# COMP 330 Notes

## 0.1 Short History of Data Science

- Science of extracting actionable knowledge from data sets
  - Data sets are often large and complex
  - Large - too big to fit in the RAM of a server machine(128 - 256 GB)
  - Complex - Non tabular data, often multi-modal
- Statistics - foundation of Data Science
  - Emerges in the 1600s - centered around economic and demographic data
  - Probability developed in the 17th century - Pascal and Fermat - Motivated by study of gambling
  - Cholera Example(1854 Broad Street Cholera Outbreak) - Mapping cholera cases to find an outbreak source
  - Becomes more rigorous in late 1800s
    - Ronald Risher - null hypothesis, linear discriminator
    - Pearson and Neyman - "Type II" error, test power, confidence intervals
- Computer Science and Big Data
  - IBM IMS - early systems, designed in 1966 for the Apollo program
  - Inventoried materials for Saturn V moon rocket and Apollo space vehicle
- Machine Learning
  - Subset of "AI"(Artificial Intelligence)
  - AI - task of programming an "intelligent agent"
    - Able to solve novel problems
    - Emergent behavior
    - Can do things not specifically programmed to do
    - Goes back to Leibnez, Hobbes, Descartes'
    - Leibniz "Calculus ratiocintor"
      - 17th century
      - Precursor to mathematical logic
      - Imagines an inference engine able to reason
    - Artificial Neural Networks
      - First computational model proposed in 1943 - inspired by brain
      - McCulloch and Pitts - ANs can perform logic
      - Contemporary with ENIAC

- Minsky builds SNARC(Stochastic Neural Analog Reinforcement Calculator)
- Early ANNs: compute model not AI

* AI: term invented by John McCarthy in 1956
* Early quick success
  - AI as search: goals, subgoals, backtracking
  - Newell and Simon - "General Problem Solver"
  - Arthur Lee Samuel - Samuel Checkers-playing Program
  - Eliza: 1964 - 1966 MIT AI Lab - mock therapist - first serious attempt to pass "Turing Test" - not close to passing
  - Turing Test - based on Turing's "Computing Machinery and Intelligence"
    1. Three "players": Interrogator, Computer, Human
    2. Interrogator passes written questions to both
    3. Tries to determine which one is human
* Early Hype but ominous signs
  - Automatic Language Processing Advisory Committee - Critical of machine translation work
  - Minsky and Papert "Perceptrons" - seemed to kill off NN
  - "AI: A General Survey" by Lighthill - formed basis for decision by British government to end support for AI research in all but two universities
* Decline of AI
  - Speech Understanding Research program(DARPA)
    1. Harpy - recognized more than 1000 words but had severe limitations
    2. Led to massive cutback in DARPA AI funding
  - 1990 - Death of Fifth-Generation project
    1. In the 80's, Japanese invested $ 400 million(1990 dollars) - not much to show

– Rise of ML

* AI traditionally relies on smart programmers
  - May be able to "reason" and search for a better solution
  - Could chain together rules but didn't really learn
* Ends with advent of ML
  - ML is an AI that learns from retrospective data or experience
  - ML is often statistical in nature
* Backpropogation - Rumelhart, Hinton, Williams use BP to train a NN via gradient descent
  - Forerunner of all modern "deep learning"
  - Easier way to optimize weights of a NN
* Reinforcement Learning(Q-learning) - Watkins

            - Paradigm where ML learns via experience
            - Not pre-collected data
        * Association rule mining - kicks off field of "data mining"
        * Random Forests - Ensemble decision tree method
        * Support Vector Machines - "Max margin" method, works well for large number features, little data
        * LSTM - Type of NN which can remember state over sequential data
    - NVIDIA and CUDA Architecture(2007)
        * SGI graphics engineers move to NVIDIA in 1999
        * Graphics pipeline added in NVIDIA GPUs
        * CG programming language for GPUs, C for Graphics
        * GPGPU
        * cuDNN
    - ImageNet - classification of images, spurred vision research
    - Word2Vec - Word embedding, easy to use ML methods on text data
    - GANS - Generative Adversarial Networks - can generate images, two neural networks:
        * One to generate data
        * One to recognize real from fake data
    - Image Classification - ML algorithms can out-perform human experts at important tasks - finding tumors

# 1 Relational Databases 1

- Database - collection of data, set of programs for managing that data
- In past, dominant data model was the network or navigational model - data was a set of records with pointers between them
    - DB code written in COBOL
    - Code was written for specific storage model
    - Changing storage / indexing required modification
    - Led to little flexibility
- 1970 - development of relational model
    - Data stored in relations
    - Relation is a table of tuples or records
    - Attributes of a tuple have no sub-structure(are atomic)
    - Querying done via "relational calculus" - declarative language

    - Give a mathematical description of the tuples you want
    - System figures out how to get those for you
    - Data independence, code has no data access specs, can change physical org, no code re-writes

- Relation Schema - all data stored in tables or relations - relation scheme consists of:
  - Relation name
  - Set of attribute_name, attribute_type pairs - each pair is referred to as an attribute or column
  - Usually denoted using NAME (attribute_name, attribute_type)
  - Defined as a set of sets
    - If $T_1, T_2, ...T_n$ are the $n$ attribute types, where each $T_i$ is a set of possible values(for integers, $-2^{31}, 2^{31} - 1$, for strings)
    - A realization of the scheme(aka relation) is a subset of $T_1 \times T_2 \times ... \times T_n$ - huge set of all possible combinations of all possible values of each of its attributes
  - A corresponding relation for LIKES (DRINKER string, BEER string) could be {("Luis", "Modelo"), ("Sinan", "PBR")} - referred to as a table
    - Entries in the relation are referred to as "rows", "tuples", or "records"
  - Entire table is an instance of the relation schema
- In the relational model, given $R(A_1, A_2, ...A_n)$, a set of attributes $K = \{K_1, ..., K_m\}$ is a key(subset of the attributes in a relation that uniquely identifies each row in the relation) if
  - For any valid realization $R'$ of $R$
  - For all $t_1, t_2$ in $R'$...
  - If $t_1[K_1] = t_2[K_1]$ and $t_1[K_2] = t_2[K_2]$ and ... $t_1[K_m] = t_2[K_m]$
  - Then $t_1 = t_2$
- Relation scheme can have many keys
- Those that are minimal are candidate keys - no subset of the attributes is a key
- One is designated as the primary key - denoted with an underline
- Surrogate Key - no real-world meaning, added to simplify primary key - only job is to uniquely identify each student
- Connecting Relations - need some notion between tuple references
  - DRINKER(DRINKER, FNAME, LNAME)
  - LIKES(DRINKER, BEER)
  - LIKES.DRINKER refers to DRINKER.DRINKER but not the other way around

