Lecture Notes on

Modern Data Science

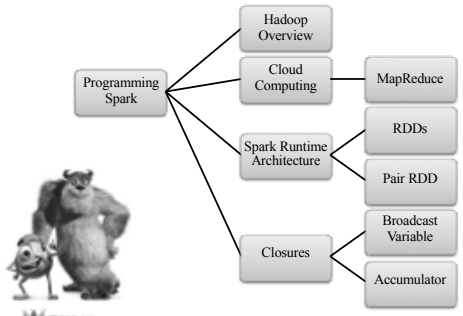
Module 06: Cloud Computing - Spark

Gang Li
School of Information Technology
Deakin University, VIC 3125, Australia

Unit Learning Outcomes

- **ULO1**:
  - Develop knowledge of and discuss new and ***emerging fields*** in data science.
    - Cloud Computing and Hadoop/Spark Platform
- **ULO4**:
  - Use appropriate ***platform*** to collect and process relatively large datasets.
    - Programming in Spark

Road map



Hadoop Essentials

- Hadoop EcoSystem
- HDFS
- MapReduce

Data Intensive Computing

- Data intensive computing
  - Concerns with the production, manipulation and analysis of data in the range of hundreds of megabytes (MB) to petabytes (PB) and beyond
  - A range of supporting parallel and distributed computing technologies to deal with the challenges of data representation, reliable shared storage, efficient algorithms and scalable infrastructure to perform analysis

Data Intensive Computing

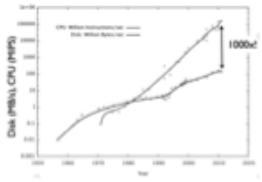
10 GB
100 GB
1 TB

Data Intensive Computing

10 TB
100 TB
XX PB

Data Intensive Computing

- Problems with Scale-Up
  - Disk streaming speed ~ 50MB/s
    - 3TB ~ 17.5 Hours
    - 1PB ~ 8 months
  - Seeking speeds even worse than r/w
    - ~10ms for a seek
    - Enough time to read 0.5MB

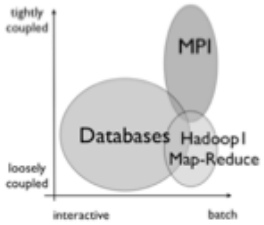


Meet the Hadoop

- What is Hadoop?
  - It's a framework for running applications on large clusters of commodity hardware which produces huge data and to process it
    - Based on 2004 Google MapReduce Paper

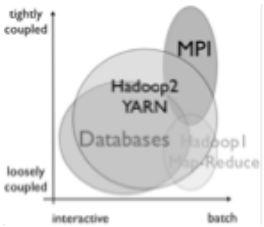
Meet the Hadoop

- HPC attitude
  - "The problem of disk-limited, loosely-coupled data analysis was solved by throwing more disks and using weak scaling"
- Flip-side:
  - A single novice developer can write real, scalable, 1000+ node data-processing tasks in Hadoop-family tools in an afternoon
  - MPI... less so



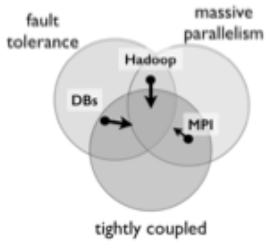
Meet the Hadoop

- Hadoop 2+
  - "An increasingly important tool for quantitative researchers to have at their disposal"
  - Spark, Pregel/Giraph for general, MPI-like communication patterns
  - Enormous data processing capabilities, batch + interactive

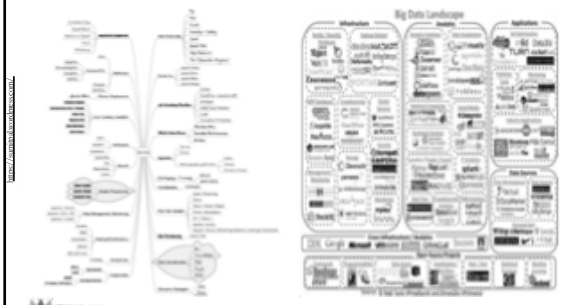


Everything is Converging

- Good ideas are good ideas
- These models are all converging at the largest scales
  - MPI is trying to grow fault tolerance
  - RDBM are scaling up
  - Hadoop is growing tightly coupled
    - Much faster than MPI codes are growing fault tolerance
    - Emerging tools tightly coupled computation atop Hadoop

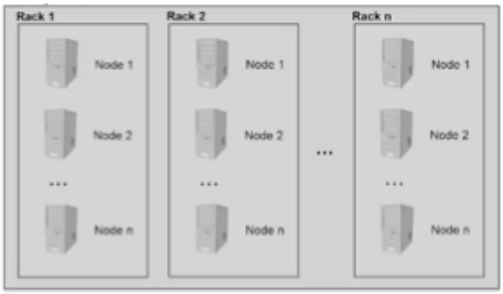


Hadoop Ecosystem



Hadoop Ecosystem

Hadoop Cluster

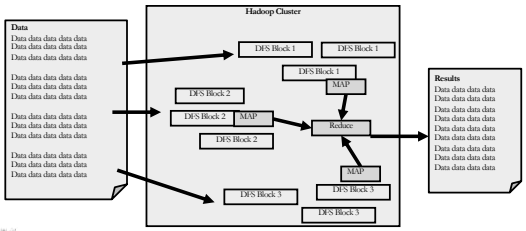


Hadoop Cluster

- Before Hadoop 2.2
  - Two main components
    - Storage: HDFS
    - Processing: MapReduce Engine
      - Framework for performing calculations on the data in the file system
      - Has a built-in resource manager and scheduler

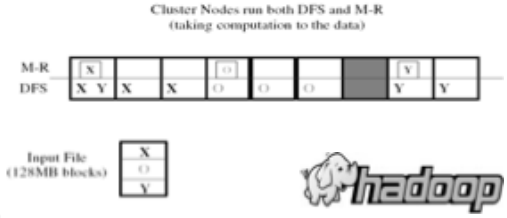
Hadoop Cluster

- Intuition
  - Moving computation is more efficient than moving data



Hadoop Cluster

- Intuition
  - Moving computation is more efficient than moving data



Hadoop Distributed File System (HDFS)

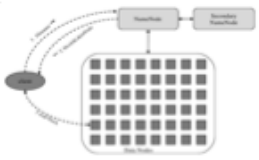
- A distributed file system designed to run on commodity hardware
  - Share similarities with existing distributed file systems and supports traditional hierarchical file organization
  - Reliable data replication and accessible via Web interface and Shell commands
  - *Benefits*: Fault tolerant, high throughput, streaming data access, robustness and handling of large data sets
  - HDFS is not a general purpose F/S

Hadoop Distributed File System (HDFS)

- What HDFS is not good for?
  - Low-latency reads
    - High-throughput rather than low latency for small chunks of data
  - Large amount of small files
    - Better for millions of large files instead of billions of small files
  - Multiple writers
    - Single write per file
    - Writes only at the end of file, no support for arbitrary offset

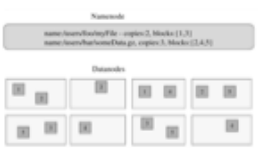
Hadoop Distributed File System (HDFS)

- DFS Master "**Namenode**"
  - Manages the filesystem namespace
  - Maintain file name to list blocks + location mapping
  - Manages block allocation/replication
  - Checkpoints namespace and journals namespace changes for reliability
  - Control access to namespace
- DFS Slaves "**Datanodes**" handle block storage
  - Stores blocks using the underlying OS's files
  - Clients access the blocks directly from datanodes
  - Periodically sends block reports to Namenode and check block integrity
- DFS "**Secondary Namenode**"
  - for checkpointing instead of backup



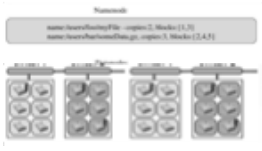
Hadoop Distributed File System (HDFS)

- Files are split into blocks
- Blocks
  - Single unit of storage: a contiguous piece of information on a disk
  - Transparent to user
  - Managed by Namenode, stored by Datanode
  - Blocks are either 64MB or 128MB



Hadoop Distributed File System (HDFS)

- Same block is replicated on multiple machines (3 by default)
  - Replicate placement are rack aware
  - 1st replica on the local rack
  - 2nd replica on the local rack but different machine
  - 3rd replica on the different rack
- Namenode determines replica placement



Hadoop Distributed File System (HDFS)

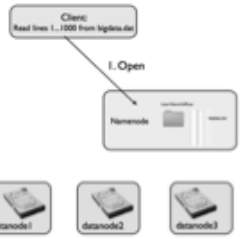
- DataNode
  - Each block replica on a DataNode is represented by two files in the local native filesystem.
    - The first file contains the data itself
    - The second file records the block's metadata including checksums for the data and the generation stamp.
  - At startup each DataNode connects to a NameNode and preforms a handshake.
    - The handshake verifies that the DataNode is part of the NameNode and runs the same version of software
  - A DataNode identifies block replicas in its possession to the NameNode by sending a block report.

Hadoop Distributed File System (HDFS)

- NameNode
  - The NameNode does not directly send requests to DataNodes. It uses replies to heartbeats to send instructions to the DataNodes
    - The instructions include commands to replicate blocks to other nodes, remove local block replicas, re-register and send an immediate block report, and shut down the node

Hadoop Distributed File System (HDFS)

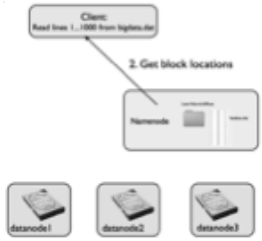
- Reading a HDFS file
  - Reading a file shorter
  - Get block locations
  - Read from replica



Jonathan Dursi https://github.com/ljdursi/hdfs-tutorial/processes.ipynb

Hadoop Distributed File System (HDFS)

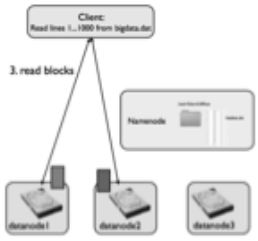
- Reading a HDFS file
  - Reading a file shorter
  - Get block locations
  - Read from replica



Jonathan Dursi https://github.com/ljdursi/hdfs-tutorial/processes.ipynb

Hadoop Distributed File System (HDFS)

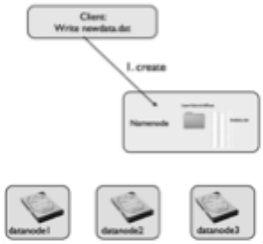
- Reading a HDFS file
  - Reading a file shorter
  - Get block locations
  - Read from replica



Jonathan Dursi https://github.com/ljdursi/hdfs-tutorial/processes.ipynb

Hadoop Distributed File System (HDFS)

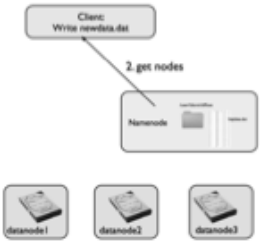
- Writing a HDFS file
  - Creating file
  - Get nodes for blocks
  - Start writing
  - Data nodes coordinate replication
  - Get Acknowledgement
  - Complete



Jonathan Dursi https://github.com/ljdursi/hdfs-tutorial/processes.ipynb

Hadoop Distributed File System (HDFS)

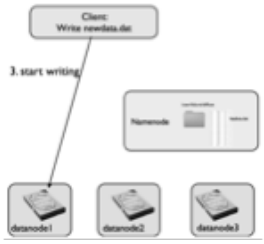
- Writing a HDFS file
  - Creating file
  - Get nodes for blocks
  - Start writing
  - Data nodes coordinate replication
  - Get Acknowledgement
  - Complete



Jonathan Dursi https://github.com/ljdursi/hdfs-tutorial/processes.ipynb

Hadoop Distributed File System (HDFS)

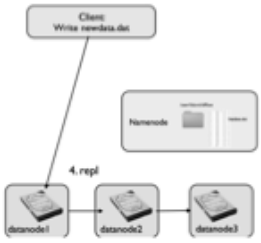
- Writing a HDFS file
  - Creating file
  - Get nodes for blocks
  - Start writing
  - Data nodes coordinate replication
  - Get Acknowledgement
  - Complete



Jonathan Dursi https://github.com/ljdursi/hdfs_hadoop_for_hpcers-tutorial

Hadoop Distributed File System (HDFS)

- Writing a HDFS file
  - Creating file
  - Get nodes for blocks
  - Start writing
  - Data nodes coordinate replication
  - Get Acknowledgement
  - Complete



Jonathan Dursi https://github.com/ljdursi/hdfs_hadoop_for_hpcers-tutorial

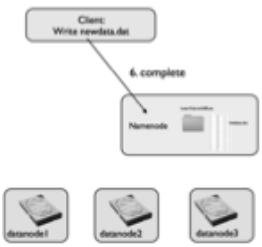
Hadoop Distributed File System (HDFS)

- Writing a HDFS file
  - Creating file
  - Get nodes for blocks
  - Start writing
  - Data nodes coordinate replication
  - Get Acknowledgement
  - Complete

Jonathan Dursi https://github.com/ljdursi/hdfs_hadoop_for_hpcers-tutorial

Hadoop Distributed File System (HDFS)

- Writing a HDFS file
  - Creating file
  - Get nodes for blocks
  - Start writing
  - Data nodes coordinate replication
  - Get Acknowledgement
  - Complete

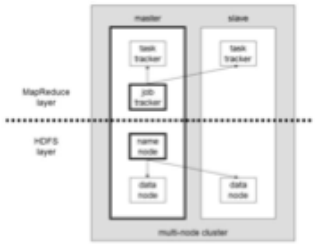


Jonathan Dursi https://github.com/ljdursi/hdfs_hadoop_for_hpcers-tutorial

Hadoop Server Roles

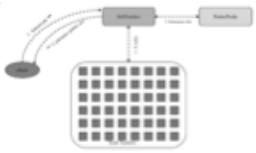
http://www.revelytix.com/?q=content/hadoop-ecosystem

Hadoop Server Roles



Job Tracker, Task Tracker and Job Client

- Map/Reduce Master "Jobtracker"
  - Accepts MR jobs submitted by users
  - Assigns Map and Reduce tasks to Tasktrackers
  - Monitors task and tasktracker status, re-executes tasks upon failure
- Map/Reduce Slaves "Tasktrackers"
  - Run Map and Reduce tasks upon instruction from the Jobtracker
  - Manage storage and transmission of intermediate output.



MapReduce Paradigm

- Programming model developed at Google
  - Sort/merge based distributed computing
  - The underlying system takes care of the partitioning of the input data, scheduling the program's execution across several machines, handling machine failures, and managing required inter-machine communication.
    - This is the key for Hadoop's success

MapReduce Paradigm

- Mapping workers to Processors
  - The input data (on HDFS) is stored on the local disks of the machines in the cluster.
  - The MapReduce master takes the location information of the input files into account and attempts to schedule a map task on a machine that contains a replica of the corresponding input data.
  - Failing that, it attempts to schedule a map task near a replica of that task's input data.
  - When running large MapReduce operations on a significant fraction of the workers in a cluster, most input data is read locally and consumes no network bandwidth.

MapReduce Paradigm

- Mapping workers to Processors
  - The input data (on HDFS) is stored on the local disks of the machines in the cluster.
  - The MapReduce master takes the location information of the input files into account and attempts to schedule a map task on a machine that contains a replica of the corresponding input data.
  - Failing that, it attempts to schedule a map task near a replica of that task's input data.
  - When running large MapReduce operations on a significant fraction of the workers in a cluster, most input data is read locally and consumes no network bandwidth.

MapReduce Paradigm

- Mapping workers to Processors
  - The input data (on HDFS) is stored on the local disks of the machines in the cluster.
  - The MapReduce master takes the location information of the input files into account and attempts to schedule a map task on a machine that contains a replica of the corresponding input data.
  - Failing that, it attempts to schedule a map task near a replica of that task's input data.
  - When running large MapReduce operations on a significant fraction of the workers in a cluster, most input data is read locally and consumes no network bandwidth.

MapReduce Paradigm

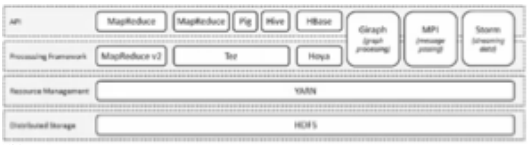
- Mapping workers to Processors
  - The input data (on HDFS) is stored on the local disks of the machines in the cluster.
  - The MapReduce master takes the location information of the input files into account and attempts to schedule a map task on a machine that contains a replica of the corresponding input data.
  - Failing that, it attempts to schedule a map task near a replica of that task's input data.
  - When running large MapReduce operations on a significant fraction of the workers in a cluster, most input data is read locally and consumes no network bandwidth.

MapReduce Paradigm

- Task Granularity
  - The map phase has M pieces and the reduce phase has R pieces.
  - M and R should be much larger than the number of **worker machines**.
  - Having each worker perform many different tasks improves dynamic load balancing, and also speeds up recovery when a worker fails.
  - Larger the M and R, more the decisions the master must make
  - R is often constrained by users because the output of each reduce task ends up in a separate output file.
    - Typically, (at Google), M = 200,000 and R = 5,000, using 2,000 worker machines.

Hadoop 2.2+

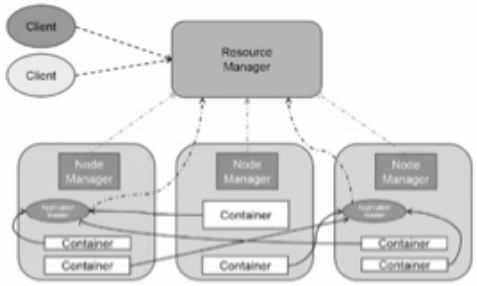
- YARN
  - MapReduce V2
  - DataNodes still exist
  - JobTrackers and TaskTrackers no longer exist



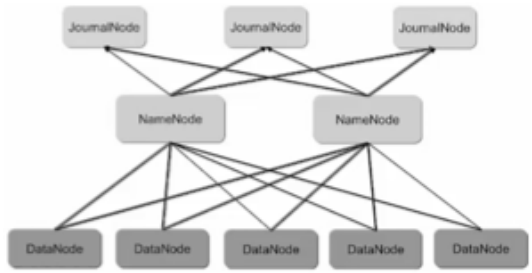
Hadoop 2.2+

- YARN
  - Provides generic scheduling and resource management
    - More than just batch processing
  - More efficient scheduling and workload management
    - No more balancing between map and reduce slots

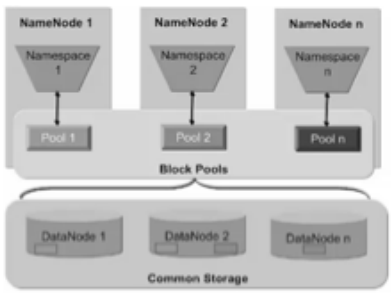
Hadoop High Availability



Hadoop High Availability



Hadoop High Availability



Map Reduce

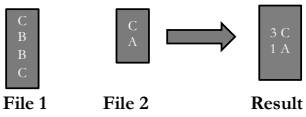
- Map + Reduce
- Apache Spark
- History Review

MapReduce

- Main Operations
  - Map
  - Reduce
- Input:
  - Key Value pairs
  - Write once and Read Many (WORM) data
- Output:
  - Key Value pairs
  - Written to disk

MapReduce

- An Example Task:
  - Counts lines in all *.txt files that match <regex> and displays the counts in descending order

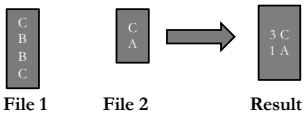


File 1 **File 2** **Result**

MapReduce

Map tasks
(0, C) → [(C, 1)]
(2, B) → [ ]
(4, B) → [ ]
(6, C) → [(C, 1)]
(0, C) → [(C, 1)]
(2, A) → [(A, 1)]

Reduce tasks
(A, [1]) → (A, 1)
(C, [1, 1, 1]) → (C, 3)



File 1 **File 2** **Result**

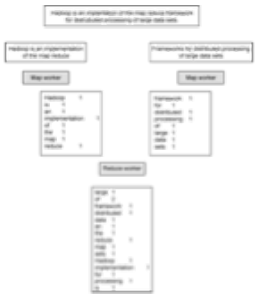
MapReduce

Map tasks
(0, C) → [(C, 1)]
(2, B) → [ ]
(4, B) → [ ]
(6, C) → [(C, 1)]
(0, C) → [(C, 1)]
(2, A) → [(A, 1)]

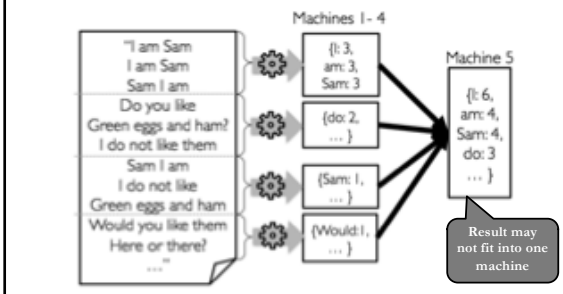
Reduce tasks
(A, [1]) → (A, 1)
(C, [1, 1, 1]) → (C, 3)

```
>>> grep -Eh 'A|C' *.txt | sort | uniq -c | sort -nr
```

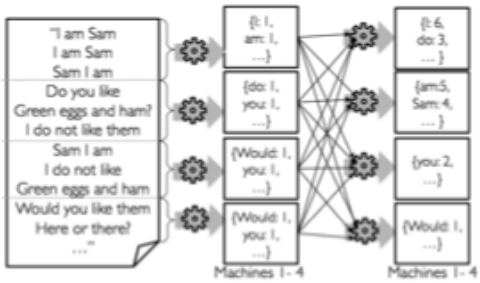

MapReduce



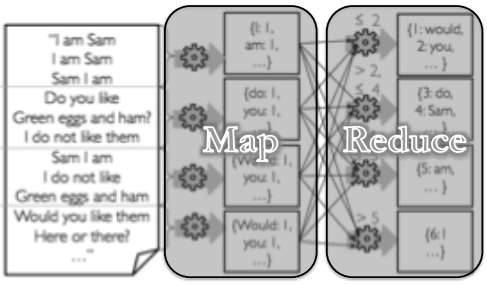
What if the Document is Really Big?



What if the Document is Really Big?



What if the Document is Really Big?



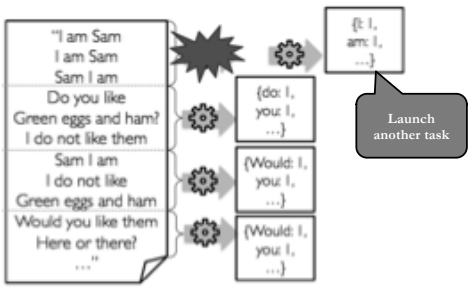
What if the Document is Really Big?

- Need to handle more data?
  - Just add more Mappers/Reducers!
- No need to handle multi-threaded code
  - Mappers and Reducers are typically single threaded and deterministic
    - Determinism allows for restarting of failed jobs
  - Mappers/Reducers run entirely independent of each other
    - In Hadoop, they run in separate JVMs

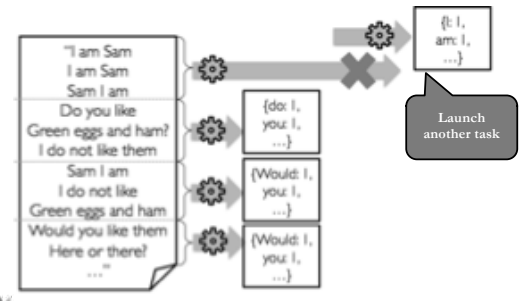
How to Deal with Failures?

