaches this barrier and waits for the other processors to finish.

In other words, there are three steps (phases) in each superstep:

Local computation: every processor performs computations using data stored in local memory - independent of what happens at other processors; a processor can contain several processes (threads)

Communication: exchange of data between processes (put and get); one-sided communication

Barrier synchronization: all processes wait until everyone has finished the communication step The following figure illustrates the actions applied in one superstep.

"Think like a vertex"..

User-defined vertex update ops (that can happen in parallel), local edge updates.

Google's implementation of BSP is called Pregel.

Trivia: why did they name it Pregel? This blog post has the answer [read the last sentence!] :)

Aside: this and this :)

Pregel -> Giraph,HAMA

Giraph is an open source version of Pregel, so is Hama, Golden Orb, Stanford GPS.

Specifically designed for iterative graph computations (and nothing else!).

Apache HAMA is a general-purpose Bulk Synchronous Parallel (BSP) computing engine on top of Hadoop. It provides a parallel processing framework for massive iterative algorithms (including ones for scientific computing, ie. 'HPC' applications).

HAMA performs a series of supersteps based on BSP - it is suitable for iterative computation, since it is possible that input data which can be saved in memory, is able to get transfered between supersteps (unlike MR).

HAMA's vertex-centric graph computing model is suggestive of MapReduce in that users focus on a local action, processing each item independently, and the system (HAMA runtime) composes these actions to run over a large dataset.

But HAMA is not merely a graph computing engine - instead it is a general purpose BSP platform, so on top of it, any arbitrary computation (graph processing, machine learning, MRQL, matrix algorithms, network algorithms..) can be implemented; in contrast, Giraph is ONLY for graph computing.

Here is a paper on the processing of one TRILLION (!) edges at Facebook. Three applications (that run on Facebook's friendships graph) are presented:

- label propagation
- PageRank
- 'friends of friends' score

Please read the paper to learn the details.

Also, here is a writeup on Facebook's use of Giraph (to enable graph searching by users).

1/20 8:23:50 ***

← →

Data Mining

[search(ing) for PATTERNS]

"You look at where you're going and where you are and it never makes sense, but then you look back at where you've been and a pattern seems to emerge."
- Robert M. Pirsig, Zen and the Art of Motorcycle Maintenance: An Inquiry Into Values

What is data mining?

Data mining is the science of extracting useful information from large datasets.

At the heart of data mining is the process of discovering **RELATIONSHIPS** between parts of a dataset.

"Data mining is the analysis of (often large) observational data sets to find unsuspected relationships and to summarize the data in novel ways that are both understandable and useful to the data owner. The relationships and summaries derived through a data mining exercise are often referred to as **models** or **patterns**".

The term **'trend'** is used to describe patterns that occur or change over time.

How is ML different from DM?

Machine learning is the process of TRAINING an algorithm on an EXISTING dataset in order to have it discover **relationships** (so as to create a model/pattern/trend), and USING the result to analyze NEW data.

Here is Andrew Ng's 'classic' Stanford course on ML that is hosted at Coursera, which he co-founded [till early 2017, Andrew was with Baidu].

Here is how data mining relates to existing fields:

By nature, most data mining is cyclical. Starting with **data**, mining leads to **discovery**, which leads to action (**"deployment"**), which in turn leads to new data - the cycle continues.

A more nuanced depiction of the cycle:

As you can imagine, data mining is useful in dozens (hundreds!) of fields!! Almost ANY type of data can be mined, and results put to use. Following are typical uses:

- predicting which customers will purchase what products and when
- deciding should insurance rates be set to ensure profitability
- predicting equipment failures, reducing unnecessary maintenance and increasing uptime to optimize asset performance
- anticipating resource demands
- predicting which customers are likely to leave and what can be done to retain them
- detecting fraud
- minimizing financial risk
- increasing response rates for marketing campaigns

Common Applications for Data Mining Across Industries

Business Question	Application	What Is Predicted?
How to better target product/service offers?	Profiling and segmentation.	Customer behaviors and needs by segment.
Which product/service to recommend?	Cross-sell and up-sell.	Probable customer purchases.
How to grow and maintain valuable customers?	Acquisition and retention.	Customer preferences and purchase patterns.
How to direct the right offer to the right person at the right time?	Campaign management.	The success of customer communications.
Which customers to invest in and how to best appeal to them?	Profitability and lifetime value.	Drivers of future value (margin and retention).

Industry-Specific Data Mining Applications

Business Question	Application	What Is Predicted?
How to assess and control risk within existing (or new) consumer portfolios?	Credit scoring (banking).	Creditworthiness of new and existing sets of customers.
How to increase sales with cross-sell/up-sell, loyalty programs and promotions?	Recommendation systems (online retail).	Products that are likely to be purchased next.
How to minimize operational disruptions and maintenance costs?	Asset maintenance (utilities, manufacturing, oil and gas).	The real drivers of asset or equipment failure.
How to reduce health care costs and satisfy patients?	Health and condition management (health insurance).	Patients at risk of chronic, treatable/preventable illness.
How to decrease fraud losses and lower false positives?	Fraud management and cybersecurity (government, insurance, banks).	Unknown fraud cases and future risks.
How to bring drugs to the marketplace quickly and effectively?	Drug discovery (life sciences).	Compounds that have desirable effects.

Data mining algorithms: categories

Practically all data mining ('DM') algorithms neatly fit into one of these 4 categories:

- **Classification**: involves LABELING data
- **Clustering**: involves GROUPING data, based on similarity
- **Association**: involves RELATING data
- **Regression**: involves COUPLING data [incl finding 'outliers']

We'll look at examples in each category, that will provide you a concrete understanding of the above summarization.

DM algorithms can also be classified in a different way - as being a 'supervised' learning method (where we need to provide categories for, ie. train, using known outcomes), or an 'unsupervised' method (where we provide just the data to the algorithm, leaving it to learn on its own), or 'semi-supervised', or even 'self-supervised'.

Algorithms!

Now we can start discussing specific algorithms (where each one belongs to the four categories we just outlined).

Challenge/fun - can each algorithm be talked about, without using any math at all? You get the 'big picture' [a clear, intuitive understanding] that way.. After that, we can optionally look at equations/code for the algorithms, to learn the details (God/the devil *is* in the details :)).

Algorithm: Decision trees (eg. C4.5, C5.0 etc.)

Classification and regression trees (aka decision trees) are machine-learning methods for constructing prediction models from data. The models are obtained by recursively partitioning the data space and fitting a simple prediction model within each partition.

The decision tree algorithm works like this:

- user provides a set of input (training) data, which consists of features (independent parameters) for each piece of data, AND an outcome (a 'label', ie. a class name)
- the algorithm uses the data to build a 'decision tree' [with feature-based conditionals (eqvt to 'if' or 'case' statements) at each non-leaf node], leading to the outcomes (known labels) at the terminals
- the user makes use of the tree by providing it new data (just the feature values) - the algorithm uses the tree to 'classify' the new item into one of the known outcomes (classes)

Should we play tennis? Depends (on the weather) :)

This is a VERY simple algorithm! The entire tree is a disjunction (where the branches are) of conjunctions (that lead down from root to leaves).

If the outcome (dependent, or 'target' or 'response' variable) consists of classes (ie. it is 'categorical'), the tree we build is called a classification tree. On the other hand if the target variable is continuous (a numerical quantity), we build a regression tree. Numerical quantities can be ordered, so such data is called 'ordinal'; categories are names, they provide 'nominal' data. Given that, nominal data results in classification trees, and ordinal data results in regression trees.

Here is another example (from 'Principles of Data Mining' by David Hand et. al.). Shown is a 'scatter plot' of a set of wines - color intensity vs alcohol content:

The classification tree for the above data is this:

When we superpose the 'decision boundaries' from the classification tree on to the data, we see this:

A sample regression tree is shown below – note that the predictions at the leaves are numerical (not categorical):

Note - algorithms that create **c**lassification trees **a**nd **r**egression **t**rees are referred to as CART algorithms (guess what CART stands for :)).

Algorithm: Support Vector Machine (SVM)

An SVM always partitions data (classifies) them into TWO sets - uses a 'slicing' hyperplane (multi-dimensional equivalent of a line), instead of a decision tree.

The hyperplane maximizes the gap on either side (between itself and features on either side). This is to minimize chances of mis-classifying new data.

On either side, the equidistant data points closest to the hyperplane are the 'support vectors' (note that there can be several of these, eg. 3 on one side, 2 on the other). Special case - if there is a single support (closest point) on either side, in 2D, the separator is the perpendicular bisector of the line segment joining the supports; if not, the separating line/plane/hyperplane needs to be calculated by finding two parallel hyperplanes with no data in between them, and maximizing their gap. The goal is to achieve "margin maximization".

How do we know that a support data point is indeed a support vector? If we move the data point and therefore the boundary moves, that is a support vector :) In other words, non-support data can be moved around a bit, that will not change the boundary (unless the movement brings it to be inside the current margin).

Note - in our simplistic setup, we make two assumptions: that our two classes of data are indeed separable (not inter-mingled), and that they are linearly separable. FYI, it is possible to create SVMs even if both the assumptions aren't true [think - how?].

Here's some geek humor for ya:

Prof. Patrick Winston @ MIT, lecturing on SVMs:
https://www.youtube.com/watch?v=_PwhiWxHK8o ...

Algorithm: kNN

kNN (k Nearest Neighbors) algorithm picks 'k' neartest neighbors, closest to our unclassified (new) point, considers the 'k' neighbors' types (classes), and attempts to classify the unlabeled point using the neighbors' type data - majority wins (the new point's type will be the type of the majority of its 'k' neighbors).

Here is an example of 'loan defaulting' prediction. With k=3, and a new point of (42,142000), our neighbors have targets of Non Default, Non Default, Default - so we assign 'Non Default' as our target.

Note that if k=1, we assign as our target, that of our closest point.

Also, to eliminate excessive influence of large numerical values, we usually normalize our data so that all values are 0..1.

kNN is a 'lazy learner' - just stores input data, uses it only when classifying an unlabeled (new) input. VERY easy to understand, and implement!

Algorithm: Naive Bayes

The 'naïve' Bayes algorithm is a probability-based, supervised, classifier algorithm (given a datum with $x_1, x_2, x_3 . . x_n$ features (ie an n-dimensional point), classify it to be one of 1,2,3...k classes).

The algorithm is so called because of its strong ('naïve') assumption: each of the x_1 ..x_n features are statistically independent of each other ["class conditional independence"] - eg. a fruit is an apple if it is red, round and ~4 in. dia [color, shape and size are independent metrics].

Other names for this algorithm: simple Bayes, independence Bayes, idiot Bayes (!).

For each training feature set, probabilities are assigned for each possible outcome (class). Given a new feature, the algorithm outputs a classification corresponding to the max of the most probable value of each class (which the algorithm calculates, using the 'maximum a posteriori', or 'MAP' decision rule).

Here is an example (from http://www.statsoft.com/textbook/naive-bayes-classifier).

Given the following distribution of 20 red and 40 green balls (ie. 60 samples, 2 classes [green,red]), how to classify the new [white] one ('X')?

