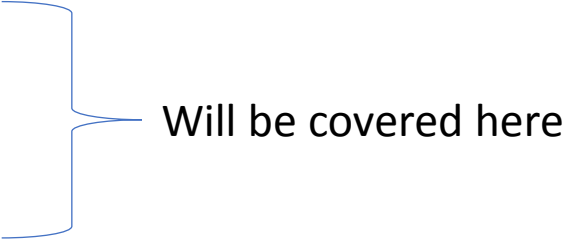


Use of Association algorithm,
Neural network and Graph
mining

What have been covered so far

- **Introduction to machine learning**
 - **Data management with Python**
 - List
 - Dictionary
 - Set
 - Tuple
 - Array
 - Data frame
 - **Introduction of data visualization (ggplot2)**
 - **Use of regression and correlation**
 - **Use of decision tree (CART)**
 - **Use of clustering algorithm**
 - **Web scraping with BeautifulSoup and Selenium**
 - Use of association algorithm
 - Use of neural network
 - Use of graph mining
- 
- Will be covered here

Finding pattern with Association rules

Association rules

- *Co-occurrence grouping or association discovery* attempts to find associations between entities based on transactions involving them.
- Why would we want to find such cooccurrences?
- There are many applications. Consider a consumer-facing application.
- Let's say that we run an online retailer.
- Based on shopping cart data, we might tell a customer, "Customers who bought the new eWatch also bought the eBracelet Bluetooth speaker companion."
- If the associations indeed capture true consumer preferences, this might increase revenue from cross-selling

Example

- By looking at the sets of purchases, one can infer that there are a couple of typical buying patterns.
- A person visiting a sick friend or family member tends to buy a get well card and flowers, while visitors to new mothers tend to buy plush toy bears and balloons.
- Such patterns are notable because they appear frequently enough to catch our interest;
- we simply apply a bit of logic and subject matter experience to explain the rule.

Transaction number	Purchased items
1	{flowers, get well card, soda}
2	{plush toy bear, flowers, balloons, candy bar}
3	{get well card, candy bar, flowers}
4	{plush toy bear, balloons, soda}
5	{flowers, get well card, soda}

Question: Can you identify the association rules manually here ?

Association rule mining

- Proposed by **Agrawal et al in 1993**.
- It is an important data mining model studied extensively by the database and data mining community.
- Assume all data are categorical.
- No good algorithm for numeric data.
- Initially used for **Market Basket Analysis** to find how items purchased by customers are related.

Bread → Milk [sup = 5%, conf = 100%]

The model: data

- $I = \{i_1, i_2, \dots, i_m\}$: a set of *items*.
- Transaction t :
 - t a set of items, and $t \subseteq I$.
- Transaction Database T : a set of transactions $T = \{t_1, t_2, \dots, t_n\}$.

Transaction data: supermarket data

- Market basket transactions:

t1: {bread, cheese, milk}

t2: {apple, eggs, salt, yogurt}

...

...

tn: {biscuit, eggs, milk}

- Concepts:

- *An item*: an item/article in a basket
- *I*: the set of all items sold in the store
- *A transaction*: items purchased in a basket; it may have TID (transaction ID)
- *A transactional dataset*: A set of transactions

The model: rules

- A transaction t contains X , a set of items (**itemset**) in I , if $X \subseteq t$.
- An **association rule** is an implication of the form:
 $X \rightarrow Y$, where $X, Y \subset I$, and $X \cap Y = \emptyset$
- An **itemset** is a set of items.
 - E.g., $X = \{\text{milk, bread, cereal}\}$ is an itemset.
- A **k -itemset** is an itemset with k items.
 - E.g., $\{\text{milk, bread, cereal}\}$ is a 3-itemset

Rule strength measures

- **Support:** The rule holds with **support** sup in T (the transaction data set) if $sup\%$ of transactions contain $X \cup Y$.
 - $sup = \Pr(X \cup Y)$.
- **Confidence:** The rule holds in T with **confidence** $conf$ if $conf\%$ of transactions that contain X also contain Y .
 - $conf = \Pr(Y \mid X)$
- An association rule is a pattern that states when X occurs, Y occurs with certain probability.

Support and Confidence

- **Support count:** The support count of an itemset X , denoted by $X.count$, in a data set T is the number of transactions in T that contain X . Assume T has n transactions.
- Then,

$$support = \frac{(X \cup Y).count}{n}$$

$$confidence = \frac{(X \cup Y).count}{X.count}$$

Example-1/2

- The **support** of an itemset or rule measures how frequently it occurs in the data.
- For instance the itemset $\{get\ well\ card,\ flowers\}$, has support of $3 / 5 = 0.6$ in the hospital gift shop data.
- Similarly, the support for $\{get\ well\ card\} \rightarrow \{flowers\}$ is also 0.6.
- The support can be calculated for any itemset or even a single item;
- for instance, the support for $\{candy\ bar\}$ is $2 / 5 = 0.4$, since candy bars appear in 40 percent of purchases.

Transaction number	Purchased items
1	$\{flowers, get\ well\ card, soda\}$
2	$\{plush\ toy\ bear, flowers, balloons, candy\ bar\}$
3	$\{get\ well\ card, candy\ bar, flowers\}$
4	$\{plush\ toy\ bear, balloons, soda\}$
5	$\{flowers, get\ well\ card, soda\}$

Example-2/2

- Essentially, the confidence tells us the proportion of transactions where the presence of item or itemset X results in the presence of item or itemset Y .
- Keep in mind that the confidence that X leads to Y is not the same as the confidence that Y leads to X .
- For example, the confidence of $\{flowers\} \rightarrow \{get\ well\ card\}$ is $0.6 / 0.8 = 0.75$.
- In comparison, the confidence of $\{get\ well\ card\} \rightarrow \{flowers\}$ is $0.6 / 0.6 = 1.0$.
- This means that a purchase involving flowers is accompanied by a purchase of a get well card 75 percent of the time, while a purchase of a get well card is associated with flowers 100 percent of the time.
- This information could be quite useful to the gift shop management.

Transaction number	Purchased items
1	<i>{flowers, get well card, soda}</i>
2	<i>{plush toy bear, flowers, balloons, candy bar}</i>
3	<i>{get well card, candy bar, flowers}</i>
4	<i>{plush toy bear, balloons, soda}</i>
5	<i>{flowers, get well card, soda}</i>

Use of Apriori algorithm

- From the example above, rules like $\{get\ well\ card\} \rightarrow \{flowers\}$ are known as **strong rules**, because they have both high support and confidence.
- One way to find more strong rules would be to examine every possible combination of the items in the gift shop, measure the support and confidence value, and report back only those rules that meet certain levels of interest.
- However, as noted before, this strategy is generally not feasible for anything but the smallest of datasets.
- Here comes the use of the Apriori algorithm

Apriori Algorithm

- The Apriori principle states that all subsets of a frequent itemset must also be frequent.
- In other words, if $\{A, B\}$ is frequent, then $\{A\}$ and $\{B\}$ must both be frequent.
- Recall also that by definition, the support indicates how frequently an itemset appears in the data. Therefore, if we know that $\{A\}$ does not meet a desired support threshold, there is no reason to consider $\{A, B\}$ or any itemset containing $\{A\}$; it cannot possibly be frequent.
- The Apriori algorithm uses this logic to exclude potential association rules prior to actually evaluating them.
- The actual process of creating rules occurs in two phases:
 1. Identifying all the itemsets that meet a minimum support threshold.
 2. Creating rules from these itemsets using those meeting a minimum confidence threshold.

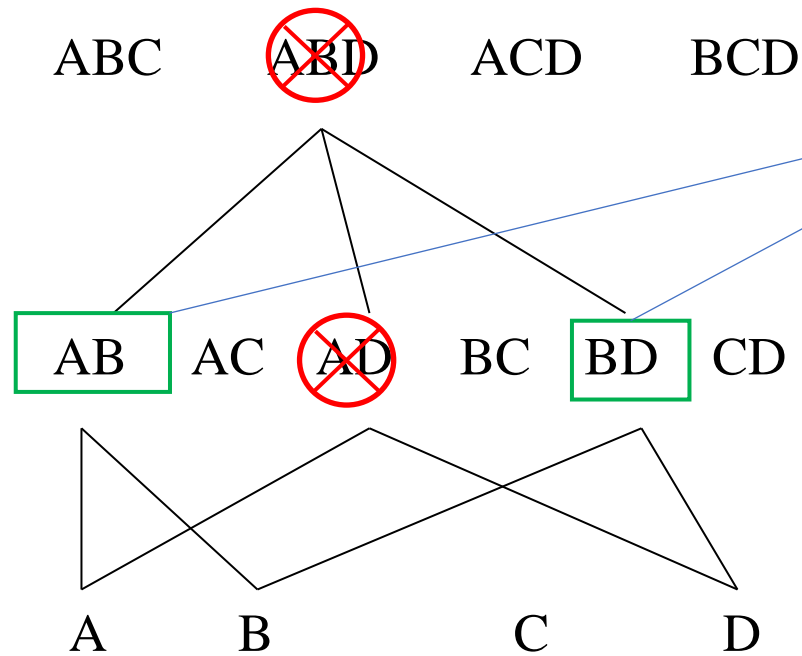
Use of Apriori algorithm-1/3

- The first phase occurs in multiple iterations. Each successive iteration involves evaluating the support of a set of increasingly large itemsets.
- For instance, iteration 1 involves evaluating the set of 1-item itemsets (1-itemsets), iteration 2 evaluates 2-itemsets, and so on.
- The result of each iteration i is a set of all the i -itemsets that meet the minimum support threshold.
- All the itemsets from iteration i are combined in order to generate candidate itemsets for the evaluation in iteration $i + 1$. But the Apriori principle can eliminate some of them even before the next round begins. If $\{A\}$, $\{B\}$, and $\{D\}$ are frequent in iteration 1 while $\{C\}$ is not frequent, iteration 2 will consider only $\{A, B\}$, $\{A, D\}$, and $\{B, D\}$.
- Thus, the algorithm needs to evaluate only three itemsets rather than the six that would have been evaluated if the sets containing C had not been eliminated *a priori*.
- Continuing with this thought, suppose during iteration 2, it is discovered that $\{A, B\}$ and $\{B, D\}$ are frequent, but $\{A, D\}$ is not. Although iteration 3 would normally begin by evaluating the support for $\{A, B, D\}$, it is not mandatory that this step should occur at all.
- Why not? The Apriori principle states that $\{A, B, D\}$ cannot possibly be frequent, since the subset $\{A, D\}$ is not. Therefore, having generated no new itemsets in iteration 3, the algorithm may stop.