- Failures are norm in commodity hardware
  - 1 server fails every 3 years → with 10k nodes 10 faults/day
- **Worker failure**
  - Detect failure via periodic heartbeats
  - Re-execute completed and in-progress *map* tasks
  - Task completion committed through master
- **Master failure**
  - Single point of failure; Resume from Execution Log
- Robust
  - Google's experience:
    - lost 1600 of 1800 machines once!, but finished fine

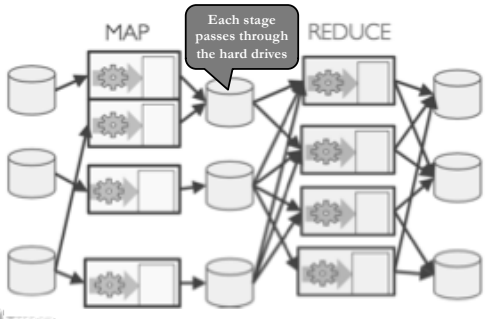
How to Deal with Failures?



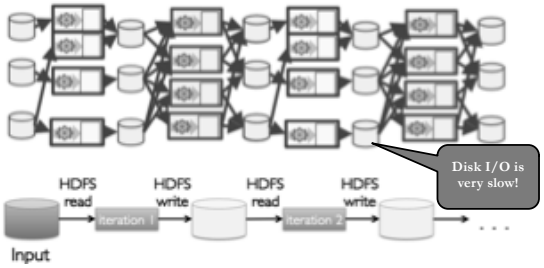
How to Deal with Slow Tasks?



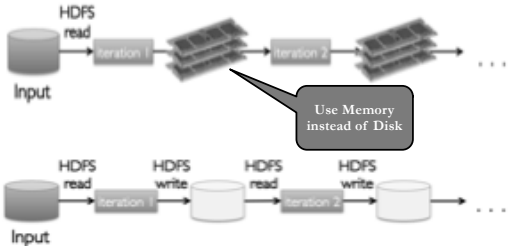
Map Reduce: Distributed Execution



Map Reduce: Iterative Jobs



Map Reduce: Iterative Jobs



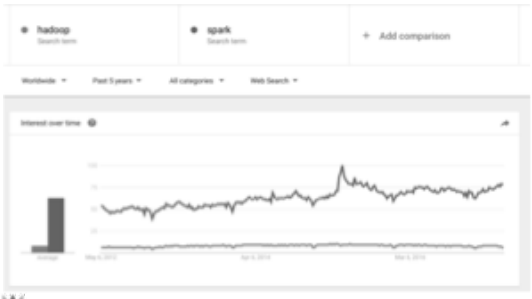
Apache Spark

- The **Spark** Computing Framework
  - Keep more data in-memory
  - A new distributed execution engine that provides programming abstraction and parallel runtime to hide complexities of fault-tolerance and slow machines

Apache Spark

- **Resilient Distributed Dataset** (RDD)
  - a read-only, partitioned collection of records spread across a cluster, stored in memory or on disk
  - RDDs can only be created through deterministic operations on either
    1. data in stable storage or
    2. other RDDs.

Spark vs Hadoop



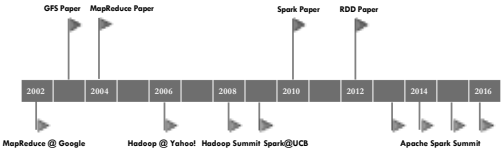
Spark vs Hadoop MapReduce

	Spark	Hadoop MapReduce
Storage	In-memory or on disk	Disk only
Operations	Map, Reduce, Join, Sample, etc.	Map and Reduce
Execution Model	Batch, Interactive, Streaming	Batch
Programming Environment	Scala, Java, R, and Python	Java

A Record in Daytona GraySort

	Hadoop MR Record	Spark Record	Spark 1 PB
Data Size	102.5 TB	100 TB	1000 TB
Elapsed Time	72 mins	23 mins	234 mins
# Nodes	2100	206	190
# Cores	50400 physical	6592 virtualized	6080 virtualized
Cluster disk throughput	3150 GB/s (est.)	618 GB/s	570 GB/s
Sort Benchmark Daytona Rules	Yes	Yes	No
Network	dedicated data center, 10Gbps	virtualized (EC2) 10Gbps network	virtualized (EC2) 10Gbps network
Sort rate	**1.42 TB/min**	**4.27 TB/min**	**4.27 TB/min**
Sort rate/node	**0.67 GB/min**	**20.7 GB/min**	**22.5 GB/min**

History Review

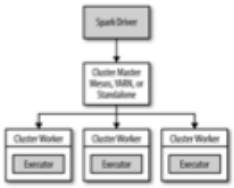


Spark Essentials

- Spark Runtime Architecture
- Spark Context
- Resilient Distributed Dataset

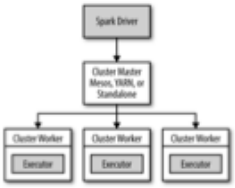
Spark Runtime Architecture

- In distributed mode, Spark uses a master/slave architecture
  - one central coordinator (**driver**) and
  - many distributed workers (**executors**)
- Spark Application contains
  - The driver runs in its own Java process
  - Each executor is a separate Java process



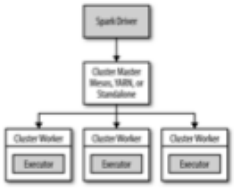
Spark Runtime Architecture

- **The Driver**
  - The process where method **`main()`** runs
  - Two duties
    - Converting a user program into tasks
      - Send works to executors as tasks
    - Scheduling tasks on executors
      - Coordinate the scheduling of individual tasks on executors
      - Complete view of application's executors



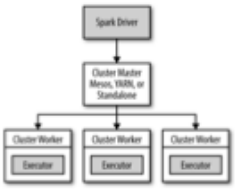
Spark Runtime Architecture

- **Executors**
  - The worker processes responsible for running individual tasks
  - Two duties
    - Running the tasks and return results to the driver
    - Providing in-memory storage for RDDs that are cached, through the Block Manager.



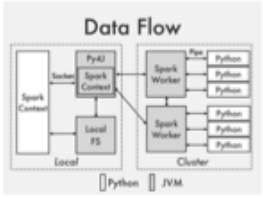
Spark Runtime Architecture

- **Cluster Manager**
  - Standalone cluster manager, or external managers such as YARN, Mesos, etc.
  - Launch executors on behave of the driver program
  - Launch the driver sometimes
  - Terms
    - **Master** and **Worker**
      - Centralized and distributed portions of the cluster manager
    - **Driver** and **Executor**
      - The processes that execute each Spark application



Python Spark (PySpark)

- PySpark is built on top of Spark's Java API
  - Data is processed in Python and cached/shuffled in JVM
  - In the Python driver, **SparkContext** uses Py4J to launch a JVM and create a **JavaSparkContext**.
    - Py4J is only used on the driver for local communication between the Python and Java SparkContext objects;
    - large data transfers are performed through a different mechanism.



Spark Context

- Driver programs access Spark through a **SparkContext** object
  - which tells Spark how and where to access a cluster
- Use **SparkContext** to create RDDs

Initializing a SparkContext

- The master parameter for a **SparkContext** determines which type and size of cluster to use

Value	Explanation
`spark://host:port`	Connect to a Spark Standalone cluster at the specified port. By default Spark Standalone masters use port 7077.
`mesos://host:port`	Connect to a Mesos cluster master at the specified port. By default Mesos masters listen on port 5050.
`yarn`	Connect to a YARN cluster. When running on YARN you'll need to set the `HADOOP_CONF_DIR` environment variable to point the location of your Hadoop configuration directory, which contains information about the cluster.
`local`	Run in local mode with a single core.
`local[N]`	Run in local mode with N cores.
`local[*]`	Run in local mode and use as many cores as the machine has.

Initializing a SparkContext

- The master parameter for a **SparkContext** determines which type and size of cluster to use

```
from pyspark import SparkConf, SparkContext

conf = SparkConf().setMaster("local").setAppName("SIT742")
sc = SparkContext(conf = conf)
```

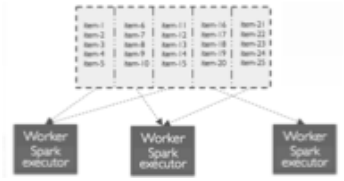
  - PySpark is not on **sys.path** by default, we can use

```
!pip install findspark
import findspark
findspark.init()

import pyspark
sc = pyspark.SparkContext(appName="SIT742")
```

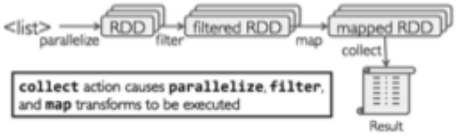

Resilient Distributed Datasets

- RDD is an ***immutable*** distributed collection of objects
  - Each RDD is split into multiple partitions, which may be computed on different nodes of the cluster
  - Track lineage to efficiently re-compute the lost data
  - Programmer specifies the number of partitions for an RDD



Resilient Distributed Datasets

- Working with RDDs
  - Create an RDD
  - Apply ***transformations*** to an RDD: map, filter, etc.
  - Apply ***actions*** to an RDD: collect, count

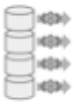


Resilient Distributed Dataset

- Creating RDDs
- RDD Operations
  - Transformation
  - Actions
- Caching RDDs

Creating RDDs

- Two ways to create RDDs:
  - ***Parallelizing*** existing Python collections
    - We can specify the number of partitions (4 here)
    - No computation occurs with **sc.parallelize**
    - Spark only records how to create the RDD with 4 partitions here

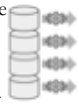


```
>>> data = [1, 2, 3, 4, 5]

>>> rdd = sc.parallelize(data, 4)
>>> rdd
ParallelCollectionRDD[6] at parallelize at PythonRDD.scala:392
```


Creating RDDs

- Two ways to create RDDs:
  - ***Loading*** an dataset from HDFS, or other storage
    - We can specify the number of partitions (4 here)
    - No computation occurs with **sc.parallelize**
    - Spark only records how to create the RDD with 4 partitions here



```
>>> fdd = sc.textfile("/path/to/filename", 4)

>>> fdd
MapPartitionsRDD[14] at textFile at
NativeMethodAccessorImpl.java:-2
```

RDD Operations

Transformations

- Operations on RDDs that return a new RDD
- ***Lazy Evaluation***
  - Results not computed right away
  - Spark remembers set of transformations applied to base dataset

Actions

- Operations that return a result to the driver program or write it to storage
- Eager
  - Cause Spark to kick off a computation
  - Mechanism for getting results out of Spark

RDD Operations: Transformation

- Python **`lambda`** Functions
  - Small anonymous functions that is
    - Restricted to a single expression
    - Not bound to a name
  - Can use **`lambda`** functions wherever function objects are required

```
lambda x: x*2

lambda x: x!= 1

lambda x: [x, x+5]
```


RDD Operations: Transformation

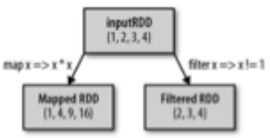
- Element-wise transformations
  - Transformation **`filter()`**
    - Takes in a function and returns an RDD that only has elements that pass the **`filter( )`** function

```
Rdd = sc.parallelize([1,2,3,4])
Rdd.filter(lambda x: x!=1)
```

RDD Operations: Transformation

- Element-wise transformations
  - Transformation **`map()`**
    - Takes in a function and applies it to each element in the RDD with the result of the function being the new value of each element in the resulting RDD

```
Rdd = sc.parallelize([1,2,3,4])
Rdd.Map(lambda x: x*2)
```



RDD Operations: Transformation

- Element-wise transformations
  - Transformation **`flatmap( )`**
    - Get back an RDD that consists of the elements from all of its iterators

RDD Operations: Transformation

- Sampling Transformation
  - **`sample()`** an RDD
    - We can specify with or without replacement, or the fraction

```
Rdd = sc.parallelize([1,2,3,3])
Sdd = Rdd.sample(false,0.5)
```

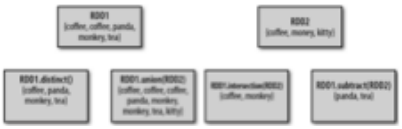
RDD Operations: Transformation

Table 3-2. Basic RDD transformations on an RDD containing {1, 2, 3, 3}

Function name	Purpose	Example	Result
map()	Apply a function to each element in the RDD and return an RDD of the result.	rdd.map(x => x + 1)	{2, 3, 4, 4}
flatMap()	Apply a function to each element in the RDD and return an RDD of the contents of the iterators returned. Often used to extract words.	rdd.flatMap(x => x.to(3))	{1, 2, 3, 2, 3, 3, 3}
filter()	Return an RDD consisting of only elements that pass the condition passed to filter().	rdd.filter(x => x != 1)	{2, 3, 3}
distinct()	Remove duplicates.	rdd.distinct()	{1, 2, 3}
sample(withReplacement, fraction, [seed])	Sample an RDD, with or without replacement.	rdd.sample(false, 0.5)	Nondeterministic

RDD Operations: Transformation

- Pseudo Set Operations
  - RDDs support many operations of mathematical sets:
    - **distinct**, **union**, **intersect**, **subtract**
    - All expensive except **union** because they involve shuffling



RDD Operations: Transformation

Table 3-3. Two-RDD transformations on RDDs containing {1, 2, 3} and {3, 4, 5}

Function name	Purpose	Example	Result
union()	Produce an RDD containing elements from both RDDs.	rdd.union(other)	{1, 2, 3, 3, 4, 5}
intersection()	RDD containing only elements found in both RDDs.	rdd.intersection(other)	{3}
subtract()	Remove the contents of one RDD (e.g., remove training data).	rdd.subtract(other)	{1, 2}
cartesian()	Cartesian product with the other RDD.	rdd.cartesian(other)	{(1, 3), (1, 4), … (3,5)}

RDD Operations: Action

- **reduce( )**
  - takes a function that operates on two elements of the type in the RDD and returns a new element of the same type
  - should be ***commutative*** and ***associative*** so that it can be computed correctly in parallel

```
>>> Rdd = sc.parallelize([1,2,3])
>>> Rdd.reduce(lambda a, b: a*b)
Value: 6
```


RDD Operations: Action

- **fold( )**
  - takes a function similarly as **reduce( )** does, but takes a "zero value" to be used for initial call on each partition
    - should be the identity element for the operation
      - 0 for +, 1 for *, etc.
  - return type the same as RDD elements

```
>>> Rdd = sc.parallelize([1,2,3])
>>> Rdd.fold(0)(lambda a, b: a+b)
Value: 6
```


RDD Operations: Action

- **aggregate( )**
  - we can also supply an initial zero value of the type we want to return
  - a 1st function to combine the elements from RDD with the accumulator
  - a 2nd function to merge two accumulators given that each node accumulates its own results locally

```
sumCount = Rdd.aggregate((0,0),
       (lambda acc, value: (acc[0]+value, acc[1]+1),
       (lambda acc1,acc2: (acc1[0]+acc2[0],acc1[1]+acc2[1]))))
return sumCount[0]/float(sumCount[1])
```

RDD Operations: Action

- **`collect( )`**
  - return the entire RDD's contents to the driver program
- **`take( )`**
  - returns n elements from the RDD
  - attempts to minimize the number of partitions it accesses, so may be biased

```
>>> Rdd = sc.parallelize([1,2,3])
>>> Rdd.take(2)
Value: [1,2]
>>> Rdd.collect()
Value: [1,2,3]
```


RDD Operations: Action

- **`top(n)`**
  - return the top n elements of the RDD
- **`count( )`**
  - returns the number of elements in the RDD

```
>>> Rdd = sc.parallelize([1,2,3])
>>> Rdd.count()
Value: 3
```


RDD Operations: Action

Table 3-4. Basic actions on an RDD containing {1, 2, 3, 3}

Function name	Purpose	Example	Result
`collect()`	Return all elements from the RDD.	`rdd.collect()`	`{1, 2, 3, 3}`
`count()`	Number of elements in the RDD.	`rdd.count()`	`4`
`countByValue()`	Number of times each element occurs in the RDD.	`rdd.countByValue()`	`{(1, 1), (2, 1), (3, 2)}`
`take(num)`	Return num elements from the RDD.	`rdd.take(2)`	`{1, 2}`
`top(num)`	Return the top num elements the RDD.	`rdd.top(2)`	`{3, 3}`
`takeOrdered(num)(ordering)`	Return num elements based on provided ordering.	`rdd.takeOrdered(2)(myOrdering)`	`{3, 3}`

RDD Operations: Action

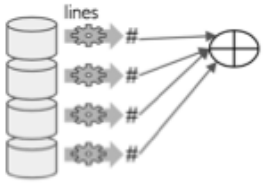
`takeSample(withReplacement, num, [seed])`	Return num elements at random.	`rdd.takeSample(false, 1)`	Nondeterministic
`reduce(func)`	Combine the elements of the RDD together in parallel (e.g., sum).	`rdd.reduce((x, y) => x + y)`	9
`fold(zero)(func)`	Same as `reduce()` but with the provided zero value.	`rdd.fold(0)((x, y) => x + y)`	9
`aggregate(zeroValue)(seqOp, combOp)`	Similar to `reduce()` but used to return a different type.	`rdd.aggregate((0, 0))((x, y) => (x._1 + y, x._2 + 1), (x, y) => (x._1 + y._1, x._2 + y._2))`	(9, 4)
`foreach(func)`	Apply the provided function to each element of the RDD.	`rdd.foreach(func)`	Nothing

Caching RDDs

```
lines = sc.textFile("...", 4)

print lines.count()
```

Count will cause Spark to
- read data
- sum within partitions
- combine sums in driver

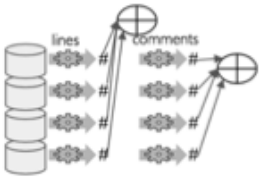


Caching RDDs

```
lines = sc.textFile("...", 4)
cmts = lines.filter(isComment)

print lines.count()
print cmts.count()
```

Count will cause Spark to
- read data (again)
- sum within partitions
- combine sums in driver



Caching RDDs

```
lines = sc.textFile("...", 4)

lines.cache()
# save, don't recompute

cmts = lines.filter(isComment)

print lines.count()
print cmts.count()

lines.unpersist()
# remove them from cache
```

Pair RDDs

- Pair RDD
- Pair RDD Operations
  - Transformations
  - Actions

Pair RDDs

- **Pair RDD**
  - Similar to MapReduce, Spark supports Key-Value pairs
  - Each element of a Pair RDD is a pair tuple
  - Allow to act on each key in parallel or regroup data across the network.

```
>>> rdd = sc.parallelize([(1, 2), (3, 4)])
RDD: [(1, 2), (3, 4)]

pairs = lines.map(lambda x: (x.split(" ")[0], x))
# create a pair RDD using the first word as the key in Python
```


Pair RDD Transformation

- **`reduceByKey(func)`**
  - Return a new distributed dataset of **`(k, v)`** pairs where the values for each key are aggregated using the given reduce function **`func`**, which must be of type **`(v, v)`** → **`v`**

```
>>> rdd = sc.parallelize([(1,2), (3,4), (3,6)])
>>> rdd.reduceByKey(lambda a, b: a + b)
RDD: [(1,2), (3,4), (3,6)] → [(1,2), (3,10)]
```


Pair RDD Transformation

- **`sortByKey(func)`**
  - Return a new dataset **`(k,v)`** pairs sorted by keys in ascending order

```
>>> rdd2 = sc.parallelize([(1,'a'), (2,'c'), (1,'b')])
>>> rdd2.sortByKey()
RDD: [(1,'a'), (2,'c'), (1,'b')] →
[(1,'a'), (1,'b'), (2,'c')]
```


Pair RDD Transformation

- **`groupByKey(func)`**
  - Return a new dataset of **`(k,iterable<V>)`** pairs
  - Be careful using this, as it can cause a lot of data movement across the network and create large **`iterable`** at works

```
>>> rdd2 = sc.parallelize([(1,'a'), (2,'c'), (1,'b')])

>>> rdd2.groupByKey()
RDD: [(1,'a'), (1,'b'), (2,'c')] →
              [(1,['a','b']), (2,['c'])]
```

Pair RDD Transformation

- **mapValues(func)**
  - Apply a function to each value of a pair RDD without changing the key
- **keys( )**
  - Return an RDD of just the keys

```
>>> rdd2 = sc.parallelize([(1,'a'), (2,'c'), (1,'b')])

>>> rdd2.mapValues(lambda x: x+'s')
RDD: [(1,'as'), (2,'cs'), (1,'bs')]

>>> rdd2.keys()
RDD: [1, 2, 1]
```


Pair RDD Transformation

- **sortByKey( )**
  - Takes a parameter called **ascending** indicating whether we want it in ascending order
  - We can also use our own comparison function

```
rdd.sortByKey(ascending=True, numPartitions=None,
      keyfunc = lambda x: str(x))
```


Pair RDD Transformation

Table 4-1. Transformations on one pair RDD (example: {(1, 2), (3, 4), (3, 6)})

Function name	Purpose	Example	Result
reduceByKey(func)	Combine values with the same key.	rdd.reduceByKey((x, y) => x + y)	{(1, 2), (3, 10)}
groupByKey()	Group values with the same key.	rdd.groupByKey()	{(1, [2]), (3, [4, 6])}
combineByKey(createCombiner, mergeValue, mergeCombiners, partitioner)	Combine values with the same key using a different result type.	See Examples 4-12 through 4-14.	
mapValues(func)	Apply a function to each value of a pair RDD without changing the key.	rdd.mapValues(x => x+1)	{(1, 3), (3, 5), (3, 7)}

Pair RDD Transformation

Function name	Purpose	Example	Result
values()	Return an RDD of just the values.	rdd.values()	{2, 4, 6}
sortByKey()	Return an RDD sorted by the key.	rdd.sortByKey()	{(1, 2), (3, 4), (3, 6)}
flatMapValues(func)	Apply a function that returns an iterator to each value of a pair RDD, and for each element returned, produce a key/value entry with the old key. Often used for tokenization.	rdd.flatMapValues(x => (x to 5))	{(1, 2), (1, 3), (1, 4), (1, 5), (3, 4), (3, 5)}
keys()	Return an RDD of just the keys.	rdd.keys()	{1, 3, 3}

Pair RDD Joins

- **X.Join(Y)** is an inner join
  - Returns RDD of all pairs of elements with matching keys in X and Y
  - Each pair is **(k, (v1,v2))** tuple, where
    - **(k,v1)** is in **X**
    - **(k,v2)** is in **Y**

```
>>> x = sc.parallelize([("a", 1), ("b", 4)])
>>> y = sc.parallelize([("a", 2), ("a", 3)])
>>> sorted(x.join(y).collect())

Value: [('a', (1, 2)), ('a', (1, 3))]
```


Pair RDD Joins

- **X.leftOuterJoin(Y)**
  - For each element **(k, v)** in X, resulting RDD will either contain
    - All pairs **(k,(v, w))** for **w** in Y
    - Or pair **(k,(v, None)** if no elements in Y have key **k**

```
>>> x = sc.parallelize([("a", 1), ("b", 4)])
>>> y = sc.parallelize([("a", 2)])
>>> sorted(x.leftOuterJoin(y).collect())

Value: [('a', (1, 2)), ('b', (4, None))]
```

Pair RDD Joins

- **`X.rightOuterJoin(Y)`**
  - For each element **`(k, w)`** in Y, resulting RDD will either contain
    - All pairs **`(k,(v, w))`** for **`v`** in X
    - Or pair **`(k,(None,w)`** if no elements in X have key **`k`**

```
>>> x = sc.parallelize([("a", 1), ("b", 4)])
>>> y = sc.parallelize([("a", 2), ("c", 5)])
>>> sorted(x.rightOuterJoin(y).collect())

Value: [('a', (1, 2)), ('c', (None,5))]
```