PRIOR probability of X being green or red:

$$\text{Prior probability for GREEN} \propto \frac{40}{60}$$

$$\text{Prior probability for RED} \propto \frac{20}{60}$$

Given the neighborhood around X, probability (LIKELIHOOD) of X being green, X being red:

$$\text{Probability of X given GREEN} \propto \frac{1}{40}$$

$$\text{Probability of X given RED} \propto \frac{3}{20}$$

POSTERIOR probabilities of X being green, X being red [posterior probability = prior probability modified by likelihood]:

$$\text{Posterior probability of X being GREEN} \propto$$
$$\text{Prior probability of GREEN} \times \text{Likelihood of X given GREEN}$$
$$= \frac{4}{6} \times \frac{1}{40} = \frac{1}{60}$$
$$\text{Posterior probability of X being RED} \propto$$
$$\text{Prior probability of RED} \times \text{Likelihood of X given RED}$$
$$= \frac{2}{6} \times \frac{3}{20} = \frac{1}{20}$$

Classify, using the max of the two posterior probabilities above [MAP]: X is classified as 'red'.

Algorithm: k-means clustering

This algorithm creates 'k' number of "clusters" (sets, groups, aggregates..) from the input data, using some measure of closeness (items in a cluster are closer to each other than any other item in any other cluster). This is an example of an unsupervised algorithm - we don't need to provide training/sample clusters, the algorithm comes up with them on its own.

If each item in our (un-clustered) dataset has 'n' attributes, it is eqvt to a point in n-dimensional space (eg. n can be 120, for Amazon!). We are now looking to form clusters in n-D space!

Approach: start with 'n' random locations ('centroids', ie means) in the dataset; assign each input point (our data) to the [current] closest centroid; compute new centroids (from our data, for each centroid's "membership"); iterate (till convergence is reached) - this is the 'mean shift' algorithm. Here is another description.

Here is a/nother clip showing the cluster creation; here is the Java .jar file (download, double click to open) used to create the clip.

This page contains JavaScript code that implements k-means - the centroid migration trail is plotted; this is the neat result.

Here is a use case for doing clustering:

Again, very simple to understand/code!

Another example - here, we classify clients of a company, into 3 categories, based on how many orders they placed, and what the orders cost:

Q: how many clusters are ideal? Here is a way to estimate it [on the Y axis, plot SSE - sum of the squared distance between each cluster point and its centroid].

Algorithm: Hierarchical clustering

In some situations (where it is meaningful to do so), it is helpful to separate items in a dataset into **hierarchical** clusters (clusters of clusters of..). There are two ways to look at this - as the merging of smaller clusters into bigger superclusters, or dividing of larger clusters into finer scale ones.

Below is a dataset that plots, for Yellowstone National Park, the waiting time between geyser eruptions, and time length (duration) of eruptions. The data points are numbered, just to be able to identify them in our cluster diagram.

Given this data, we run a hierarchical clustering algorithm, whose output is a 'dendogram' (tree-like structure) that shows the merging of clusters:

Another example:

How do we decide what to merge? A popular strategy - pick clusters that lead to the smallest increase in sum of squared distances (in the above diagram, that is what the vertical bar lengths signify). The dendogram

shows that we need to start by merging #18 and #27, which we can verify by looking at the scatter plot (those points almost coincide!).

Algorithm: EM (Expectation Maximization)

EM is a rather 'magical' algorithm that can solve for a Catch-22 (circular reference) set of values! Here we use it, to cluster data.

Imagine we have a statistical model that has some parameters, regular variables, and some 'latent' (hidden) variables [eg. data columns with missing values]. The model is expected to operate on new data (predict/classify), given training/pre-existing data. Sounds like a straightforward DM algorithm (eg. k-means clustering), except that it is not!

We do not know the values for the model parameters, or latent/missing variables! We DO have observed/collected/measured data, which the model should be able to act on, ie we have 'outcomes'. **The question is, what model parameters would explain (result in, produce...) the outcomes?** In other words, we want to explain why/how those outcomes result.

The algorithm works as follows:

- pre-step: start with random (!) values for the model parameters
- step1: use current param values to compute probabilities for all possible values for each hidden (latent) var, then do a weighted average (weighted by probability) to compute the best value for each latent var (this is the 'E' step)
- step2: use the hidden vars' values found in the above step, to improve/update the model parameters ('M' step) - do this by maximizing likehood [for the outcomes, given the params]
- iterate the above two steps till param values converge

Note: in step1, we're using estimated param values as 'true' values; in step2, we're using estimated variable values as 'observed' values.

As counter-intuitive as it sounds, this does work!

Here is another explanation, from a StackOverflow post:

There's a chicken-and-egg problem in that to solve for your model parameters you need to know the distribution of your unobserved data; but the distribution of your unobserved data is a function of your model parameters.

E-M tries to get around this by iteratively guessing a distribution for the unobserved data, then estimating the model parameters by maximizing something that is a lower bound on the actual likelihood function, and repeating until convergence:

- start with guess for values of your model parameters
- E-step: For each datapoint that has missing values, use your model equation to solve for the distribution of the missing data given your current guess of the model parameters and given the observed data (note that you are solving for a distribution for each missing value, not for the expected value). Now that we have a distribution for each missing value, we can calculate the expectation of the likelihood function with respect to the unobserved variables. If our guess for the model parameter was correct, this expected likelihood will be the actual likelihood of our observed data; if the parameters were not correct, it will just be a lower bound.
- M-step: Now that we've got an expected likelihood function with no unobserved variables in it, maximize the function as you would in the fully observed case, to get a new estimate of your model parameters.
- repeat until convergence.

EM is frequently used to 'auto cluster' data.

Here is an example of EM being used to compute clustering on a dataset (centroids are the model params, cluster memberships are the latent vars).

EM is an example of a family of maximum likelihood estimator [MLE] algorithms - others include gradient descent, and conjugate gradient algorithms [which are optimization algorithms, that can, among other things, be used for MLE].

One more example: EM clustering of Yellowstone's Old Faithful eruption data:

Note that in the above, the initial random model consists of two flattened spheres.

Classif vs clustering algorithm[s]: a clarification

What is the difference between classification and clustering? Aren't they the same?

Classification algorithms (when used as learning algorithms) have one ultimate purpose: given a piece of new data, to place it into one of several pre-exising, LABELED "buckets" - these labels could be just names/nonimal (eg. ShortPerson, Yes, OakTree..) or value ranges/ordinal (eg. 2.5-3.9, 100,000-250,000)..

Clustering algorithms on the other hand (again, when used as learning algorithms) take a new piece of data, and place it into a pre-existing group - these groups are UN-LABELED, ie. don't have names or ranges.

Also, in parameter (feature/attribute) space, each cluster would be distinct from all other clusters, by definition; with classification, just 'gaps' don't need to exist.

Note that we use multiple terms to denote data. If we imagine data to be tabular, each row would constitute one sample, with each column being refered to as an attribute/parameter/feature/input/independent variable/descriptor/dimension, and the label column (if present) being refered to as a label/class/target/output/dependent variable/response variable/dimension.

Each sample (row) would constitute a single point in the multidimensional space/axes/coordinate system created by the columns, including the label column (if present), and the collection of rows/samples would lead to a distribution - data mining consists of finding patterns in such a multi-dimensional point distribution.

Algorithm: A priori

Looking for hidden relationships in large datasets is known as association analysis or association rule learning.

The A priori algorithm comes up with association rules (relationships between existing data, as mentioned above). Here is an example: outputting "items purchased together" (aka 'itemsets'), eg. [chip, dip, soda], from grocery transaction records.

Inputs:

- data ("shopping basket"), ie. raw data of items bought during multiple transactions
- size of the desired itemsets (eg. 2, 3, 4..) - # of items bought together
- the support (number of times the itemset occurs, divided by the total # of data items)
- confidence (conditional probability of an item being in a datum, given another item - in other words, the ratio of supports)

Transaction number	Items
0	soy milk, lettuce
1	lettuce, diapers, wine, chard
2	soy milk, diapers, wine, orange juice
3	lettuce, soy milk, diapers, wine
4	lettuce, soy milk, diapers, orange juice

In the above, support for {diapers,wine} is 3/5, and support for {diapers} alone is 4/5; the confidence for {diapers} -> {wine} [given a purchase of diapers, would there also be a purchase of wine] is support({diapers,wine})/support({diapers}), which is 3/5 over 4/5, which is 0.75 (note that 5, ie the total, cancels out). In other words, 75% of the time, a purchase of diapers is also accompanied by the purchase of wine.

What is specified to the algorithm as input, is a **[support (ie. frequency),confidence (ie. certainty, ie. accuracy)] pair** - given these, the algorithm outputs all matching associations that satisfy the [support,confidence] criteria. We would then make appropriate use of the association results (eg. co-locate associated items in a store, put up related ads in the search page, print out discount coupons for associated products while the customer is paying their bill nearby, etc.).

Below is another 'shopping basket' example, with products in columns, customers in rows.

basket-id	A	B	C	D	E		
t_1	1	0	0	0	0		
t_2	1	1	1	1	0		
t_3	1	0	1	0	1		
t_4	0	0	1	0	0		
t_5	0	1	1	1	0		
t_6	1	1	1	0	0		
t_7	1	0	1	1	0		
t_8	0	1	1	0	1		
t_9	1	0	0	1	0		
t_{10}	0	1	1	0	1		

If we set a frequency (ie support) threshold of 0.4 and confidence of 0.5, we can see that the matching items are {A}, {B}, {C}, {D}, {A,C}, {B,C} (each of these occur >=4 times). We can see that {A,C} has an accuracy (ie. confidence) of 4/6=2/3 [occurences-of-AC/occurences-of-A], and {B,C} has an accuracy of 5/5=1 [EVERY time B is purchased, C is also purchased!].

Summary: given a basket (set of itemsets), we can look for associations between itemsetA and itemsetB, given:

- size (count) of itemsetA (eg. 2, eg. {A,C} above)
- size (count) of itemsetB (eg. 1, eg. {A} above)
- threshold/support (S) for itemsetA, eg. 0.4 above
- threshold/support (S) for itemsetB, eg. 0.4 again, above
- confidence for S(itemsetA)/S(itemsetB), eg. 0.5 above

So if our five inputs listed above are (2,1,0.4,0.4,0.5) in the above basket example, the outputs would be (A,C) and (B,C), which means we'd consider co-locating or co-marketing (A,C) and also (B,C) - this is the actionable result we get, from mining the basket data.

Note - such association-mining is also useful in recommendation engines, where it is classified as a form of 'collaborative filtering' (as opposed to 'content-based filtering') algorithm.

Algorithm: Linear regression

This (mining) technique is straight out of statistics - given a set of training pairs for a feature x and outcome y, fit the best line describing the relationship between x,y. The line describes the relationship/pattern.

Given that we have Y_i which depends on $X_{i1}, X_{i2}, X_{i3} \ldots X_{ip}$ [i is the sample index (one "row" of data/one observation/...), 1,2,3..p are the variable indices (dimensions)], we seek to model the dependency relationship as $Y_i = f\left(X_i\right) + \varepsilon_i$ where f is the unknown function (that we seek), and ε is the error with mean=0. Specifically, we are calculating the y-intercept c and slope m of the "regression line", whose equation would be $Y = f\left(X\right) = mX + c$.

Here is an example:

A different kind of linear regression called ARMA is what is applied to more 'noisy' linear variation [eg. a noisy cycle superposed over a linear trend]. This in turn is part of a larger field called 'time series analysis' [eg. see this].