Use of Apriori algorithm-2/3

- At this point, the second phase of the Apriori algorithm may begin. Given the set of frequent itemsets, association rules are generated from all possible subsets.
- For instance, $\{A, B\}$ would result in candidate rules for $\{A\} \rightarrow \{B\}$ and $\{B\} \rightarrow \{A\}$.
- These are evaluated against a minimum confidence threshold, and any rule that does not meet the desired confidence level is eliminated.

Use of Apriori algorithm-3/3



{A,B} and {B,D} will be selected and it will calculate the confidence of each possible mapping:

A->B

B->A

B->D

D->B

Exercise

- Here is a list of Transactions from a sample department store. By using Apriori algorithm create the association rules for the following data with 33% support level and 66% confidence.
- Each row here represents one transaction. A to G refers to the names of the products in the store.

1	A	B	C	E	F	G
2	B	E	F	G		
3	A	C	E	F		
4	B	C	F	G		
5	A	C	E	F	G	
6	C	F	G			
7	A	D	F	G		
8	D	E	F			
9	A	B	D	E		
10	A	B	C	F	G	
11	B	D	E	G		
12	A	C	D	E	F	

Use of Neural Network

What is Artificial Neural Network (ANN)

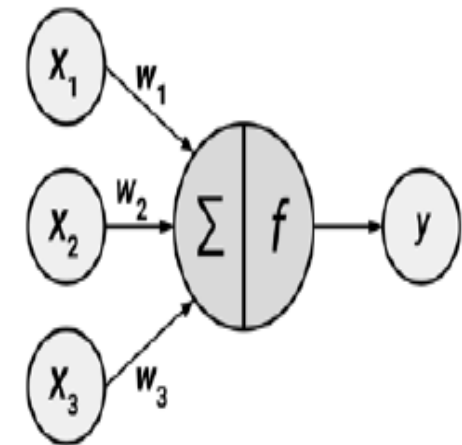
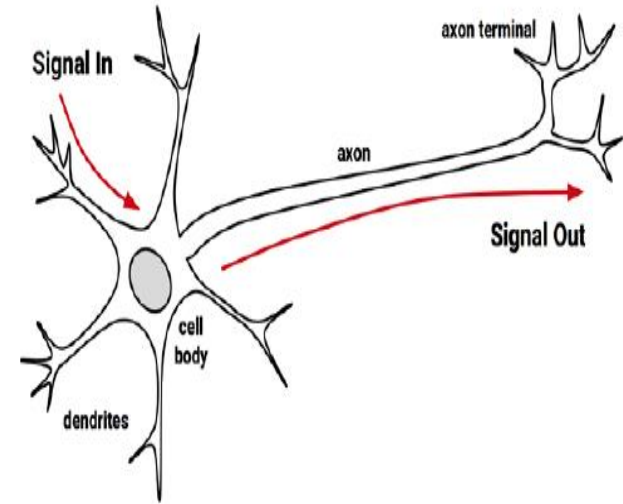
- An **Artificial Neural Network (ANN)** models the relationship between a set of input signals and an output signal using a model derived from our understanding of how a biological brain responds to stimuli from sensory inputs.
- Just as a brain uses a network of interconnected cells called **neurons** to create a massive parallel processor, ANN uses a network of artificial neurons or **nodes** to solve learning problems.
- In engineering, these are referred to as **black box** processes because the mechanism that transforms the input into the output is obfuscated by an imaginary box.

Common use cases of ANN

- Speech and handwriting recognition programs like those used by voicemail transcription services and postal mail sorting machines
- The automation of smart devices like an office building's environmental controls or self-driving cars and self-piloting drones
- Sophisticated models of weather and climate patterns, tensile strength, fluid dynamics, and many other scientific, social, or economic phenomena

The model of a single artificial neuron

- The model of a single artificial neuron can be understood in terms very similar to the biological model.
- As depicted in the following figure, a directed network diagram defines a relationship between the input signals received by the dendrites (x variables), and the output signal (y variable). Just as with the biological neuron, each dendrite's signal is weighted (w values) according to its importance.
- The input signals are summed by the cell body and the signal is passed on according to an **activation function** denoted by f :

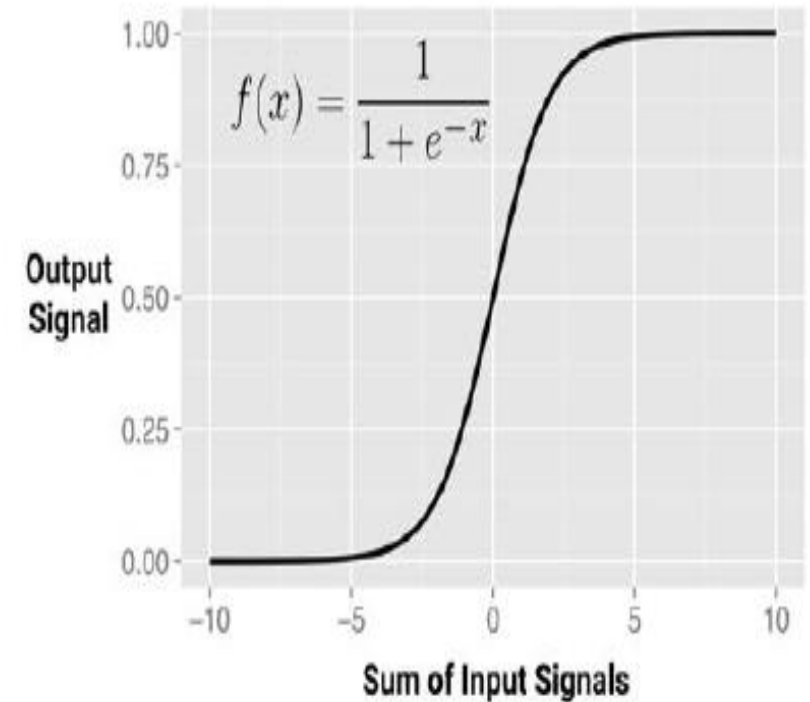


How to model an artificial neural network

- Neural networks use neurons defined this way as building blocks to construct complex models of data.
- Although there are numerous variants of neural networks, each can be defined in terms of the following characteristics:
 - An **activation function**, which transforms a neuron's combined input signals into a single output signal to be broadcasted further in the network
 - A **network topology** (or architecture), which describes the number of neurons in the model as well as the number of layers and manner in which they are connected
 - The **training algorithm** that specifies how connection weights are set in order to inhibit or excite neurons in proportion to the input signal

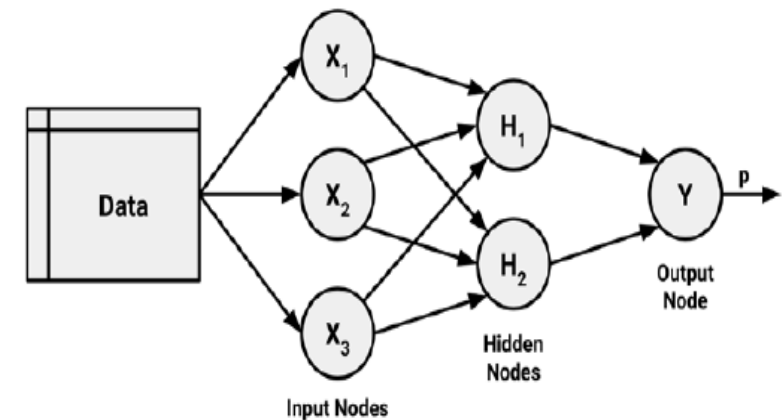
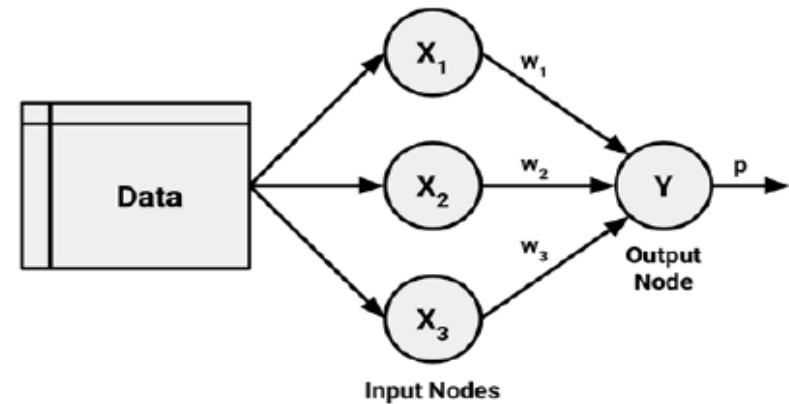
Activation function

- Perhaps the most commonly used alternative is the **sigmoid activation function** (more specifically, the *logistic* sigmoid) shown in the following figure.
- Note that in the formula shown, e is the base of the natural logarithm (approximately 2.72).
- Although it shares a similar step or "S" shape with the threshold activation function, the output signal is no longer binary; output values can fall anywhere in the range from 0 to 1.
- Additionally, the sigmoid is **differentiable**, which means that it is possible to calculate the derivative across the entire range of inputs.
- As you will learn later, this feature is crucial to create efficient ANN optimization algorithms.