Pair RDD Joins

- **`X.fullOuterJoin(Y)`**
  - For each element **`(k,v)`** in X, resulting RDD will either contain all pairs **`(k,(v,w))`** for **`w`** in Y, or **`(k,(v,None)`** if no match in Y have **`k`**
  - For each element **`(k,w)`** in Y, resulting RDD with either contain all pairs **`(k,(v,w))`** for **`v`** in X, or **`(k,(None,w)`** if no match in X have **`k`**

```
>>> x = sc.parallelize([("a", 1), ("b", 4)])
>>> y = sc.parallelize([("a", 2), ("c", 5)])
>>> sorted(x.fullOuterJoin(y).collect())

Value: [('a', (1, 2)), ('b', (4, None)), ('c', (None,5))]
```


Pair RDD Actions

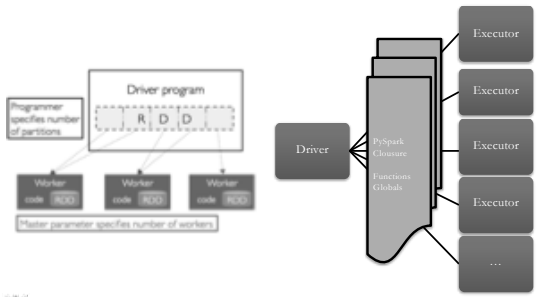
Table 4-3. Actions on pair RDDs (example {(1, 2), (3, 4), (3, 6)})

Function	Description	Example	Result
countByKey()	Count the number of elements for each key.	rdd.countByKey()	{(1, 1), (3, 2)}
collectAsMap()	Collect the result as a map to provide easy lookup.	rdd.collectAsMap()	Map({1, 2), (3, 4), (3, 6)}
lookup(key)	Return all values associated with the provided key.	rdd.lookup(3)	[4, 6]

Closures

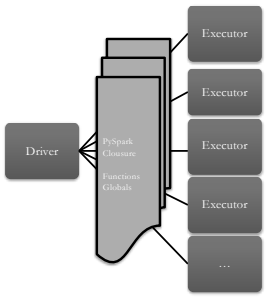
- PySpark Closures
- PySpark Shared Variables
  - Broadcast Variables
  - Accumulators

PySpark Closures



PySpark Closures

- Spark automatically creates **closures** for
  - ***Functions*** that run on RDDs at workers
  - Any ***global variables*** used by those workers
- One closure per worker
  - Sent for every task
  - No communication between workers
  - Changes to global variables at workers are not sent to the driver



PySpark Closures

- Consider the code below (***Local*** mode)
  - the code sums the values within the RDD and store it in **`counter`**, because both the RDD and the variable **`counter`** are in the same memory space on the driver node.

```
counter = 0
rdd = sc.parallelize(data)

# Wrong: Don't do this!!
rdd.foreach(lambda x: counter += x)

print("Counter value: " + counter)
```


PySpark Closures

- Consider the code below (***Cluster*** Mode)
  - Spark computes/serializes/send the closure to executors
  - Closure contains variables and methods which must be visible for the executor to perform its computations
  - The executors running on separate worker nodes each have their own copy of the closure.

```
counter = 0
rdd = sc.parallelize(data)

# Wrong: Don't do this!!
rdd.foreach(lambda x: counter += x)

print("Counter value: " + counter)
```


PySpark Closures

- Consider the code below (***Cluster*** Mode)
  - The variables within the closure to each executor are copies and thus, when **`counter`** is referenced within the **`foreach`** function, it's no longer the **`counter`** on the driver node. The **`counter`** in the driver node still there but no longer visible to the executors!

```
counter = 0
rdd = sc.parallelize(data)

# Wrong: Don't do this!!
rdd.foreach(lambda x: counter += x)

print("Counter value: " + counter)
```


Spark HelloWorld!: Word Count

```
text_file = sc.textFile("hdfs://...")

counts = text_file.flatMap(lambda line: line.split(" ")) \
             .map(lambda word: (word, 1)) \
             .reduceByKey(lambda a, b: a + b)

counts.saveAsTextFile("hdfs://...")
```


PySpark Shared Variables

Broadcast Variables

- keep a read-only variable cached on each machine rather than shipping a copy of it with tasks
- Spark also attempts to distribute broadcast variables using efficient broadcast algorithms to reduce communication cost.

Accumulators

- Aggregate values from workers back to driver
- Only driver can access value of accumulator
- For tasks, accumulators are write-only

PySpark Shared Variables: Broadcast Variables

- Spark automatically broadcasts the common data needed by tasks within each stage.
  - The data broadcasted this way is cached in serialized form and deserialized before running each task.
- Creating **broadcast variables** is only useful when tasks across multiple stages need the same data or when caching the data in deserialized form is important.

```
>>> broadcastVar = sc.broadcast([1, 2, 3])
<pyspark.broadcast.Broadcast object at 0x102789f10>
>>> broadcastVar.value
[1, 2, 3]
```

PySpark Shared Variables: Broadcast Variables

```
# Lookup the locations of the call signs on the
# RDD contactCounts. We load a list of call sign
# prefixes to country code to support this lookup
signPrefixes = loadCallSignTable()

def processSignCount(sign_count, signPrefixes):
        country = lookupCountry(sign_count[0], signPrefixes)
        count = sign_count[1]
        return (country, count)

countryContactCounts = (contactCounts
        .map(processSignCount)
        .reduceByKey((lambda x, y: x+ y)))
```


PySpark Shared Variables: Broadcast Variables

```
# Lookup the locations of the call signs on the
# RDD contactCounts. We load a list of call sign
# prefixes to country code to support this lookup
signPrefixes = sc.broadcast(loadCallSignTable())

def processSignCount(sign_count, signPrefixes):
        country = lookupCountry(sign_count[0], signPrefixes.value)
        count = sign_count[1]
        return (country, count)

countryContactCounts = (contactCounts
        .map(processSignCount)
        .reduceByKey((lambda x, y: x+ y)))
```


PySpark Shared Variables: Accumulators

- **Accumulator** provides a simple syntax for aggregating values from worker nodes back to the driver program.
- Only driver can read its value
- Used to efficiently implement parallel counters and sums

```
>>> accum = sc.accumulator(0)
>>> rdd = sc.parallelize([1, 2, 3, 4])
>>> def f(x):
>>>   global accum
>>>   accum += x
>>> rdd.foreach(f)
>>> accum.value
Value: 10
```


PySpark Shared Variables: Accumulators

```
file = sc.textFile(inputFile)
# Create Accumulator[Int] initialized to 0
blankLines = sc.accumulator(0)

def extractCallSigns(line):
        global blankLines # Make the global variable accessible
        if (line == ""):
                blankLines += 1
        return line.split(" ")

callSigns = file.flatMap(extractCallSigns)
print "Blank lines: %d" % blankLines.value
```


PySpark Shared Variables: Accumulators

- **Accumulators** can be used in actions or transformations:
  - Actions:
    - each task's update to accumulator is applied only once
  - Transformations:
    - no guarantees (use only for debugging)

Spark File Performance

- Issues to Consider
- File R/W Performance
  - Binary vs. Text
  - Compressed vs. Uncompressed
  - Read vs. Write

Considerations for File Performance

- Data Sources
  - Where to read data from?
  - Where to save data to?
- Issues to consider
  - Read vs Write Performance
  - Text vs Binary format
  - Uncompressed vs Compressed
  - Environment: Python vs Scala/Java

File Read/Write Performance

- Test run on 626 MB Text File vs 787 MB Binary File
  - **Python (Pandas)** performance is dependent on library
    - Pandas has no binary file I/O library
  - **Scala/Java**: 6 seconds is the time for sustained read/write. It is often faster due to system catching
  - Read and Write time comparable
  - Binary I/O much faster than Text I/O

	Read Time (Text)	Write Time (Text)	Read Time (Binary)	Write Time (Binary)
Pandas (Python)	36 secs	45 secs	**	**
Scala/Java	18 secs	21 secs	1-6* secs	1-6* secs

File Read/Write Performance

- Scala/Java Language
  - Write times much larger than Read
  - Large range of compression time
  - Small range of compressed file sizes

Binary File	Read Time	Write Time	File Size
Gzip level 6 (Java default)	4 secs	75 secs	286 MB
Gzip level 3	4 secs	20 secs	313 MB
Gzip level 1	4 secs	14 secs	328 MB
LZ4 fast	2 secs	4 secs	423 MB
Raw binary file	1-6 secs	1-6 secs	787 MB

File Read/Write Performance

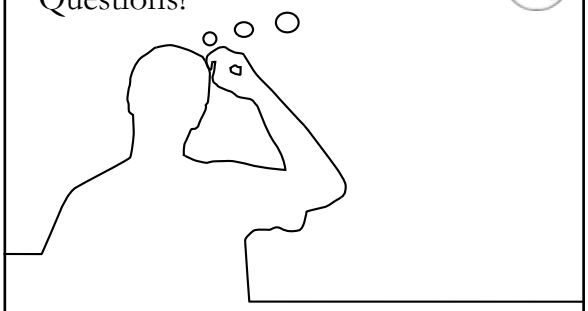
- Scala/Java Language
  - Binary I/O still much faster than Text I/O
  - LZ4 Compression ~ Raw I/O Speed

Text File	Read Time	Write Time	File Size
Gzip level 6 (default)	26 secs	98 secs	243 MB
Gzip level 3	25 secs	46 secs	259 MB
Gzip level 1	25 secs	33 secs	281 MB
LZ4 fast	22 secs	24 secs	423 MB
Raw text file	18 secs	21 secs	626 MB

File Read/Write Performance

- Summary
  - Uncompressed read and write times are comparable
  - Binary I/O is much faster than Text I/O
  - Compressed reads much faster than compressed writes
    - LZ4 is better than gzip
    - LZ4 compression times approach raw I/O times

Questions?



This Session's Readings

- Spark Programming Guide
  - https://spark.apache.org/docs/latest/programming-guide.html
- YouTube Video:
  - https://www.youtube.com/watch?v=xc7Lc8RA8wE&list=PLxwbieuTaYXmWTBovyyw2NibPfUaJk-h4&index=2

Lecture Notes on

Modern Data Science

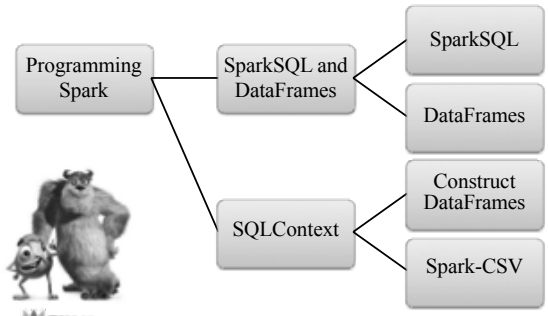
Module 06: Cloud Computing - SparkSQL

Gang Li
School of Information Technology
Deakin University, VIC 3125, Australia

Unit Learning Outcomes

- **ULO1**:
  - Develop knowledge of and discuss new and ***emerging fields*** in data science.
    - Cloud Computing and Hadoop/Spark Platform
- **ULO4**:
  - Use appropriate ***platform*** to collect and process relatively large datasets.
    - Programming in Spark

Road map



Spark SQL and DataFrames

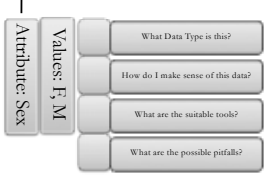
- Tabular Data
- From RDD to DataFrames
- DataFrames

Example of Datasets

visitorid	locationid	firstaccess	lastaccess	accesscount	macaddress	name
1893824	718	23:27.7	56:36.5	38		City Moorabool3 Nth
1893825	718	23:52.1	24:04.0	5		City Moorabool3 Nth
1893826	718	24:05.1	24:05.1	NaN		City Moorabool3 Nth
1893833	412	30:09.2	30:25.2	8	B4:07:F9:B9:F8:61	City Johnstone Pk
1893836	718	39:40.6	53:50.9	88		City Moorabool3 Nth
1893838	718	44:23.0	44:38.1	5.1		City Moorabool3 Nth
1893839	718	44:38.9	44:38.9	1		City Moorabool3 Nth

Example of Datasets

Family Name	Given Name	Sex	Height	Addr.
Lee	George	M	1.75	VIC
Robins	Tim	M	3.87	SA
Chan	Ruth	F	1.63	NSW
Smith	Jim	M	-	Victoria
Chiang	Yin	F	0.65	TAS
Enders	Robert	M	-	VIC



Tabular Data

***attribute**, variables, feature, field*

*object, **instance**, example, **record***

Day	Outlook	Temperature	Humidity	Wind	Play Football
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Weak	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Tabular Data

- One of the most common data formats
  - CSV, EXCEL, ARFF etc.
- A *table* is a collection of *rows* and *columns!*
  - Each row has an *index* and
  - Each column has a *name!*
- A *cell* is specified by an **`(index, name)`** *pair!*
  - A cell may or may not have a *value!*
  - A cell's *type* is inferred from its value

Tabular Data

- Limitations
  - Format not well-defined
  - May be missing data values
  - Types may be incorrectly inferred ("2" vs "2.0")
  - No Support for Versioning of Format

Spark Model

- *Write programs in terms of transformations on distributed datasets*
- **Resilient Distributed Datasets** (RDDs)
  - Collections of objects that can be stored in memory or disk across a cluster
  - Parallel functional transformations (map, filter, …)
  - Automatically rebuilt on failure

Data Challenges and Solutions

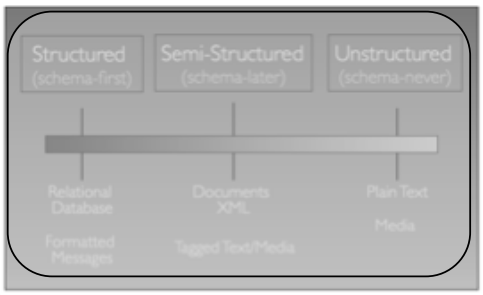
Challenges

- Perform ETL to and from various (semi- or unstructured) data sources
- Perform advanced analytics (e.g. machine learning, graph processing) that are hard to express in relational systems.

Solutions

- **SparkSQL** and **DataFrame** API that can perform relational operations on both external data sources and Spark's built-in RDDs.
- A highly extensible optimizer, ***Catalyst***, that uses features of Scala to add composable rule, control code generation, and define extensions.

Data Characteristic Hierarchy: Data Records



Spark SQL

- **SQL** = *Structured Query Language*
  - Some of the Functionality SQL provides:
    - Create, modify, delete relations
    - Add, modify, remove tuples
    - Specify queries to find tuples matching criteria

```
Select * from Users Where Users.age < 21


Select S.name, E.UID
From Students S, EnrolledUnits E
Where S.SID = E.SID
```


Spark SQL

- ***Spark SQL*** allows you to manipulate distributed data with SQL queries. Currently, two SQL dialects are supported.
  - If you're using a Spark **`SQLContext`**, the only supported dialect is "**`sql`**", a rich subset of SQL 92.
  - If you're using a **`HiveContext`**, the default dialect is "**`hiveql`**", corresponding to Hive's SQL dialect.
    - "**`sql`**" is also available, but "**`hiveql`**" is a richer dialect.

Spark SQL

- You issue SQL queries through a **`SQLContext`** or **`HiveContext`**, using the **`sql()`** method which returns a **DataFrame**.
  - You can mix DataFrame methods and SQL queries in the same code.
- To use SQL, you *must* either:
  - query a persisted Hive table, or
  - make a *table alias* for a DataFrame, using **`registerTempTable()`**

DataFrames

- **DataFrame** is a size-mutable, potentially heterogeneous tabular data structure with labeled axes (i.e., rows and columns).
  - a recent extension to **Spark RDD API** (early 2015)
  - *Distributed* collection of data organized into named columns
    - Equivalent to **Python/Pandas** and **R DataFrame**, but distributed and with richer optimizations under the hood
    - designed from the ground-up to support modern big data and data science applications
  - Types of columns inferred from values

DataFrames

- **DataFrame** has the following features:
  - Ability to scale from kilobytes of data on a single laptop to petabytes on a large cluster
  - Support for a wide array of data formats and storage systems
  - State-of-the-art optimization and code generation through the Spark SQL Catalyst optimizer
  - Seamless integration with all big data tooling and infrastructure via Spark
  - APIs for Python, Java, Scala, and R (in development via SparkR)

DataFrames

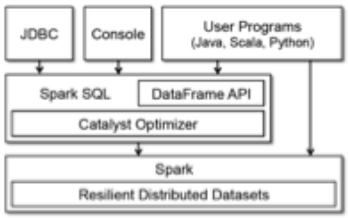
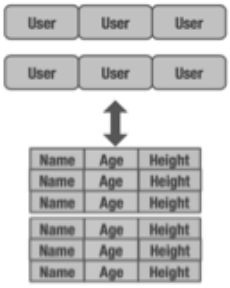


Figure 1: Interfaces to Spark SQL, and interaction with Spark.

From RDDs to DataFrames

- **RDDs**
  - Functional transformations on partitioned collections of opaque objects
- **DataFrames**
  - Declarative transformations on partitioned collections of tuples

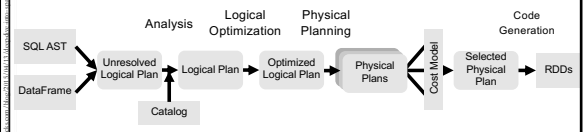


From RDDs to DataFrames

- **DataFrames** are built on top of the Spark RDD API.
  - You can use normal RDD operations on DataFrames.
  - Stick with the DataFrame API, wherever possible.
    - Using RDD operations will often give you back an RDD, not a DataFrame.
    - The DataFrame API is likely to be more efficient, because it can optimize the underlying operations with Catalyst.

From RDDs to DataFrames

- Plan Optimization & Execution
  - Represented internally as a "*logical plan*"
  - Execution is lazy, allowing it to be optimized by Catalyst

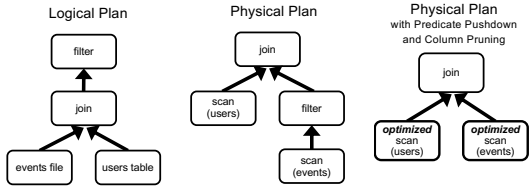


DataFrames and SQL share the same optimization/execution pipeline

From RDDs to DataFrames

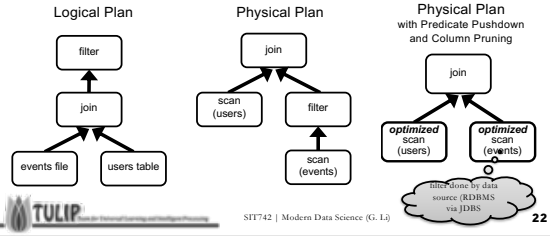
- Plan Optimization & Execution

```
joined = users.join(events, users.id == events.uid)
filtered = joined.filter(events.date >= "2015-01-01")
```



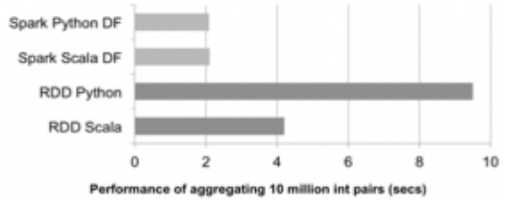
From RDDs to DataFrames

- Plan Optimization & Execution
  - Data Sources API can automatically prune columns and push filters to the source



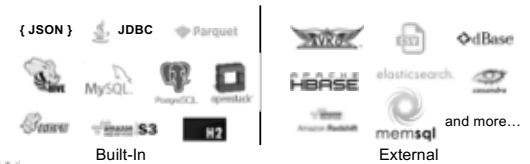
From RDDs to DataFrames

- DataFrames can be significantly faster than RDDs
- Performance is irrelevant to language



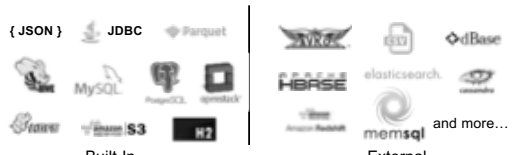
Data Sources Supported by DataFrames

- DataFrame execution is done internally with RDDs making interoperation with outside sources easy.
- Easy loading/saving DataFrames



Data Sources Supported by DataFrames

- Efficient data access by Spark SQL query optimizer
  - Have interfaces allowing optimizations to be pushed down to data source
    - e.g. avoid reading unnecessary data for a query



SparkSQL and DataFrames

- SparkSession
- Construct DataFrame
  - Table
  - File
  - RDD
- Spark-CSV

SparkSession

- Entry to all relational functionality in Spark is **`SparkSession`** class or its decedents.

```
from pyspark.sql import SparkSession

sqlContext = SparkSession \
.builder \
.appName("Python Spark SQL basic example") \
.config("spark.some.config.option", "some-value") \
.getOrCreate()
```


Construct a DataFrame from Tables

- **`DataFrames`** can be constructed from a wide array of sources such as:
  - tables in Hive, external databases, structured data files, or existing RDDs.
  - The example shows creation from a Hive Table

```
#Construct a DataFrame from a "users" table in Hive.
df = sqlContext.table("users")

# Displays the content of the DataFrame to stdout
df.show()
```


Construct a DataFrame from Data with Schema

- The (changing) role of Schema
  - **Schema** specify the **structure** and **types** of a data repository, e.g. the types of each column in a table.
  - They may also specify constraints **within** or **between** data fields.
  - Traditional databases are **schema-on-write**.
    - You cannot load data into a table without a schema.
  - Newer (NoSQL) data stores are **schema-on-read** or **schemaless**:
    - You can defer applying a schema until you read the data, or avoid schema altogether.

Construct a DataFrame from Data with Schema

- We can easily create **`DataFrames`** from those data formats that have a schema, which includes the column names and data types of **`DataFrames`**.
  - A **Parquet** table has a schema that Spark can use, it also allows Spark to be efficient about how it parses data
  - The example shows creation from a Parquet file

```
# Construct a DataFrame from a PARQUET file in installed folder.
df = sqlContext.read.load("examples/src/main/resources/users.parquet")

# Displays the content of the DataFrame to stdout
df.show()
```


Construct a DataFrame from Data with Schema

- We can easily create **`DataFrames`** from those data formats that have a schema, which includes the column names and data types of **`DataFrames`**.
  - The example shows creation from a JSON file

```
# Construct a DataFrame from a JASON file in installed folder.
df = sqlContext.read.json("examples/src/main/resources/people.json")

# Displays the content of the DataFrame to stdout
df.show()
```


Construct a DataFrame from RDD

- Spark SQL can convert an **RDD** of **`Row`** objects to a **`DataFrame`**, inferring the data types.
  - **`Rows`** are constructed by passing a list of key/value pairs as kwargs to the **`Row`** class.
    - The keys of this list define the column names of the table, and
    - The types are inferred by looking at the first row.