Algorithm: Non-linear regression

Here we fit a higher order polynomial equation (parabola, ie. $ax^2 + bx + c$) to the observed data:

Algorithm: Two parameter, non-linear regression

Here we need to fit a higher order, non-linear surface (ie. non-planar) to the observed data:

Algorithm: Non-parametric modeling

Parametric methods (eg. regression analysis which we just looked at) involve parameter estimation (regression coefficients).

Non-parametric methods - no assumptions on what our surface (the dependent variable) would look like; we would need much more data to do the surface fitting, but we don't need the surface to be parameter-based!

Example: A Thin-Plate Spline Estimate

- Non-linear regression methods are more flexible and can potentially provide more accurate estimates.

While non-parametric models might be better suited to certain data distributions, they could lead to a poor estimate as well (if there is over-fit)..

A Poor Estimate

- Non-linear regression methods can also be too flexible and produce poor estimates for *f*.

Algorithm: Logistic regression

Logistic regression is a classification (usually binary) algorithm:

- compute regression coeffs (linear) corresponding to a 'decision boundary' (line dividing two classes of training data)
- use the derived regression coeffs to compute outcome for new data
- transform the outcome to a logistic regression value, and use the 0..1 result to predict a binary outcome (class A or class B)

Figure: *Logistic function* $y = \frac{1}{a+be^{-cx}}$

Result (the whole point of doing the three steps above) - we are transforming a continuous, regression-derived value (which can be arbitrarily large or small) into a 0..1 value, which in turn we transform into a binary class (eg. yes or no) [similar to using an SVM or creating two clusters via k Means]. Note that if we use the simpler form of the logistic equation (a=b=c=1), we'd need to transform our regression results to a 0-centered distribution before using the logistic equation - this is because 1/(1+exp(-x)) is 0.5, when x=0.

(Meta) Algorithm: Ensemble Learning

What if we used a training dataset to train several different algorithms - eg. decision tree, kNN, neural net(s)..? You'll most likely get (slightly!) different results for target prediction.

We could use a voting scheme, and use the result as the overall output. Eg. for a yes/no classification, we'd return a 'yes' ('no') if we got a 'yes' ('no') majority.

This method of combining learners' results is called 'boosting', and resulting combo learner is called an 'ensemble learner'. Why do this? "Wisdom of the crowds" :) We do this to minimize/eliminate variances between the learners.

FYI - 'AdaBoost' (Adaptive Boosting) is an algorithm for doing ensemble learning - here, the individual learners' weights are iteratively and adaptively tweaked so as too minimize overall classification errors [starting from a larger number of features used by participating learners to predict outcomes, the iterative training steps select only those features known to improve the predictive power of the overall model].

FYI - 'Bagging' (bootstrap aggregating) is a data-conditioning-related ensemble algorithm (where we employ 'bootstrap resampling' of data - divide the data into smaller subsets, use all the subsets for training different 'variations' of a model, use all the resulting models to predict outcomes, transform these into a single 'ensemble' outcome).

Algorithm: RandomForest (TM)

RandomForest(TM) is an *ensemble* method where we:

- grow a 'forest' (eg. with count=500) decision trees, run our new feature through all of them, then use a voting or averaging scheme to derive an ensemble classification result
- keep each tree small - use sqrt(k) features for it, chosen randomly from the overall 'k' samples

1/18 8:24:20 ***

← →

Machine learning

A gentle introduction

ML: BFD!!

Our 'menu' for today

We humans (not other animals!) have been to outer space, invented agriculture, found cures for diseases, invented countless things, create and use STEM, have radically altered the environment... But, these pale in comparison, when it comes to the promise/potential/dream of machine intelligence.

Our specific topic today - a DATA-DRIVEN approach to AI.

We will necessarily leave out the underlying math - courses related to ML (CS566, CS567...) will provide you that [almost all the math falls into these three categories: **statistics, probability** - for data sampling, experiment design, simulation, model building; **linear algebra** - for data description and analysis (eg. in NNs), geometric operations on data; **calculus** - for function optimization (eg error reduction, reward maximization)]

Note, "for the record" - I'm *not* against AI, I don't "hate" ML, etc!! :) That said, I do have strong thougths about future directions, reasons for shortcomings, etc.

Types of AI, Machine Learning ("ML"), types of ML

Here is a good way [after Arend Hintze] to **classify AI types** (not just techniques!)..

Type I: Reactive machines - make optimal moves - no memory, no past 'experience'. Ex: game trees.

Type II: Limited memory - human-compiled/provided , one-shot 'past' 'experiences' are stored for lookup. Ex: expert systems, neural networks.

Type III: Theory of Mind - "the understanding that people, creatures and objects in the world can have thoughts and emotions that affect the AI programs' own behavior".

Type IV: Self-awareness - machines that have consciousness, that can form representations about themselves (and others).

Type I AI is simply, application of rules/logic (eg. chess-playing machines).

Type II AI is where we are, today - specifically, this is what we call 'machine learning' - it is **"data-driven AI"**! Within the last decade or so, spectacular progress has been made in this area, ending what was called the 'AI Winter'.

As of now, types III and IV are in the realm of speculation and science-fiction, but in the general public's mind, they appear to be certainty in the near term :)

Practically speaking, there are exactly three types of AI that have been pursued, in the quest for human-level AI:

- inference-based: 'symbolic'
- goals/rewards-based: 'reinforcement'
- connection-based: 'neuro'

ML is the ONE subset of AI that is revolutionizing the world.

"Machine learning focuses on the construction and study of systems that can learn from data to optimize a performance function, such as optimizing the expected reward or minimizing loss functions. The goal is to develop deep insights from data assets faster, extract knowledge from data with greater precision, improve the bottom line and reduce risk."
- Wayne Thompson, SAS

ML comes in several flavors - the key types of machine learning include:

- Supervised learning
- Unsupervised learning
- Semisupervised learning
- Reinforcement learning

Here is a classification:

Supervised learning algorithms are "trained" using examples (**DATA!**] where in addition to features [inputs], the desired output [label, aka target] is known. The goal is to LEARN the patterns inherent in the training dataset, and use the knowledge to PREDICT the labels for new data.

Unsupervised learning is a type of machine learning where the system operates on unlabeled examples. In this case, the system is not told the "right answer." The algorithm tries to find a hidden structure or manifold in unlabeled data. The goal of unsupervised learning is to explore the data to find intrinsic structures within it using methods like clustering or dimension reduction.

For Euclidian space data: k-means clustering, Gaussian mixtures and principal component analysis (PCA)

For non-Euclidian space data: ISOMAP, local linear embedding (LLE), Laplacian eigenmaps, kernel PCA.

Use matrix factorization, topic models/graphs for social media data.

Here is a WIRED mag writeup on unsupervised learning:

Let's say, for example, that you're a researcher who wants to learn more about human personality types. You're awarded an extremely generous grant that allows you to give 200,000 people a 500-question personality test, with answers that vary on a scale from one to 10. Eventually you find yourself with 200,000 data points in 500 virtual "dimensions" - one dimension for each of the original questions on the personality quiz. These points, taken together, form a lower-dimensional "surface" in the 500-dimensional space in the same way that a simple plot of elevation across a mountain range creates a two-dimensional surface in three-dimensional space.

What you would like to do, as a researcher, is identify this lower-dimensional surface, thereby reducing the personality portraits of the 200,000 subjects to their essential properties - a task that is similar to finding that two variables suffice to identify any point in the mountain-range surface. Perhaps the personality-test surface can also be described with a simple function, a connection between a number of variables that is significantly smaller than 500. This function is likely to reflect a hidden structure in the data.

In the last 15 years or so, researchers have created a number of tools to probe the geometry of these hidden structures. For example, you might build a model of the surface by first zooming in at many different points. At each point, you would place a drop of virtual ink on the surface and watch how it spread out. Depending on how the surface is curved at each point, the ink would diffuse in some directions but not in others. If you were to connect all the drops of ink, you would get a pretty good picture of what the surface looks like as a whole. And with this information in hand, you would no longer have just a collection of data points. Now you would start to see the connections on the surface, the interesting loops, folds and kinks. This would give you a map.

Here is a practical use for unsupervised learning.

Semisupervised learning is used for the same applications as supervised learning. But this technique uses both labeled and unlabeled data for training - typically, a small amount of labeled data with a large amount of unlabeled data. The primary goal is unsupervised learning (clustering, for example), and labels are viewed as side information (cluster indicators in the case of clustering) to help the algorithm find the right intrinsic data structure.

With **reinforcement learning** 'RL'), the algorithm discovers for itself which actions yield the greatest rewards through trial and error. Reinforcement learning has three primary components:
1. agent - the learner or decision maker
2. environment - everything the agent interacts with
3. actions - what the agent can do

The objective is for the agent to choose actions that maximize the expected reward over a given period of time. The agent will reach the goal much quicker by following a good policy, so the goal in reinforcement learning is to learn the best policy. Reinforcement learning is often used for robotics and navigation.

Markov decision processes (MDPs) are popular models used in reinforcement learning. MDPs assume the state of the environment is perfectly observed by the agent. When this is not the case, we can use a more general model called partially observable MDPs (or POMDPs).

And there's also, hierarchical RL: https://sites.google.com/view/hrl-ep3

The magic

A fascinating documentary on (data-driven) ML: https://www.pbs.org/wgbh/frontline/film/in-the-age-of-ai - ~2 hours, every second of which is worth watching (because this is how our future is being shaped). Set aside 2 hours, watch it; or, as my friend Lurong would exclaim, "just-tu do it!!" :) **For now, we simply want a TL;DR - so, let's watch just till 1:15.**

Knosis.ai: https://www.youtube.com/watch?v=3RJ_YPh-1t8 - glimpses of how ML is FUELED by data!!

The core ideas [backprop, nonlinear activation, composition]

Neuro/connectionist/data-driven AI, works like so:

- data is gathered and cleaned; if necessary, each datum is assigned a **LABEL**; or the data used as-is
- the data+labels are used to TRAIN an algorithm (NN) iteratively - to reduce mislabeling 'loss'; more training is done if necessary
- the trained model is DEPLOYED to production - it can now label NEW data on its own [the 'machine' has 'learned' (the PATTERNS in the DATA)!]

Our brains contain about 100 billion of them - each neuron is like a function, with inputs ("dendrites"), and an output ("axon"):

Neurons ENCODE memory, learning... There are many types of neurons:

A neurons CONNECTS, via dendrites (inputs) and axon (output), to other neurons:

A neural network is a form of 'AI' - uses neuron-like connected units to **learn patterns** in training (existing) data that has known outcomes, and uses the learning to be able to gracefully respond to new (non-training, 'live') data.

Definition: a neural net(work) is **an interconnected set of weighted, nonlinear functions** [this compact definition will become clear[(er), soon]:

The overall idea is this:

- existing data is used to TRAIN a neural network - the network 'learns' patterns in the data, by adapting weights in each interconnected unit ('neuron')
- the network can now go 'live', ie. be deployed
- new data can be processed on the DEPLOYED network, which would make predictions about it based on the patterns learned

Guess why you are able to recognize these?