    - One person can like multiple beers but cannot have multiple entires for the same person in the DRINKER table but can have in the LIKES table
    - Can have a drinker than does not like beer but cannot have a beer that is liked by a non-drinker - someone can be in the drinker table and not in the likes table but a drinker cannot be in the likes table while not being in the drinker table
  - Using Foreign Keys - given relation schemas $R_1, R_2$:
    - Say a set of attributes $K_1$ from $R_1$ is a foreign key to a set of attributes $K_2$ from $R_2$ if
    - $K_2$ is a candidate key for $R_2$ and
    - For any valid realizations $R_1', R_2'$ of $R_1, R_2$...
    - For each $t_1 \in R_1'$ it MUST be the case that there exists $t_2 \in R_2'$ such that
    - $t_1[K_{1,1}] = t_2[K_{2,1}]$ and $t_1[K_{1,2}] = t_2[K_{2,2}]$ and ... $t_1[K_{1,m}] = t_2[K_{2,m}]$
    - Foreign key $k_1$ must be a set of attributes that uniquely identify a record in another table $R_2$
    - The combination of attribute values present in every tuple of $R_1$ must also be present in $R_2$
  - Cascading Deletes - having elements in one foreign key of child table but not in parent table - would delete all entries of that element
  - Failed Inserts - database system does not allow merging if foreign key in child table has an element while the same key in parent table $R$ does not occur

# 2 Relational Databases 2

- Relational Calculus is a variant on first-order logic
- "Give me all tuples t where $P(t)$ holds
  - $P(t)$ is a predicate - boolean function - can use logical operations and quantification(first order logic)
  - Important Equivalence $\forall X(P(X))$is equivalent to $\neg\exists(X)(\neg P(X))$

# 3 Relational Calculus and Algebra

Using syntax $\{t|P(t)\}$ Relational Calculus

- Start with {} and add the "such that" bar "|"
- Add what you are looking for to the left of the "|" - description of the tuples you want back

- Provide predicate that evaluates to True over all variables that appear on the left - if predicate is True, then tuple will be included
- Could treat as a pseudocode to find the relevant data

Relational Algebra - "abstract machine" of relational databases

- Algebra - set(domain) with a number of operations - this domain is closed under those operations
- In RA - the domain is the set of all valid relations - set of operations include $\pi, \sigma, \times, \bowtie, \cap, \cup, -$
- Projection - removes attributes $\pi_A(R)$
  - $A$ is set of attributes of relation $R$
  - Removes all attributes not in $A$ from $R$
  - Cardinality of output can differ from $R$ - can remove tuples if there are duplicates
  - Output is a relation
  - Like "select" in R
- Selection - removes tuples $\sigma_B(R)$
  - $B$ is a boolean predicate that can be applied to a single tuple from $R$
  - Simply removes all tuples not accepted by $B$
  - Output is a relation
  - Like "filter"
- Rename - $\rho_{A/B}(R)$ - renames attribute $B$ to $A$ in relation $R$ - output is a relation
  - $\rho^{(A_1,A,2,\ldots,A_n)}(R)$ - renames relation $R$ to $S$ and renames all attributes as specified output is a relation
  - Like "Rename"
- Assignment $X \leftarrow$ RA statement
  - Assigns output of an RA statement, which is a relation, to a temporary relation - for convenience
- Join : Cartesian Product - combines tuples, join is cartesian product
  - $R \times S$ - returns $r \cdot s$ for all $r \in R, s \in S$
  - Like an inner join
  - Cardinality is $|r \times s|$
- Theta Join - shorthand for $\sigma_B(R \times S)$ is $R \bowtie_B S$ - interested in only some of the tuples generated by the cross product

- Natural Join - want to join two relations
  - Using an equality check on all attributes having the same name
  - Then project away redundant attributes
  - Shorthand $R * S$
  - Like inner join

  Set-Based Operations
  - Types and numbers of input attributes must match
  - Attribute names come from LHS
  - $R \cup S$ - all tuples in $R$ or in $S$
  - $R \cap S$ - all tuples in R and in S
  - $R - S$ - all tuples in $R$ and not in $S$

# 4 SQL 1

De-facto standard DB programming language - consists of

- Declarative DML
- Imperative DML
- DDL(data definition language) - tells how you define schemas

In SQL:

- Duplicates are not automatically eliminated
- Not all SQL implementations(mySQL vs SQLserver etc.) are compatible
  - Support different set of operators
  - Date and time syntax
  - Comparison case sensitivity
- SQL extends RC / RA
  - Aggregate functions
  - Schema modifications

All RA have SQL equivalents:

- Projection - SELECT
- Join - FROM

- Selection - WHERE
- Union - UNION
- Intersection - JOIN / EXISTS / IN
- Difference - EXCEPT
- Rename - AS - technically optional, could just be "FREQUENTS f" instead of "FREQUENTS as f"
- Assignment - INTO

Declarative Query Structure:

```
SELECT <attribute list>
FROM <tables>
WHERE <condition>(Optional)
```

- Should also include what relation the attribute comes from even if it does not matter(e.g. "f.Drinker")
- Since SQL does not clean up duplicates, need DISTINCT keyword
- Can cartesian join through the syntax "FREQ SERVES as S"

As Keyword:

- Rename from RA
- Works on tables as well as attributes
- Actual keyword "AS" is optional

Where Clause:

- $< attribute >=< value >$
- $< attribute > BETWEEN [value1] AND [value2]$
- $< attribute > IN([value1], [value2], ...)$
- $< attribute > LIKE'SST\%'$ - true if LHS is a string that starts with "SST"
- $< attribute > LIKE'SST\_'$ - matches strings that start with "SST" and one character after
- $< attribute >$ IS NULL and $[attribute]$ IS NOT NULL
- Logical combinations with AND and OR

- Mathematical functions $<>, != , >, <, ...$
- Subqueries

Subqueries:

- Can have subquery in the WHERE clause - linked with keywords
- EXISTS
  - EXISTS $< subquery >$
  - If subquery returns at least one tuple, the clause evaluates to TRUE
  - Also a NOT EXISTS - subquery must return no tuples
  - Like existential quantifer
- IN
  - $< expression > IN < subquery > / < expression > NOTIN < subquery >$
  - Checks a value from a resultant table from a subquery is in that table
- ALL
  - $< expression >< boolOP > ALL < subquery >$
  - TRUE if every item in the subquery makes the boolOP evaluate to TRUE
  - Like for all
- SOME
  - $< expression >< boolOP > SOME/ANY < subquery >$
  - TRUE if some item in the subquery can make the boolOP evaluate to TRUE
- As we iterate over tuples of outer query - inner query is evaluated for each tuple
- Some can be evaluated just once - if not reliant on other parts of the our query
- Some require the subquery to be evaluated for every value assignment in the outer query

Views:

- "Common" (non-materialized) views are just macros
  - Unexecuted query
  - Can be used in place of a table
  - Convenient way to simplify a query
  - Query is executed when view is used by another query
  - Its results are not stored

Style

- Declarative SQL codes tend to be very short
- Good because effort, bugs $\propto$ code length
- Bad because sometimes difficult to understand
- Always alias tuple variables and indent carefully
- Only one major keyword per line
- Pick a capilization schema and religiously stick to it
- Make frequent use of views

# 5 SQL 2

Can compute simple statistics using SQL:

- SUM
- AVERAGE(AVG)
- MAX
- MIN

Can also compute aggregate at a finer level of granularity using GROUP BY

```
SELECT r.BEER, AVERAGE (r.RATING)
FROM RATES r
GROUP BY r.BEER
```

- First groups relation into subsets
- Every tuple in the subset has the same value for r.BEER
- Aggregate run over each subset independently
- Can only have the attribute being grouped by in SELECT outside of an aggregate function in an aggregate query
- For NULL:
  - COUNT(*) will count every row
  - COUNT(< *attribute* >) will count NON-NULL values
  - AVG, MIN, and MAX will ignore NULL
  - GROUP BY includes a row for NULL

- Subquery in FROM:
  - Can have a subquery in FROM clause
  - Treated as a temporary table
  - MUST be assigned an alias
- Can use TOP (n) to get top of a relation
  - Can optionally use PERCENT keyword
  - Can add WITH TIES
  - Can also choose ASC or DESC
  - ORDER BY can be used without TOP

# 6 SQL 3

- HAVING - allows us to check for conditions on aggregate functions
- JOINS
  - TABLE t1 JOIN TABLE2 t2 ON pred
  - TABLE t1 INNER JOIN TABLE2 t2 ON pred
  - Exact Same, classic join
  - TABLE t1 NATURAL JOIN TABLE2 t2 - no need for attributes since common attributes are found
- Can do CROSS JOINs - whichare cross product: TABLE t1 CROSS JOIN TABLE2 t2 is the same as TABLE1 t1, TABLE2 t2
- In FROM, can have joins of the form

```
TABLE t1 LEFT OUTER JOIN TABLE2 t2 ON pred
TABLE t1 RIGHT OUTER JOIN TABLE2 t2 ON pred
TABLE1 t1 FULL OUTER JOIN TABLE2 t2 ON pred
```

  - Includes all tuples from the outer side
  - Assigns NULLS if there is no matching tuple
  - Pick one and stick with it
- Every attribute type can take the value NULL
  - NULL is a special value
  - Used to signal a missing value

  - Nearly all non-comparison ops taking NULL as input return NULL - ISNULL(exp)

- SQL uses 3-value logic:
  - Values are true, fale, and unknown
  - Truth tables generally make sense - true and unknow give unknown, true or unknown gives true
  - Any comparison with NULL returns unknown - for a where to accept the tuple, must get a true

- Data Manipulation Language(DML)
  - Data retrieval(SELECT)
  - Data insertion(INSERT)
  - Data deletion(DELETE)
  - Data modification (UPDATE)

- Data Definition Language(DDL)
  - Relation definition(CREATE TABLE)
  - Relation scheme update (ALTER TABLE)
  - Relation deletion(DROP TABLE)
  - Creating tables:

```
CREATE TABLE RATES {
DRINKER VARCHAR (30),
BEER VARCHAR (30),
SCORE INTEGER
}
```

  - Can define a primary key using

```
PRIMARY KEY (DRINKER, BEER)
```

  - No primary key allows for duplicates

- UNIQUE can accept NULL values - can only be on primary key but multiple unique fields / combinations

- Foreign Keys:

```
ALTER TABLE RATES ADD CONSTRAINT FK
FOREIGN KEY (DRINKER, BEER)
REFERENCES LIKES (DRINKER, BEER)
```

- FK is used as a constraint to ensure that the FK values exist in the parent table

- Can manually add tuples using INTERSERT INTO relation VALUES (val1, val2, val3)

# 7 Imperative SQL 1

- Encapsulation - make it easy for programmer
- Safety - protect database from the programmer
- Performance - fewer end-to-end trips
- Can respond to events, dynamic queries
- Common form of imperative code:
  - Procedure whose code is stored in the DB
  - Can be invoked from the command line, external program
  - Or from another stored procedure
-
```
CREATE PROCEDURE procName
/* list params */
AS BEGIN
/* code here */
END;
```

- Example: Write a stored procedure to get the peak count in a given region - return overall count if no region given

```
PEAK(NAME, ELEV, DIFF, MAP, REGION)
CREATE PROCEDURE getNumPeaks
/* list params */
@whichRegion VARCHAR (8000) = NULL
AS BEGIN
DECLARE @queryString VARCHAR(8000);
SET @queryString = 'SELECT COUNT(*) FROM PEAK' +
ISNULL(' WHERE region = ''' + @whichRegion + '''', ' ');
EXECUTE (@queryString);
END;
```

  - All local vars need a DECLARE
  - @ - used to declare variable names
- Susceptible to injection -

```
'Southern Sierra'; DROP TABLE PEAK; --
```

- Leads to the query

```
SELECT COUNT(*) FROM PEAK WHERE region = 'Southern Sierra';
DROP TABLE PEAK; --';
```

- "–" makes everything after a comment
- Then to call:

```
EXECUTE getNumPEAKS
@whichRegion = 'Corocoran to Whitney';

CREATE VIEW MyView as SELECT * FROM Employees
WHERE Department = 'Sales';
SELECT * FROM MyView;

vs.

CREATE VIEW myView AS SELECT * FROM Employees
WHERE Department = 'Sales';
GO
SELECT * FROM MyView;
```