Construct a DataFrame from RDD (1)

```
# sc is an existing SparkContext.
from pyspark.sql import SQLContext, Row
sqlContext = SQLContext(sc)

# Load a text file and convert each line to a Row.
lines = sc.textFile("examples/src/main/resources/people.txt")
parts = lines.map(lambda l: l.split(","))
people = parts.map(lambda p: Row(name=p[0], age=int(p[1])))

# Infer the schema, and register the DataFrame as a table.
schemaPeople = sqlContext.createDataFrame(people)

schemaPeople.registerTempTable("people")
```


Construct a DataFrame from RDD (2)

```
# Import SQLContext and data types
from pyspark.sql import SQLContext
from pyspark.sql.types import *

# sc is an existing SparkContext.
sqlContext = SQLContext(sc)
# Load a text file and convert each line to a tuple.
lines = sc.textFile("examples/src/main/resources/people.txt")
parts = lines.map(lambda l: l.split(","))
people = parts.map(lambda p: (p[0], p[1].strip()))

# The schema is encoded in a string.
schemaString = "name age"
fields = [StructField(field_name, StringType(), True) for field_name in
                                                    schemaString.split()]
schema = StructType(fields)
# Apply the schema to the RDD.
schemaPeople = sqlContext.createDataFrame(people, schema)
# Register the DataFrame as a table.
schemaPeople.registerTempTable("people")
```


Example: Construct a DataFrame from CSV

- Suppose we have a text file that has no schema
- We can infer the schema

```
First Name: string
Last Name: string
Gender: string
Age: integer
```

```
Rosalita,Ramirez,F,14
Ally,Garcia,F,39
Claire,McBride,F,23
Abigail,Cottrell,F,75
José,Rivera,M,59
Ravi,Dasgupta,M,25
```

Example: Construct a DataFrame from CSV (1)

```
from pyspark.sql import Row

rdd = sc.textFile("examples/src/main/resources/people.csv")
Person = Row('first_name', 'last_name', 'gender', 'age')

def line_to_person(line):
        cells = line.split(",")
        cells[3] = int(cells[3])
        return Person(*cells)

peopleRDD = rdd.map(line_to_person)

df = peopleRDD.toDF()
# DataFrame[first_name: string, last_name: string, gender: string, age: bigint]
```


Example: Construct a DataFrame from CSV (2)

```
rdd = sc.textFile("examples/src/main/resources/people.csv")


def line_to_person(line):
        cells = line.split(",")
        return tuple(cells[0:3] + [int(cells[3])])

peopleRDD = rdd.map(line_to_person)

df = peopleRDD.toDF(("first_name","last_name","gender", "age"))
```


Extensions of DataFrames API

- **`DataFrames`** API can be extended to understand additional input formats (or, input *sources*).
  - For instance, if you're dealing with CSV files, a *very* common data file format, you can use the *spark-csv* package
    - This package augments the DataFrames API so that it understands CSV files.

Spark-CSV

- Let us assume our data file has a header

```
firstname,lastname,gender,age
Rosalita,Ramirez,F,14
Ally,Garcia,F,39
Claire,McBride,F,23
Abigail,Cottrell,F,75
José,Rivera,M,59
Ravi,Dasgupta,M,25
```


Spark-CSV

- With *spark-csv*, we can simply create a **`DataFrame`** directly from our CSV file.
  - *spark-csv* uses the header to infer the schema, but the column types will always be *string*.

```
# Python
df = sqlContext.read.format("com.databricks.spark.csv").\
              load("people.csv", header="true")

// df: org.apache.spark.sql.DataFrame = [fname: string,
// lname: string, gender: string, age: string]
```


Spark-CSV

- You can also declare the schema programmatically, which allows you to specify the column types.

```
from pyspark.sql.types import *

schema = StructType([StructField("firstName", StringType(), False),
        StructField("lastName", StringType(), False),
        StructField("gender", StringType(), False),
        StructField("age", IntegerType(), False)])

df = sqlContext.read.format("com.databricks.spark.csv").\
              schema(schema).\
              load("people.csv")
```

DataFrame Operations

- Transformations
- Actions

DataFrame Operations

- DataFrames are *lazy*
  - ***Transformations*** contribute to the query plan, but they don't execute anything.
    - filter, select, drop, intersect, join etc.
  - ***Actions*** cause the execution of the query
    - Spark initiates a distributed read of the data source
    - The data flows through the transformations (the RDDs resulting from the Catalyst query plan)
    - The result of the action is pulled back into the driver JVM.

DataFrames Operations

DataFrames Operations

- **Columns**
  - A DataFrame *column* is an abstraction. It provides a common column-oriented view of the underlying data, *regardless* of how the data is really organized.
  - A place (a *cell*) for a data value to reside, within a row of data. This cell can have several states:
    - empty (null), missing (not there at all), or contains a (typed) value (non-null)
  - A collection of those cells, from multiple rows

DataFrames Operations

- **Columns**
  - Assume we have a **`DataFrame`**, **`df`**, that reads a data source with "**`first`**", "**`last`**", and "**`age`**" columns.
  - In Python, it's possible to access a **`DataFrame`** 's columns either by attribute (**`df.age`**) or by indexing (**`df['age']`**).

```
# Python
df = sqlContext.read.format("com.databricks.spark.csv").\
               load("people.csv", header="true")

// df: org.apache.spark.sql.DataFrame = [fname: string,
// lname: string, gender: string, age: string]
```


DataFrames Operations

- **`Show()`**
  - You can look at the first *n* elements in a **DataFrame** with the **`show()`** method.
    - If not specified, *n* defaults to 20.
  - This method is an *action*: It:
    - reads (or re-reads) the input source
    - executes the RDD DAG across the cluster
    - pulls the *n* elements back to the driver JVM
    - displays those elements in a tabular form

DataFrames Operations

- **`Select()`**
  - behaves like a SQL SELECT, allowing you to limit the results to specific columns.
  - also allows you create on-the-fly *derived columns*.

```
df.select(df['first_name'], df['age'], df['age'] > 49).show(5)

+----------+---+----------+
|first_name|age|(age > 49)|
+----------+---+----------+
|      Erin| 42|     false|
|    Claire| 23|     false|
|    Norman| 81|      true|
|    Miguel| 64|      true|
+----------+---+----------+
```


DataFrames Operations

- **`Select()`**
  - you can also use SQL

```
df.registerTempTable("names")
sqlContext.sql("SELECT first_name, age, age > 49 FROM names").\
           show(5)

+----------+---+----------+
|first_name|age|(age > 49)|
+----------+---+----------+
|    Norman| 81|      true|
|    Miguel| 64|      true|
+----------+---+----------+
```


DataFrames Operations

- **`filter()`**
  - allows you to filter rows out of your results.

```
df.filter(df['age'] > 49).\
       select(df['first_name'], df['age']).\
       show()
+---------+---+
|firstName|age|
+---------+---+
|   Norman| 81|
|   Miguel| 64|
|  Abigail| 75|
+---------+---+
```


DataFrames Operations

- **`filter()`**
  - allows you to filter rows out of your results.
  - Here's the SQL version.

```
SQLContext.sql("SELECT first_name, age FROM names " + \
               "WHERE age > 49").show()

+---------+---+
|firstName|age|
+---------+---+
|   Norman| 81|
|   Miguel| 64|
|  Abigail| 75|
+---------+---+
```


DataFrames Operations

- **`orderby()`**
  - allows you to sort your results.

```
df.filter(df['age'] > 49).\
       select(df['first_name'], df['age']).\
       orderBy(df['age'].desc(), df['first_name']).show()

+---------+---+
|firstName|age|
+---------+---+
|   Norman| 81|
|  Abigail| 75|
|   Miguel| 64|
+---------+---+
```


DataFrames Operations

- **`orderby()`**
  - allows you to sort your results.
  - Here's the SQL version.

```
SQLContext.sql("SELECT first_name, age FROM names " + \
       "WHERE age > 49 ORDER BY age DESC, first_name").show()

+---------+---+
|firstName|age|
+---------+---+
|   Norman| 81|
|   Miguel| 64|
|  Abigail| 75|
+---------+---+
```

DataFrames Operations

- **groupBy()**
  - Often used with **count()**, **groupBy()** groups data items by a specific column value

```
df.groupBy("age").count().show()

+---+-----+
|age|count|
+---+-----+
| 39|    1|
| 42|    2|
| 64|    1|
| 75|    1|
| 81|    1|
+---+-----+
```


DataFrames Operations

- **groupBy()**
  - and SQL is not surprising

```
SQLContext.sql("SELECT age,'count(age) FROM names " + \
       "GROUP BY age").show()
+---+-----+
|age|count|
+---+-----+
| 39|    1|
| 42|    2|
| 64|    1|
| 75|    1|
| 81|    1|
+---+-----+
```


DataFrames Operations

- **alias()**
  - allows you to rename a column.
  - It's especially useful with generated columns.

```
df.select(df['first_name'], df['age'],\
       (df['age'] <'30).alias('young')).show(5)
+----------+---+-----+
|first_name|age|young|
+----------+---+-----+
|      Erin| 42|false|
|    Claire| 23| true|
|    Norman| 81|false|
|    Miguel| 64|false|
|  Rosalita| 14| true|
+----------+---+-----+
```


DataFrames Operations

- **alias()**
  - and of course, in SQL

```
SQLContext.sql("SELECT firstName,age,age<30 AS young " \
       FROM names".show()
+----------+---+-----+
|first_name|age|young|
+----------+---+-----+
|      Erin| 42|false|
|    Claire| 23| true|
|    Norman| 81|false|
|    Miguel| 64|false|
|  Rosalita| 14| true|
+----------+---+-----+
```


DataFrames Operations

- **join()**
  - Let's assume we have a second file, a JSON file that contains records like this:

```
{
"firstName": "Erin",
"lastName": "Shannon",
"medium":"oil on canvas"
},
{
"firstName": "Norman",
"lastName": "Lockwood",
"medium": "metal
(sculpture)"
},
…
```


DataFrames Operations

- **join()**
  - We can load and then join a second DataFrame

```
df2'='sqlContext.read.json("artists.json")
#"Schema"inferred"as"DataFrame[firstName:" string,"lastName:"string,"medium:" string]

df.join(
  df2,'
  df.first_name =='df2.firstName'and df.last_name =='df2.lastName
 ).show()
+----------+---------+------+---+---------+--------+------------------+
|first_name|last_name|gender|age|firstName|lastName|            medium|
+----------+---------+------+---+---------+--------+------------------+
|    Norman| Lockwood|     M| 81|   Norman|Lockwood| metal (sculpture)|
|      Erin|  Shannon|     F| 42|     Erin| Shannon|     oil on canvas|
|  Rosalita|  Ramirez|     F| 14| Rosalita| Ramirez|          charcoal|
|    Miguel|     Ruiz|     M| 64|   Miguel|    Ruiz|     oil on canvas|
+----------+---------+------+---+---------+--------+------------------+
```

DataFrames Operations

- **join()**
  - We can also select some of the columns

```
df2 = sqlContext.read.json("artists.json")
#"Schema"inferred"as"DataFrame[firstName:" string,"lastName:"string,"medium:" string]
df.join(df2,
  df.first_name == df2.firstName and df.last_name == df2.lastName )
df3.select("first_name",'"last_name','"age','"medium").show()

+----------+---------+---+------------------+
|first_name|last_name|age|            medium|
+----------+---------+---+------------------+
|    Norman| Lockwood| 81| metal (sculpture)|
|      Erin|  Shannon| 42|      oil on canvas|
|  Rosalita|  Ramirez| 14|          charcoal|
|    Miguel|     Ruiz| 64|      oil on canvas|
+----------+---------+---+------------------+
```


Example of DataFrames Operations

```
from pyspark.sql import SQLContext
from pyspark.sql.types import *
# sc is an existing SparkContext.
sqlContext = SQLContext(sc)
# Load a text file and convert each line to a tuple.
lines = sc.textFile("examples/src/main/resources/people.txt")
parts = lines.map(lambda l: l.split(","))
people = parts.map(lambda p: (p[0], p[1].strip()))
# The schema is encoded in a string.
schemaString = "name age"
fields = [StructField(field_name, StringType(), True) for field_name in  \
                    schemaString.split()]
schema = StructType(fields)
# Apply the schema to the RDD.
schemaPeople = sqlContext.createDataFrame(people, schema)
# Register the DataFrame as a table.
schemaPeople.registerTempTable("people")
# SQL can be run over DataFrames that have been registered as a table.
results = sqlContext.sql("SELECT name FROM people")
# The results of SQL queries are RDDs and support all the normal RDD operations.
names = results.map(lambda p: "Name: " + p.name)
for name in names.collect(): print(name)
```


Writing DataFrames

- to Files
- to HIVE Tables
- Save Modes

Writing DataFrames into Files

- You can write DataFrames out, as well.
  - When doing ETL, this is a *very* common requirement.
  - In most cases, if you can read a data format, you can write that data format, as well.
  - If you're writing to a text file format (e.g., JSON), you'll typically get multiple output files.

```
df.write.format("json").save("/path/to/directory")
df.write.format("parquet").save("/path/to/directory")
```


Writing DataFrames into Hive

- When with a HiveContext, you can save a DataFrame as a persistent table, with the saveAsTable() method.
  - Unlike registerTempTable(), saveAsTable() materializes the DataFrame (i.e., runs the DAG) and creates a pointer to the data in the Hive metastore.
  - Persistent tables will exist even after your Spark program has restarted.
  - By default, saveAsTable() will create a *managed table*:
    - the metastore controls the location of the data.
    - Data in a managed table is also deleted automatically when the table is dropped.

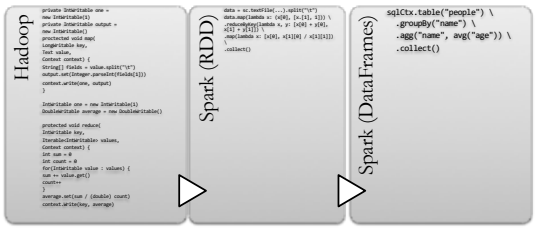
Writing DataFrames: Save Modes

Scala/Java	Python	Meaning
`SaveMode.ErrorIfExists` (default)	`"error"`	If output data or table already exists, an exception is expected to be thrown.
`SaveMode.Append`	`"append"`	If output data or table already exists, append contents of the DataFrame to existing data.
`SaveMode.Overwrite`	`"overwrite"`	If output data or table already exists, replace existing data with contents of DataFrame.
`SaveMode.Ignore`	`"ignore"`	If output data or table already exists, do not write DataFrame at all.

Writing DataFrames: Save Modes

- **These save modes do *not* utilize any locking and are *not* atomic.**
  - Thus, it is *not* safe to have multiple writers attempting to write to the same location.
  - Additionally, when performing an overwrite, the data will be deleted before writing out the new data.

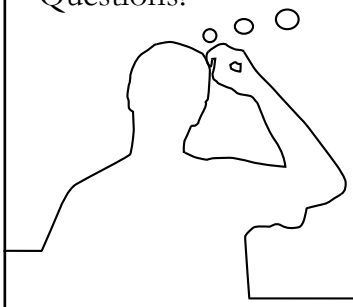
MapReduce→ RDD → DataFrames →…



What is Next? Spark Datasets!

- Spark Datasets, an extension of the DataFrame API that provides a *type-safe, object-oriented programming interface*.
  - Spark 1.6 and later versions includes an API preview of Datasets, but no Python API in the current release
  - Like DataFrames, Datasets take advantage of Spark's Catalyst optimizer by exposing expressions and data fields to a query planner.
  - A Dataset is a strongly-typed, immutable collection of objects that are mapped to a relational schema.

Questions?



This Session's Readings

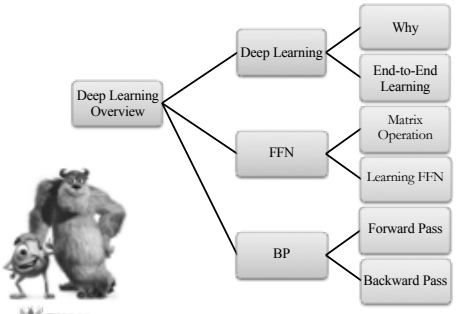
- Spark SQL, DataFrames and Datasets
  - http://spark.apache.org/docs/latest/sql-programming-guide.html
  - https://databricks.com/blog/2015/02/17/introducing-dataframes-in-spark-for-large-scale-data-science.html
  - http://spark.apache.org/docs/latest/sql-programming-guide.html#starting-point-sparksession
- Usage
  - https://databricks.com/blog/2015/06/02/statistical-and-mathematical-functions-with-dataframes-in-spark.html

Lecture Notes on
Advanced Data Analytics

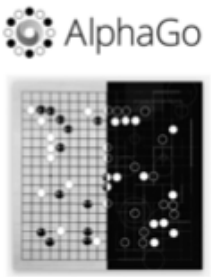
Module 05: Deep Learning Overview

Gang Li
School of Information Technology
Deakin University, VIC 3125, Australia

Road map

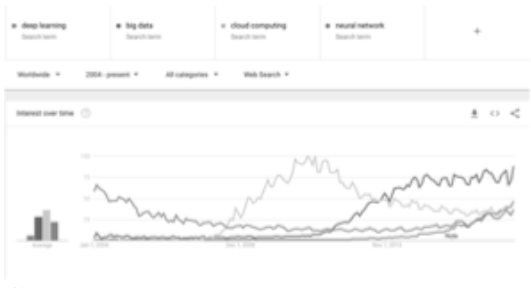


Deep Learning



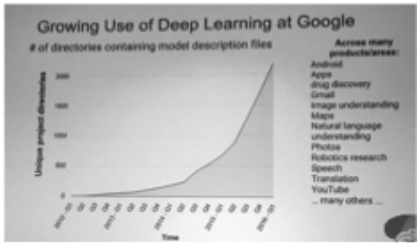
- Deep Learning
- Why Deep Learning?
  - Modularization
  - End-to-End Learning

Google Trends



Deep learning trends at Google

- SIGMOD 2016, Jeff Dean



What is Deep Learning?

- Fast answer:
  - Fast answer: multilayer perceptron (aka deep neural networks) of the 1980s rebranded in 2006.
  - But early nets go stuck at 1-2 hidden layers.

- Slow answer:
  - distributed representation, multiple steps of computation, modelling the compositionality of the world, a better prior, advances in compute, data & optimization, neural architectures, etc.

From ANN to Deep Learning

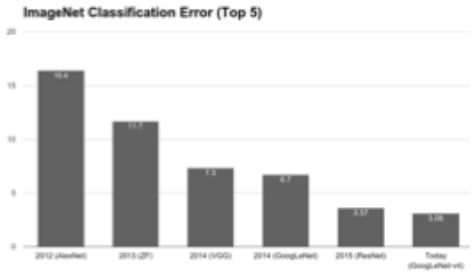
- 1958: Perceptron (linear model)
- 1969: Perceptron has limitation
- 1980s: Multi-layer perceptron
  - Do not have significant difference from DNN today
- 1986: Backpropagation
  - Usually more than 3 hidden layers is not helpful
- 1989: 1 hidden layer is "good enough"
  - why deep?



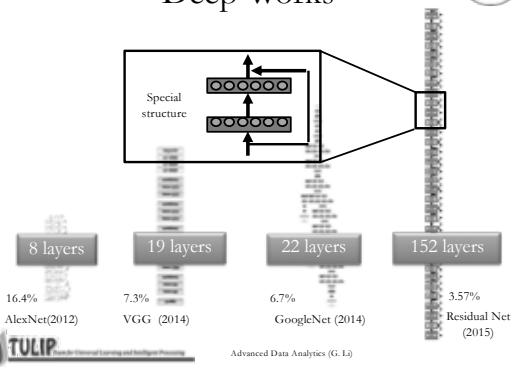
From ANN to Deep Learning

- 2006: RBM initialization
- 2009: GPU
- 2012: win ILSVRC image competition
- 2015.2:
  - Image recognition surpassing human-level performance
- 2016.3: Alpha GO beats Lee Sedol
- 2016.10:
  - Many system as good as humans

Deep Works



Deep Works

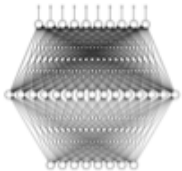


The Best of Machine Learning

- Strong/flexible priors:
  - Good features: Feature engineering
  - Data structure: HMM, CRF, MRF, Bayesian nets
  - Model structure, VC-dimension, regularization, sparsity:
    - SVM, compressed sensing
  - Manifold assumption, class/region separation:
    - Metric + semi-supervised learning
  - Factors of variation: PCA, ICA, FA
- Uncertainty quantification:
  - Bayesian, ensemble: RF, GBM
- Sharing statistical strength:
  - model reuse: transfer learning, domain adaption, multitask learning, lifelong learning

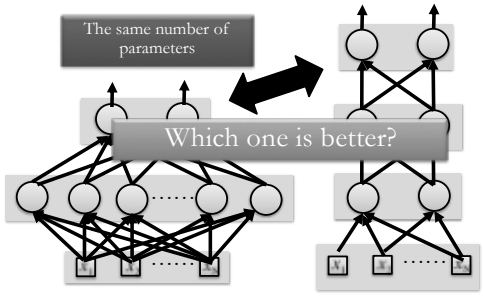
Universality Theorem

- Shallow network can represent any function.
- Given **enough** hidden neurons, Any continuous function f: $R^N \rightarrow R^M$ can be realized by a network with one hidden layer
- However, using deep structure is more effective.



Reference for the reason:
http://neuralnetworksanddeeplearning.com/chap4.html

Fat + Short v.s. Thin + Tall



Fat + Short v.s. Thin + Tall

Layer X Size	Word Error Rate (%)	Layer X Size	Word Error Rate (%)
1 X 2k	24.2		
2 X 2k	20.4		
3 X 2k	18.4		
4 X 2k	17.8		
5 X 2k	17.2	1 X 3772	22.5
7 X 2k	17.1	1 X 4634	22.6
		1 X 16k	22.1

Why?

- Seide, Frank, Gang Li, and Dong Yu. "Conversational Speech Transcription Using Context-Dependent Deep Neural Networks." *Interspeech*. 2011.

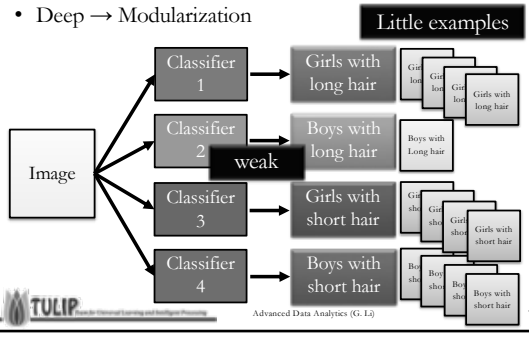
Modularization

- Deep → Modularization
  - Don't put everything in your main function.

http://rinuboney.github.io/2015/10/18/theoretical-motivations-deep-learning.html

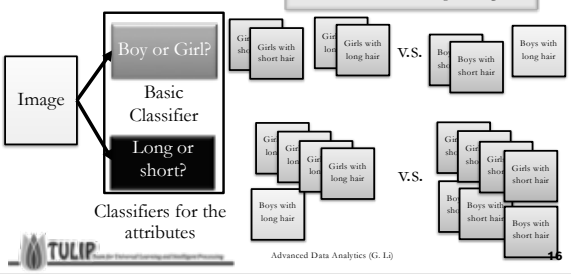
Modularization

- Deep → Modularization



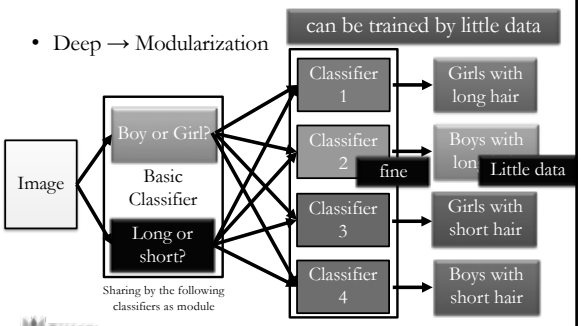
Modularization

- Deep → Modularization
  - Each basic classifier can have sufficient training examples.



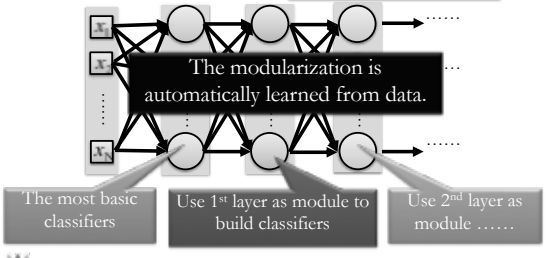
Modularization

- Deep → Modularization



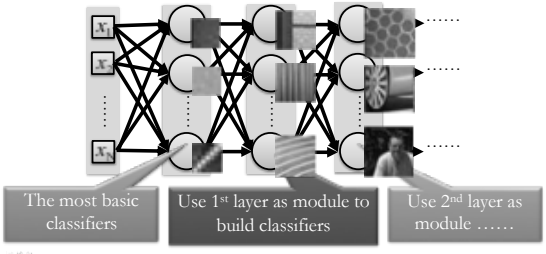
Modularization

- Deep → Modularization → Less training data?