Neural networks (NNs) can be used to:

- recognize/classify features - traffic, terrorists, expressions, plants, words..
- detect anomalies - unusual CC activity, unusual machine states, gene sequences, brain waves..
- predict exchange rates, 'likes'..
- calculate numerical values (eg. home prices)
- ... [HUNDREDS, if not THOUSANDS, of uses - ANY form of data, that has ANY pattern in it, can be learned!!]

Here is some early NN work.

As you can imagine, 'Big Data' can help in all of the above! The bigger the training set, the better the learning, and therefore, better the result.

Below is an overview of how NNs work..

The brain (specifically, learning/training) is modeled after strengthening relevant neuron connections - neurons communicate (through axons and dendrites) dataflow-style (neurons send output signals to other neurons):

How the brain works on one slide!

- Each neuron receives inputs from other neurons
  - A few neurons also connect to receptors.
  - Cortical neurons use spikes to communicate.
- The effect of each input line on the neuron is controlled by a synaptic weight
  - The weights can be positive or negative.
- The synaptic weights adapt so that the whole network learns to perform useful computations
  - Recognizing objects, understanding language, making plans, controlling the body.
- You have about 10^{11} neurons each with about 10^4 weights.
  - A huge number of weights can affect the computation in a very short time. Much better bandwidth than a workstation.

Linear (identity), 'leaky' output: input values get passed through 'verbatim' (not very useful to us, does not happen in real brains!):

Linear neurons

- These are simple but computationally limited
  - If we can make them learn we may get insight into more complicated neurons.

$$y = b + \sum_i x_i w_i$$

A better model is when a neuron outputs a 1 (stays 0 to start with) ("fires") if and when its combined inputs exceed a threshold value:

Binary threshold neurons

- McCulloch-Pitts (1943): influenced Von Neumann.
  - First compute a weighted sum of the inputs.
  - Then send out a fixed size spike of activity if the weighted sum exceeds a threshold.
  - McCulloch and Pitts thought that each spike is like the truth value of a proposition and each neuron combines truth values to compute the truth value of another proposition!

Another option is to convert the 'step' pulse to a ramp:

Rectified Linear Neurons
(sometimes called linear threshold neurons)

They compute a linear weighted sum of their inputs.
The output is a non-linear function of the total input.

$$z = b + \sum_i x_i w_i$$

$$y = \begin{cases} z & \text{if } z > 0 \\ 0 & \text{otherwise} \end{cases}$$

Even better - use a smoother buildup of output:

Sigmoid neurons

- These give a real-valued output that is a smooth and bounded function of their total input.
  - Typically they use the logistic function
  - They have nice derivatives which make learning easy (see lecture 3).

$$z = b + \sum_i x_i w_i \qquad y = \frac{1}{1+e^{-z}}$$

Even better - use a sigmoidal probability distribution for the output:

Stochastic binary neurons

- These use the same equations as logistic units.
  - But they treat the output of the logistic as the probability of producing a spike in a short time window.
- We can do a similar trick for rectified linear units:
  - The output is treated as the Poisson rate for spikes.

$$z = b + \sum_i x_i w_i \qquad p(s=1) = \frac{1}{1+e^{-z}}$$

The functions we use to generate the output, are called activation functions - the ones we looked at are identity, binary threshold, rectifier and sigmoid. The gradients of these functions are used during backprop. There are more (look these up later) - symmetrical sigmoid, ie. hyperbolic tangent (tanh), soft rectifier, polynomial kernels...

This is from an early ('87) newsletter - today's NNs are not viewed as systems of coupled ODEs - instead we use 'training' to make processing element 'learn' how to respond to its inputs:

Basic theory

In engineering terminology, a neural network is a highly parallel dynamic system with the topology of a directed graph. NN nodes are referred to as "processing elements", and the directed links (information channels) are called "interconnects". Each processing element receives multiple inputs and generates a single output signal that branches into multiple copies that are sent to other processing elements as input signals.

Differential equations

Each processing element's operation is determined by differential equations that define how the output signal develops in time as a function of the input signals. These equations are called the "transfer function" equations of the processing element. Other differential equations ("learning laws") can be used for modifying adjustable coefficients in the processing elements' main transfer function equations. Hence, a complete artificial NN can be described as a large system of coupled, ordinary differential equations.

With the above info, we can start to build our neural networks!

* we create LAYER upon LAYER of neurons - each layer is a set (eg. column) of neurons, which feed their (stochastic) outputs downstream, to neurons in the next (eg. column to the right) layer, and so on

* each layer is responsible for 'learning' some aspect of our target - usually the layers operate in a hierarchical (eg. raw pixels to curves to regions to shapes to FEATURES) fashion

* a layer 'learns' like so: its input weights are adjusted (modified iteratively) so that the weights make the neurons fire when they are given only 'good' inputs.

Here is how to visualize the layers.

The above steps can be summarized this way:

- We start by choosing a model-class: $y = f(\mathbf{x}; \mathbf{W})$
  - A model-class, *f*, is a way of using some numerical parameters, **W**, to map each input vector, **x**, into a predicted output y.
- Learning usually means adjusting the parameters to reduce the discrepancy between the target output, t, on each training case and the actual output, y, produced by the model.
  - For regression, $\frac{1}{2}(y-t)^2$ is often a sensible measure of the discrepancy.
  - For classification there are other measures that are generally more sensible (they also work better).

Learning (ie. iterative weights modification/adjustment) works via 'backpropagation', with iterative weight adjustments starting from the last hidden layer (closest to the output layer) to the first hidden layer (closest to the input layer). Backpropagation aims to reduce the ERROR between the expected and the actual output [by finding the minimum of the [quadratic] loss function], for a given training input. Two hyper/meta parameters guide convergence: learning rate [scale factor for the error], momentum [scale factor for error from the previous step]. To know more (mathematical details), look at this page, and this.

Here is backprop again, in equation and code form:

For each input/target pair $(\mathbf{x}^{(n)}, t^{(n)})$ $(n = 1, \ldots, N)$, compute $y^{(n)} = y(\mathbf{x}^{(n)}; \mathbf{w})$, where

$$y(\mathbf{x}; \mathbf{w}) = \frac{1}{1 + \exp(-\sum_i w_i x_i)}, \tag{39.18}$$

define $e^{(n)} = t^{(n)} - y^{(n)}$, and compute for each weight w_i

$$g_i^{(n)} = -e^{(n)} x_i^{(n)}. \tag{39.19}$$

Then let

$$\Delta w_i = -\eta \sum_n g_i^{(n)}. \tag{39.20}$$

```
global x ;        # x is an N * I matrix containing all the input vectors
global t ;        # t is a vector of length N containing all the targets

for l = 1:L       # loop L times

   a = x * w  ;                          # compute all activations
   y = sigmoid(a) ;                      # compute outputs
   e = t - y   ;                         # compute errors
   g = - x' * e ;                        # compute the gradient vector
   w = w - eta * ( g + alpha * w )  ;    # make step, using learning rate eta
                                         #    and weight decay alpha
endfor

function f = sigmoid ( v )
   f = 1.0 ./ ( 1.0 .+ exp ( - v ) ) ;
endfunction
```

Here is backprop again, and resulting error reduction:

To quote MIT's Alex "Sandy" Pentland: "The good magic is that it has something called the credit assignment function. What that lets you do is take stupid neurons, these little linear functions, and figure out, in a big network, which ones are doing the work and encourage them more. It's a way of taking a random bunch of things that are all hooked together in a network and making them smart by giving them feedback about what works and what doesn't. It sounds pretty simple, but it's got some complicated math around it. That's the magic that makes AI work."

As per the above, here is a schematic showing how we could look for a face:

Note that a single neuron's learning/training (backprop-based calculation of weights and bias) can be considered to be equivalent to multi-linear regression - the neuron's inputs are features (x_0, x_1..), the learned weights are corresponding coefficients (w_0,w_1..) and the bias 'b' is the y intercept! We then take this result ('y') and non-linearize it for output, via an activation function. So overall, this is equivalent to applying logistic regression to the inputs. When we have multiple neurons in multiple layers (all hidden, except for inputs and outputs), we are chaining multiple sigmoids, which can approximate ANY continuous function! THIS is the true magic of ANNs. Such 'approximation by summation' occurs elsewhere as well - the Stone-Weierstrass theorem, Fourier/wavelet analysis, power series for trig functions...

A simpler example - a red or blue classifier can trained, by feeding it a large set of (x,y) values and corresponding blueness values - the learned weights in this case are the coefficients a and b, in the line equation ax+by=c [equivalently, m and c, in y=mx+c]:

Here is a simple network to learn XOR(A,B) - here all the 6 weights (1,1,1,1,-1,1) are learned:

X1 Value	X2 Value	AND Node $\sum W_i X_i$	OR Node $\sum W_i X_i$	XOR (from AND *-1 + from OR *1)
0	0	∑=0 → 0	∑=0 → 0	0*-1 + 0*1 =0 → 0 = False
0	1	∑=1 → 0	∑=1 → 1	0*-1 + 1*1 =1 → 1 = True
1	0	∑=1 → 0	∑=1 → 1	0*-1 + 1*1 =1 → 1 = True
1	1	∑=2 → 1	∑=2 → 1	1*-1 + 1*1 =0 → 0 = False

The following clip shows how a different NN (with one middle ('hidden') layer with 5 neurons) learns XOR - as the 5 neurons' weights (not pictured) are repeatedly modified, the 4 inputs ((0,0), (0,1), (1,0), (1,1)) progressively lead to the corresponding expected XOR values of 0,1,1,0 [in other words, the NN learns to predict XOR-like outputs when given binary inputs, just by being provided the inputs as well as expected outputs]:

This page has clear, detailed steps on weights updating.

In the above examples, there was a single neuron at the output layer, with a single 0 to 1 probability value as its output; if we had multiple neurons (one for each class we want to identify), we'd like their probabilities to sum up to 1.0 - we'd then use a 'Softmax' classifier [a generalization of the sigmoid classifier shown above]. A Softmax classifier takes an array of 'k' real-valued inputs, and returns an array of 'k' 0..1 outputs that sum up to 1.

NN-based learning has started to REVOLUTIONIZE AI, thanks to three advances:

- Big Data (BILLIONS of images, tens of thousands of hours of video/audio, terabytes of text, billions of tweets..), to use for training: more training predictably leads to better learning
- better algorithms - fruits of decades' worth of academic research in ML; more recently, 'industry' (Google, Facebook, Microsoft, IBM) seems to be taking the lead
- faster cloud computing platforms and libraries

Here is a (Jupyter) notebook with an NN implementation, if you want to play with it [you can simply look at the rendered/static version on GitHub, or interact with it].

'Summary':

And, REMEMBER [click]:

Imagine God's voice, BOOMING DOWN this answer:
THE SIGMOID IS THE *LINCHPIN* THAT HOLDS THE TRILLION DOLLAR AI ECONOMY TOGETHER!!!!!!!!!!
I cannot say this any louder, climbing on any taller mountain, lol.

IF we omit the sigmoid, what we'll have is a GIANT **LINEAR** equation (agree?) THAT WILL NOT CONVERGE, ie WON'T LEARN, even after trillions of backprop iterations!!! It's THAT important - **NO SIGMOID, NO ML!!!!!!!!!! Period.**

Again: if you remove the sigmoid, the network cannot learn the patterns in input data.

What we do in (supervised) ML is IDENTICAL to what we do in BI, DM!