- GO allows the SQL query to execute, without GO just compiles

```
CREATE PROCEDURE getNumPeaks
@whichRegion VARCHAR (8000),
@result INT OUTPUT
AS BEGIN
DECLARE myRes CURSOR FOR
SELECT COUNT(*) FROM peak WHERE region = @whichRegion;
OPEN myRes;
FETCH myRes INTO @result;
CLOSE myRes;
DEALLOCATE myRes;
END
```

- Now have a 'cursor'
  - Standard abstraction for dealing with record sets
  - Essentially an iterator
  - For CLOSE - releases resources associated with a cursor
  - For DEALLOCATE - removes output obtained from query, removes cursor's reference and frees all associated resources

- To call procedure:

```
DECLARE @myResult INT
EXECUTE getNumPeaks
@whichReqion = 'Kaweahs and West',
@result = @myResult output;
PRINT @myResult;
```

# 8 Imperative SQL 2

Writing a stored procedure giving the tallest height in a region

```
CREATE PROCEDURE getTallestPeak
@whichRegion VARCHAR (8000),
@result VARCHAR (8000) OUTPUT
AS BEGIN
...
END
```

Body 1: Declare control vars and cursor

```
    DECLARE @peakName VARCHAR (8000);
DECLARE @bestName VARCHAR (8000);
DECLARE @peakHeight INT;
DECLARE @bestHeight INT;
SET @bestHeight = -1;
DECLARE myRes CURSOR FOR
SELECT name, elev FROM peak WHERE region = @whichRegion;
```

Body 2: Open cursor and loop to find the tallest peak

```
    OPEN myRes;
FETCH NEXT FROM myRes INTO @peakName, @peakHeight;
WHILE (@@FETCH_STATUS = 0) BEGIN
IF @peakHeight > @bestHeight BEGIN
SET @bestHeight = @peakHeight;
SET @bestName = @peakName;
END
FETCH NEXT FROM myRes INTO @peakName, @peakHeight;
END
```

FETCH

- 0 - good
- -1 - fail or beyond result set

- -2 - fetched row missing
- -9 - cursor not performing a fetch operation
- FETCH NEXT
- FETCH FIRST
- FETCH LAST
- FETCH ABSOLUTE n - get nth record
- FETCH RELATIVE n - go n records from here

Body 3: return the result

```
    CLOSE myRes;
DEALLOCATE myRes;
SET @result = @bestName;
END;
```

To call:

```
    DECLARE @myResult VARCHAR (8000);
EXECUTE getTallestPeak
@whichRegion = 'Corocoran to Whitney',
@result = @myResult output;
PRINT @myResult;
GO
```

- Shouldn't really write this
- TOP k would be shorter, easier, and faster
- use as much declarative code
- Only use loops, etc. when you have to
- Sometimes 3+ orders of magnitude speed difference

TSQL Functions

```
    CREATE FUNCTION foo (@myArg INTEGER)
RETURNS INTEGER AS BEGIN
...
RETURN (@someValue);
...
```

- Can be called from witin a SQL statement

```
    SELECT *
FROM SOME_TABLE s
WHERE foo (s.SOME_ATT) = 12;
```

Can have table-valued functions

```
    CREATE FUNCTION ProductsCostingMoreThan(@cost money)
RETURNS TABLE
AS
RETURN
SELECT ProductID, UnitPrice
FROM Products
WHERE UnitPrice > @cost

SELECT *
FROM ProductsCostingMoreThan (2567) t
WHERE t.ProductID = 12;
```

Triggers - stored procedures that fire in response to some event(e.g. trigger catches updates to peak table, prints error message and does not process

```
    CREATE TRIGGER checkHeight ON peak FOR UPDATE AS BEGIN
...
END
```

- FOR / AFTER - run only once triggering action succeeds - FOR occurs before or during an update, after occurs after update
- INSTEAD OF - don't run triggering action; run this instead
- Can have UPDATE, INSERT, DELETE

Body 1: Exit check and declarations

```
    IF @@ROWCOUNT = 0 BEGIN
RETURN;
END
DECLARE myRes CURSOR FOR
SELECT d.name, d.elev, i.elev FROM inserted i, deleted d
WHERE i.name = d.name AND i.elev <> d.elev;
DECLARE @peakName VARCHAR (8000);
DECLARE @oldHeight INT;
DECLARE @newHeight INT;
```

- @@ROWCOUNT - number of tuples affected
- deleted - table containing old versions of records

- inserted - table containing new versions

Body 2: Loop through and reject updates

```
    OPEN myRes;
FETCH NEXT FROM myRes INTO @peakName,
@oldHeight, @newHeight;
WHILE (@@FETCH_STATUS = 0) BEGIN
PRINT 'You changed the height of ' + @peakName +
' from ' + CAST (@oldHeight AS VARCHAR(100)) +
' to ' + CAST (@newHeight AS VARCHAR(100)) +
'. I am ignoring it.';
UPDATE peak SET elev = @oldHeight
WHERE name = @peakName;
FETCH NEXT FROM myRes INTO
@peakName,
@oldHeight, @newHeight;
END
CLOSE myRes;
DEALLOCATE myRes;
```

No arrays, linked lists, etc. in TSQL - everything is a table

```
    DECLARE @myMap TABLE (
myKey INTEGER,
myValue VARCHAR (200),
PRIMARY KEY (myKey);
);
DECLARE @my2DArray TABLE (
xPos INTEGER,
yPos INTEGER,
value DOUBLE
PRIMARY KEY (xPos, yPos);
);
```

# 9 Connected Components & Page Rank

Connected Component:

- Subgraph where any two nodes are connected to each other by paths, not connected to other nodes in the supergraph
- Found using BFS

PageRank:

- Measure associated to graph nodes

- Shows relative importance of the nodes
- Higher PR value $\implies$ more importance
  - A node is important when being pointed by lots of other nodes or by other important nodes
- PR models a random web surfer
- Random web surfer starts at a random page
- Randomly clicks on the out-going links to go other pages
- Can keep continue doing this(d probability) or jump to a random page(1-d)
- d is the Damping Factor
- PR value of a webpage is the probability that a random web surfer stops clicking when s/he lands on that webpage
- PR of node $j$ = probability that random surfer lands on $j$
  - Can come from an in-coming link to node $j$
  - Can also have a random jump to node $j$
- PR formulation for node $j$ for a graph $n$:

$$((1-d) \times 1/n) + (d \times \{PR\ value\ when\ surfer\ comes\ from\ an\ incoming\ link)$$