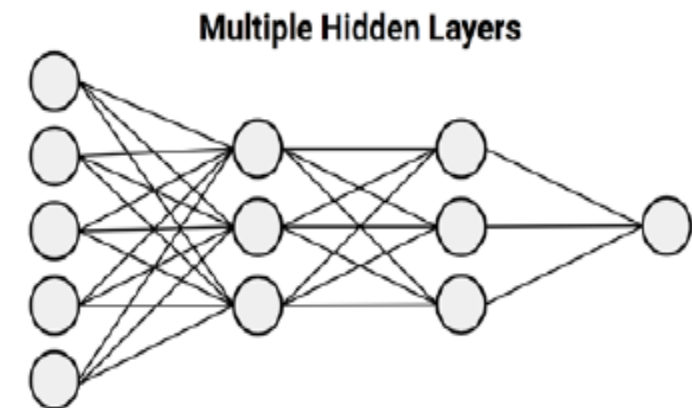
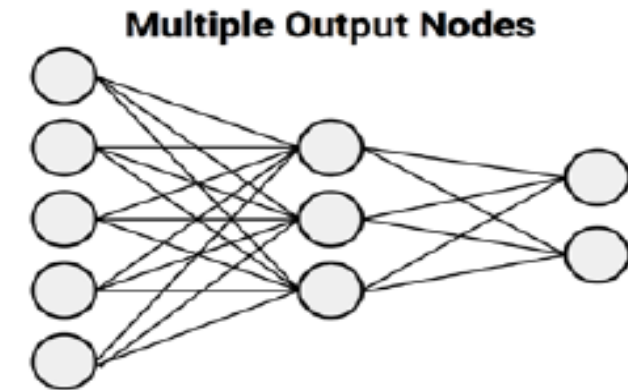
Network topology

- The ability of a neural network to learn is rooted in its **topology**, or the patterns and structures of interconnected neurons.
- Although there are countless forms of network architecture, they can be differentiated by three key characteristics:
 - The number of layers
 - Whether information in the network is allowed to travel backward
 - The number of nodes within each layer of the network



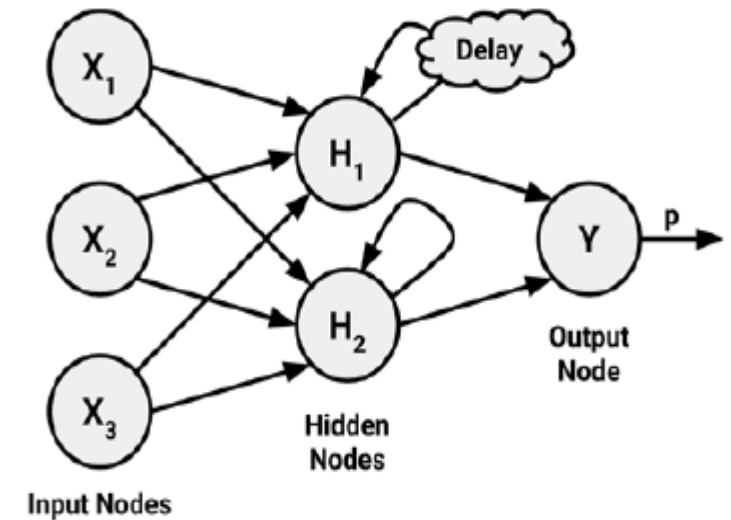
Feedforward network

- You may have noticed that in the prior examples, arrowheads were used to indicate signals traveling in only one direction.
- Networks in which the input signal is fed continuously in one direction from connection to connection until it reaches the output layer are called **feedforward** networks.
- In spite of the restriction on information flow, feedforward networks offer a surprising amount of flexibility.
- For instance, the number of levels and nodes at each level can be varied, multiple outcomes can be modeled simultaneously, or multiple hidden layers can be applied.
- A neural network with multiple hidden layers is called a **Deep Neural Network (DNN)** and the practice of training such network is sometimes referred to as **deep learning**.



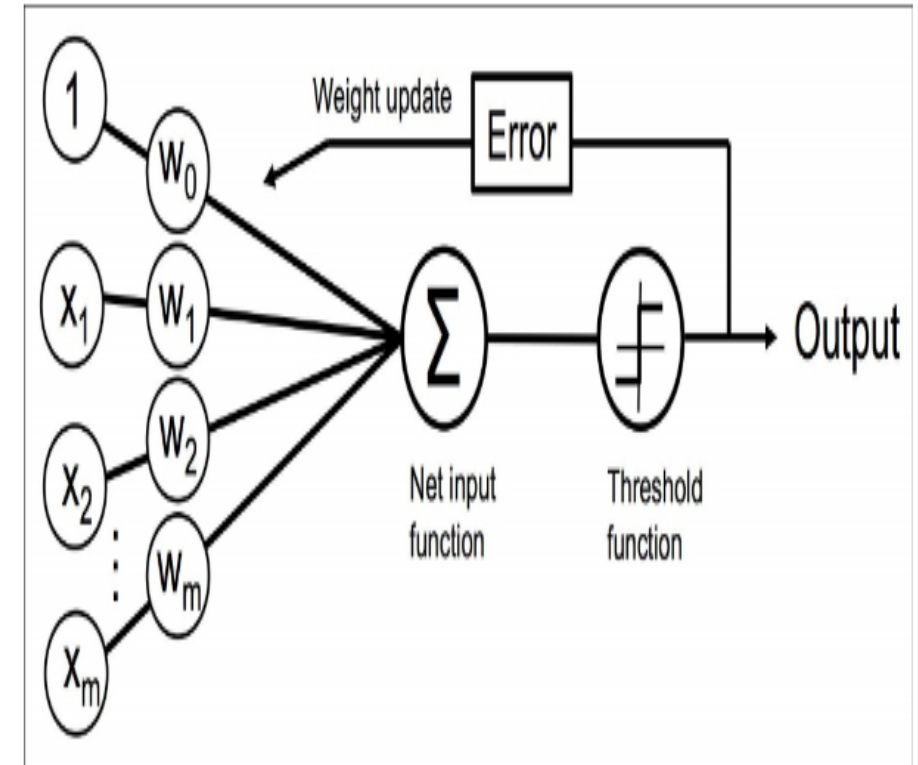
Recurrent network vs Multilayer Perceptron(MLP)

- In contrast, a **recurrent network** (or **feedback network**) allows signals to travel in both directions using loops. This property, which more closely mirrors how a biological neural network works, allows extremely complex patterns to be learned.
- The addition of a short-term memory, or **delay**, increases the power of recurrent networks immensely.
- Notably, this includes the capability to understand the sequences of events over a period of time. This could be used for stock market prediction, speech comprehension, or weather forecasting.
- In spite of their potential, recurrent networks are still largely theoretical and are rarely used in practice.
- On the other hand, feedforward networks have been extensively applied to real-world problems.
- In fact, the multilayer feedforward network, sometimes called the **Multilayer Perceptron (MLP)**, is the de facto standard ANN topology. If someone mentions that they are fitting a neural network, they are most likely referring to a MLP



Training neural networks with backpropagation

- Training a neural network by adjusting connection weights is very computationally intensive.
- Consequently, though they had been studied for decades prior, ANNs were rarely applied to real-world learning tasks until the mid-to-late 1980s, when an efficient method of training an ANN was discovered.
- The algorithm, which used a strategy of back-propagating errors, is now known simply as **backpropagation**.



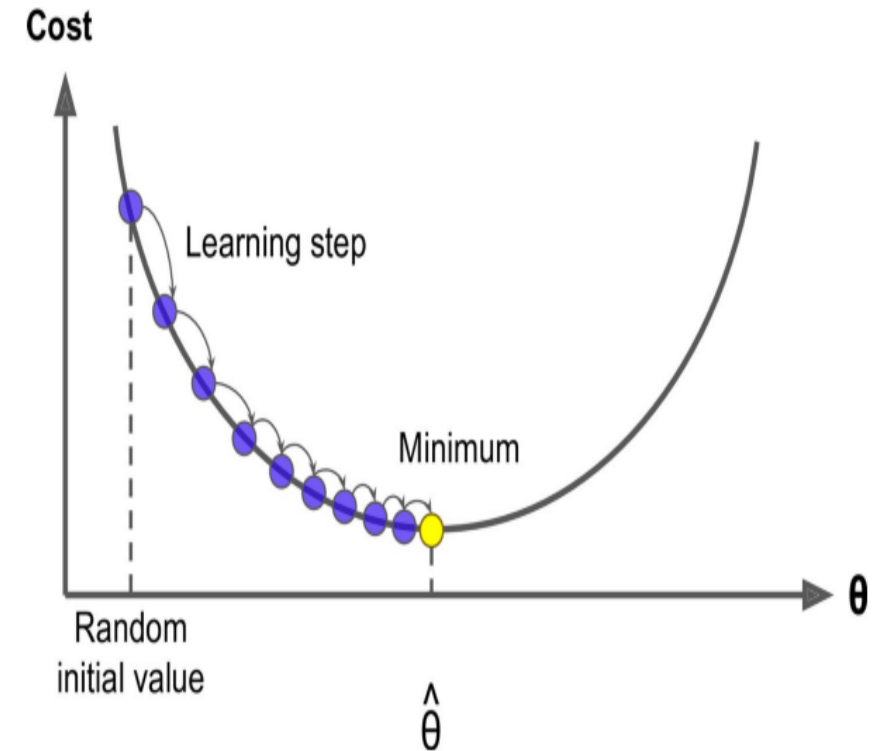
The Perceptron learning rule

$$w_{i,j}^{(\text{next step})} = w_{i,j} + \eta(y_j - \hat{y}_j)x_i$$

- $w_{i,j}$ is the connection weight between the i^{th} input neuron and the j^{th} output neuron.
- x_i is the i^{th} input value of the current training instance.
- \hat{y}_j is the output of the j^{th} output neuron for the current training instance.
- y_j is the target output of the j^{th} output neuron for the current training instance.
- η is the learning rate.