It's ALL about **calculating quantities derived from patterns in existing data**.

EVERY neural network (which is really (supervised) ML is) is simply, a giant, deterministic, non-linear EQUATION!!!

y = F(X0,X1,X2...)

f(x0,x1,x2...) ⟶ y

(x0,x1,x2...) is a single piece (row) of data. Given it, WHAT IS 'y'? In other words, **WHAT IS f()? [wtf, lol]**

How can we calculate f()?

- using LOTS of data
- by iteration - via 'hyper params' such as architecture, learning rate, momentum; by optimizing our solving; by computing and using error between computed and expected outputs
- using nonlinearity
- using LOTS of small, simple 'neuron' subfunctions (out of which our f() is COMPOSED) [connected using specific *architectures*]

'Deep Learning' is starting to yield spectacular results, to what were once considered intractable problems..

Why now? Massive amounts of learnable data, massive storage, massive computing power, advances in ML.. Here is NVIDIA's response (to 'why now')..

In Deep Learning, we have large numbers (even 1000!) of hidden layers, each of which learns/processes a single feature. Eg. here is a (non-so-deep) NN:

"Deep learning is currently one of the best providers of solutions regarding problems in image recognition, speech recognition, object recognition, and natural language with its increasing number of libraries that are available in Python. The aim of deep learning is to develop deep neural networks by increasing and improving the number of training layers for each network, so that a machine learns more about the data until it's as accurate as possible. Developers can avail the techniques provided by deep learning to accomplish complex machine learning tasks, and train AI networks to develop deep levels of perceptual recognition."

Q: so what makes it 'deep'? A: the number of intermediate layers of neurons.

Applications, demos

ML is a runaway **engineering** success, which is sure to lead to 1000s (!) of applications, covering every human activity! Remember - if ANYTHING has a 'PATTERN' (that a. sets it APART from others, and b. has VARIATIONS within itself), it can be LEARNED!

Below is an arbitrary ("random") sampling of applications [some we talked about or encountered earlier]. The point is that "AI", ie. ML, is now mature enough, widely deployable enough that we can start dreaming up NEW USES for it!

- 'YOLO' in the browser
- tensorflow playground
- pages at GCP (eg Cloud Vision API) etc.
- https://www.youtube.com/channel/UCWN3xxRkmTPmbKwht9FuE5A - make the data 'lit'!
- SDCs, eg. https://www.youtube.com/watch?v=tiwVMrTLUWg
- Face detection, EU airports: https://www.cnn.com/travel/article/ai-lie-detector-eu-airports-scli-intl/index.html
- Face detection, Chinese classrooms(!): https://www.youtube.com/watch?v=3H1hj_C8F_A
- https://developer.amazon.com/blogs/alexa/post/ca34b954-1c5d-4a59-b326-f45c8df7c89c/alexa-skill-tech-for-good-challenge-winners
- https://ai.googleblog.com/2018/11/improved-grading-of-prostate-cancer.html
- ASL: https://www.youtube.com/watch?v=a4zvhJsBPa0
- 'master key' (uh oh): https://boingboing.net/2018/11/15/masterprints.html
- Google's Quick Draw, AutoDraw [here is an alternate implementation of QuickDraw; and, here is Google's dataset!]
- Inceptionism, DL portrait morph
- https://aiportraits.com/#
- neural style transfer [incl this clip :)]
- deepfakes, eg. https://www.bbc.co.uk/programmes/p06r8g4l [most are NSFW!!]
- https://medium.freecodecamp.org/chihuahua-or-muffin-my-search-for-the-best-computer-vision-api-cbda4d6b425d, http://www.evolvingai.org/fooling [easily foolable!]
- https://gallery.azure.ai/browse
- https://lobe.ai/
- https://research.google.com/seedbank/seeds
- https://thispersondoesnotexist.com/ [and https://thisrentaldoesnotexist.com/]
- https://ganbreeder.app/category/random
- https://www.askforgametask.com/html5/tutorials/tetris_ai_bot/source/ - an agent trained to play Tetris
- https://teachablemachine.withgoogle.com/

GPUs and other forms of hardware are used to accelerate deep learning - advantages: massively parallel processing, and possibility of arbitrary speed increases over time just by upgrading hardware!

GPUs (multi-core, high-performance graphics chips made by NVIDIA etc.) and DNNs seem to be a match made in heaven!

NVIDIA has made available a LOT of resources related to DNNs using GPUs, including a framework called DIGITS (Deep Learning GPU Training System). NVIDIA's DGX-1 is a deep learning platform built atop their Tesla P100 GPUs. Here is an excellent intro' to deep learning - a series of posts. Here is a GPU-powered self-driving car (with 'only' 37 million neurons) :)

Microsoft has created a GPU-based network for doing face recognition, speech recognition, etc.

Untether: https://www.technologyreview.com/the-download/613258/intel-buys-into-an-ai-chip-that-can-transfer-data-1000-times-faster/

The following are GPU-based NN implementations, by others:

- AMD
- Fujitsu
- Inspur

TPU (TensorFlow Processing Unit) is a Google-developed chip, for DNNs [in their Waymo cars].

Intel has its Neural Compute Stick...

FPGAs also offer a custom path to DNN creation.

Also: TeraDeep, CEVA, Synopsis, Alluviate..

A new form of CPU, involving 'chiplets' (from AMD) might also be a suitable platform...

Intel also has Nervana NNP-T.

There is a push to also deploy models on edge devices - SoCs, smartphones, browsers...

Eg. one trend is to build ML into cameras, eg. as done in Pixy2.

Google's TensorFlow can also run on the browser.

Here is a ConvNet (ie CNN) demo, running in the browser.

Here is an example of language processing on a smartphone.

Crop disease detection, in Kenya, using TF on an Android smartphone: https://www.youtube.com/watch?v=NlpS-DhayQA ...

We can also do simple object detection in the browser!

Players (People, companies, gov's)

- Amazon uses deep learning for product recommendations, Alexa...
- Google: self-driving cars, TensorFlow, DeepMind
- Microsoft: ImageNet entry, CNTK, Skype Translator... Here are a bunch of their AI demos.
- Facebook: DeepFace can search 800M faces in <5 sec! Also, Facebook is planning to open source its hardware setup. Deep learning is also used in Instagram, for recognizing content in images, including text.
- IBM: Watson, AlchemyAPI, Watson Analytics
- Apple uses deep learning for Siri, iTunes, etc.
- Many others: Alibaba, Baidu, Tencent, Uber, Netflix, Visa, LinkedIn...

The top three players - Amazon, Google, Microsoft - all have cloud-based APIs. Others - eg. FloydHub, Paperspace... other cloud-based ML training and hosting.

The 2018 Turing Award was for ML.

AI (ML, really) is tranforming world economies - everyone wants to participate, and WIN:

- US: this and this
- China: world domination
- India: address societal needs
- EU: strategy

Again - > watch the ~ 2 hour PBS that we brought up earlier.

Look up papers/blogs by:

- **Jürgen Schmidhuber**
- Andrew Ng
- Yann LeCun, Geoff Hinton, Yoshua Bengio
- Andrej Karpathy
- Chris Olah
- Brandon Rohrer
- Fei-Fei Li
- Yejin Choi

History...

Here is 'a' history of AI, to help set the stage. What's not shown, is the 'Cyc' project ('84-'94), which I worked on (for just a year). Also, here is my own material on the history of AI.

RNN, LSTM, **Transformers**

An RNN is a history-dependent network where past predictions are used for future ones (by having outputs fed back):

An LSTM is a special kind of RNN, for being able to process longer chains of dependencies.

RNNs/LSTMs are especially good for 'sequence' problems such as speech recognition, language translation, etc.; they are not massively parallelizable the way CNNs can be.

Here is more on LSTMs.

Temporal Convolution Nets (TCNs) are a good, parallelizable alt to RNNs; Numenta's HTM - also a better alternative to RNNs.

NN architectures

Specific architectures (numbers and types of layers) exist, for different NN tasks - eg. look at this page:
https://medium.com/@sidereal/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5

Before embarking on a big task, it is important to first identify, or create, a suitable architecture - otherwise, learning efficiency, and/or performance accuracy, will suffer.

CNN [Convolutional Neural Network]

In signal processing, a convolution is a blending (or integrating) operation between two functions (or signals or numerical arrays) - one function is convolved (pointwise-multiplied) with another, and the results summed.

Here is an example of convolution - the 'Input' function [with discrete array-like values] is convolved with a 'Kernel' function [also with a discrete set of values] to produce a result; here this is done six times:

Convolution, applied 6 times [from Ian Goodfellow's ML book]

Convolution is used heavily in creating image-processing filters for blurring, sharpening, edge-detection, etc. The to-be-processed image represents the convolved function, and a 'sliding' "mask" (grid of weights), the convolving function (aka convolution kernel):

From the River Trail documentation

Here is [the result of] a blurring operation:

Derived from the Gimp documentation

Here you can fill in your own weights for a kernel, and examine the resulting convolution.

So - how does this relate to neural nets? In other words, what are CNNs?

CNNs are biologically inspired - (convo) filters are used across a whole layer, to enable the entire layer as a whole to detect a feature. Detection regions are overlapped, like with cells in the eye.

Here is an*excellent* talk on CNNs/DNNs, by Facebook's LeCun.

Here is a *great* page, with plenty of posts on NNs - with lots of explanatory diagrams.

In essence, a CNN is where we represent a neuron's weights as a matrix (kernel), and slide it (IP-style) over an input (an image, a piece of speech, text, etc.) to produce a convolved output.

In what sense is a neuron's weights, a convolution kernel?

We know that for an individual neuron, its output y is expressed by

$y = x_0 \cdot w_0 + w_1 \cdot x_1 + \ \ldots\ . \ + w_n \cdot x_n + b$ where the w_is represent the neuron's weights, and the x_is, the incoming signals [b is the neuron's activation bias]. The multiplications and summations resemble a convolution! The incoming 'function' is $[x_0, x_1, x_2, \ \ldots\ . x_n]$, and the neuron's kernel 'function', $[w_0, w_1, w_2, \ \ldots\ . w_n]$.

Eg. if the kernel function is $[0, 0, 0\ \ldots\ w_0, w_1, 0, 0\ \ldots\ ]$ [where we only process our two nearest inputs], the equivalent network would look like so [fig from Chris Olah]:

The above could be considered one 'layer' of neurons, in a multi-layered network. The convolution (each neuron's application of w_0 and w_1 to its inputs) would produce the following:
$y_0 = x_0 \cdot w_0 + x_1 \cdot w_1 + b_0$
$y_1 = x_1 \cdot w_0 + x_2 \cdot w_1 + b_1$
$y_2 = x_2 \cdot w_0 + x_3 \cdot w_1 + b_2$
....

Pretty cool, right? Treating the neuron as a kernel function provides a convenient way to represent its weights as an array. For 2D inputs such as images, speech and text, the kernels would be 2D arrays that are coded to detect specific features (such as a vertical edge, color..).

EACH NEURON IS CONVOLVED OVER THE ENTIRE INPUT (again, IP-style), AND AN OUTPUT IS GENERATED FROM ALL THE CONVOLUTIONS. The output gets 'normalized' (eg. clamped), and 'collapsed' (reduced in size, aka 'pooling'), and the process repeats down several layers of neurons: input -> convolve -> normalize -> reduce/pool -> convolve -> normalize -> reduce/pool -> ... -> output.