- Essentially a weighted average
- PR value when surfer comes from an incoming link
  - Assume each node has a PR value
  - Each link is assigned a weight by dividing PR into number of out-going links
  - PR value of a node when surfer comes from an incoming link is the sum of the weights of in-coming links
- Assuming $n$ nodes, initialize PR of all nodes equal $1/n$
- Iteratively calculate new PR values based on old values for all nodes

$$PR_{new}(node_j) = ((1-d) * 1/n) + (d * \sum_{k \in \{nodes\ pointing lnode_j\}} \frac{PR_{old}(node_k)}{\#\ of\ outgoing\ links\ of\ node_k}$$

- Stop when PR values converge

$$\sum_j |PR_{new}(node_j) - PR_{old}(node_j)| \leq \epsilon$$

- Sinks - nodes with no out-going link
  - With current formulation, sum of all PR values is normalized to 1
  - If we have sinks in the graph, sum of PR values will be less than 1
  - Solution - add links to all nodes from a sink including itself - PR of sink will be evenly distributed to all nodes

# 10 Modeling

- Traditional statistical definition - set of assumptions regarding the stochastic process that generated the data
- More modern - a mathematical object that enables an analyst to use data to understand the past and present, and make predictions about the future
- Real data - big, complex, difficult to understand
- Model is compact and simple - more about simplification

Modeling Process

- Choosing the model
  - Select distribution family
  - Choosing hyperparameters - external to model and not based on data

# 11 Gradient Descent

Optimization

- Adjusting model parameters to minimize or maximize an objective function
- Optimization at the heart of all learning frameworks
- Desired properties - easily applied, scalable, fast

Gradient Descent

- Have some function $L(\theta_0, \theta_1)$, where we want $min_{\theta_0,\theta_1} L(\theta_0, \theta_1)$
- Start with some $\theta_0, \theta_1$
- Keep changing $\theta_0, \theta_1$ to reduce $L(\theta_0, \theta_1)$ until we hopefully end up at minimum

Algorithm

- Repeat until convergence:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} L(\theta_0, \theta_1), j = 0, j = 1$$

- Simultaneous update

$$temp0 := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} L(\theta_0, \theta_1)$$
$$temp1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_0} L(\theta_1, \theta_1)$$
$$\theta_0 := temp0$$
$$\theta_1 := temp1$$

$$\Theta_1 \leftarrow \text{non-stupid guess for } \Theta^*$$
$$i \leftarrow 1$$
$$repeat\{$$
$$\Theta_{i+1} \leftarrow \Theta_i - \lambda \nabla L(\Theta_i);$$
$$i \leftarrow i + 1;$$
$$\} \; while(||\Theta_i - \Theta_{i-1}||_1 > \epsilon)$$

Gradient Descent for Linear Regression

$$h_\theta(x) = \theta_0 + \theta_1 x$$
$$L(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)} - y^{(i)})$$
$$\frac{\partial}{\partial \theta_j} L(\theta_0, \theta_1) = 2 \cdot \frac{1}{2m} \sum_{1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot 1$$
$$\frac{\partial}{\partial \theta_0} L = ...$$
$$\frac{\partial}{\partial \theta_1} L = 2 \cdot \frac{1}{2m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x^{(i)}$$
$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{1}^{m} (h_\theta(x^{(i)}) - y^{(i)})$$
$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x^{(i)}$$

## 11.1 Normal Equations

- Closed form solution to find regression coefficients - $\theta = (X^T X)^{-1} X^T y$
- Slower for larger n

- If $X^TX$ is non-invertible - get redundant features or too many features

Choosing Learning Rate

- Line Search
- Bold Driver
  - Make large conservative guess for $\alpha$
  - At each iteration, compute $L(\Theta i)$
  - If $L(\Theta i) < L(\Theta_{i-1})$
    - Reaching minimum loss
    - Incremenet $\alpha$ a little but
    - Increase slowly so don't miss convergence
  - If $L(\Theta_i) \geq L(\Theta_{i-1})$
    - Indicative of divergence
    - Reduce $\alpha$ by a lot
    - Don't update the parameters in that iteration
  - One evaluation of the loss function per iteration(less expensive than line search)
- "Batch" Gradient Descent - each step of gradient descent uses all training examples
- Stochastic Gradient Descent - at each step pick one random point, continue as if entire dataset was just one point
- Minibatch Gradient Descent - pick small subset of data points, continue as if your entire dataset was just this subset

# 12 Intro to Big Data

- "Big" - too large to fit in RAM of an expensive server machine(5GB in 2002, 6 TB in 2025)

MapReduce

- Data processing paradigm
  - Leverages cluster of commodity machines
  - Distributed processing and fault tolerance
  - Requires distributed file system
  - Well-suited for calculations that can be computed independently
- To process a data set - have two pieces of user-supplied code: Map function and reduce function

- Huge shared-nothing compute cluster
  - Using three data processing phases
  - A Map phase
  - A shuffle phase
  - Reduce phase
- Scalability and fault tolerance
- Functional model already existed - consider map and reduce functions
  - Map, applies function onto a list
  - Reduce, recursively applies function of iterable until converges to single number
- Shared-Nothing
  - Store/analyze data on large number of commodity machines
    - Local, non-shared storage attached to each of them
    - Only link is via a LAN
    - Shared nothing refers to no sharing of RAM, storage - NAS is common now, "pure" shared-nothing rarer
  - Benefits
    - Inexpensive, built out of commodity components
    - Compute resources scales nearly linearly with money
    - Contrast to shared RAm machine with uniform memory access
    - Easier to program than shared memory systems
- The Map Phase
  - Input data are stored in a huge file - contains a simple list of pairs of type (key1, value1)
  - Have a UDF of the form MAP(key1, value1) - outputs a list of pairs of the form (key2, value2)
  - In the Map phase of the MapReduce computation - Map function is called for every record in the input, instances of Map run in parallel all over the cluster
- The Shuffle Phase
  - Accepts all of the (key2, value2) pairs from the Map phase - groups together
  - After grouping all of pairs from all over the cluster having the same key2 value are merged into a single (key2, list(value2)) pair
  - Called a "Shuffle" - where a potential all-to-all transmission can happen, can be expensive, implemented using a hash function to map keys to nodes