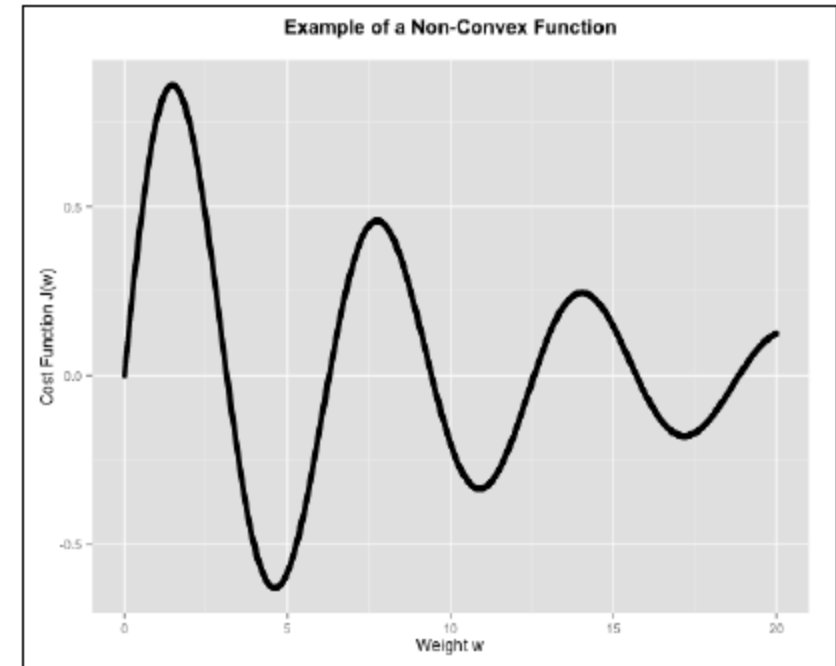
Gradient Descent

- In its most general form, the backpropagation algorithm iterates through many cycles of two processes. Each cycle is known as an **epoch**.
- Because the network contains no *a priori* (existing) knowledge, the starting weights are typically set at random. Then, the algorithm iterates through the processes, until a stopping criterion is reached. Each epoch in the backpropagation algorithm includes:
 - A **forward phase** in which the neurons are activated in sequence from the input layer to the output layer, applying each neuron's weights and activation function along the way. Upon reaching the final layer, an output signal is produced.
 - A **backward phase** in which the network's output signal resulting from the forward phase is compared to the true target value in the training data. The difference between the network's output signal and the true value results in an error that is propagated backwards in the network to modify the connection weights between neurons and reduce future errors.



Gradient descent and local minima-1/2

- Gradient descent methods rely on the idea that the cost function that is being minimized is a **convex function**.
- We'll skip the mathematical details of this and just say that a convex function is a function that has, at most, a single global minimum.
- Let's look at an example of a non-convex cost function in terms of a single weight w :



Gradient descent and local minima-2/2

- The global minimum of this function is the first trough on the left for a value of w , close to 4.5. **If our initial guess for the weight w is 1, the gradient of the cost function points towards the global minimum, and we will progressively approach it until we reach it.**
- **If our initial guess of the weight is 12, then the gradient of the cost function will point downwards towards the trough near the value 10.5. Once we reach the second trough, the gradient of the cost function will be 0 and consequently, we will not be able to make any progress towards our global minimum because we have landed in a local minimum.**
- Detecting and avoiding local minima can be very tricky, especially if there are many of them. **One way to do this is to repeat the optimization with different starting points and then pick the weights that produce the lowest value of the cost function across the different times the optimization is run.** This procedure works well if the number of local minima is small and they are not too close together.

Use of ANN with Scikit-learn

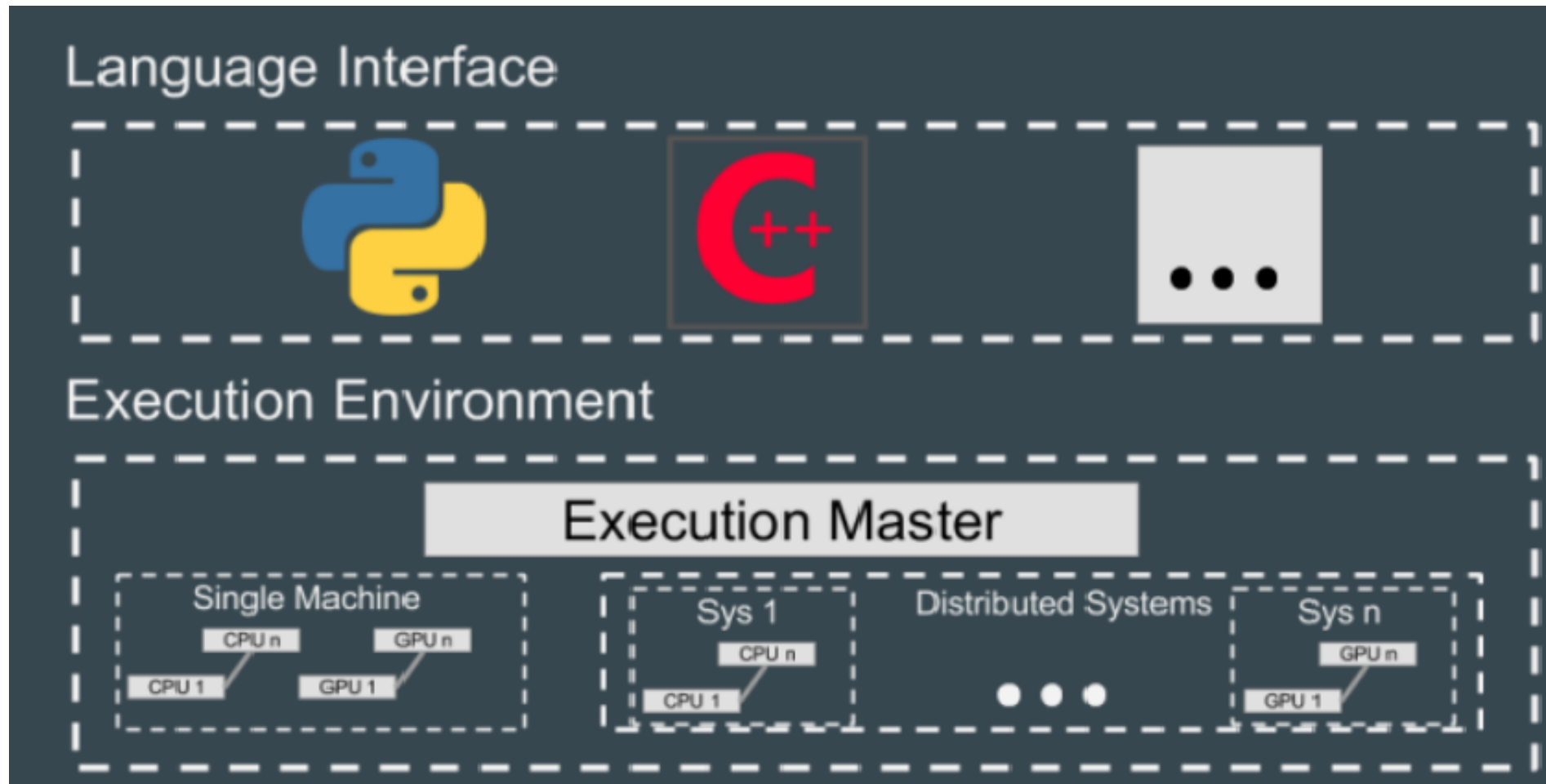
```
from sklearn.neural_network import MLPClassifier
```

- Refer to separate iPython notebook for details

What is Tensorflow

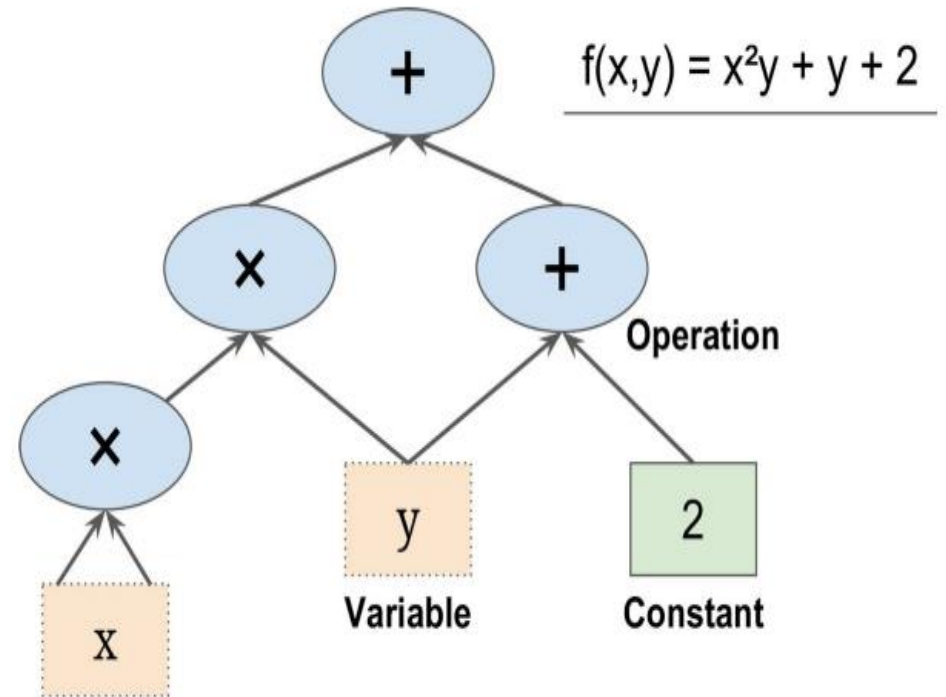
- An interface for expressing machine learning algorithms and an implementation for executing such algorithms; A framework for creating ensemble algorithms for today's most challenging problems. – *Google Brain Team*