The following pics are from a talk by Brandon Rohrer (Microsoft). You DON'T need to know the details of the steps - just understand that PIXELs are input, classification is the output.

What we want:

The input can be RST (rotation, scale, translation) of the original:

How can we compute similarity, but not LITERALLY (ie without pixel by pixel comparison)?

Useful pixels are 1, background pixels are -1:

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	1	-1	-1	-1
-1	-1	1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	1	-1	-1
-1	-1	-1	1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	1	-1	-1	-1
-1	-1	1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	1	-1	-1
-1	-1	-1	1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

We match SUBREGIONS:

Convolutional neurons that check for these three features:

CONVOLVE, ie. do $x_i \cdot w_i$ then average, output a value:

1 -1 -1
-1 1 -1
-1 -1 1
1 x 1 = 1
1

1 -1 -1
-1 1 -1
-1 -1 1
-1 x -1 = 1
1 1

1 -1 -1
-1 1 -1
-1 -1 1
-1 x -1 = 1
1 1 1

1 -1 -1
-1 1 -1
-1 -1 1
1 x 1 = 1
1 1 1
1 1 1
1 1 1

Need to center the kernel at EVERY pixel (except at the edges) and compute a value for that pixel!

We end up with a 7x7 output grid, just for this (negative slope diagonal) feature:

Each neuron (feature detector) produces an output - so a single input image produces a STACK of output images [three in our case, one from each feature detector]:

To collapse the outputs, we do 'max pooling' - replace an mxn (eg. 2x2) neighborhood of pixels with a single value, the max of all the m*n pixels.

Next, create a ReLU - rectified linear unit - replace negative values with 0s:

After a single stage of convolution, ReLU, pooling (or eqvt'ly, convolution, pooling, ReLU):

Usually there are multiple stages:

The resulting output values (12 in our case) are equivalent to VOTES: values at #0, #3, #4, #9, #10 contribute to voting for an 'X'; by repeated training with X-like images, which produce high-valued outputs for exactly those values at #0,#3,#4,#9,#10, the RECEIVER of all the 12 values, ie . the 'X' detector, learns to adjust its weights so that those inputs at #0,#3,#4,#9,#10 matter more (get assigned higher weight multipliers) compared to the other inputs such as #1,#2..:

Likewise, if we fed O image detectors kernels' results (also an array of 12 values) to the O receiver, the O receiver would classify it as an O - because the O detector has been separately trained, using several O-like images and O-feature detector neurons!!

After training, a new ('test') image is fed to BOTH the X feature detector neurons AND to the O feature detector neurons, who outputs are all combined to produce a 12-element array as before. Now we feed that array to both the X-decider neuron and the O-decider neuron:

Here's the output for X and O - the results average to 0.91 for X, and 0.52 for O - the NN would therefore classify this as an X:

If we feed the network an O-like image instead, the X and O detectors will go to work, and produce an output array where the O features (at #1,#2..) would be higher. So when this array is fed to the X decider and the O decider, we expect the image to be classified as O, eg. because the output probabilities from the X decider and O decider come out to be 0.42 and 0.89.

Repeat for each class that needs to be learned: test input => class detectors => outputs => train classifier.

This is very roughly equivalent to creating a "regression line" that "best fits" available data.

In summary:

In real world situations, these voting outputs can also be cascaded:

'All together now':

In the above, if we had fed an O-like image instead, the output probability would be higher for O.

Errors are reduced via backpropagation. Error is computed by taking the absolute differences' sums between expected and observed outputs:

	Right answer	Actual answer	Error
X	1	0.92	0.08
O	0	0.51	0.49
		Total	0.57

In RL, we'd use thousands of images for each class (outcome/label), and create a network that can detect dozens of classes - eg. here is a pictorial representation of an NN that can classify dogs:

For each feature, each weight (one at a time) is adjustly slightly (+ or -, using the given learning rate) from its current value, with the goal of reducing the error (use the modified weights to re-classify, recompute error, modify weights, reclassify.. iterate till convergence) - this is called backpropagation:

A Capsule Network (CapsNet) is a more robust (compared to regular CNNs) architecture for object detection; see also this page.

That was a whirlwind tour of the world of CNNs! Now you can start to understand how an NN can detect faces, cars..:

When is a CNN **not** a good choice? Answer: when data is not spatially laid out, ie. scrambling rows and columns of the data would still keep the data intact (like in a relational table) but would totally throw off the convolutional neurons!

GANs! And encoder-decoder pairs...

Adversarial learning methods, [esp GANs, that have dueling ["zero sum"] Generator and Discriminator networks] are very interesting.

GANs have MANY variations!

EBMs (a GAN alternative): https://openai.com/blog/energy-based-models/

As an alternative to GANs, a similar idea, called an Encoder-Decoder pair, can ALSO generate data (faces, words, music...). The encoder, specifically a 'VAE' learns to create a representation, a 'data generating distribution', of its input data, using latent-space features [of the input data]. Roughly, it learns to map an input datum into a point in multi-dim latent space. REVERSING this, **ANY random point in the latent feature space can be used to GENERATE (via a decoder) a NEW datum!** Here is more on VAEs.

Architecture pruning

By eliminating 'weak' (small weights) connections (or entire neurons), we can retain overall accuracy, and dramatically improve performance (esp on edge devices).

Problems...

Because it's ALL based on DATA, issues arise:

- bias
- deepfakes
- easy foolability
- lack of explainability
- unchecked power

And there are MUCH, MUCH, M-U-C-H bigger problems :(

- Rod Brooks
- Melanie Mitchell
- Gary Marcus
- ...

XAI

Rather than accept that an NN is a 'blackbox', XAI attempts to crack it open.

Diff eqs!

'Past' solutions (from t=0 to t=current) of a diff eq (ie for fluid flow, diffusion, vibration, EM propagation...) can be used as training data for an NN, which can then predict future evolution!

Rather than have NN layers (which are discrete), why not have a continuous NN (in the mathematical sense), and solve for weights using ODEs?

Current work (research directions)

Here is state-of-the-art...

- the Transformer architecture (Google, 2017) is a 'game changer' for language processing - it STACKS encoders and decoders, providing longer 'attention' spans. This has given rise to HUGE pre-trained language models: GPT-3, GPT-4, M'soft+NVIDIA, Wu Dao 2.0...
- optics-based NN
- another approach to AI is to model the brain's structure, in software or in hardware. IBM has its SyNAPSE chip, and TrueNorth NN chip. Numenta is another player in neuromorphic computing. Another approach to neuromorphic chips is to incorporate some wetware into them.
- Vision Transformers! (ViT)
- Neuro-symbolic integration, DeepRL... [combos]
- Geoff Hinton: GLOM
- GANsformers
- synthesizing images from text descriptions, eg. DALL-E and CLIP
- GNN - express data as a graph, learn the graph's structure, then predict properties given a new graph
- GDL - learn real-world shapes (topology)
- hmmm: https://spectrum.ieee.org/special-reports/the-great-ai-reckoning and https://bdtechtalks.com/2021/05/03/artificial-intelligence-fallacies/
- contrastive learning, eg. CLIP (these are behind the magic of 'AI art generators' - a form of "multimodal" learning
- LLMs!!!!! Including HFRL, OPL (?!)...

Summary

As we saw, there is a LOT going on, in ML! How to keep up?

- https://arxiv.org/search/?query=ML&searchtype=all
- https://www.louisbouchard.ai/research-papers/
- https://github.com/louisfb01/best_AI_papers_2021
- paperswithcode
- Two Minute Papers [https://wandb.ai/site/papers]
- Alpha Signal
- ...

1/10 8:24:54 ***

← →

DM/ML tools

Theory => practice

In the two previous lectures, we looked at a variety of algorithms for DM and ML, eg. kNN, clustering, neural networks.

Now we'll examine how these have been/can be implemented - using tools/frameworks or APIs/languages/hardware.

Note that in 'industry' (ie. outside academia and gov't), tools/APIs... are heavily used to build RL products - so you **need** to be aware of, and be knowledgeable in, as many of them as possible.

APIs/frameworks: part 1

These are the most heavily used:

- TensorFlow ('TF')
- Spark MLlib: https://spark.apache.org/mllib/ and https://spark.apache.org/docs/2.2.0/ml-pipeline.html
- Keras: https://keras.io/ [a higher level lib, compared to TF etc]; here are all the types of Keras layers
- Torch, PyTorch: https://pytorch.org, http://torch.ch/
- scikit-learn: https://scikit-learn.org/stable/
- Caffe: https://caffe2.ai/, http://caffe.berkeleyvision.org/ [→ Caffe2 → PyTorch]
- Apache mxnet: https://mxnet.apache.org/ [multi-language APIs, GPU and cloud support...]
- CNTK: https://docs.microsoft.com/en-us/cognitive-toolkit/

TL;DR: simply learn Keras or PyTorch, and if necessary, TF.

APIs/frameworks: part 2

Upcoming/lesser-used/'internal'/specific:

- here is FBLearner Flow - Facebook's version of TensorFlow :)
- Apache Mahout - a collection of ML algorithms, in Java/Scala
- .NET ML: https://dotnet.microsoft.com/apps/machinelearning-ai/ml-dotnet
- fastai [on top of PyTorch]: https://github.com/fastai/fastai
- OpenVINO: https://software.intel.com/en-us/openvino-toolkit and https://www.youtube.com/watch?v=rUwayTZKnmA&t=1s [a tutorial]
- Turi: an alternative to Apple's CreateML: https://github.com/apple/turicreate
- LibSVM: https://www.csie.ntu.edu.tw/~cjlin/libsvm/
- LightGBM: https://github.com/Microsoft/LightGBM
- XGBoost: https://xgboost.ai/ [and, look at Tianqi's slides and talk]
- CatBoost: https://tech.yandex.com/catboost/
- Google - SEED: https://ai.googleblog.com/2020/03/massively-scaling-reinforcement.html
- Uber's 'Fiber', for distributed ML training: https://venturebeat.com/2020/03/26/uber-details-fiber-a-framework-for-distributed-ai-model-training/
- LOTS of smaller efforts: https://github.com/EthicalML/awesome-production-machine-learning

Here is an article about deep learning tools.

Cloud

The virtually unlimited computing power and storage that a cloud offers, make it an ideal platform for data-heavy and computation-heavy applications such as ML.

Amazon: https://aws.amazon.com/machine-learning/ Their latest offerings make it possible to 'plug in' data analysis anywhere.

Google: https://cloud.google.com/products/ai/ [in addition, Colab is an awesome resource!]

Microsoft: https://azure.microsoft.com/en-us/services/machine-learning-studio/ [and AutoML] [aside: alternatives to brute-force 'auto ML' include 'Neural Architecture Search' [incl. this], pruning, and better network design (eg using ODEs - see this).

IBM Cloud, Watson: https://www.ibm.com/cloud/ai [eg. look at https://www.ibm.com/cloud/watson-language-translator]

Others:

- h2o: https://www.h2o.ai/products/h2o/ [supports R, Python, Java, Scala, JSON, native Flow GUI [similar to Jupyter], REST...]
- BigML: https://bigml.com/features#platform
- FloydHub: https://www.floydhub.com/
- Paperspace: https://ml-showcase.paperspace.com/
- Algorithmia, eg. https://info.algorithmia.com/ and https://demos.algorithmia.com/

With so much available out of the box, it's time for citizen data scientists?