Modularization

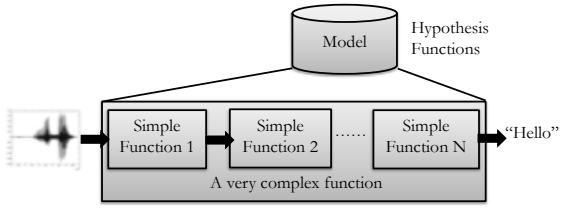
- Deep → Modularization



Reference: Zeiler, M. D., & Fergus, R. (2014). Visualizing and understanding convolutional networks. In *Computer Vision–ECCV 2014* (pp. 818-833)

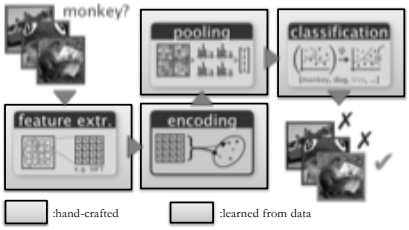
End-to-end Learning

- End-to-end training:
  - What each function should do is learned automatically



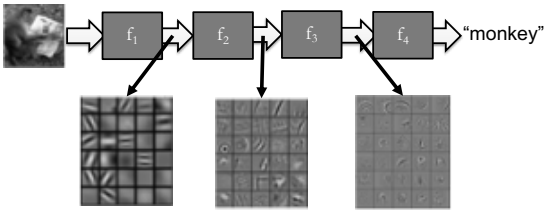
End-to-end Learning

- Shallow Approach
  - Each box is a simple function in the production line:



End-to-end Learning

- Deep Learning
  - All functions are learned from data

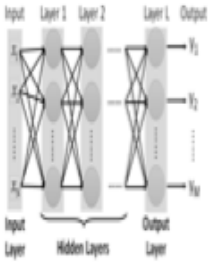


To learn more …

- Do Deep Nets Really Need To Be Deep? (by Rich Caruana)
  - http://research.microsoft.com/apps/video/default.aspx?id=232373&r=1

keynote of Rich Caruana at ASRU 2015

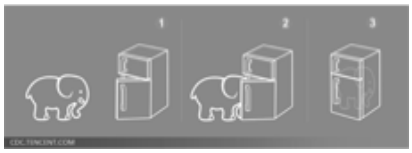
FFN



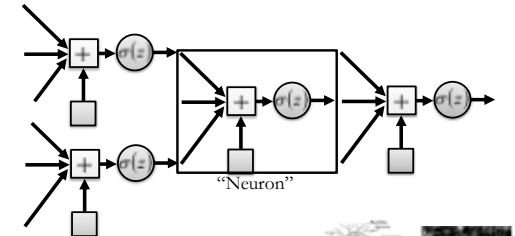
- FFN
- Matrix Operation
- Learning

Deep Learning is so simple ……

Step 1: define a set of function → Step 2: goodness of function → Step 3: pick the best function

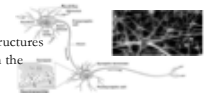


Neural Network

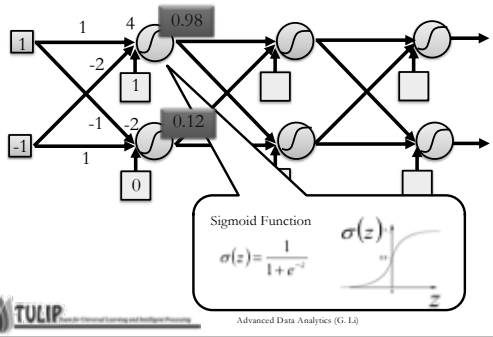


Neural Network:

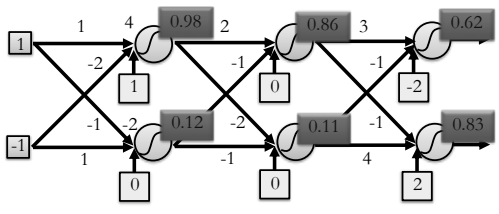
- Different connection leads to different network structures
- Network parameter θ: all the weights and biases in the "neurons"



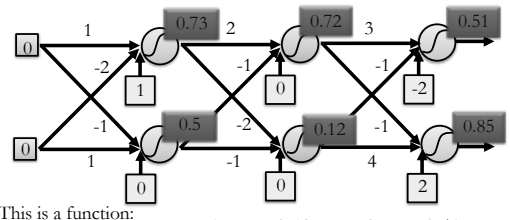
Fully Connected Feedforward Network



Fully Connected Feedforward Network



Fully Connected Feedforward Network



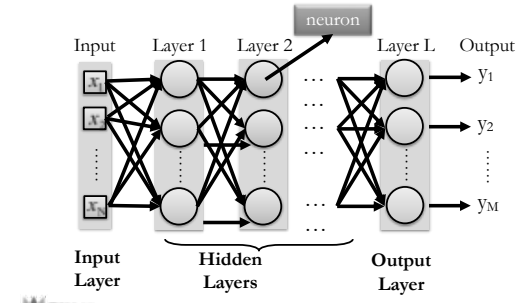
This is a function:

- input vector
- output vector

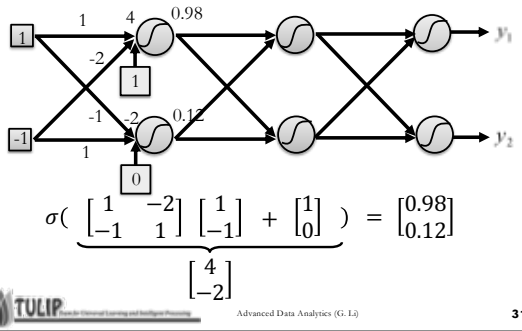
$f\left(\begin{bmatrix}1\\-1\end{bmatrix}\right)=\begin{bmatrix}0.62\\0.83\end{bmatrix}$ $f\left(\begin{bmatrix}0\\0\end{bmatrix}\right)=\begin{bmatrix}0.51\\0.85\end{bmatrix}$

Given network structure, define ***a function set***

Fully Connected Feedforward Network



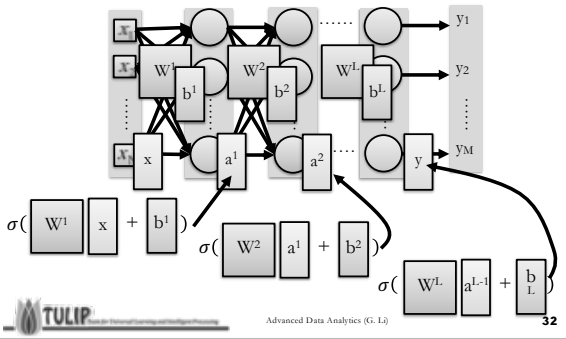
Matrix Operation



$$\sigma\left(\begin{bmatrix}1 & -2\\ -1 & 1\end{bmatrix}\begin{bmatrix}1\\ -1\end{bmatrix}+\begin{bmatrix}1\\ 0\end{bmatrix}\right)=\begin{bmatrix}0.98\\ 0.12\end{bmatrix}$$

$$\begin{bmatrix}4\\ -2\end{bmatrix}$$

Fully Connected Feedforward Network

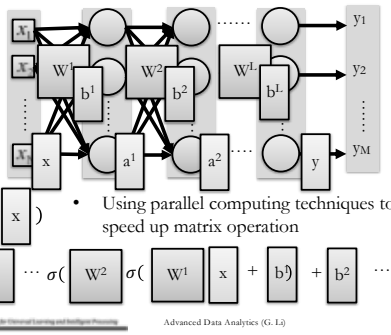


$$\sigma(W^1 x + b^1)$$

$$\sigma(W^2 a^1 + b^2)$$

$$\sigma(W^L a^{L-1} + b_L)$$

Fully Connected Feedforward Network



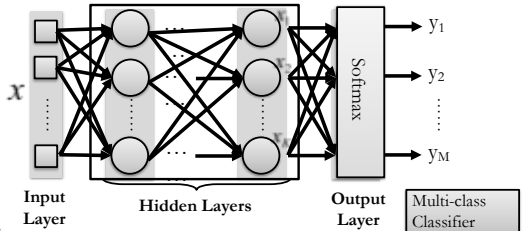
$$y = f(x)$$

- Using parallel computing techniques to speed up matrix operation

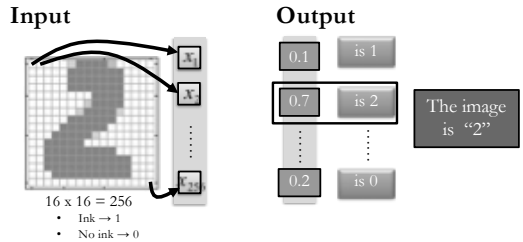
$$= \sigma(W^L \cdots \sigma(W^2 \sigma(W^1 x + b^1) + b^2) \cdots + b^L)$$

Fully Connected Feedforward Network

- Multi-class classifier
  - Feature extractor replacing feature engineering

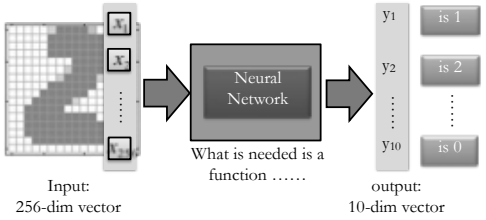


Example Application



Example Application

- Handwriting Digit Recognition

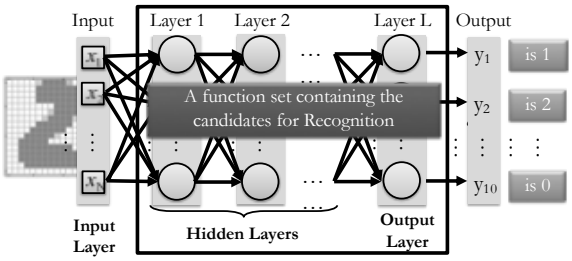


Input: 256-dim vector

What is needed is a function ……

output: 10-dim vector

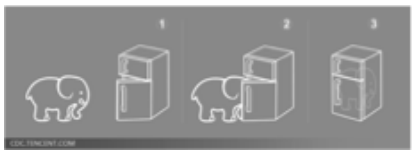
Example Application



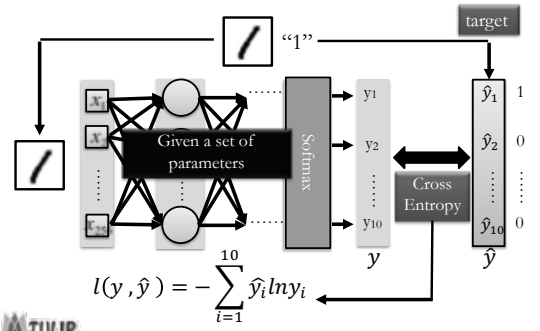
- You need to decide the network structure to let a good function in your function set.

Deep Learning is so simple ……

Step 1: define a set of function → Step 2: goodness of function → Step 3: pick the best function

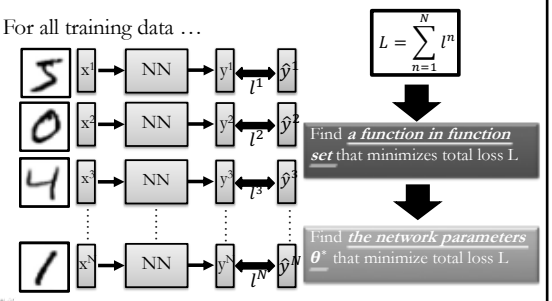


Loss for an Example



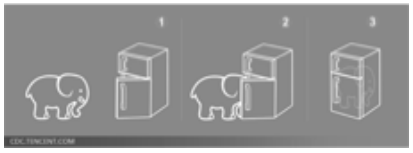
Total Loss

For all training data …



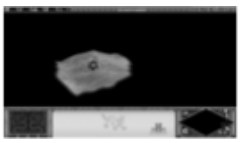
Deep Learning is so simple ……

Step 1: define a set of function → Step 2: goodness of function → Step 3: pick the best function

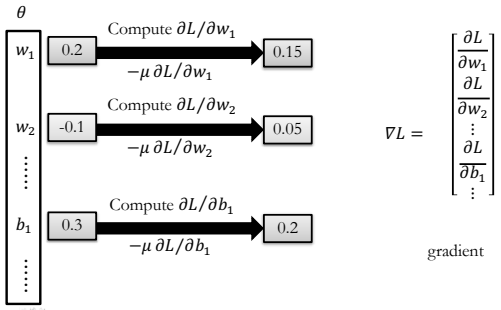


Gradient Descent

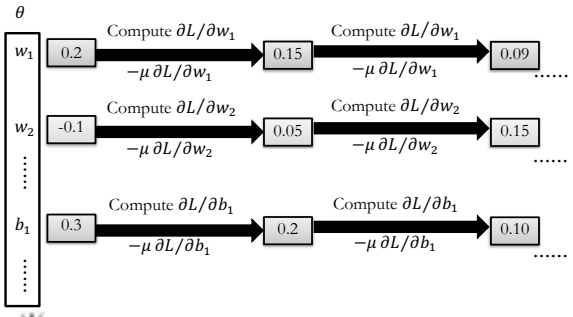
- Gradient Descent is the "learning" of machines in deep learning ……
  - Even alpha go using this approach.

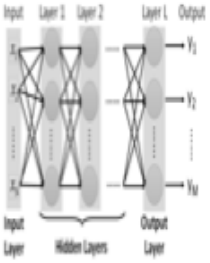
Gradient Descent



Gradient Descent



Back Propagation



- Forward Pass
- Backward Pass

Gradient Descent

Network parameters $\theta = \{w_1, w_2, \cdots, b_1, b_2, \cdots\}$

Starting Parameters $\theta^0 \rightarrow \theta^1 \rightarrow \theta^2 \rightarrow \cdots\cdots$

$$\nabla L(\theta) = \begin{bmatrix} \partial L(\theta)/\partial w_1 \\ \partial L(\theta)/\partial w_2 \\ \vdots \\ \partial L(\theta)/\partial b_1 \\ \partial L(\theta)/\partial b_2 \\ \vdots \end{bmatrix}$$

Compute $\nabla L(\theta^0)$ $\quad \theta^1 = \theta^0 - \eta \nabla L(\theta^0)$

Compute $\nabla L(\theta^1)$ $\quad \theta^2 = \theta^1 - \eta \nabla L(\theta^1)$

Millions of parameters ……

To compute the gradients efficiently, we use ***backpropagation***.

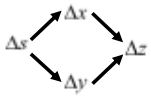
Chain Rule

Case 1 $\quad y = g(x) \quad z = h(y)$

$$\Delta x \rightarrow \Delta y \rightarrow \Delta z \qquad \frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

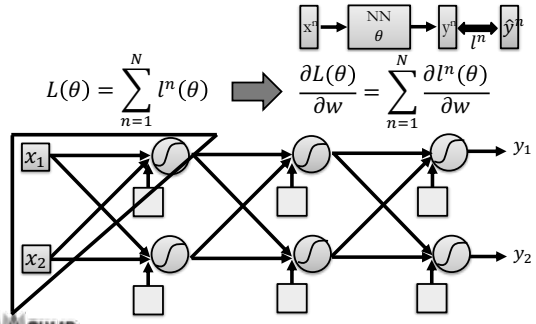
Case 2

$$x = g(s) \quad y = h(s) \quad z = k(x, y)$$

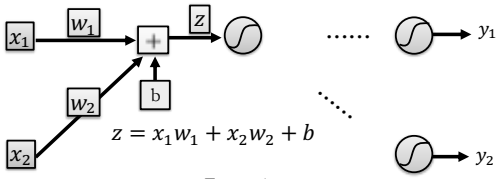


$$\frac{dz}{ds} = \frac{\partial z}{\partial x}\frac{dx}{ds} + \frac{\partial z}{\partial y}\frac{dy}{ds}$$

Backpropagation



Backpropagation



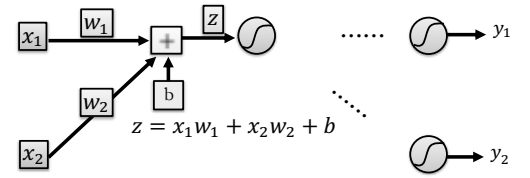
$\frac{\partial l}{\partial w} = ? \quad \frac{\partial z}{\partial w}\frac{\partial l}{\partial z}$ (Chain rule)

Forward pass:
Compute $\partial z/\partial w$ for all parameters

Backward pass:
Compute $\partial l/\partial z$ for all activation function inputs z

Backpropagation – Forward pass

Compute $\partial z/\partial w$ for all parameters

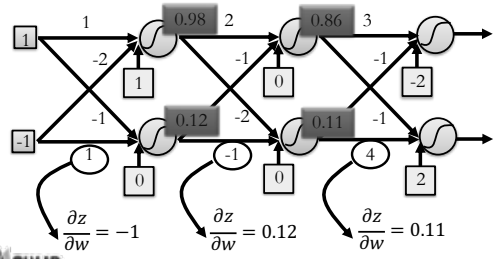


$\partial z/\partial w_1 = ?\ x_1$
$\partial z/\partial w_2 = ?\ x_2$

The value of the input connected by the weight

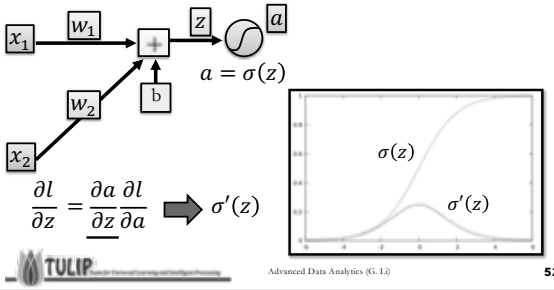
Backpropagation – Forward pass

Compute $\partial z/\partial w$ for all parameters



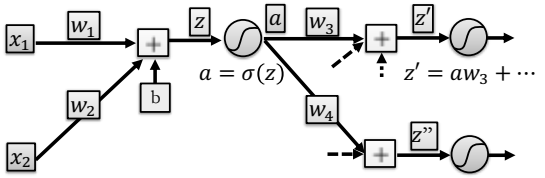
Backpropagation – Backward pass

Compute $\partial l/\partial z$ for all activation function inputs z



Backpropagation – Backward pass

Compute $\partial l/\partial z$ for all activation function inputs z



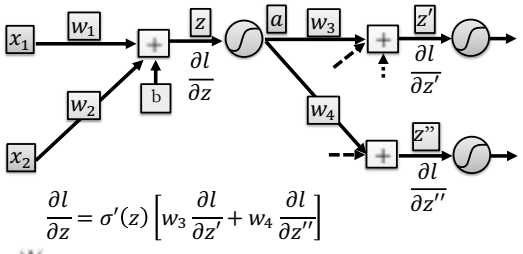
$\frac{\partial l}{\partial z} = \frac{\partial a}{\partial z}\frac{\partial l}{\partial a}$ $\quad \frac{\partial l}{\partial a} = \frac{\partial z'}{\partial a}\frac{\partial l}{\partial z'} + \frac{\partial z''}{\partial a}\frac{\partial l}{\partial z''}$ (Chain rule)

w_3 ? w_4 ?

Assumed it's known

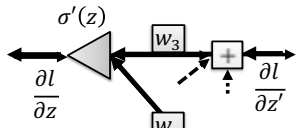
Backpropagation – Backward pass

Compute $\partial l/\partial z$ for all activation function inputs z



$$\frac{\partial l}{\partial z} = \sigma'(z)\left[w_3\frac{\partial l}{\partial z'} + w_4\frac{\partial l}{\partial z''}\right]$$

Backpropagation – Backward pass

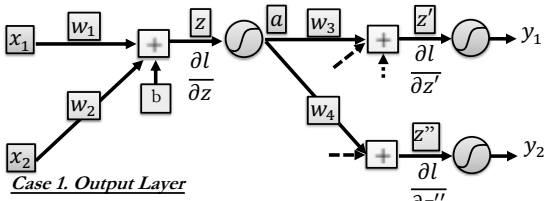


$\sigma'(z)$ is a constant because z is already determined in the forward pass.

$$\frac{\partial l}{\partial z} = \sigma'(z)\left[w_3\frac{\partial l}{\partial z'} + w_4\frac{\partial l}{\partial z''}\right]$$

Backpropagation – Backward pass

Compute $\partial l/\partial z$ for all activation function inputs z



Case 1. Output Layer

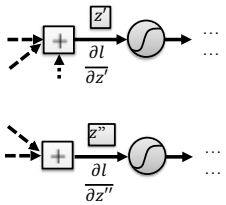
$$\frac{\partial l}{\partial z'} = \frac{\partial y_1}{\partial z'}\frac{\partial l}{\partial y_1} \qquad \frac{\partial l}{\partial z''} = \frac{\partial y_2}{\partial z''}\frac{\partial l}{\partial y_2}$$

Done!

Backpropagation – Backward pass

Compute $\partial l/\partial z$ for all activation function inputs z

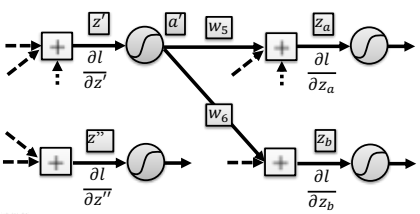
Case 2. Not Output Layer



Backpropagation – Backward pass

Compute $\partial l/\partial z$ for all activation function inputs z

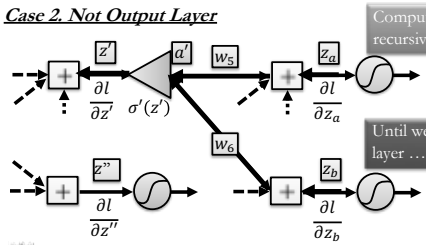
Case 2. Not Output Layer



Backpropagation – Backward pass

Compute $\partial l/\partial z$ for all activation function inputs z

Case 2. Not Output Layer

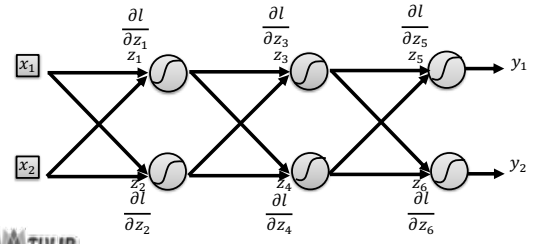


Compute $\partial l/\partial z$ recursively

Until we reach the output layer ……

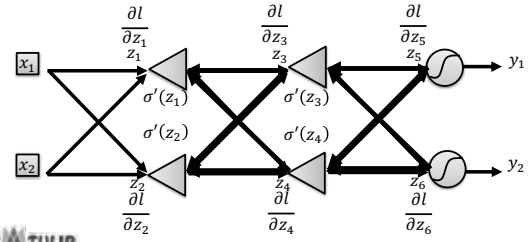
Backpropagation – Backward Pass

Compute $\partial l/\partial z$ for all activation function inputs z
Compute $\partial l/\partial z$ from the output layer



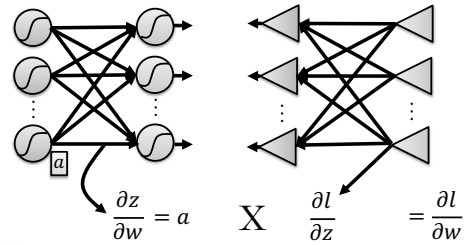
Backpropagation – Backward Pass

Compute $\partial l/\partial z$ for all activation function inputs z
Compute $\partial l/\partial z$ from the output layer



Backpropagation – Summary

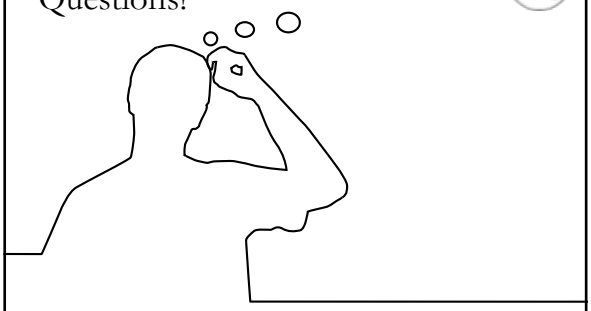
Forward Pass ***Backward Pass***



This Week's Readings

- Deep Learning: Theoretical Motivations (Yoshua Bengio)
  - http://videolectures.net/deeplearning2015_bengio_theoretical_motivations/
- Connections between physics and deep learning
  - https://www.youtube.com/watch?v=5MdSE-N0bxs
- Why Deep Learning Works: Perspectives from Theoretical Chemistry
  - https://www.youtube.com/watch?v=kIbKHIPbxiU

Questions?