Tensorflow environment

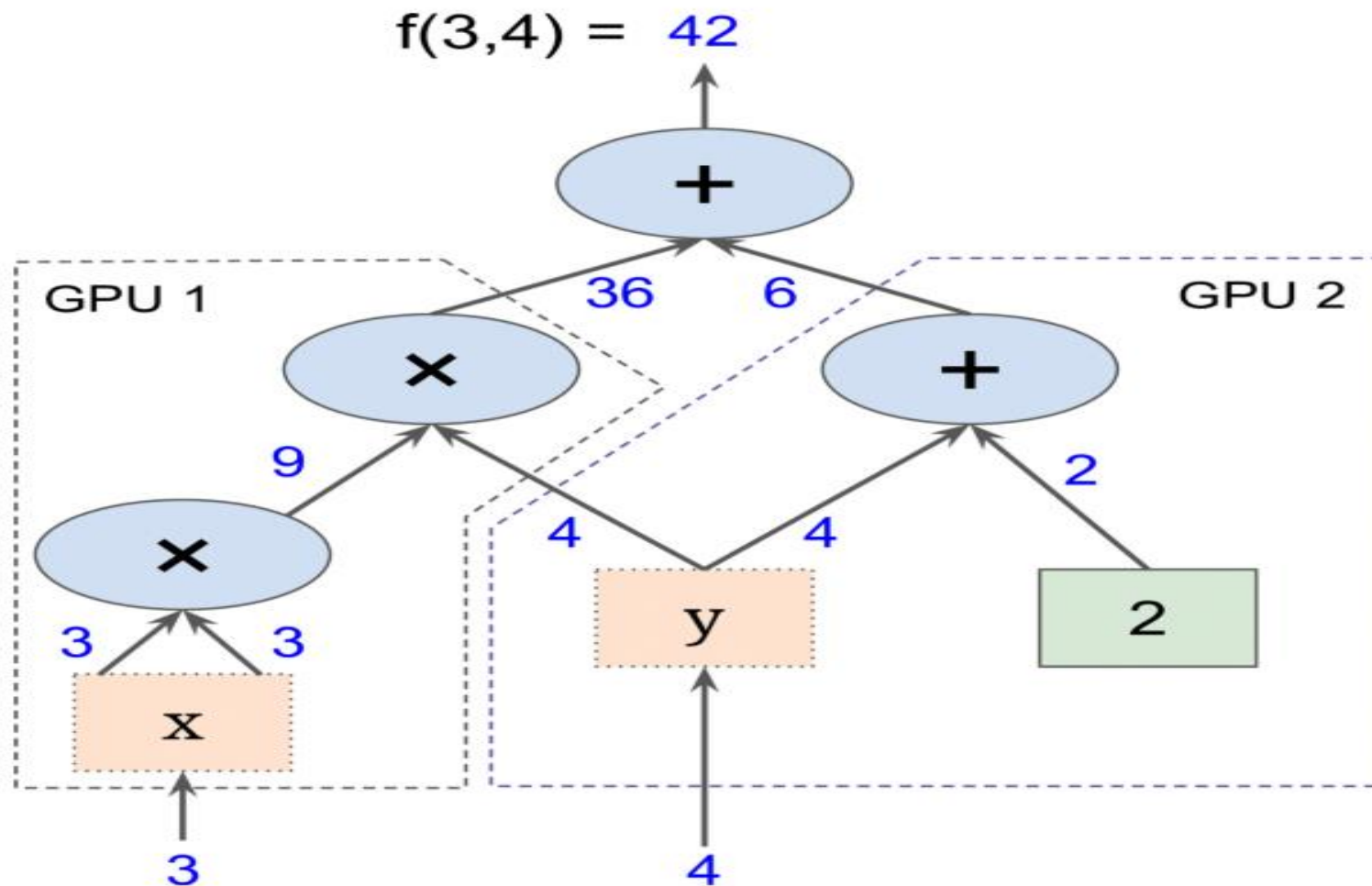


Introduction to tensorflow

- TensorFlow is a powerful open source software library for numerical computation, particularly well suited and fine-tuned for large-scale Machine Learning.
- Its basic principle is simple: you first define in Python a graph of computations to perform (for example, the one in Figure), and then TensorFlow takes that graph and runs it efficiently using optimized C++ code.



Parallel computation on multiple CPUs/ GPUs/ servers



Install Tensorflow in Python

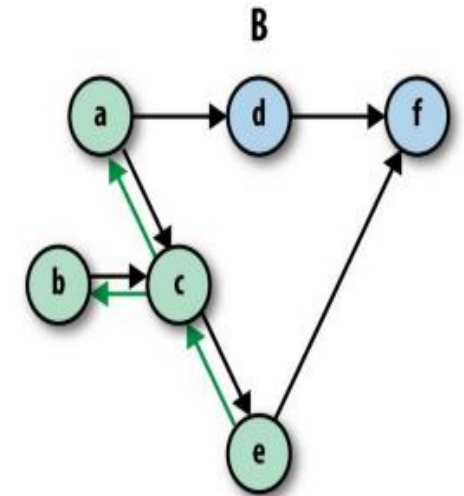
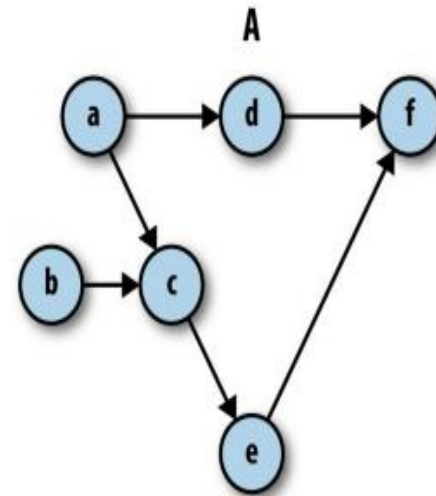
- `py -m pip install --upgrade tensorflow`

Tensorflow basic

- TensorFlow allows us to implement machine learning algorithms by creating and computing operations that interact with one another. These interactions form what we call a “**computation graph**,” with which we can intuitively represent complicated functional architectures.
- In TensorFlow, each of the graph’s nodes represents an operation, possibly applied to some input, and can generate an output that is passed on to other nodes.
- By analogy, we can think of the graph computation as an assembly line where each machine (node) either gets or creates its raw material (input), processes it, and then passes the output to other machines in an orderly fashion, producing subcomponents and eventually a final product when the assembly process comes to an end.

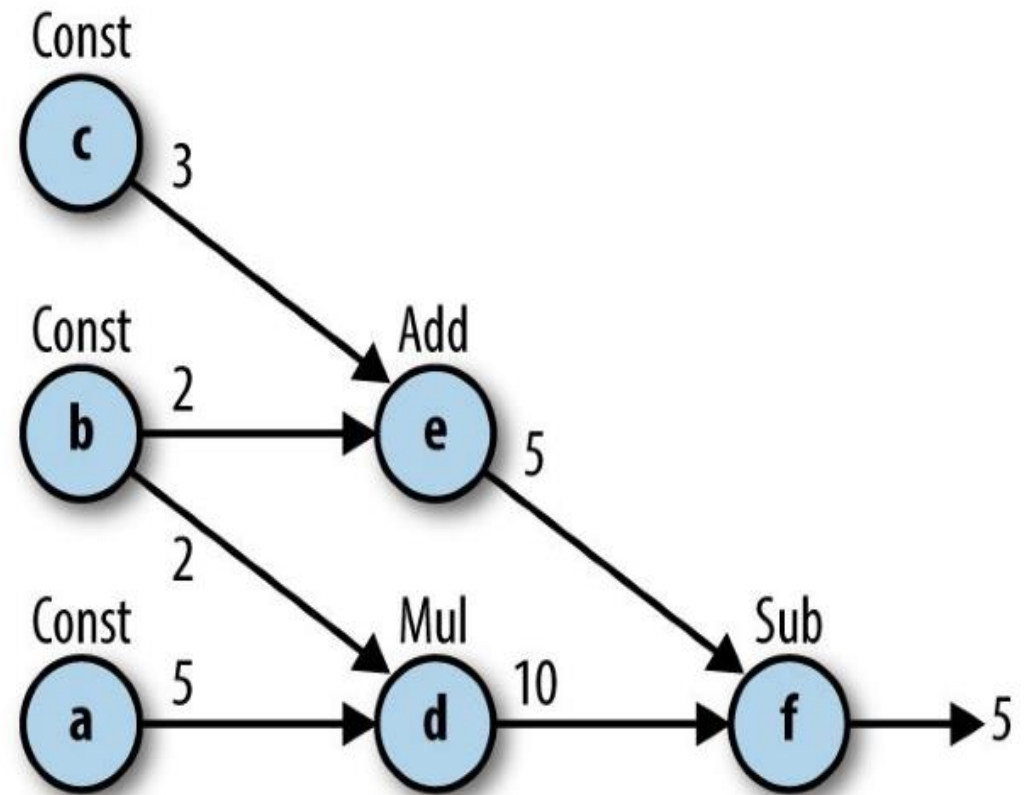
The benefits of graph computation

- TensorFlow optimizes its computations based on the graph's connectivity.
- Each graph has its own set of node dependencies. When the input of node y is affected by the output of node x , we say that node y is dependent on node x . We call it a direct dependency when the two are connected via an edge, and an indirect dependency otherwise.
- Being able to locate dependencies between units of our model **allows us to both distribute computations across available resources and avoid performing redundant computations of irrelevant subsets, resulting in a faster and more efficient way of computing things.**