Pretrained ML models

A pre-trained model includes an architecture, and weights obtained by training the architecture on specific data (eg. flowers, typical objects in a room, etc) - ready to be deployed.

Eg. this is simple object detection in the browser! You can even run this detector on a command line.

TinyMOT: https://venturebeat.com/2020/04/08/researchers-open-source-state-of-the-art-object-tracking-ai

Apple's CreateML is useful for creating a pre-trained model, which can then be deployed (eg. as an iPad app) using the companion CoreML product. NNEF and ONNX are other formats, for NN interchange.

Pre-trained models in language processing, include Transformer-based BERT and GPT-2. Try this demo (of GPT etc). There is GPT-3 currently available, GPT-4 in the works, Wu Dao 2.0, MT-NLG...

There are also, combined (bimodal) models, based on language+image data.

Tools

Several end-to-end applications exist, for DM/ML. Here popular ones.

Weka is a Java-based collection of machine learning algorithms.

RapidMiner uses a dataflow ("blocks wiring") approach for building ML pipelines.

KNIME is another dataflow-based application.

TIBCO's 'Data Science' software is a similar (to WEKA etc) platform. Statistica [similar to Mathematica] is a flexible, powerful analytics software [with an old-fashioned UI].

bonsai is a newer platform.

To do ML at scale, a job scheduler such as from cnvrg.io can help.

SynapseML is a new ML library from Microsoft.

There are a variety of DATAFLOW ('connect the boxes') tools! This category is likely to become HUGE:

- Perceptilabs: https://www.perceptilabs.com/
- Lobe: https://insights.dice.com/2018/05/07/lobe-deep-learning-platform/
- https://www.producthunt.com/posts/datature
- smartpredict: https://smartpredict.ai/
- StackML: https://stackml.com/ [RIP]
- Baseet: https://baseet.ai/ [RIP]

Languages

These languages are popular, for building ML applications (the APIs we saw earlier, are good examples):

- Python
- R
- Julia [Python 'replacement'?!]
- Wolfram
- JavaScript - this is a good list of JS-based libraries [look at ConvnetJS for nice demos]
- Scala - a functional+OO language - here is a roundup of libraries [these are in addition to Spark's MLlib Scala API]
- Java - another robust language for building ML libs [we already saw WEKA] and apps
- Jupyter [an environment, not a language] (eg. here is a collection of ML notebooks - as an exercise, run them all in Colab!) [also, here are notebooks for 'everything'!]
- ...

Hardware

Because (supervised) ML is computationally intensive, and detection/inference needs to happen in real-time almost always, it makes sense to accelerate the calculations using hardware. Following are examples.

Google TPU: TF is in hardware! Google uses a specialized chip called a 'TPU', and documents TPUs' improved performance compared to GPUs. Here is a pop-sci writeup, and a Google blog post on it.

Amazon Inferentia: a chip, for accelerating inference (detection): https://aws.amazon.com/machine-learning/inferentia/

NVIDIA DGX-1: an 'ML supercomputer': https://www.nvidia.com/en-us/data-center/dgx-1/ [here is another writeup]

Intel's Movidius (VPU): https://www.movidius.com/ - on-device computer vision

In addition to chips and machines, there are also boards and devices:

- Pixy2: https://pixycam.com/ - camera + ML in a single board
- Coral: https://coral.withgoogle.com/
- Jetson Nano: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-nano/
- Movidius NCS: https://software.intel.com/en-us/movidius-ncs
- ...

Overall, there's an explosion/resurgence in 'chip design', for accelerating AI training, inference. In April '21, NVIDIA announced its new A30 and A10 GPUs, at the annual [GTC] conference.

Summary

We looked at a plethora of ways to 'do' ML. Pick a few, and master them – they complement your coursework-based (theoretical) knowledge, and, make you marketable to employers! Aside: LOTS of salaries etc., revealed here :)

Also, FYI – in industry (G-MAFIA/FAANG/MAMMA MIA, BAT, more!), ML is part of a bigger 'production pipeline':

So, how do you prepare for the Data Roles?

- **SQL ! SQL ! SQL !** Practice a lot of SQL
- Be a savvy **Python** Developer
- **Think like a Product Manager**. Take up your favorite Application, and think of KPI (Key Performance Indicators). Determine the criteria for Decision Making
- **Teamwork and Collaboration** are essential skills needed for any Data Role. Be a good Communicator. Whether it be an interview or a Team Meeting, make sure to speak your mind
- Learn different **Visualization Techniques** and present your findings in the best way possible. Make it impressive
- Study **Datawarehousing concepts** for Data Engineering Roles
- Have a **basic understanding** of Data pipelines, MapReduce Concepts, Graph Models, Data Analytics platforms, Database Concepts, Kubernetes, Containers, and various open-source Apache Products. (Depth Knowledge is not needed – But, basic information will help you understand the bigger picture)

1/16 8:25:24 ***

← →

Visualization

of data

(https://multimedia.scmp.com/culture/article/SCMP-printed-graphics-memory/lonelyGraphics/201505A51.html)

What are we visualizing, and why?

Data!

Data visualization ("data viz") involves (the study of) tools and techniques for **turning data into images/graphics** - to obtain BETTER INSIGHT into the data.

In other words, this is about **graphical depictions of data**. **Why do it?** To understand, communicate, act/decide/utilize.

In what follows, we are going to look at what can be visualized, and how. Note: it's not all 'Big Data' viz, it's not all 'mined' results either.

Viz: a single variable

Classically, a pie-chart can be used to express relative fractions of a quantity:

A histogram/bar-chart can be used as well:

Also lookup: double histogram, density plot.

Bubble plots are also useful:

Wordles can be used to indicate relative strengths of keywords/topics. It is easy to create your own.

Simple graphics (bar charts, pie charts...) can be made more pleasing, using modern typography and layout techniques - here is a case in point.

Multivariate data - possibly the BEST viz ever?!

Edward Tufte calls the graphic below, ""The greatest statistical graphic ever drawn". Adapted from Wikipedia : 'Charles Minard's 1869 chart showing the number of men in Napoleon's 1812 Russian campaign army, their movements, as well as the temperature they encountered on the return path.' Specifically, the graph shows these 6 types of data (in 2D!): the number of Napoleon's troops; the distance traveled; temperature; latitude and longitude; direction of travel; and location relative to specific dates.

You can read more, here.

Viz: spatial data

Plotting spatial data (eg. incidence locations) on a map reveals patterns/trends in a 'direct' way - maps are 'intuitive' to humans...

Cholera outbreak map, 1854, London [plotting of reported cases as black dots reveals the source of the outbreak [highlighted region]):

It is quite useful for planning purposes, to visualize data over a map - eg. here are Starbucks locations..

Mined data, eg. associations, can be superposed over a map, eg. in a grocery store. Results can be used to redo the layout. A related topic is product placement.

As we saw earlier, a choropleth map shows spatial, aggregated data (that covers the entire region shown). These come in two varieties - unclassed (continuous scale), classed (discrete ranges).

And, more topical - here's a COVID-19 distribution map [choropleth, bubble types]. This shows a classed choropleth, segmented by US counties.

Viz: spatio-temporal data

Superposing time-varying data on a map reveals course, trends, etc. Such data could be visualized as animations, too.

Here is a different way to display spatio-temporal data [clickable] - using a speedometer-like needle/gauge symbol.

Viz: interactivity

Being able to INTERACT with data provides MORE understanding - we can selectively turn items on/off, drill down or roll up, explore the time dimension..

Crime map, USC area :(

Crimean War casualties, by Florence Nightingale (!) Here is a non-interactive version [I did the interaction and recorded a clip of it], and some background on the 'evidence-based healthcare' that Ms. Nightingale pioneered.

NOAA, Historic Hurricane Tracks [eg. under Name/Year, type in 'Katrina 2005']

Viz: animation

Even passively watching data being animated, provides us fresh perspectives.

Eg. here is a spatio-temporal hurricane map.

Population change, as measured by census data, is always interesting [there are more lovely, informative animations at the parent subreddit.]

Here is a different way to visualize relative change of a single variable over time. This is another currently relevant example.

Viz: real-time!

Real-time visualization provides a level of immediacy/freshness/relevance/interest that is simply absent in non-real-time data..

- world population growth [even more real-time stats!]
- local traffic (click on Options->Road Conditions->Fast--Slow) [amazing]
- earthquakes!
- stocks
- cybercrimes (!!); also this
- radio stations!

Viz: networks – node attrs

Network visualization is a very popular category - shows RELATIONSHIPS between entities.

A diagram that maps email exchanges between family members:

In the above, we are lacking DETAIL that can be added using extra ATTRIBUTES, and LABELS. Here is an enriched version that uses attrs and labels:

Viz: networks – edge attrs

We can use edge attrs (type, eg. arrows, dashes.., color, thickness etc.) to quantify data. The diagram below shows trade quantities between countries (2012, top 12 countries as per GDP):

Grab Bag of goodies

An assortment of 'cool' (visually appealing) and USEFUL data visualizations:

Clickstream Map of Science
Legend
Data

Points per attempt
Low
High
Few
Many
Number of attempts
All NBA field goal attempts
2006-2011
Data & Design: Kirk Goldsberry
KIRK GOLDSBERRY

CONFUSED ABOUT SYRIA? A GUIDE TO THE WAR'S FRIENDS, ENEMIES, AND FRENEMIES.
Slate
Syrian Government
Syrian Rebels
ISIS
Jabhat al-Nusra
Kurds
U.S. and Allies
Iraq
Iran and Hezbollah
Russia
Saudi Arabia, Gulf States
Turkey

wind map
Dec. 2, 2015
10:36 am EST
(time of forecast download)
top speed: 29.1 mph
average: 8.1 mph
1 mph
3 mph
5 mph
10 mph
15 mph
30 mph
Seattle
Denver
Chicago
Columbus
New York
Los Angeles
San Diego
Phoenix
Dallas
San Antonio
Houston

[from here]

Here is more eye candy.

This is a 'histomap' that puts history in perspective.

HOW to GENERATE data viz??

During the past lectures, we've looked at a few data viz examples (eg. GIS data). Here is a systematic breakdown of ways to create all manner of data viz.

Data science software

- Weka
- KNIME
- RapidMiner

Using code

- R, Shiny, ggplot2...
- matplotlib [eg. here is a walkthrough]
- d3 (JS) and Protovis
- PGFPlots (LaTeX)

Online tools

- META-CHART
- datavisual
- infogram
- Online Charts
- Slemma

Math, analysis and plotting packages

- Mathematica
- MATLAB
- OriginLab [and alternatives]
- good old Excel

3rd party data-viz software [many are complete platforms, offering 'dashboards']

- Periscope [here is a clip]
- Tableau
- SiSense
- Qlik [intro' video]
- Salesforce dashboards; design tips; how to create one
- domo
- JMP (SAS)
- Mode

As an exercise, learn to use AS MANY of these as you can! Use notebooks for R and Python, and CodePen/jsfiddle for JS.

Where is the 'science'??

Data viz is an art AND a science - there are principles, choices, tradeoffs. As for the principles, these encompass diverse disciplines such as visual perception, color theory, composition (grouping, contrast, harmony, symmetry..), design elements (line, tone, form, texture..), semiotics, etc.