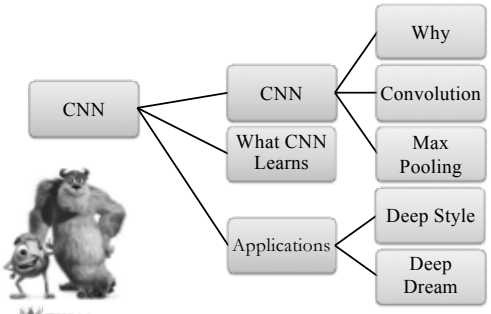
Lecture Notes on

Advanced Data Analytics

Module 05: Convolution Neural Network

Gang Li
School of Information Technology
Deakin University, VIC 3125, Australia

Road map



Convolution Neural Network

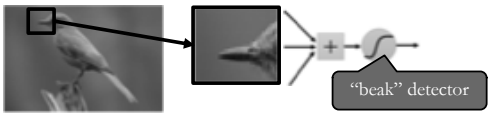
- Why CNN?
- Convolution
- Max Pooling

Why CNN for Image (1)

- Some patterns are much smaller than the whole image

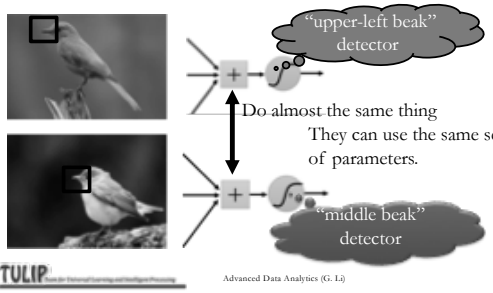
A neuron does not have to see the whole image to discover the pattern.

Connecting to small region with less parameters



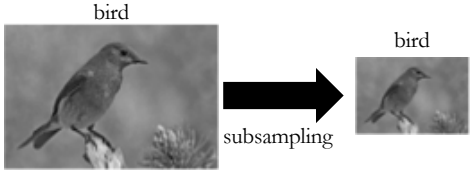
Why CNN for Image (2)

- The same patterns appear in different regions.



Why CNN for Image (3)

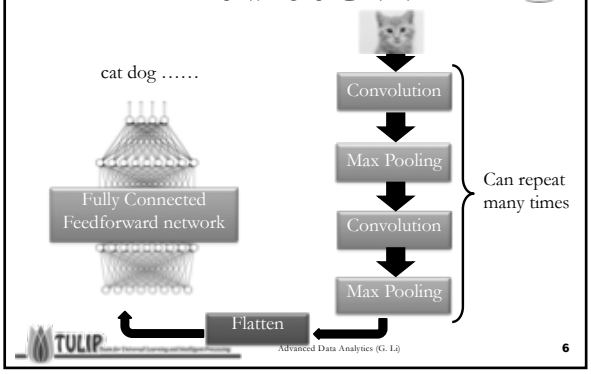
- Subsampling the pixels will not change the object



We can subsample the pixels to make image smaller

➡ Less parameters for the network to process the image

The whole CNN

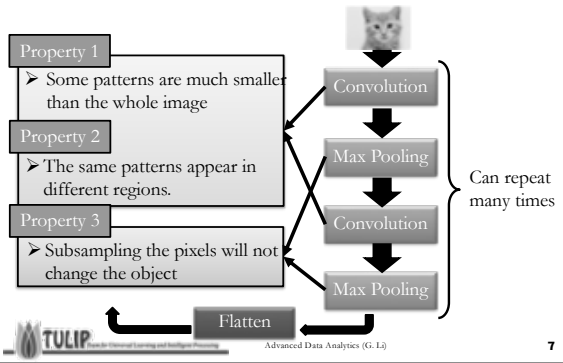


The whole CNN

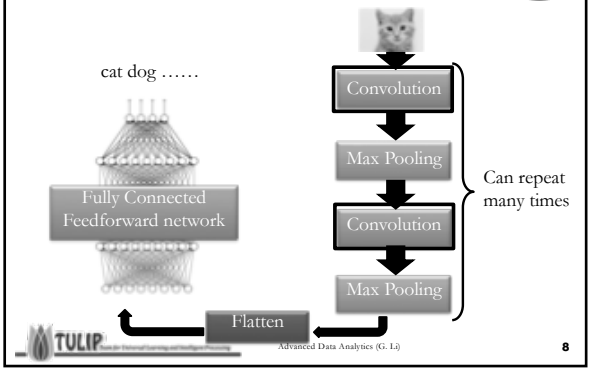
Property 1
- Some patterns are much smaller than the whole image

Property 2
- The same patterns appear in different regions.

Property 3
- Subsampling the pixels will not change the object



The whole CNN



CNN – Convolution

Those are the network parameters to be learned.

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1 Matrix

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2 Matrix

⋮ ⋮

Property 1 Each filter detects a small pattern (3 x 3).

CNN – Convolution

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

3 -1

CNN – Convolution

If stride=2

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

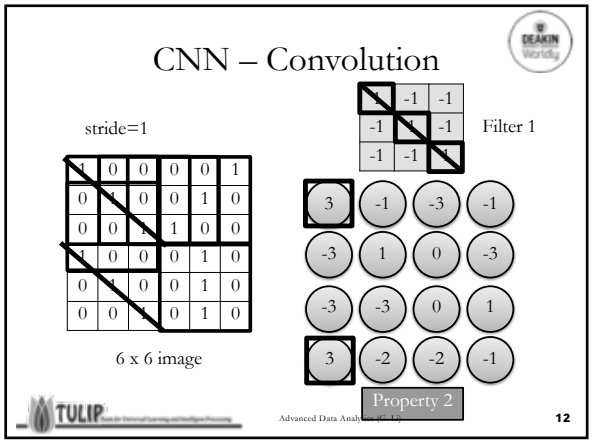
6 x 6 image

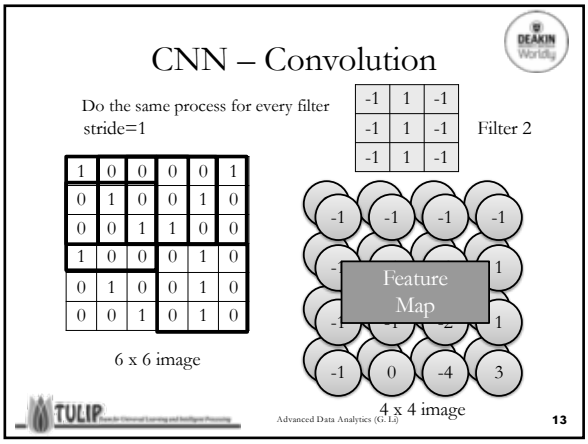
1	-1	-1
-1	1	-1
-1	-1	1

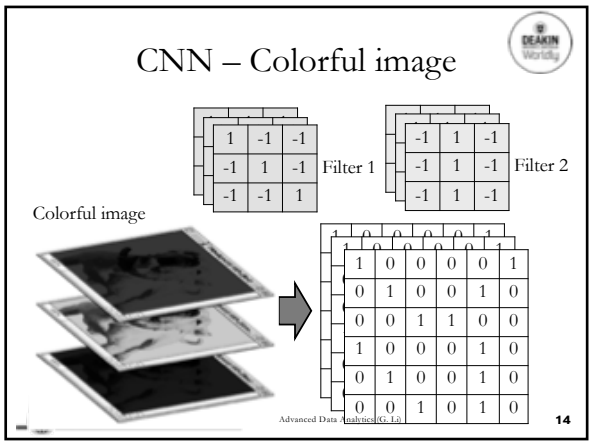
Filter 1

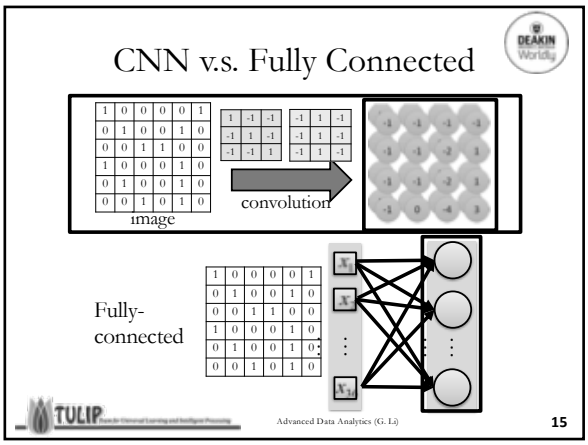
3 -3

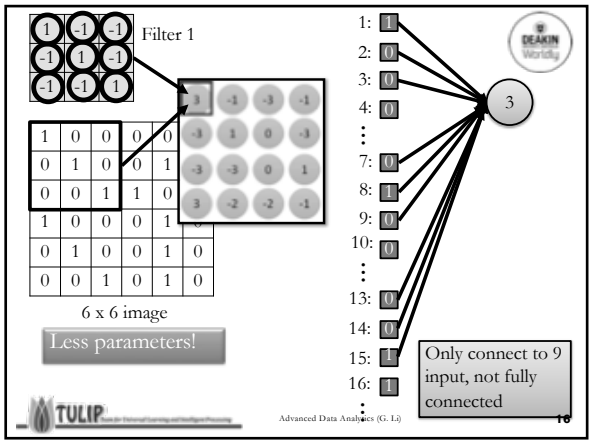
We set stride=1 below

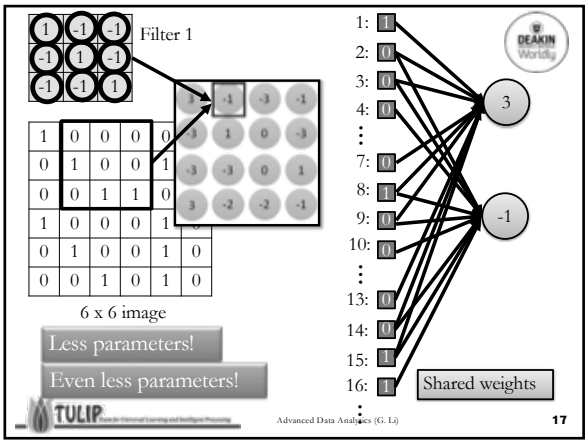

CNN – Convolution
stride=1
Filter 1
6 x 6 image
Property 2
Advanced Data Analytics (G. Li)
12


CNN – Convolution
Do the same process for every filter
stride=1
Filter 2
6 x 6 image
Feature Map
4 x 4 image
Advanced Data Analytics (G. Li)
13

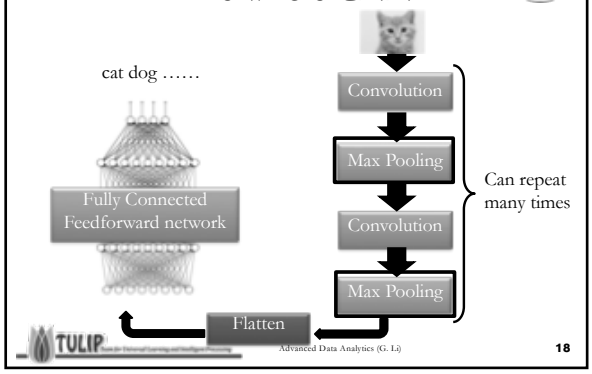

CNN – Colorful image
Filter 1
Filter 2
Colorful image
Advanced Data Analytics (G. Li)
14


CNN v.s. Fully Connected
image
convolution
Fully-connected
Advanced Data Analytics (G. Li)
15

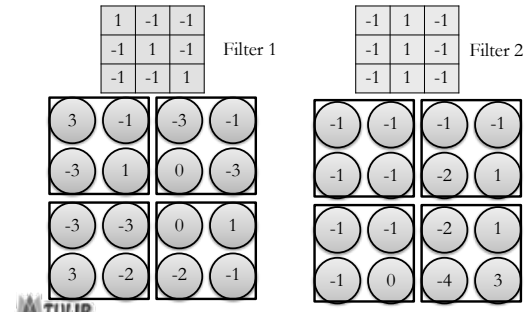

Filter 1
6 x 6 image
Less parameters!
Only connect to 9 input, not fully connected
Advanced Data Analytics (G. Li)
16


Filter 1
6 x 6 image
Less parameters!
Even less parameters!
Shared weights
Advanced Data Analytics (G. Li)
17

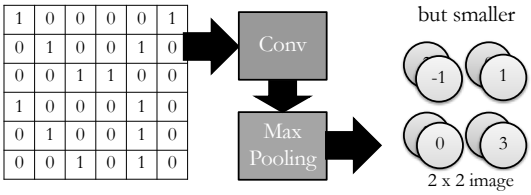
The whole CNN



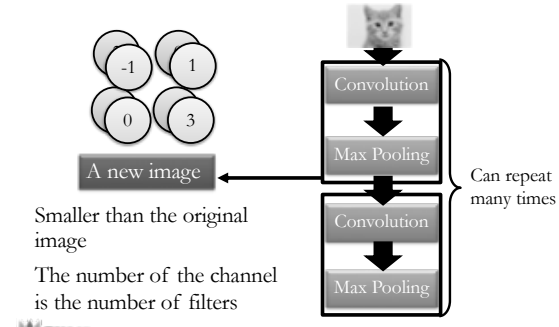
CNN – Max Pooling



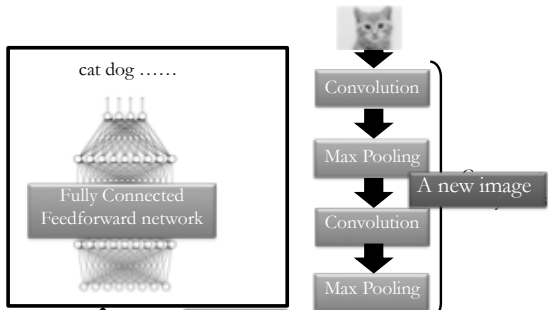
CNN – Max Pooling



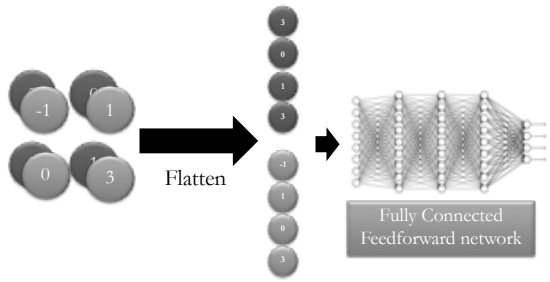
The whole CNN



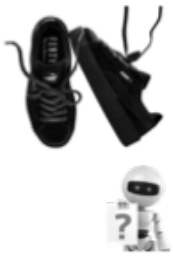
The whole CNN



Flatten



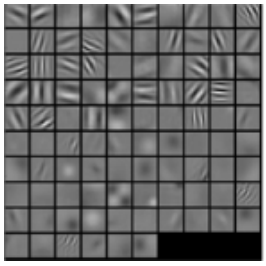
What CNN Learns?



- What CNN Learns?

First Convolution Layer

- Typical-looking filters on the trained first layer



11 x 11 (AlexNet)

http://cs231n.github.io/understanding-cnn/

How about higher layers?

- Which images make a specific neuron activate

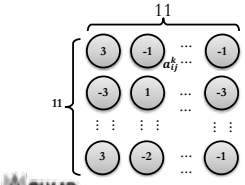
- Ross Girshick, Jeff Donahue, Trevor Darrell, Jitendra Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR, 2014

What does CNN learn?

The output of the k-th filter is a 11 x 11 matrix.

Degree of the activation of the k-th filter: $a^k = \sum_{i=1}^{11}\sum_{j=1}^{11} a_{ij}^k$

$x^* = arg\max_x a^k$ (gradient ascent)



$\frac{\partial a^k}{\partial x_{ij}}$

x input

25 3x3 filters

Convolution

Max Pooling

50 3x3 filters

Convolution

50 x 11 x 11

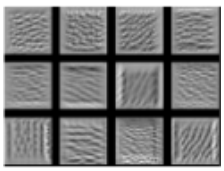
Max Pooling

What does CNN learn?

The output of the k-th filter is a 11 x 11 matrix.

Degree of the activation of the k-th filter: $a^k = \sum_{i=1}^{11}\sum_{j=1}^{11} a_{ij}^k$

$x^* = arg\max_x a^k$ (gradient ascent)



$\frac{\partial a^k}{\partial x_{ij}}$

x input

25 3x3 filters

Convolution

Max Pooling

50 3x3 filters

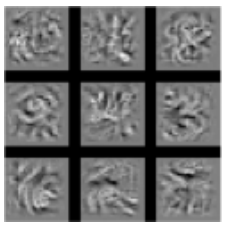
Convolution

50 x 11 x 11

Max Pooling

What does CNN learn?

Find an image maximizing the output of neuron: $x^* = arg\max_x a^j$



Each figure corresponds to a neuron

input

Convolution

Max Pooling

Convolution

Max Pooling

flatten

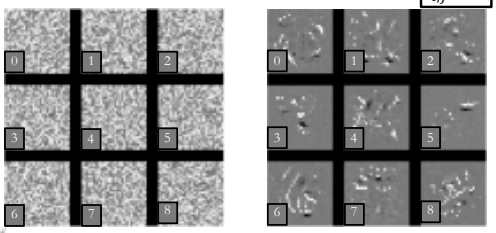
a_j

What does CNN learn?

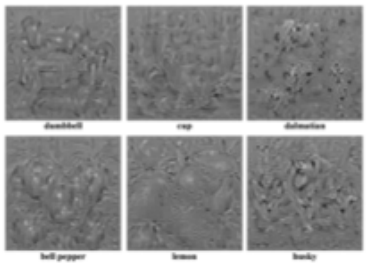
$$x^* = arg\max_x y^i$$

$$x^* = arg\max_x \left(y^i - \boxed{\sum_{i,j} |x_{ij}|} \right)$$

Over all pixel values



What does CNN learn?

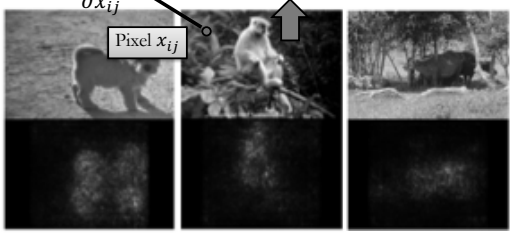


- Karen Simonyan, Andrea Vedaldi, Andrew Zisserman, "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps", ICLR, 2014

What does CNN learn?

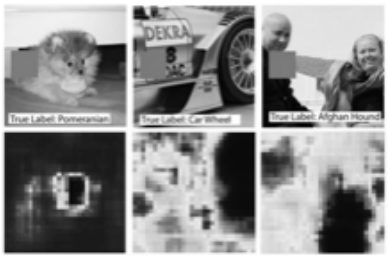
$|\frac{\partial y_k}{\partial x_{ij}}|$ y_k: the predicted class of the model

Pixel x_{ij}



- Karen Simonyan, Andrea Vedaldi, Andrew Zisserman, "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps", ICLR, 2014

What does CNN learn?



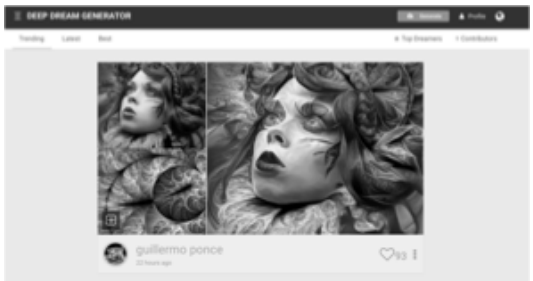
- Reference: Zeiler, M. D., & Fergus, R. (2014). Visualizing and understanding convolutional networks. In Computer Vision–ECCV 2014 (pp. 818-833)

Applications

- Deep Dream
- Deep Style

Deep Dream Generator

https://deepdreamgenerator.com

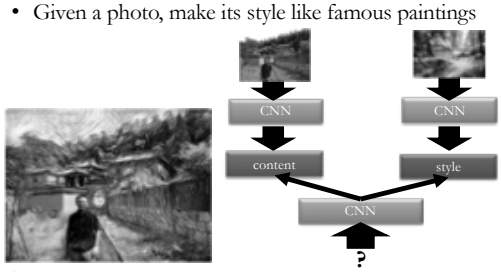


Deep Style

- Given a photo, make its style like famous paintings

Deep Style

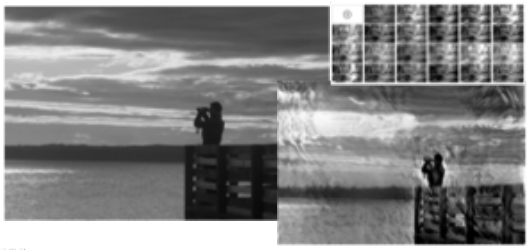
- Given a photo, make its style like famous paintings



- A Neural Algorithm of Artistic Style (https://arxiv.org/abs/1508.06576)

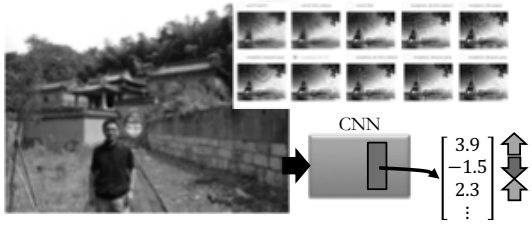
Deep Style

- Given a photo, make its style like famous paintings



Deep Dream

- Given a photo, machine adds what it sees ……

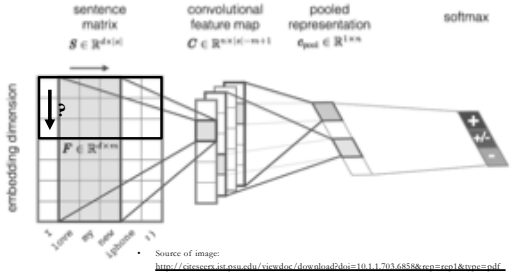


- CNN exaggerates what it sees

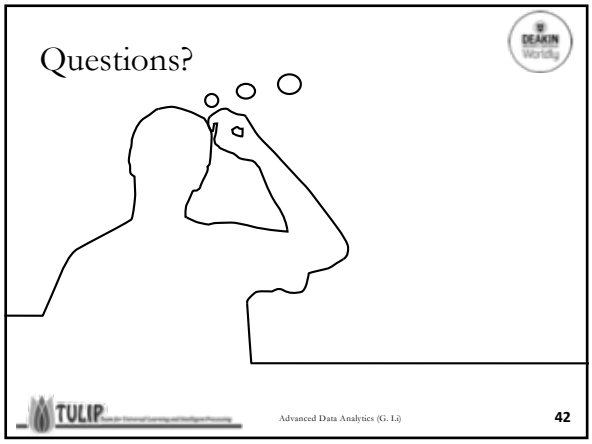
Deep Dream

- Given a photo, machine adds what it sees ……

More Application: Text



- Source of image: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.703.6858&rep=rep1&type=pdf