Simple example

```
import tensorflow as tf
a = tf.constant(5)
b = tf.constant(2)
c = tf.constant(3)
d = tf.multiply(a,b)
e = tf.add(c,b)
f = tf.subtract(d,e)
sess = tf.Session()
outs = sess.run(f)
sess.close()
print("outs = {}".format(outs))
```



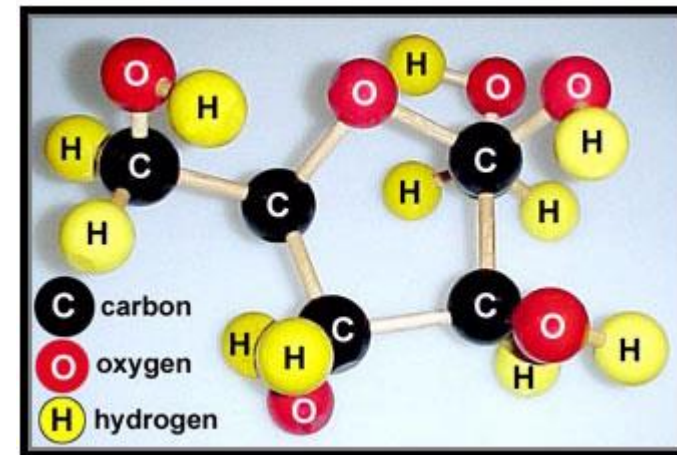
Another example

- Deploy linear regression with tensorflow
- Refer to the separate Ipython notebook for details

Use of Graph mining

What Graphs are good for?

- Most of existing data mining algorithms are based on **transaction representation**, i.e., sets of items.
- Datasets with structures, layers, hierarchy and/or geometry often do not fit well in this transaction setting. For e.g.
 - Numerical simulations
 - 3D protein structures
 - Chemical Compounds
 - Generic XML files.



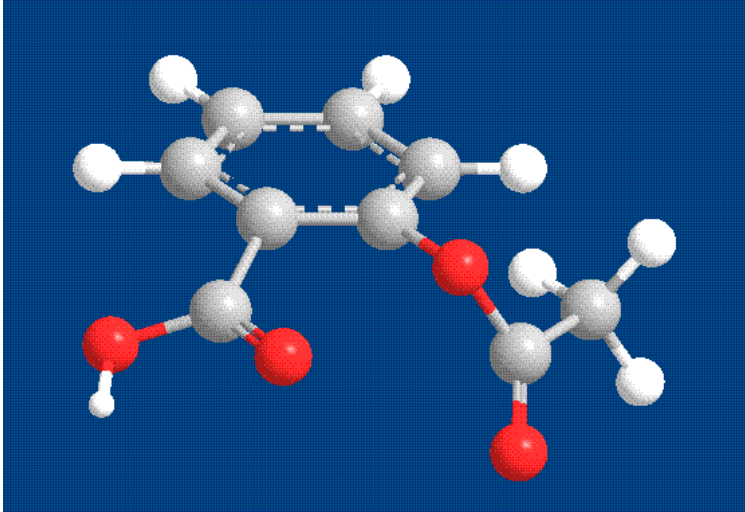
Graph Based Data Mining

- Graph Mining is the problem of discovering repetitive subgraphs occurring in the input graphs.
- Motivation:
 - finding subgraphs capable of compressing the data by abstracting instances of the substructures.
 - identifying conceptually interesting patterns

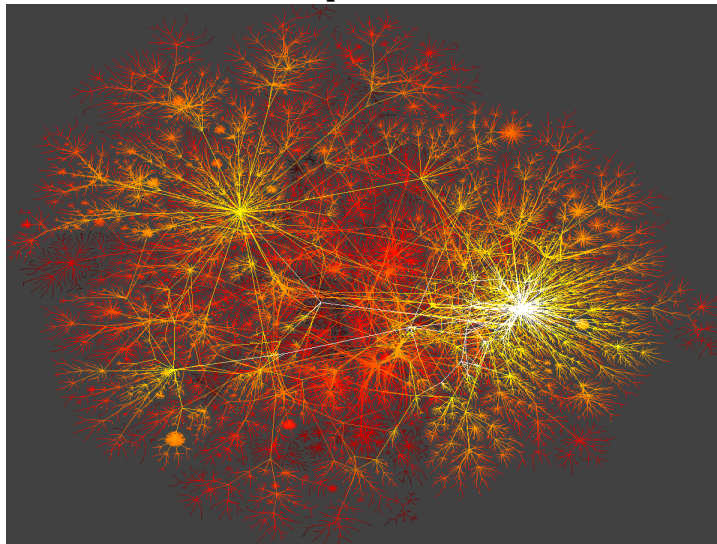
Why Graph Mining?

- Graphs are everywhere
 - Chemical compounds (Cheminformatics)
 - Protein structures, biological pathways/networks (Bioinformatics)
 - Program control flow, traffic flow, and workflow analysis
 - XML databases, Web, and social network analysis
- Graph is a general model
 - Trees, lattices, sequences, and items are degenerated graphs
- Diversity of graphs
 - Directed vs. undirected, labeled vs. unlabeled (edges & vertices), weighted, with angles & geometry (topological vs. 2-D/3-D)
- Complexity of algorithms: many problems are of high complexity (NP complete or even P-SPACE !)

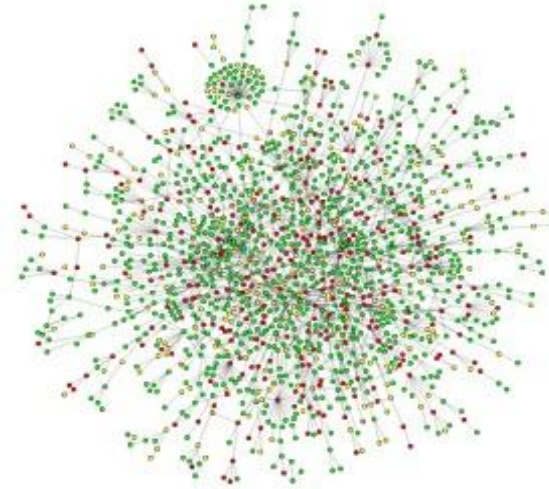
Graphs, Graphs, Everywhere



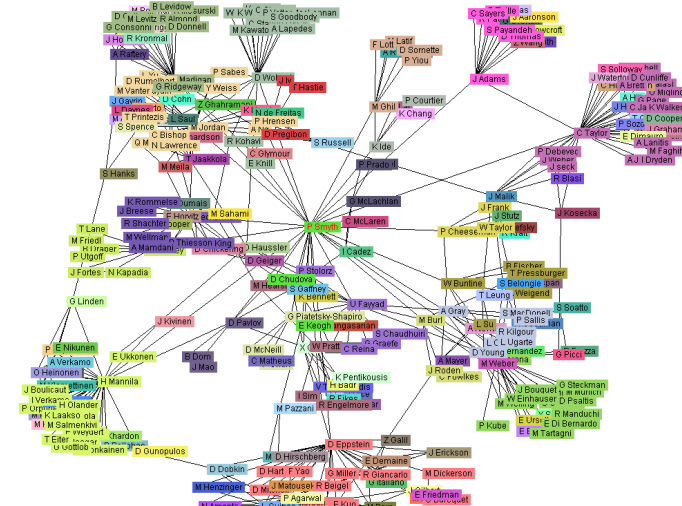
Aspirin



Internet



Yeast protein interaction network



Co-author network

from H. Jeong et al Nature 411, 41 (2001)