As for what type of graphic to generate for a given type of data analysis, we can follow the guidelines here (from 'Atlas of Knowledge'):

Insight Need Types page 26	Data Scale Types page 28	Visualization Types page 30	Graphic Symbol Types page 32	Graphic Variable Types page 34	Interaction Types page 26
• categorize/cluster • order/rank/sort • distributions (also outliers, gaps) • comparisons • trends (process and time) • geospatial • compositions (also of text) • correlations/relationships	• nominal • ordinal • interval • ratio	• table • chart • graph • map • network layout	• geometric symbols point line area surface volume • linguistic symbols text numerals punctuation marks • pictorial symbols images icons statistical glyphs	• spatial position • retinal form color optics motion	• overview • zoom • search and locate • filter • details-on-demand • history • extract • link and brush • projection • distortion

Visualization Types (Reference Systems)

1. **Charts:** No reference system—e.g., Wordle.com, pie charts
2. **Tables:** Categorical axes that can be selected, reordered; cells can be color coded and might contain proportional symbols. Special kind of graph.
3. **Graphs:** Quantitative or qualitative (categorical) axes. Timelines, bar graphs, scatter plots.
4. **Geospatial maps:** Use latitude and longitude reference system. World or city maps.
5. **Network layouts:** Node position might depends on node attributes or node similarity. **Trees:** hierarchies, taxonomies, genealogies. **Networks:** social networks, migration flows.

USC's INF554 is a data viz course.

What makes for a good design?

Induce the viewer to think about the substance rather than about methodology, graphic design, the tech of graphic production, or something else. (Edward Tufte)

More...

Here is a brief whitepaper from SAS, that discusses data visualization.

Here's an inspiring TED talk...

InfoWeTrust: https://infowetrust.com/

Terrible Maps: https://www.reddit.com/r/terriblemaps/ - fun w/ mapping...

http://www.datavis.ca/gallery/

Some you've seen, some you haven't: https://www.tableau.com/learn/articles/best-beautiful-data-visualization-examples

Facebook!

1/16 8:26:04 ***

← →

"Etc."

What about all those 'peripheral' topics?

CGPEST

Our topics for today

'Data' handling is not just about storage, processing, and utilization! There is a constellation of 'issues' (topics/items) that surround the core, and these items are just as (if not, more) important [why?].

Here is what we'll be looking at:

- privacy
- security
- ethics
- trust
- compliance
- governance

Please care about the above! In the coming years, they will be become even more important, as data pipelines become rather commonplace. What are other examples of this?

PRIVACY

Background: 'Fair Information Practices', 1970s (FEP): The FIP efforts in organizations followed five tenants:

1. Openness.

2. Disclosure.

3. Secondary usage limits.

4. Correctability.

5. Security.

Today, e-commerce companies collect LOTS of info about customers - for immediate sales, and for analytics. Too much data collection makes them vulnerable to theft/leakage etc.

Simply put, our lives ARE NOT 'PRIVATE' ANYMORE!

1. Practically all web sites track us - e-commerce, social media... - and exchange data among themselves and brokers.

2. Your medical data belongs to - your hospital!

3. You are under surveillance, esp. if you are in China or Japan [for now].

4. Your search data is not private. Through data inference, dots can be connected...

5. 'They' know where you've been! Eg. malls, public security cameras, your own phone - all know where you are...

The notion of "privacy" needs to be redefined?!

Our digital conveniences have a flip side:

SECURITY

Unfortunately, data breaches are commonplace, and involve theft/exposure of value, identity...

Novel fronts: attacks on IoT [devices, data], including cars.

ETHICS

'Ethical' use of data involves multiple aspects!

First, there's the potential for 'rogue' AI. Eg. autonomous drones, and robot soldiers can act with bias, or equally badly, without bias and without morality.

Then there is the issue of fairness. This plays out for ex, in face recognition. IBM is contributing a dataset to help mitigate this.

A gov't can unfairly target protected groups...

Data, via ML, can be used to generate fake news, eg. fakevideo. This is a specific form of 'disinformation'. Disinformation was defined in Great Soviet Encyclopedia (1952) as "false information with the intention to deceive public opinion". Question: what, then, is 'misinformation'?

Here is one way to reduce bias, at the algorithm level.

Another angle to approach fairness is to improve the interpretability of ML.

You can learn more, here.

TRUST

How (much) can we (individuals) TRUST organizations (business, government, non-profit...) to RESPONSIBLY use **our** data?

Trust is proportional to transparency, value delivery, consequence acceptance. You can learn more, here.

COMPLIANCE

Compliance is a LEGAL/REGULATORY issue - what laws be passed, to help citizens have/gain control over their data?
Aside: what is the difference between a law, regulation, and policy?

In the EU, there's GDPR [General Data Protection Regulation].

Effective 5/25/18, the EU has a SINGLE set of privacy guidelines for its member countries and citizens, called GDPR, which requires businesses to protect the "personal data and privacy of EU citizens for transactions that occur within the EU."

Here is more on GDPR.

Interestingly (or not so), the US has a maze of regulations when it comes to digital privacy ("patch quilt protections"). To be fair, so did Europe, pre-GDPR.

GOVERNANCE

What are the RULES, related to the data we use?

The goal is to tease apart, learn about, and explore the connections between the following (data-related items): governance, curation, stewardship, MDM, provenance, metadata, security, privacy.

As you know, the purpose of capturing and storing data, is to process and benefit from it - this involves the use of statistics, data mining and machine learning.

But, that is not all there is to it! What about policies, procedures, rules, guidelines, practices... regarding the collection, storage and use of data?

Data Curation ['raw' => 'curated' datasets]

Regardless of collection procedures, analysis and usage, the ONE prime characteristic of data is QUALITY ('GIGO').

'Data curation' refers to set of processes and technologies ("methods and tools") that are focused on maintaining high-quality data in an organization, for the purposes of:

- visibility
- accessibility
- interoperability/heterogeneity
- reuse
- repurpose
- transparency
- ...

Any (which means ALL!) data-driven organization/s need(s) a 'data curation infrastructure' that supports curation practices and software.

Curation: key elements

Below are important points to keep in mind, while undertaking a data curation effort:

- the type of benefit derived from curated data, depends on the type of organization utilizing the curated datasets [eg. industrial R&D vs government vs new media companies]
- curation can be stimulated via incentives [that help justify the COST of curating]
- economic impact of curation can also help justify its need
- facilitating human-data interaction helps with curating - needs ways for non-technical users to handle data [eg. via natural language interfaces, semantic searching, viz, summarizing, transforming...] - building METADATA is a crucial step towards this
- large-scale curation efforts need to be hybrid between automated and human-involved efforts (curation by demonstration ['CBD'], crowdsourcing platforms, integration with enterprise data...]
- data curators need permission to access data they are curating; they need to be able to assign permissions (digital rights) to end-users of curated data; curators also need 'provenance' (data trail) info in order to determine curation specifics
- standards-based data models and representations (eg. ontology modeling using OWL) is necessary, for curation to include third-party/crowdsourced etc. data

Here is a description, by Craigin et. al. [Cragin, M., Heidorn, P., Palmer, C. L.; Smith, Linda C., An Educational Program on Data Curation, ALA Science & Technology Section Conference, (2007)]: "Data curation is the active and on-going management of data through its lifecycle of interest and usefulness; ... curation activities enable data discovery and retrieval, maintain quality, add value, and provide for re-use over time".

Data Governance (DG)

From Wikipedia: 'Data governance is a data management concept concerning the capability that enables an organization to ensure that high data quality exists throughout the complete lifecycle of the data'.

In other words, governance == curation?

NOT REALLY: As per the DAMA International Data Management Book of Knowledge, "Data Governance (DG) is defined as the exercise of authority and control (planning, monitoring, and enforcement) over the management of data assets." In other words, Governance is about POLICIES, which can be seen as complementary to Curation.

So an organization would have Governance **policies** in place, which would aid in Curation's producing **customized business data.**

Data Provenance

Provenance ~= "lineage".

'Data provenance documents the inputs, entities, systems, and processes that influence data of interest, in effect providing a historical record of the data and its origins.'

Provenance has to do with origins, while lineage has to do with tracing data's 'journey' to the current point of usage.

Provenance/lineage is a form of metadata that needs to be added to data, during curation.

Provenance helps establish TRUST (or lack thereof) in data. With scientific data for example, this is crucial [incorrect/invalid data would lead of the acceptance of incorrect hypotheses!].

Here is an interesting list of provenance-related issues related to scientific research (just fyi).

Lineage could have life or death consequences.

MDM [Master Data Management]

MDM is the management of "master data" (similar to a master key): In business, master data management (MDM) is a method used to define and manage the critical data of an organization to provide, with data integration, a single point of reference. [Wikipedia]

The idea is to maintain a single (meaningful, accurate, complete, timely) reference for data that is shared - this is done in order to maintain consistency. The alternative (to replicate such data for each request) would be highly problematic - would lead to inconsistency, errors, wasted disk space, increased network traffic, etc.

Here is more on MDM.

Governance, Security, Privacy - a TRINITY!

Security breaches are almost 'normal' - Yahoo, Home Depot, Facebook, Uber, Equifax... what is going on?

Governance is not being followed properly - policies for handling data and accountability, culture of/training in handling data, (pro)actively managing (sensitive) data - these are missing.

Privacy and security breaches are very costly, literally - lost revenue in the form of customer attrition, fines (eg. levied by SEC), lawsuit awards... These losses are monumental, compared to investing in technologies and policies that guard against breaches!

Note that maintaining security and privacy both involve minimizing RISK - something that every business ought to be concerned about.

Here is more on governance, as it relates to training ML models.

Security vs Privacy..

Data security/protection has to do with (preventing) UNAUTHORIZED access to data. Data privacy on the other hand, has to do with (limiting) AUTHORIZED access to data - related, but not identical!

Protection/security is a technical issue, related to protecting servers, encrypting data, restricting access (eg via passwords or biometrics), etc; privacy compliance on the other hand is a legal issue.

Also: data needs to be protectable first, before privacy can be ensured!

Discussion

Gov't surveillance - China

Social Credit - China

Ethics, and legalities, of fakevideo/fakeaudio/fakeimage/faketext... these are all weapons of 'information warfare'. Here is a related talk, and another.

COVID-19: privacy issues - location

COVID-19: provenance

COVID-19: disinformation

COVID-19: discrimination

COVID-19: data breach

...?

1/5 8:26:50 ***

← →

"Data data data!"

[what else could it be?!]

The centrality of data...

Society (government, politics, education, law enforcement...), commerce, environment, science/STEM, medicine, entertainment, communication, agriculture, manufacturing... (EVERYTHING) - DATA is an **integral** part!

In this course, you learned principles: of data organization, usage, applications.

The focus of the course: **practical** aspects of creating, storing, querying, using... data.

The vast 'tech' infrastructure: processing (cloud, GPUs...), storing (cloud...), communications (the Web, handhelds...) all are [seem] **tailor-made** for handling DATA!

RL connections...

'Extras' has a lot of useful, practical, current items!

Back to data: where do you/we go from here?

Ride the subway - back and forth, up and down town, round and round... never get off (NEVER STOP LEARNING)!!

So, it's all...

D•A•T•A !!!!