- Reduce Phase
  - Apply user-supplied function of the form Reduce(key2, list(value2))
  - Outputs a list of value3 objects
  - In reduce phasee of the MapReduce computation
    * Reduce function is called for every key2 value output by the Shuffle
    * Instances of Reduce run in parallel all over the compute cluster
    * Output of all of those instances is collected
    * Put in a huge output file
- Map reduce is a data processing paradigm - requires with idea of distributed file system
  - Allows data to be stored / accessed across machines in a network
  - Abstracts away differences between local and remote data
  - Same API to read/write data - no matter where data is located in the network
- Distributed File System - MapReduce DFS sits on top of each machine's OS - lives in "user space"
- On top of OS for heterogeneity and easily portable

# 13 Second-Order Methods

- Class of iterative optimization methods
  - Use not only first partial but second
  - Speeds convergence
  - More complex, quadratic in number of variables

## 13.1 Newton's Method

- Classic second order method for optimization
- Comes from Newton's method for finding zero of a function F()
- Recall that zeroes are roots / solutions
- Use approach to keep finding better approximations to the zeros of a function
  - Make an initial guess
  - Approximate $F(\theta)$ with a line tangent to F() at that guess
  - Update $\theta$
  - $\theta$ is the value of the model parameter

```
theta <- initial guess;
while theta keeps changing, do:
       theta <- theta - F'(theta)^(-1) F(theta)
```

- Intuition:
  - Pick a value for theta
  - Evaluate F(theta)
  - Evaluate the derivative of the function at theta, F'(theta)
  - Revise theta based on these values
  - Repeat until convergence of theta
- Difference from Gradient Descent - want to find zero of function, root finding not minimum finding
- Don't want zero in dats science - max/min of loss function, so $F(\theta) \to L'(\theta)$

```
theta <- initial \\
while theta keeps changing do
theta <- theta - L'(theta) / L''(theta)
```

- Say we have multi-variate function $F : R^d \to R^d$, d is number of dimension
  - $ith$ output of $F$ is given by the function $F_i$
  - So $F(\theta) = (F_1(\theta), F_2(\theta), ..., F_d(\theta))$
  - Want to find a zero of F, find $\theta =< \theta_1, \theta_2, ...\theta_d >$ such that

$$F_1(\theta_1, \theta_2, ..., \theta_d) = 0$$
$$\vdots$$
$$F_d(\theta_1, \theta_2, ..., \theta_d) = 0$$

- To do this - need to find the Jacobian of $F$ - $d \times d$ matrix of functions
- Rows cover the different parameters for a single dimension
- Columns cover the value of $F$ for each dimension for a single parameter
- Can evaluate it at any set of parameter values, so $J_F(\theta)$ is a matrix of scalars
- For multi-variate optimization:

- Need to min/max at $\Theta$ such that

$$\frac{\partial L}{\partial \theta_1}(\Theta) = 0$$
$$\frac{\partial L}{\partial \theta_2}(\Theta) = 0$$
$$...$$
$$\frac{\partial L}{\partial \theta_d}(\Theta) = 0$$

- Want $\Theta$ such that $\nabla L(\Theta) =< 0, 0, ..., 0 >$
- Need to find Hessian - each entry is 2nd derivative of loss function wrt to each parameter
- Each row is the Jacobian given a set of model parameters
- Pros and Cons of Newton's
  - Pros
    - Convergence is quadratic - error decreases quadratically
    - Hundreds/ Thousands of iterations becomes tens
    - No learning rate to set
    - Doesn't require $F(\Theta)$ to be convex
  - Con
    - More complicated than gradient descent
    - Quadratic cost each iteration(linear gradient descent) - Hessian is quadratic in the number of variables
    - Cost is worse than quadratic, since matrix has to be inverted
    - Second derivative has to exist
- Not used much in practice since in high dimensions, $d \times d$ is too big
- Usable for $< 100K$ parameters, really hard at $1M$
- Quasi-Newton methods are used instead
- Typically use just a portion or estimation of the Hessian matrix

# 14 Expectation Maximization

- Widely-used MLE algorithm for dealing with missing data
- Often impossible to drop(needing to be continuous and differentiable) missing data or unrealistic to replace with mean

- Formally, want to compute an MLE for $L(\theta|x_1, x_2, ..., z_1, z_2, ...)$
- $x_1, x_2$ are observed data, $z_1, z_2$ are missing
- $L(\theta|x_1, x_2, ..., z_1, z_2) = f(x_1, x_2, ...z_1, z_2, ...|\theta)$
- When z's are missing, choose $\theta$ to max

$$\int_{z_1,z_2,...} f(x_1, x_2, ..., z_1, z_2, ...|\theta)d(z_1, z_2, ...)$$

- "Integrating out" a variable - e.g. if we have height, weight pairs - find probability that a pair is "tall" as $\sum_{weightw} Pr[(tall, w)]$
- Basic Idea:
  - Have an estimate $\Theta^{iter}$ for each iteration
  - Repeatedly update $\Theta^{iter}$ until convergence

$$Q(\Theta^{iter}, \Theta^{iter-1}) = E[logf(x_1, x_2, ..., z_1, z_2, ...|\Theta^{iter})|x_1, x_2, ..., \Theta^{iter-1}]$$

  - Treat $z_1, z_2...$ as random variables
  - With distribution $f(z_1, z_2, ...|x_1, x_2, ...\Theta^{iter-1})$
  - Q-function is expected value of LLH wrt this distribution
- Continuous version is

$$Q(\Theta^{iter}, \Theta^{iter-1}) = E[logf(x_1, x_2, ..., z_1, z_2, ...|\Theta^{iter})|x_1, x_2, ..., \Theta^{iter-1}]$$

- If $Q$ is discrete:

$$(\Theta^{iter}, \Theta^{iter-1}) = E \sum_{z_1,z_2,...} f(z_1, z_2, ...|x_1, x_2, ...\Theta^{iter-1})logf(x_1, x_2, ..., z_1, z_2, ...|\Theta^{iter})$$

- EM Algorithm
  - Start with reasonable guess as to best $\Theta$, call this $\Theta^0$, do not make parameters equivalent, will not converge
  - Choose $\Theta^{iter}$ to maximize the expected value of the log-likelihood
  - Computing $f(x_i|\Theta^{iter-1})$ requires pass over the data - "E-step"
  - For Q function, let $c_{i,j} = f(z_i = j|x_i, \Theta^{iter-1})$
  - Write Q function as:

$$\sum_{z_1}\sum_{z_2}\sum_{z_3}...(\prod_i c_{i,z_i})logf(x_1, x_2, ..., z_1, z_2, ...|\Theta^{iter})$$