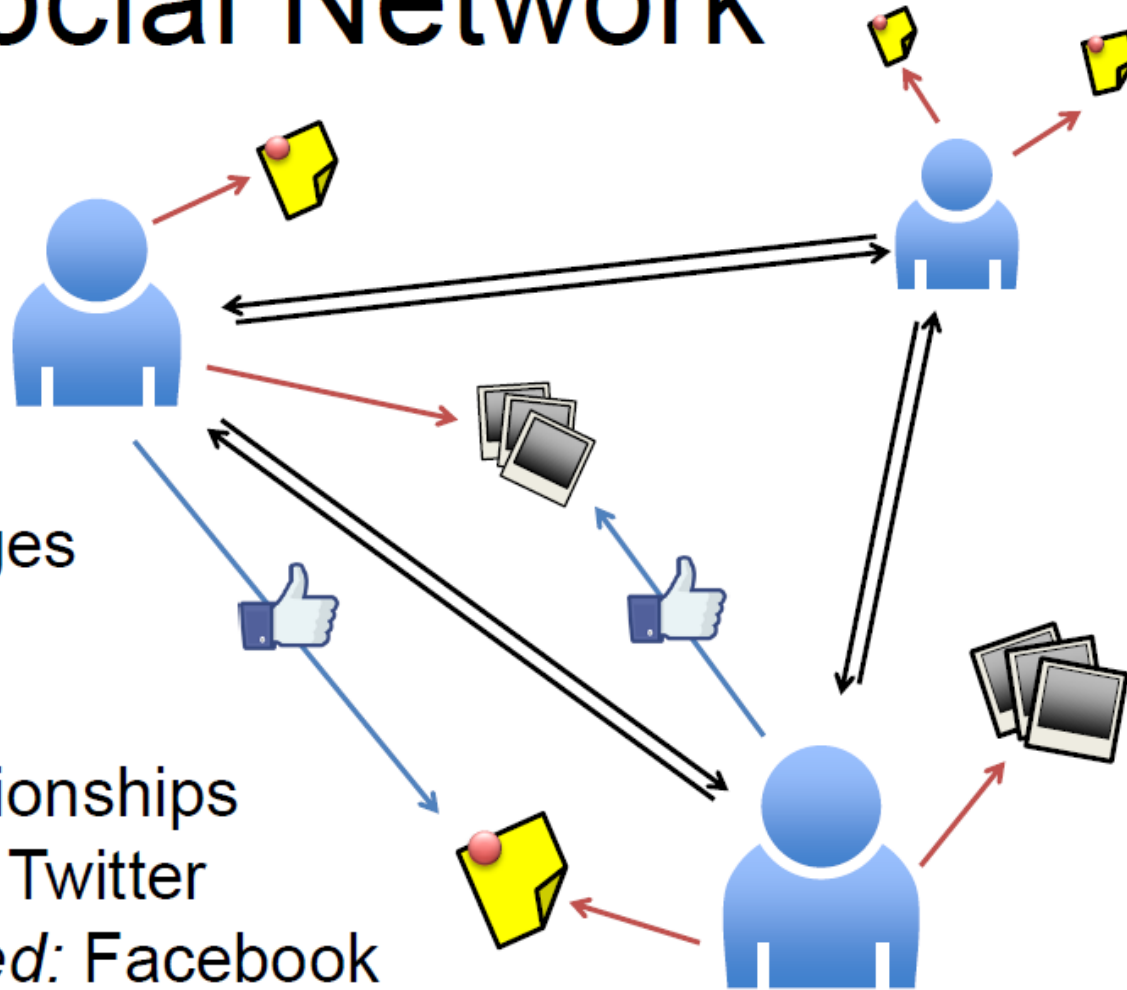
Social Network

Vertices

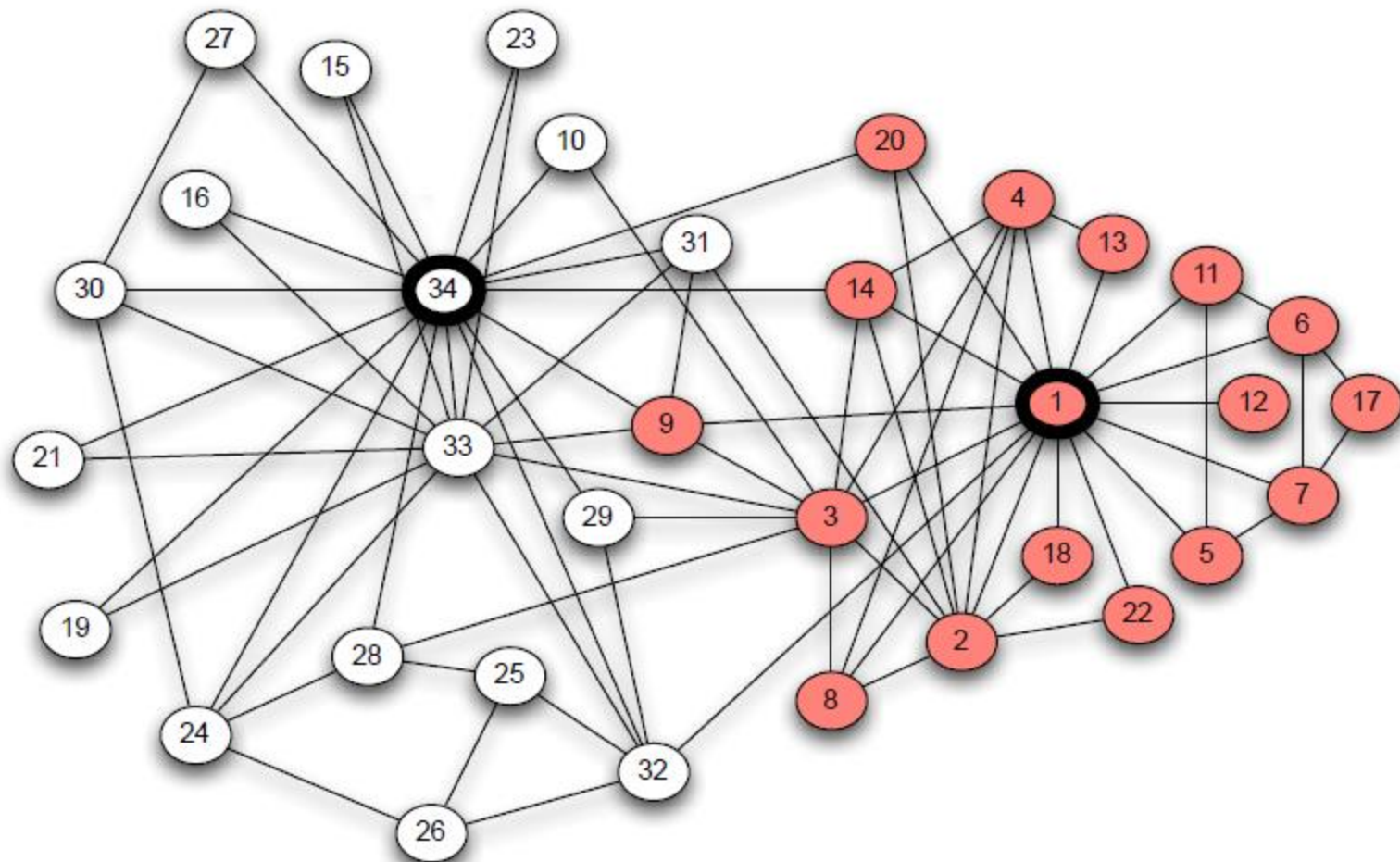
- Users
- Posts / Images

Edges

- Social Relationships
 - *Directed*: Twitter
 - *Undirected*: Facebook
- Likes



Actual Social Graph



Semantic Networks

Organize Knowledge

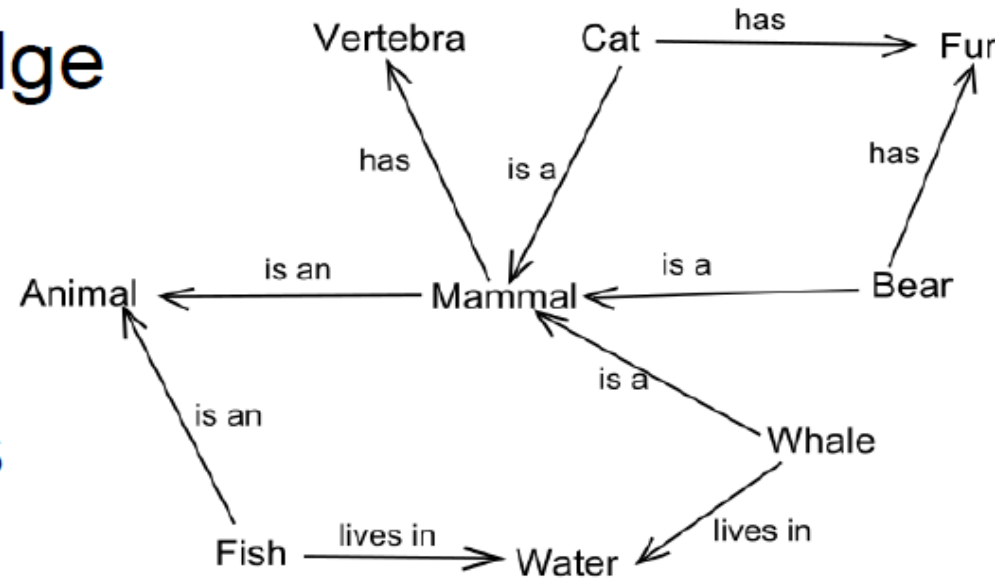
Vertices: Subject,
Object

Edges: Predicates

Example:

Google Knowledge Graph

- 570M Vertices
- 18B Edges



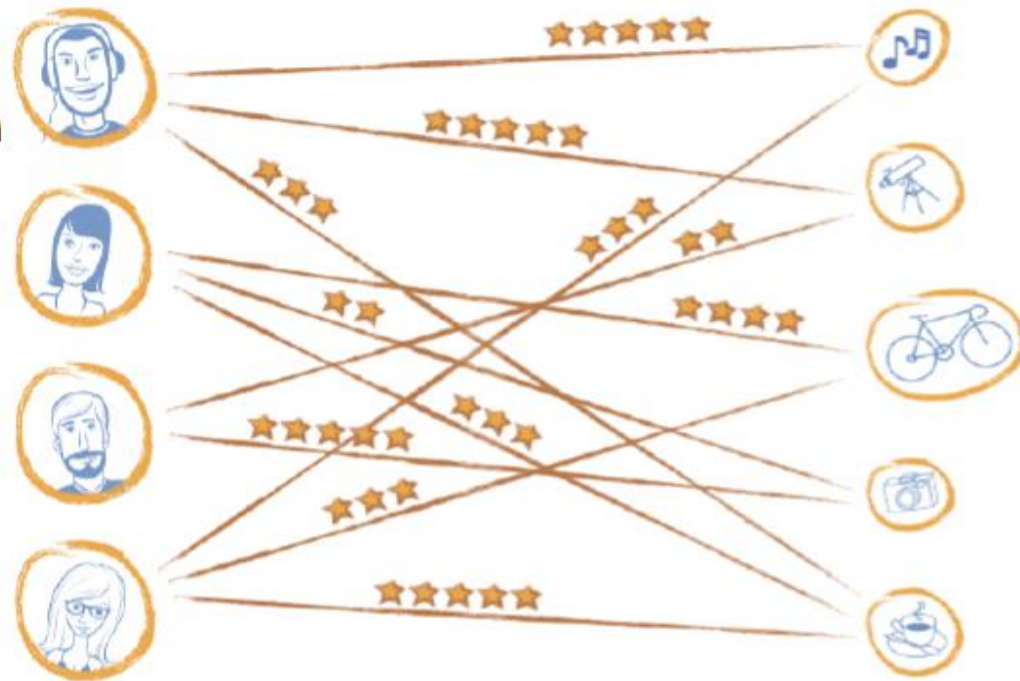
<http://wiki.dbpedia.org>

User - Item Graphs (Recommender Systems)