Questions?
TULIP
Advanced Data Analytics (G. Li)
42

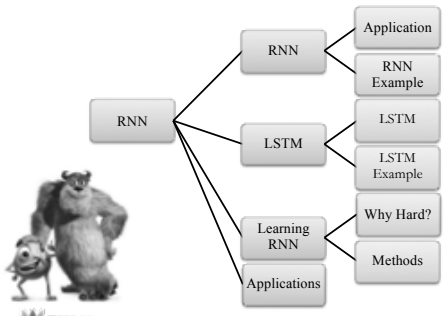
Lecture Notes on

Advanced Data Analytics

Module 05: Recurrent Neural Network (RNN)

Gang Li
School of Information Technology
Deakin University, VIC 3125, Australia

Road map

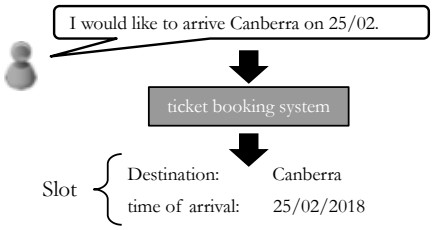


Recurrent Neural Network

- Why RNN?
- RNN
- RNN Example

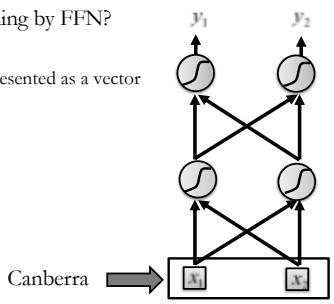
Application: Booking System

- Slot Filling



Application: Booking System

- Solving slot filling by FFN?
  - Input:
    - Word represented as a vector



One Hot Encoding

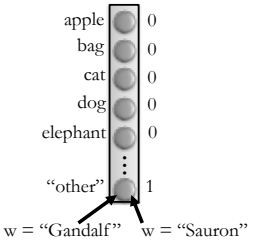
- The vector is lexicon size.
- Each dimension corresponds to a word in the lexicon
- The dimension for the word is 1, and others are 0

lexicon = {apple, bag, cat, dog, elephant}

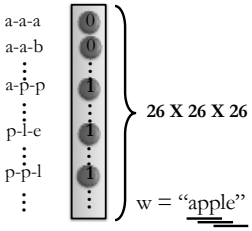
apple = [1 0 0 0 0]
bag = [0 1 0 0 0]
cat = [0 0 1 0 0]
dog = [0 0 0 1 0]
elephant = [0 0 0 0 1]

Beyond One Hot Encoding

Dimension for "Other"

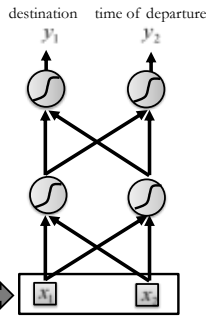


Word hashing

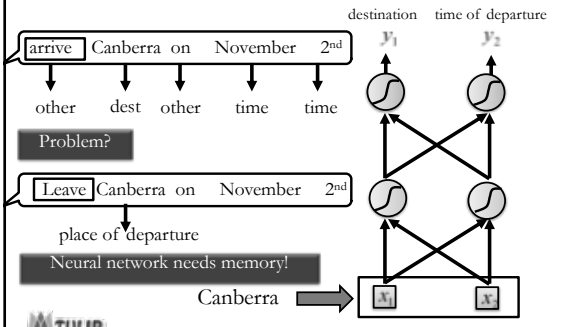


Application: Booking System

- Solving slot filling by FFN?
  - Input:
    - Word represented as a vector
  - Output:
    - Probability distribution that the input word belonging to the slots

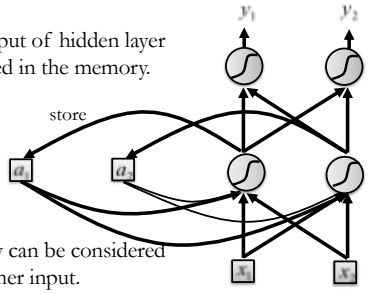


Example Application



Recurrent Neural Network (RNN)

The output of hidden layer are stored in the memory.

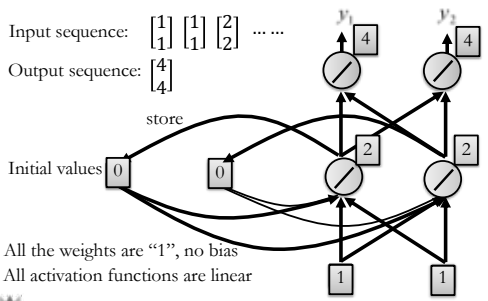


Memory can be considered as another input.

Example

Input sequence: $\begin{bmatrix}1\\1\end{bmatrix}\begin{bmatrix}1\\1\end{bmatrix}\begin{bmatrix}2\\2\end{bmatrix}\cdots\cdots$

Output sequence: $\begin{bmatrix}4\\4\end{bmatrix}$

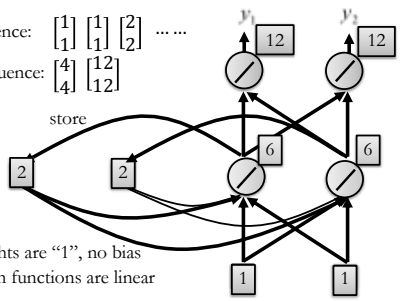


All the weights are "1", no bias
All activation functions are linear

Example

Input sequence: $\begin{bmatrix}1\\1\end{bmatrix}\begin{bmatrix}1\\1\end{bmatrix}\begin{bmatrix}2\\2\end{bmatrix}\cdots\cdots$

Output sequence: $\begin{bmatrix}4\\4\end{bmatrix}\begin{bmatrix}12\\12\end{bmatrix}$

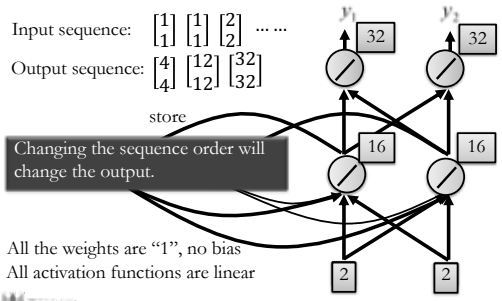


All the weights are "1", no bias
All activation functions are linear

Example

Input sequence: $\begin{bmatrix}1\\1\end{bmatrix}$ $\begin{bmatrix}1\\1\end{bmatrix}$ $\begin{bmatrix}2\\2\end{bmatrix}$ ……

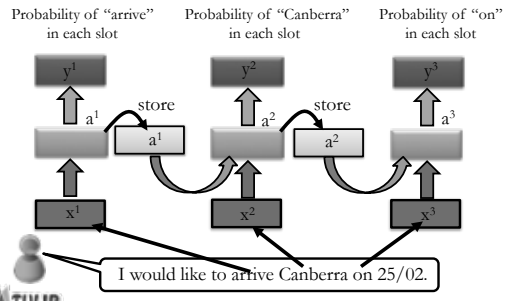
Output sequence: $\begin{bmatrix}4\\4\end{bmatrix}$ $\begin{bmatrix}12\\12\end{bmatrix}$ $\begin{bmatrix}32\\32\end{bmatrix}$



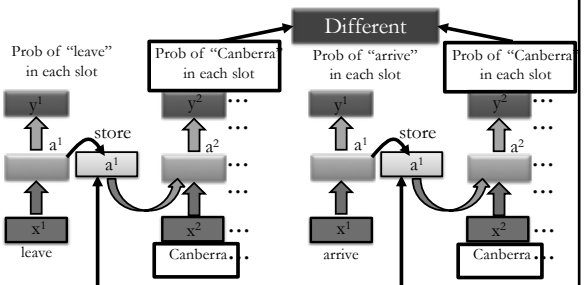
Changing the sequence order will change the output.

All the weights are "1", no bias
All activation functions are linear

RNN

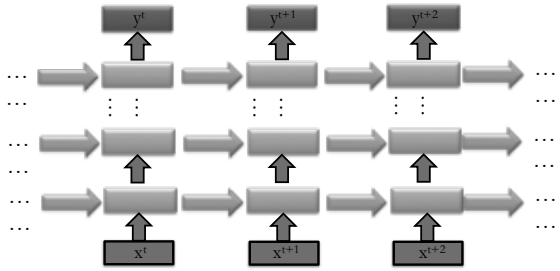


RNN

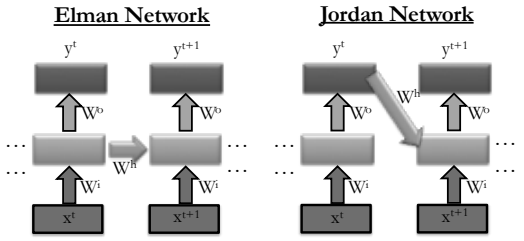


The values stored in the memory is different.

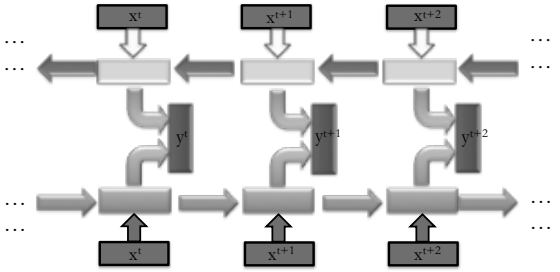
Of course it can be deep …



Elman Network & Jordan Network



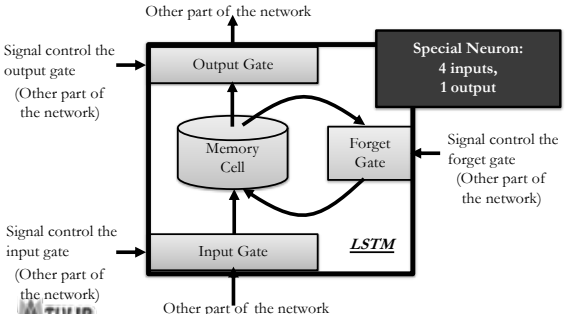
Bidirectional RNN



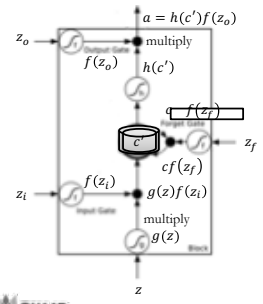
LSTM

- Why LSTM?
- LSTM
- LSTM and FFN

Long Short-Term Memory (LSTM)



Long Short-Term Memory (LSTM)

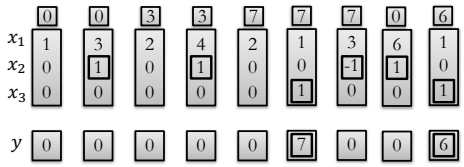


Activation function f is usually a sigmoid function

- Between 0 and 1
- Mimic open and close gate

$$c' = g(z)f(z_i) + cf(z_f)$$

LSTM - Example

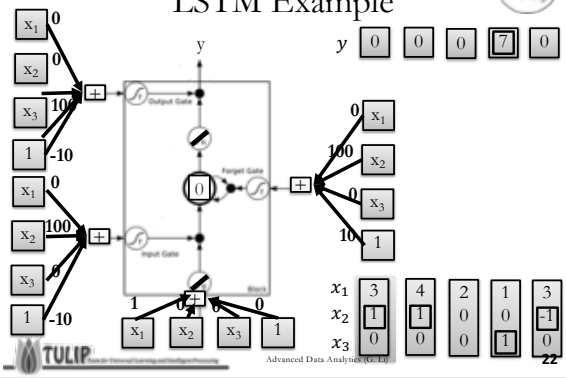


When $x_2 = 1$, add the numbers of x_1 into the memory

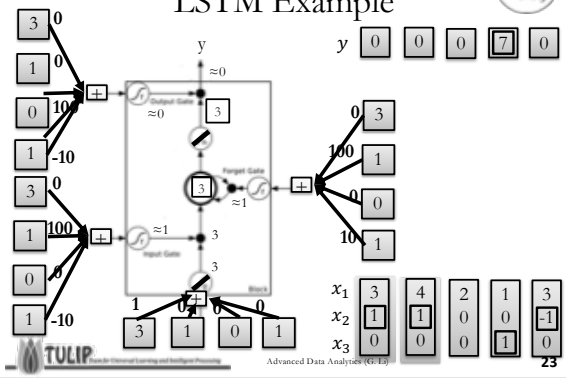
When $x_2 = -1$, reset the memory

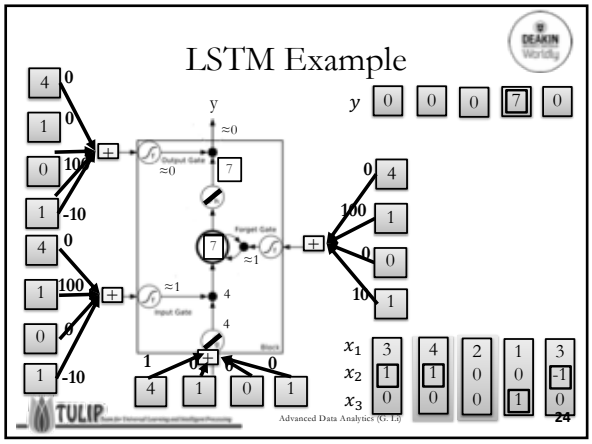
When $x_3 = 1$, output the number in the memory.

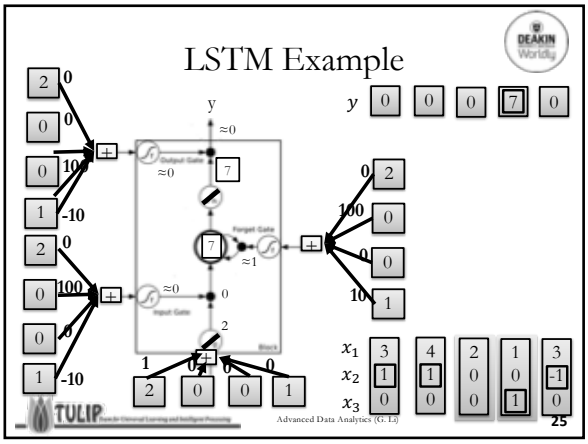
LSTM Example

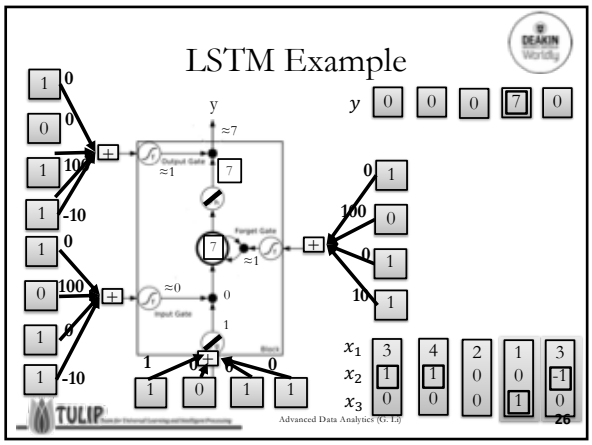


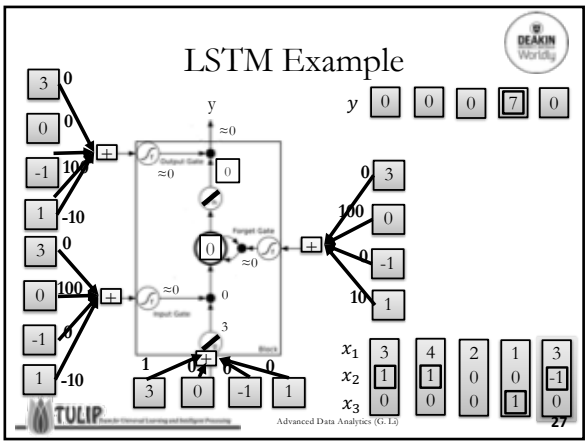
LSTM Example

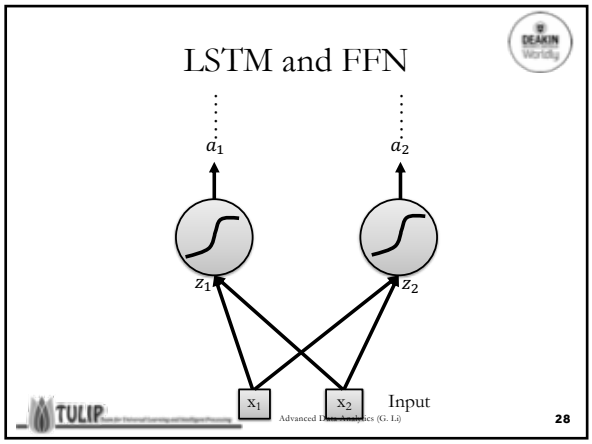


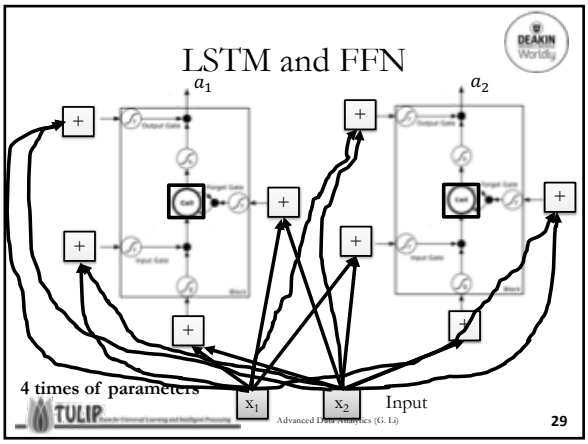

LSTM Example
y
x_1
x_2
x_3
Advanced Data Analytics (G. Li)
24


LSTM Example
y
x_1
x_2
x_3
Advanced Data Analytics (G. Li)
25

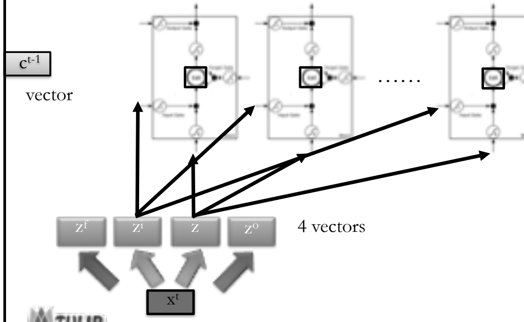

LSTM Example
y
x_1
x_2
x_3
Advanced Data Analytics (G. Li)
26


LSTM Example
y
x_1
x_2
x_3
Advanced Data Analytics (G. Li)
27

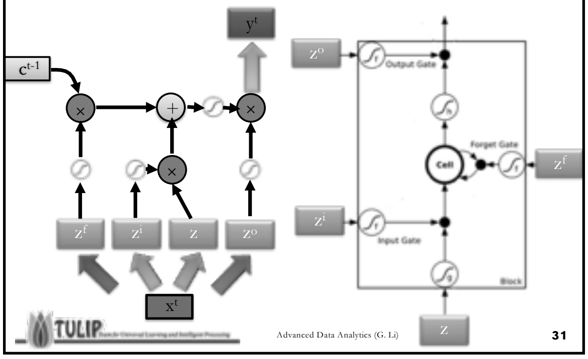

LSTM and FFN
a_1
a_2
z_1
z_2
x_1
x_2
Input
Advanced Data Analytics (G. Li)
28


LSTM and FFN
a_1
a_2
4 times of parameters
x_1
x_2
Input
Advanced Data Analytics (G. Li)
29

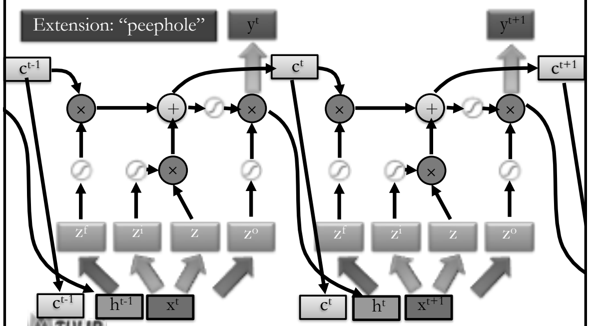
LSTM and FFN



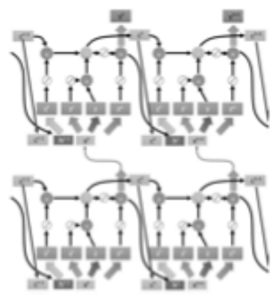
LSTM and FFN



LSTM



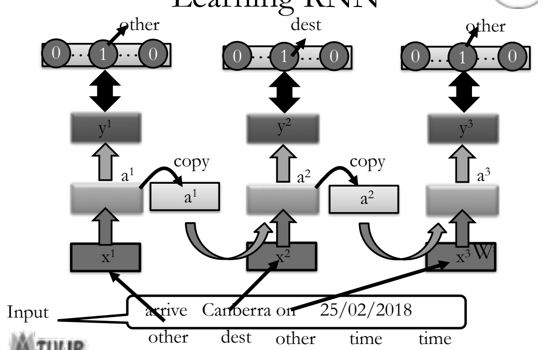
Multiple-layer LSTM



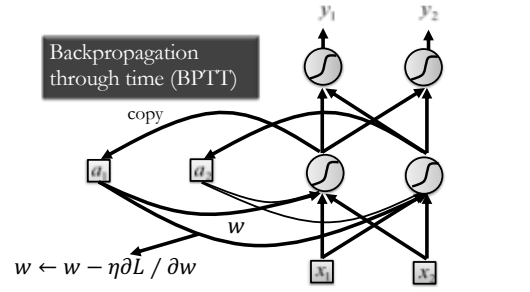
Learning RNN

- Why Difficult?
- Helpful Techniques

Learning RNN



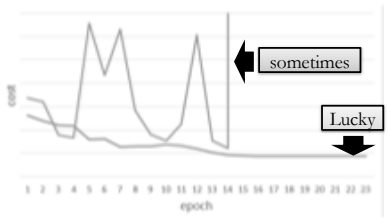
Learning RNN



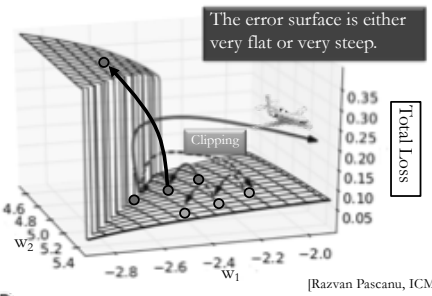
Learning RNN

- RNN-based network is not always easy to learn

Real experiments on Language modeling



Learning RNN



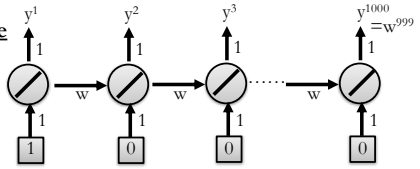
[Razvan Pascanu, ICML'13]

Why Difficult?

$w = 1 \Rightarrow y^{1000} = 1$
$w = 1.01 \Rightarrow y^{1000} \approx 20000$
Large $\partial L/\partial w$ → Small Learning rate?

$w = 0.99 \Rightarrow y^{1000} \approx 0$
$w = 0.01 \Rightarrow y^{1000} \approx 0$
small $\partial L/\partial w$ → Large Learning rate?

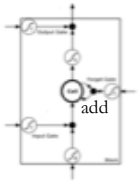
Toy Example



Helpful Techniques

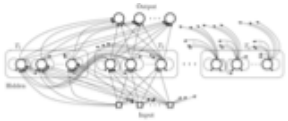
- Long Short-term Memory (LSTM)
  - Can deal with gradient vanishing (not gradient explode)
  - Memory and input are **added**
    - The influence never disappears unless forget gate is closed

No Gradient vanishing
(If forget gate is opened.)