  - Need to maximize

# 15 Spark

Hadoop - "Pure" MapReduce Software

- Data reread from DFS for each MR job
- Bad for iterative data processing like gradient descent
- Can only do Map or MapReduce - everything in terms of those operations
- New alternative softwares - streaming and batch processing applications
- Generally oriented more towards in-memory computing - more expressive APIs

Apache Spark

- Platform for efficient distributed data analytics - runs on JVM
- Written in Scala - bindings for Java, Scala, Python, R
- Doesn't do storage - focus exclusively on compute
- RDDs - Resilient Distributed Data Set(RDD) - basic abstraction
  - Resilient - fault tolerance
  - Distributed - across machins in a cluster
  - Read-only
  - Data set buffered in RAM by Spark
  - Lazy - computations done on demand
  - Ephemeral - computations are discarded when not used
  - Lineage - computation sequence to generate an RDD is saved

```
myRDD = sc.textFile(someFileName) #sc is Spark context
data = [1,2,3,4,5]
myRDD = sc.parallelize(data) or
myRDD = sc.parallelize(range(20000))
```

  - Collect: returns the entire RDD - brings entire RDD to local environment - can lead to crashes

```
RDD.collect()
```

  - Top: returns n entries based on implicit ordering

```
myRDD.top(3)
```

  - Take: returns first n entries

```
myRDD.take(5)
```

- Computations - series of transformations over RDDs
- Lambda - nameless function that we can pass like a variable
  - Can "capture" its surroundings at creation
  - Can also accept parameters
  - Can return many items but must return something
- flatMap:

```
def countWord(fileName):
    textFile = sc.textFile(fileName)
    lines = textFile.flatMap(lambda line: line.split(" "))
```

- Process every data item in the RDD
- Apply lambda to it
- Lambda arguement to return zero or more results
- Each result added into resulting RDD
- map vs flatMap
  - Use map when want one-to-one transformation - input and output have same size
  - use flatmap when you want a transformation that can generate zero or more elements from each input
- reduceByKey()

```
def countWords (fileName):
    textFile = sc.textFile (fileName)
    lines = textFile.flatMap (lambda line: line.split(" "))
    counts = lines.map (lambda word: (word, 1))
    aggCounts = counts.reduceByKey (lambda a, b: a + b)
```

  - Data must be Key,Value pairs
  - Shuffle so that all (K,V) pairs with same K on same machine
  - Organize into $(K, (V_1, V_2, ..., V_n))$ pairs
  - Use lambda to "reduce" the list to a single value
- top()

- ```
  def countWords (fileName):
  textFile = sc.textFile (fileName)
  lines = textFile.flatMap (lambda line: line.split(" "))
  counts = lines.map (lambda word: (word, 1))
  aggCounts = counts.reduceByKey (lambda a, b: a + b)
  return aggCounts.top (200, key=lamda p: p[1])
  ```
  - Data must be (Key,Value) pairs
  - Takes two parameters, first is number to return
  - Second(optional) - lambda to use to obtain key for comparison
  - top collects an RDD, moving from cloud to local - so result is not an RDD

- Spark uses lazy evaluation - if running

  ```
  textFile = sc.textFile (fileName)
  lines = textFile.flatMap (lambda line: line.split(" "))
  counts = lines.map (lambda word: (word, 1))
  aggCounts = counts.reduceByKey (lambda a, b: a + b)
  ```

- Nothing happens(other than spark remembering the ops) - does not execute until attempt made to collect an RDD

- By waiting until last second - opportunities for "pipelining" exploited, only ops that require a shuffle can't be pipelined

- groupByKey()
  - Data must be (Key, Value) pairs
  - Shuffle so that all (K,V) pairs with same K on same machine
  - Organize into $(K, (V_1, V_2, ..., V_n))$ pairs
  - Store each list as a resultiterable for future processing
  - Like reduceByKey() but without reduce

- join()
  - Given two data sets rddOne, rddTwo - join using

    ```
    rddOne.join(rddTwo)
    ```

  - Returns $(K, (V_1, V_2))$ pairs
  - Constructed from all $(K_1, V_1)$ from $rddOne$, $(K_2, V_2)$ from $rddTwo$, where $K_1 = K_2$
  - Can blow up RDD size if join is many-to-many, requires expensive shuffle
    ```

- reduce() - not a transform from RDD to RDD
  - Repeatedly applies a lambda to each item in RDD to get single result
- aggregate()
  - Data must be in Key,Value pairs
  - Organized into $(K, (V_1, V_2, ..., V_n))$ pairs
  - Aggregate the list, like reduce()
  - Takes three args
    - The "zero" to init the aggregation
    - Lambda that takes $X_1, X_2$ and aggs them, where $X_1$ already aggregared, $X_2$ not
    - Lambda that takes $X_1, X_2$ and aggs them, where both aggregated

# 16 Supervised Learning

Machine Learning Review:

- Objective: Predicting/classifying new data based on what we learn from existing data
- Requires training data to be representative of the data we want to predict/classift
- Data collection procedure is paramount
- Tasks:
  - Choosing the model - family, complexity, hyperparameters
  - Learning the model - "fit" the model by learning its parameters
  - Validating the model - make sure model matches data
  - Applying the model - use model to explain past/present and make predictions on future

Supervised Learning

- Given a bunch of $(x_i, y_i)$ pairs
- Goal: learn how to predict value of y from x
- Called "supervised" because have examples of correct labeling
- Common Examples - classification and regression
- Simplest model is a linear regression - others like kNN, SVMs

Measuring Regression Performance

- View list of prediction errors - $\epsilon_i = (f(t_i) - x_i)$
- Can have many loss functions, corresponding to norms
- A norm is a function that maps a vector to a non-negative scalar
- Given a vector of errors $\epsilon_i, l_p$ norm defined as

$$(\sum_{i=1}^{n} |\epsilon_i|^p)^{\frac{1}{p}}$$

- $l_0$ - sum of non-zero values
- $l_1$ - mean absolute error
- $l_2$ - mean squared error
- $l_\infty$ - max absolute value

Measuring Classification Performance

- Accuracy - easy to understand and compute but terrible with unbalanced classes
- Measured through a confusion matrix - accuracy in terms of True Positive and True Negative, False Positive and False Negative
- False Positive Rate - Number False Positive / Number False - $\frac{FP}{N} = \frac{FP}{TN+FP}$
- False Negative - Number False Negative / Number of True - $\frac{FN}{P} = \frac{FN}{TP+FN}$
- Recall: $\frac{TP}{P} = \frac{TP}{TP+FN}$
- Precision - $\frac{TP}{TP+FP}$
- Combined in $F_1$ score
  - $F_1 = \frac{2 \times Precision \times Recall}{Precision + Recall}$
  - Ranges from 0-1 - single number
  - Doesn't consider false positive/recall trade-off