Bipartite Graphs

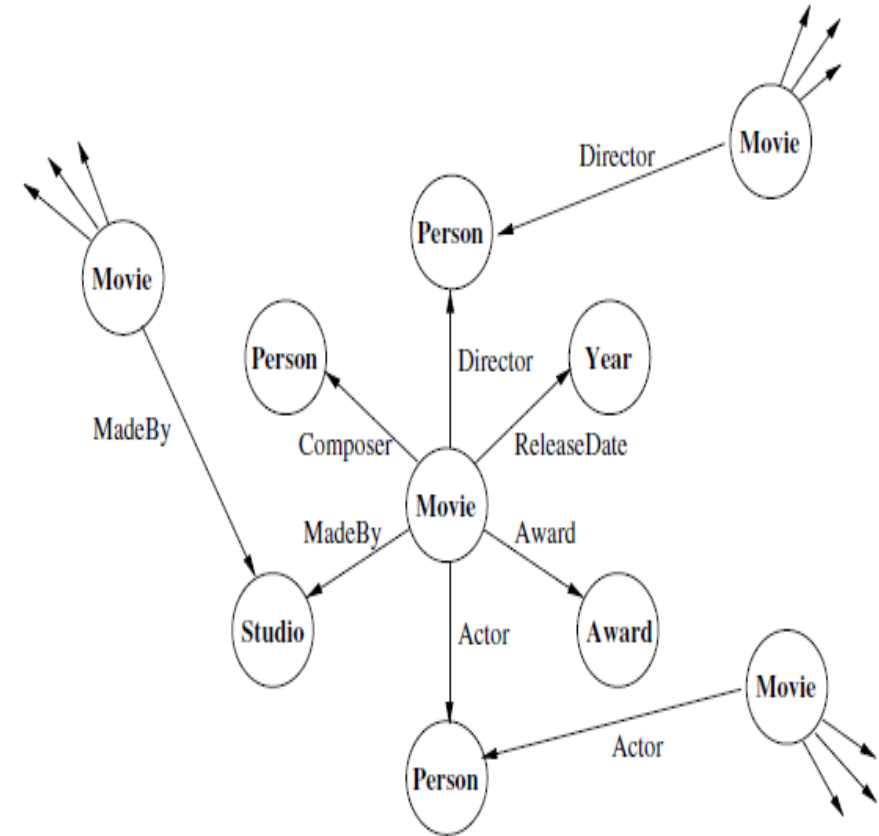
Vertices: *Users and Items*

Edges: *Ratings*



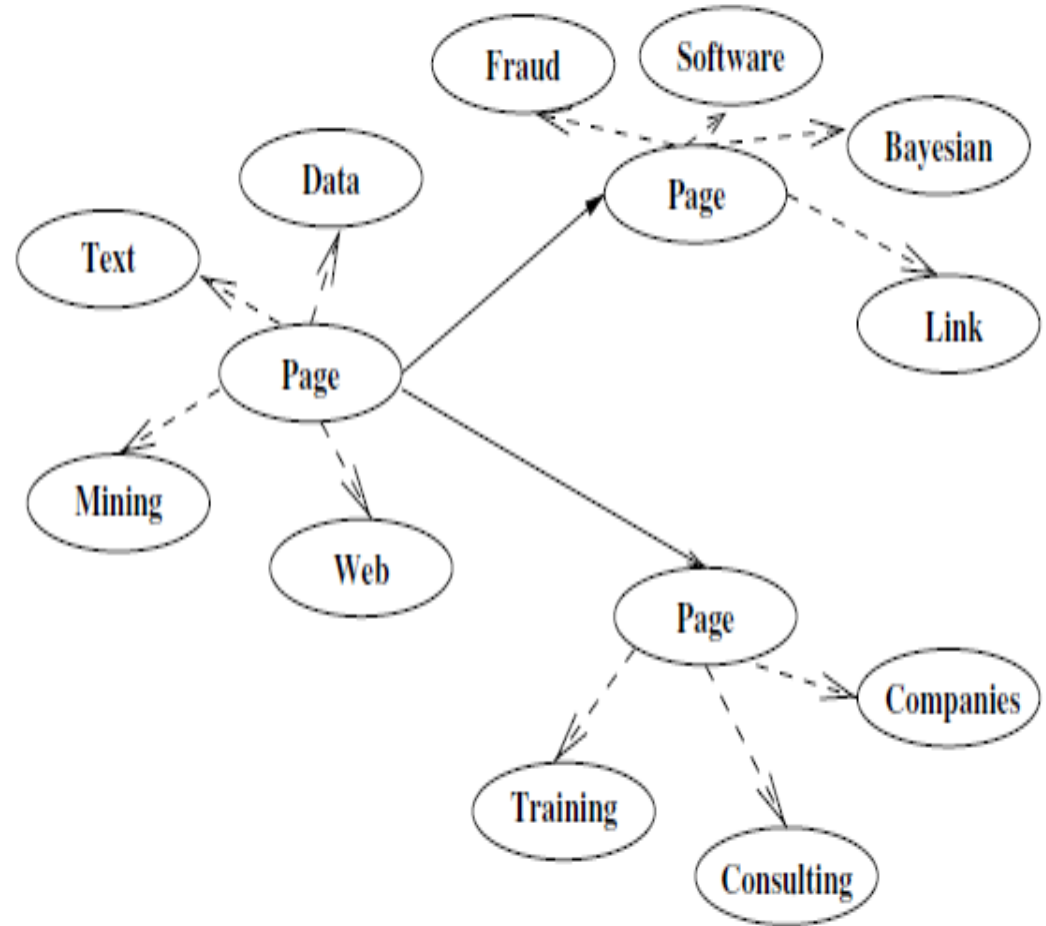
Example one

- To predict whether a movie will make more than \$2 million in its opening weekend



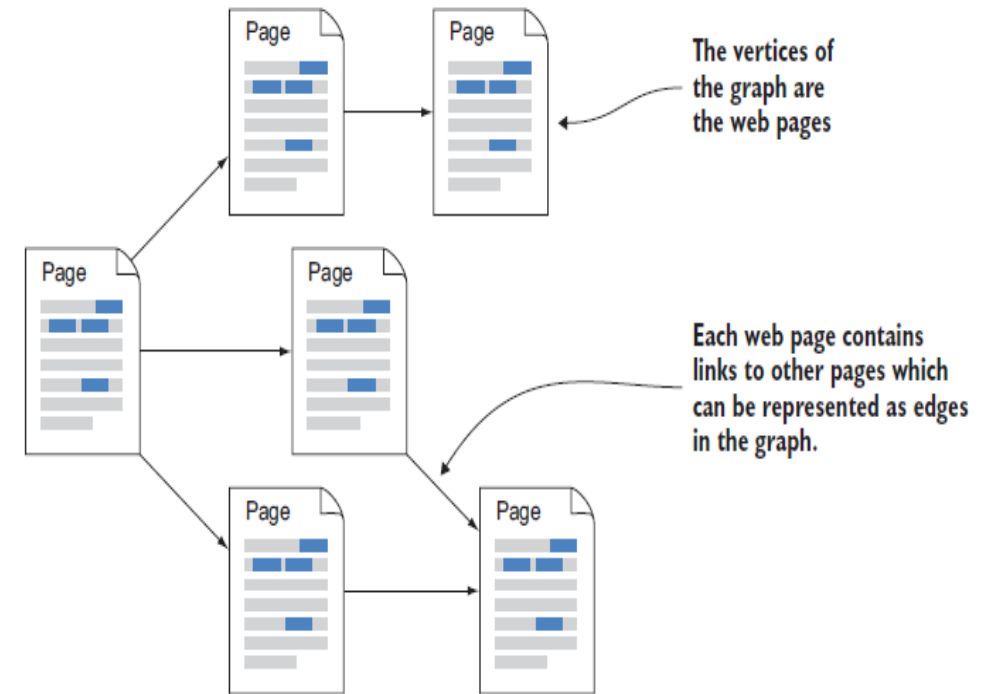
Example two

- **To show how new or emerging communities of Web pages can be identified from such a graph.**
- Analysis of this graph leads to identification of topic hubs and authorities .
- Authorities in this case are highly ranked pages on a given topic, and hubs represent overview sites with links to strong authority pages.
- **What commonalities can we find in Web navigation patterns?**
- Answering this question is the problem of Web usage mining.
- Although mining clickstream data on the client side has been investigated, **data is most easily collected and mined from Web servers.**



Example of the use of graph

- As the figure shows, you can represent these pages and links as a graph.
- You can then use the structure of the graph to provide information on the relative authority of each page.
- You can visualize this as each page providing a vote for each page it points to.
- But not all pages are equal; you might imagine that a page on a major news site has more importance than a posting by an unknown blogger.
- This is the problem that's solved by the **PageRank algorithm**.



Other use cases

- Finding the **shortest route** in a geo-mapping app
- **Recommending products, services, personal contacts, or media** based on **other people with similar-looking graphs**
- **Converting a tangle of interconnected topics into a hierarchy for organizational schemes** that require a hierarchy (computer file system folders, a class syllabus, and so forth)
- Determining the **most authoritative scholarly papers**