ROC Plot

- Can modify recall by changing internal parameter in classifier
- Increase $c$ - fewer false positives, lower recall
- ROC - "Receiver Operating Characteristic"
- Measures effect of increasing recall on False Positive rate - plots TPR against FPR at each threshold c(ROC Plot)

- AUC-ROC curve converts plot to single number - usually between 0.5 - 1 - less than 0.5 means actively bad

Feature Engineering / Selection

- Focus in supervised learning on models
- Models are almost always less important than feature engineering
- How to vectorize features

# 17 GLM

Linear Regression is just a model with normal error

$$x_i, y_i \sim N(r \cdot x_i, \sigma^2)$$
$$Likelihood - \Pi_i N(y_i | r \cdot x_i, \sigma^2) = \Pi_i \sigma^{-1} (2\pi)^{-1/2} e^{-1/2(y_i - r \cdot x_i)^2 \sigma^{-2}}$$

So for MLE, we try to maximize the same loss function as LR

$$-\sum_i (y_i - r \cdot x_i)^2$$

GLM is generalization of LR, allows error to be generated by wide variety of distribution. Any probability distribution can be writte nas

$$p(y|\theta) = b(y) exp(\theta T(y) - f(\theta))$$

Say we have prediction problem where

- Want to predict output $y$ from an input vector x
- Natural to assume randomness or error on y is produced by some exponential family
- The ExpFam parameter $\theta$ is linearly related to $x$, assuming $x$ vector-valued, $\theta$ is scalar valued

$$\theta = \sum_j x_j \times r_j$$

# 18 Overfitting

Detecting Over-Fitting

- Sniff Test
- Independent validation and test sets

- Prioritize simplest solutions

Bias-Variance Trade-Off

- Bias - error from incorrect model assumptions
- Variance - sensitivity of model to bad data
- Expected squared error of any prediction is

$$E[(Y - \hat{f}(x))^2]$$

- $x$ is the data used for prediction
- $Y$ produces the value we are trying to predict from $x$
- $\hat{f}(.)$ is the model we are learning

$$Var(Y) = E[Y^2] - E^2[Y]$$
$$Bias^2(\hat{f}(x)) = E^2[\hat{f}(x) - Y]$$
$$E[(Y - \hat{f}(x))^2] = Var(Y) + Var(\hat{f}(x)) + Bias^2(\hat{f}(x))$$

- Means error of supervised learner is a sum of
  - "Losseness" of relationship between $x$ and $Y$, $(Var(Y))$
  - Sensitivity of the learner to variability of the training data$(Var(\hat{f}(x))$
  - Inability of learner $\hat{f}$ to learn relationship between $X$ and $Y$, $Bias^2(\hat{f}(x))$

Regularization

- Give learning algorithm ability to choose complexity of model
- Simpler model - lowers $Var(\hat{f}(X))$, raises inductive bias
- Complex Model - raises $Var(\hat{f}(x))$, lowers inductive bias
- Balances trade-off between bias and variance to get lowest error
- Penalizes model for complexity
- Penality based on $l_p$ norm

For logistic regression:

- $p = 1$ - LASSO Regression
- $p = 2$ - ridge regression

# 19 Outliers

Outliers - unexpected or unusual points - need some sort of reference as unusual data is not the same as an outlier

Distance-Based Outlier

- Point is an outlier if it is far from all other points
- Let $d(x_i)$ be the distance to points $x_i$'s kth NN in the data set
- Given data set $x_1, x_2, ...$ want to compute the set $O$ such that
- $|O| = m$ and $\forall(x_0 \in O, x_i \in X - O), d(x_i) \leq d(x_0)$
- Loop through all points, for all other points, check distance - keep track of closest $n$ neighbors
- The highest $m$ points would then be considered outliers
- Choosing distance function - use $l_p$ norm
- Choosing $m$ - start small, gradually increase until nothing "interesting" is added
- Choose $k$ - determined using validation set, try $\sqrt{N}$

Model-Based Outlier

- Learn a model for what "typical" is
- Outlier is data point with low score according to the model

# 20 Deep Neural Networks

At highest level - nonlinear function represented usually as a graph

- Nodes - neurons:
  - Little computational units
  - Computed function must be differentiable since learning is done via gradient descent
- Output of neuron is called an "activation"
- Neurons organized into layers
- Top layer is "output layer"
- Bottom layer is "input layer" - activation of input neurons read directly from input features

- "Hidden Layers" between those two layers
- Each link between neurons has a "weight"

Function of Each Neuron

- Need neuron to compute some function $f(v)$
- Sigmoid function - neuron goes from "off" to "on" - smoothly so we can differentiate
- Classically two are used in NNs
  - Hypoerbolic Tangent - $tanh(v)$
  - Logistic Function - $(1 + e^{-v})^{-1}$
  - Modern Alternative is ReLU - $max(0, v)$

# 21 RNNs

FNNs don't easily handle sequences

- Standard idea - Use FF network with enough input units
- Pad unused tokens with special chars, set number of inputs to max sequence
- Problemated for predicting on longer sequences, last inputs are not trained very much

Elman Network

- Split hidden layer into tokens
- Tokens can be anything - characters or words
- Usage of state layer after a hidden layer which feeds back into the hidden layer
- Have set of context nodes(state layer)
- Read value of hidden layer - non-trainable where the value is simply remembered for one time tick

Jordan Network

- Similar to Elmann - copies output values, not hidden values
- Must be producing an output at each time tick
- Can be used for sequence-to-sequence
- Easier learner but harder to remember long-term dependencies
- Not suitable for representartion learning

Training

- Classic algorithm is back-propagation
- View RNN as a compact representation of a complex graph
- Unroll the complex graph
- Use backpropogation on that - weights are constrained to repeat

Vanishing Gradient

- Errors fall off exponentially as backpropped through layers
- Derivative of loss wrt activation function often much smaller than 1
- Repeatedly multiplying causes backpropped erros to end to zero
- Backprop does not affect weights in first few layers

Solutions

- Gated Neurons
- Special RNN designed to deal with vanishing gradient problem
- Use gates to control flow of information
- Move away from continuous S-shaped activation feature
- Simpler, piece-wise-linear activation functions