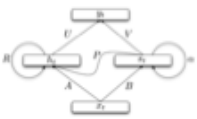
Helpful Techniques

Clockwise RNN



[Jan Koutnik, JMLR'14]

Structurally Constrained Recurrent Network (SCRN)



[Tomas Mikolov, ICLR'15]

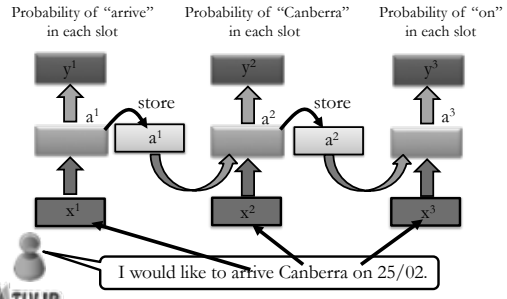
Vanilla RNN Initialized with Identity matrix + ReLU activation function [Quoc V. Le, arXiv'15]

- Outperform or be comparable with LSTM in 4 different tasks

RNN/LSTM Applications

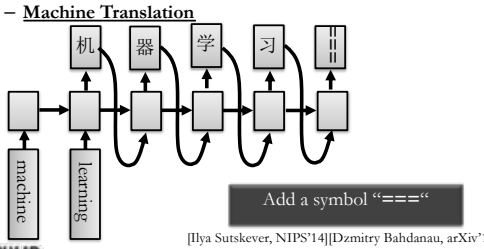
- Applications
- Attention-based Model
- Visual Question Answering

Applications: Many to Many



Many to Many (No Limitation)

- Both input and output are both sequences ***with different lengths***. → ***Sequence to sequence learning***
  - **Machine Translation**

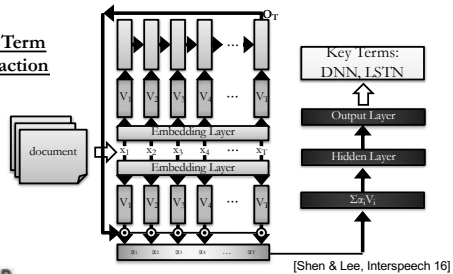


[Ilya Sutskever, NIPS'14][Dzmitry Bahdanau, arXiv'15]

Applications: Many to One

- Input is a vector sequence, but output is only one vector

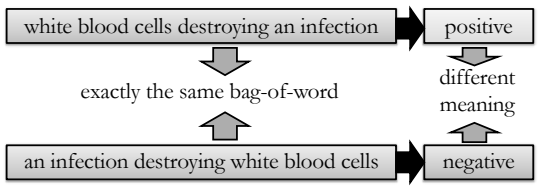
Key Term Extraction



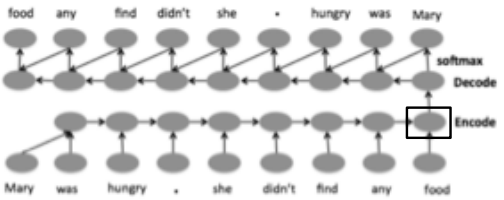
[Shen & Lee, Interspeech 16]

Sequence-to-sequence Auto-encoder - Text

- To understand the meaning of a word sequence, the order of the words can not be ignored.

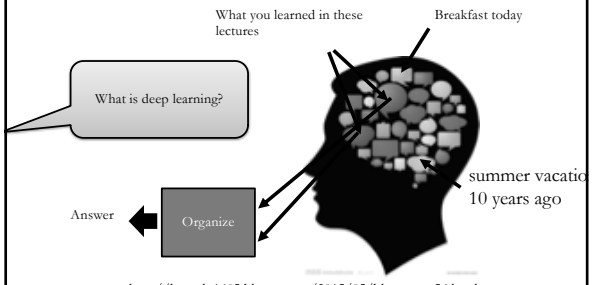


Sequence-to-sequence Auto-encoder - Text



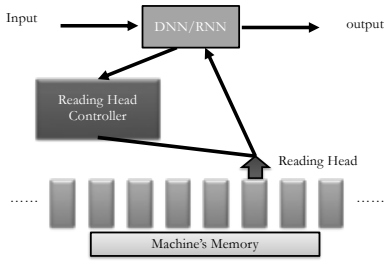
- Li, Jiwei, Minh-Thang Luong, and Dan Jurafsky. "A hierarchical neural autoencoder for paragraphs and documents." *arXiv preprint arXiv:1506.01057*(2015).

Attention-based Model

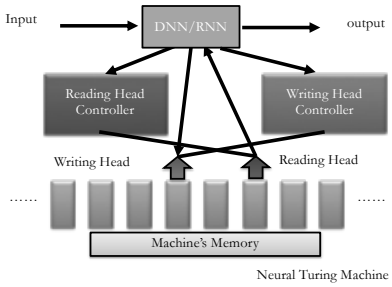


http://henrylo1605.blogspot.tw/2015/05/blog-post_56.html

Attention-based Model

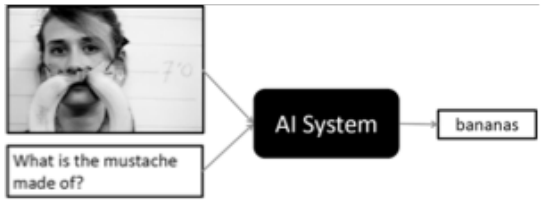


Attention-based Model v2



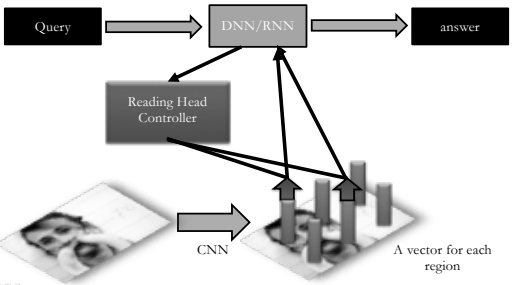
Neural Turing Machine

Visual Question Answering



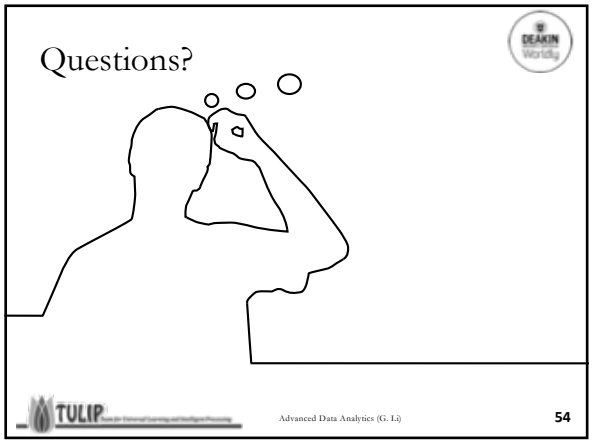
source: http://visualqa.org

Visual Question Answering



This Session's Readings

- The Unreasonable Effectiveness of Recurrent Neural Networks
  - http://karpathy.github.io/2015/05/21/rnn-effectiveness/
- Understanding LSTM Networks
  - http://colah.github.io/posts/2015-08-Understanding-LSTMs/


Questions?
TULIP
Advanced Data Analytics (G. Li)
54

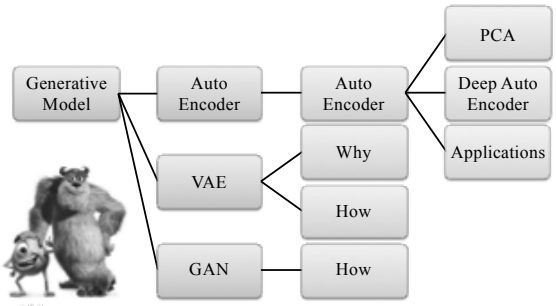
Lecture Notes on

Advanced Data Analytics

Module 05: Generative Model

Gang Li
School of Information Technology
Deakin University, VIC 3125, Australia

Road map



Generative Models

- Unsupervised Learning
- Generation

Unsupervised Learning

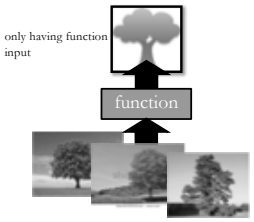
- "We expect unsupervised learning to become far more important in the longer term. Human and animal learning is largely unsupervised: we discover the structure of the world by observing it, not by being told the name of every object."

  LeCun, Bengio, Hinton, Nature 2015

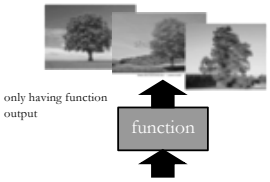
- As I've said in previous statements: most of human and animal learning is unsupervised learning. If intelligence was a cake, unsupervised learning would be the cake, supervised learning would be the icing on the cake, and reinforcement learning would be the cherry on the cake. We know how to make the icing and the cherry, but we don't know how to make the cake.
  - Yann LeCun, March 14, 2016 (Facebook)

Unsupervised Learning

Dimension Reduction

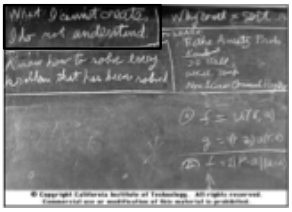


Generation



Generation

- Generative Models:
  - https://openai.com/blog/generative-models/



What I cannot create, I do not understand.

Richard Feynman

- https://www.quora.com/What-did-Richard-Feynman-mean-when-he-said-What-I-cannot-create-I-do-not-understand

Generation

Now  v.s. 

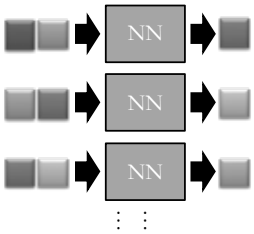
In the future

Machine draws a cat

Component-by-component

- Image generation

E.g. 3 x 3 images

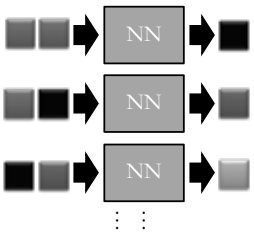


Can be trained just with a large collection of images without any annotation

Component-by-component

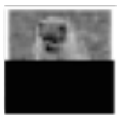
- Image generation

E.g. 3 x 3 images

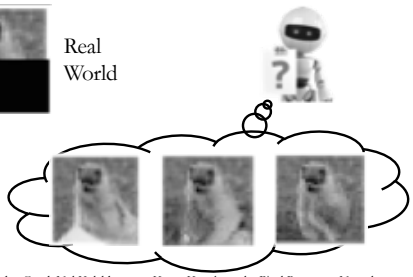


Can be trained just with a large collection of images without any annotation

PixelRNN

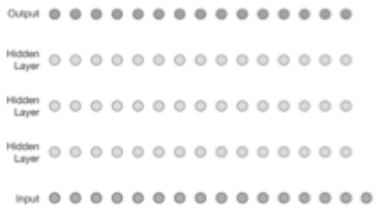


Real World



- Aaron van den Oord, Nal Kalchbrenner, Koray Kavukcuoglu, Pixel Recurrent Neural Networks, arXiv preprint, 2016

More than images ……

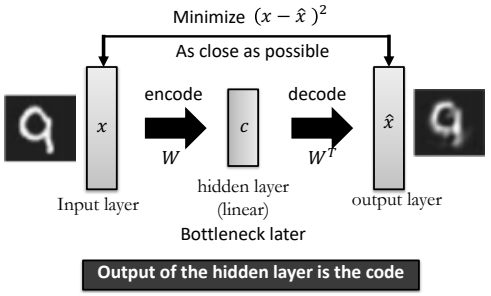


- Audio: Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, Koray Kavukcuoglu, WaveNet: A Generative Model for Raw Audio, arXiv preprint, 2016
- Video: Nal Kalchbrenner, Aaron van den Oord, Karen Simonyan, Ivo Danihelka, Oriol Vinyals, Alex Graves, Koray Kavukcuoglu, Video Pixel Networks , arXiv preprint, 2016

Auto-Encoder

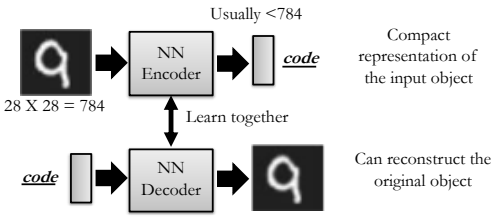
- Auto-Encoder
- Deep Auto-Encoder
- Auto-Encoder Applications

PCA and FFN



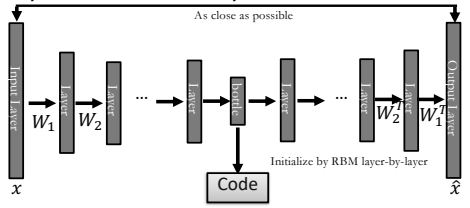
Output of the hidden layer is the code

Auto-encoder



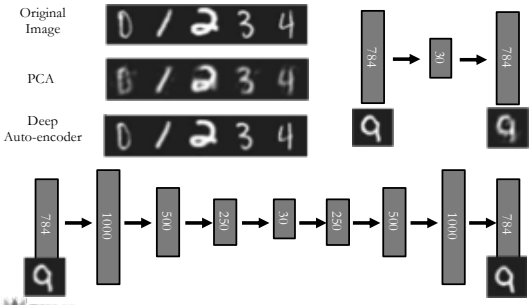
Deep Auto-encoder

- Auto-encoder can be deep
  - symmetric is not necessary.

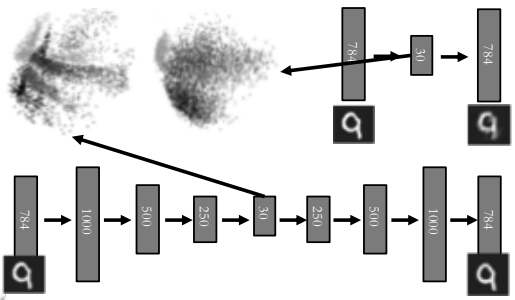


- Reference: Hinton, Geoffrey E., and Ruslan R. Salakhutdinov. "Reducing the dimensionality of data with neural networks." *Science* 313.5786 (2006): 504-507

Deep Auto-encoder

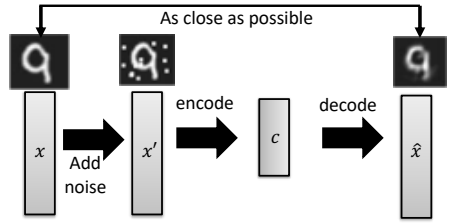


Deep Auto-encoder



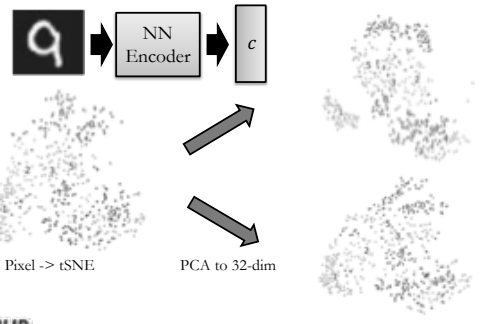
Deep Auto-encoder

- De-noising auto-encoder



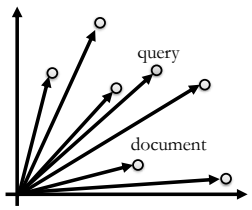
- Vincent, Pascal, et al. "Extracting and composing robust features with denoising autoencoders." *ICML*, 2008.

Deep Auto-encoder



Auto-encoder – Text Retrieval

Vector Space Model



Bag-of-word

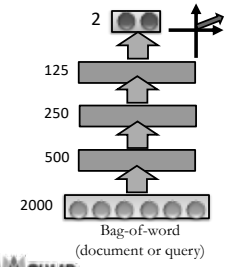
word string: "This is an apple"

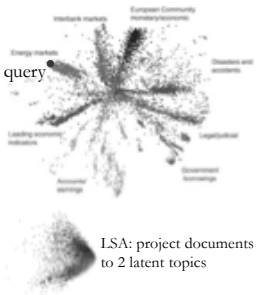
this	1
is	1
a	0
an	1
apple	1
pen	0
...	

Semantics are not considered.

Auto-encoder – Text Retrieval

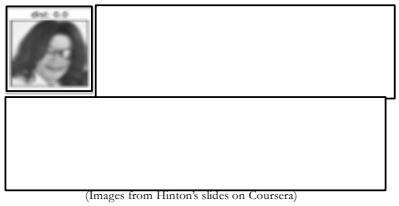
The documents talking about the same thing will have close code.





Auto-encoder: Similar Image Search

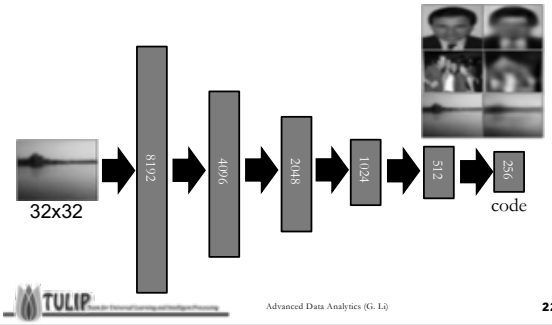
Retrieved using Euclidean distance in pixel intensity space



(Images from Hinton's slides on Coursera)

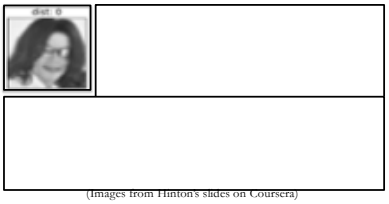
- Reference: Krizhevsky, Alex, and Geoffrey E. Hinton. "Using very deep autoencoders for content-based image retrieval." *ESANN*. 2011.

Auto-encoder: Similar Image Search



Auto-encoder: Similar Image Search

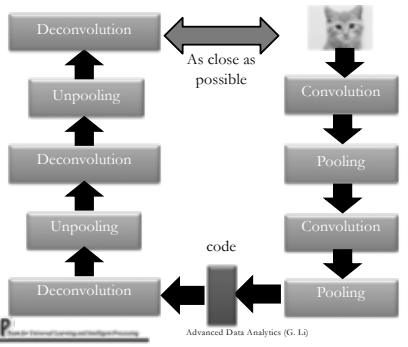
- Retrieved using 256 codes using Euclidean distance



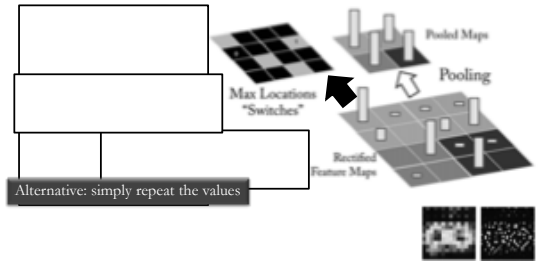
(Images from Hinton's slides on Coursera)

- Reference: Krizhevsky, Alex, and Geoffrey E. Hinton. "Using very deep autoencoders for content-based image retrieval." *ESANN*. 2011.

Auto-encoder for CNN

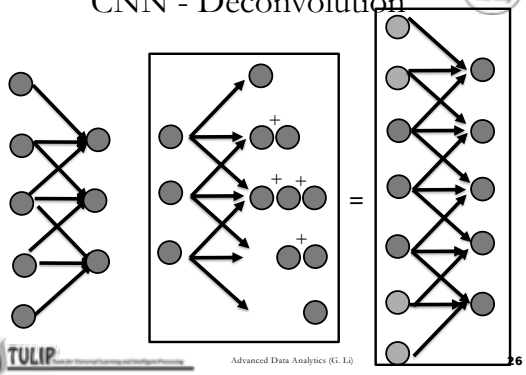


CNN -Unpooling



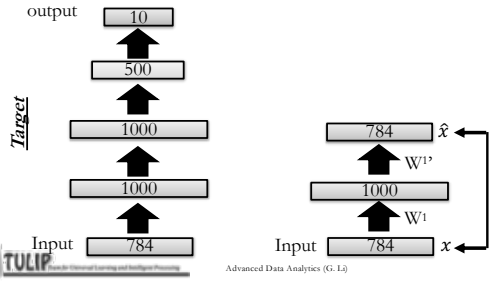
Source of image :
- https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/image_segmentation.html

CNN - Deconvolution



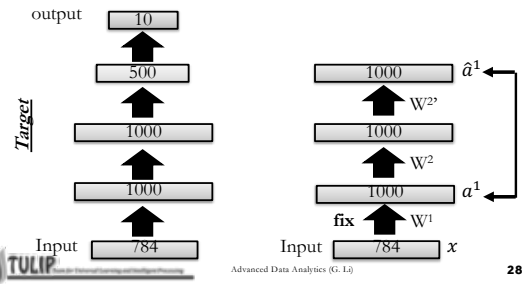
Auto-encoder: Pre-training DNN

- Greedy Layer-wise Pre-training *again*



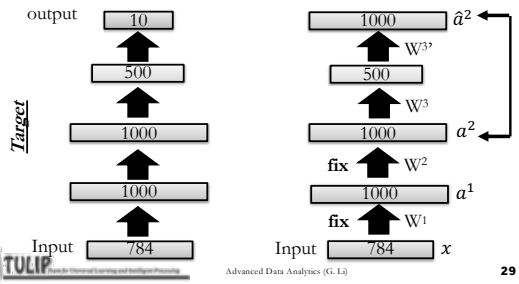
Auto-encoder: Pre-training DNN

- Greedy Layer-wise Pre-training *again*



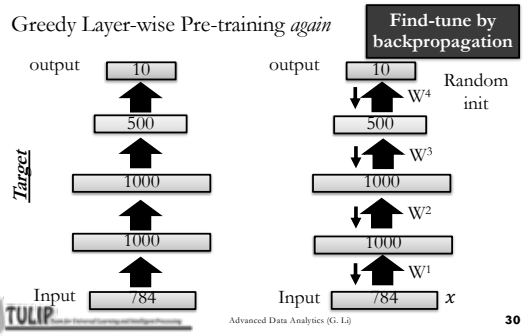
Auto-encoder: Pre-training DNN

- Greedy Layer-wise Pre-training *again*

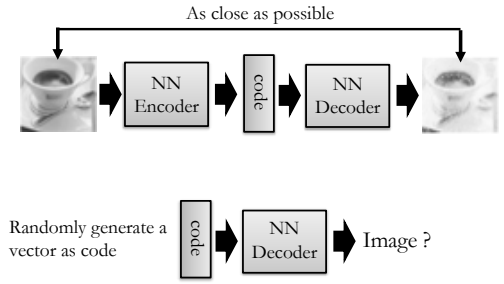


Auto-encoder: Pre-training DNN

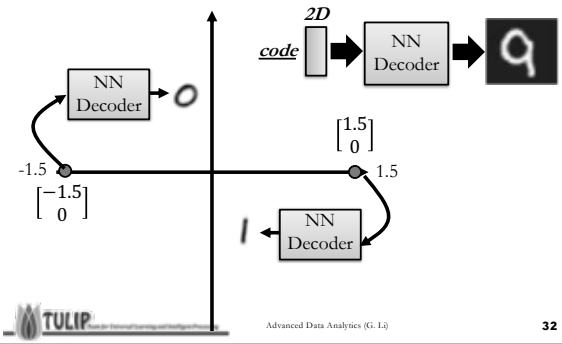
- Greedy Layer-wise Pre-training *again*



Auto-encoder: Generating New Things

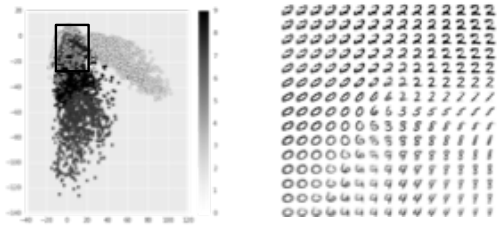


Auto-encoder: Generating New Things



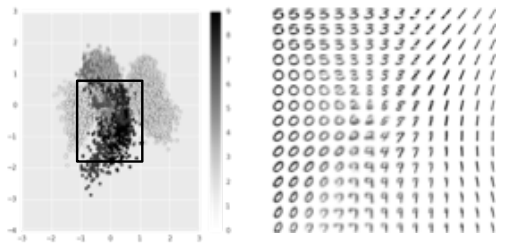
Auto-encoder: Generating New Things

- Can we use decoder to generate something?



Auto-encoder: Generating New Things

- Can we use decoder to generate something?

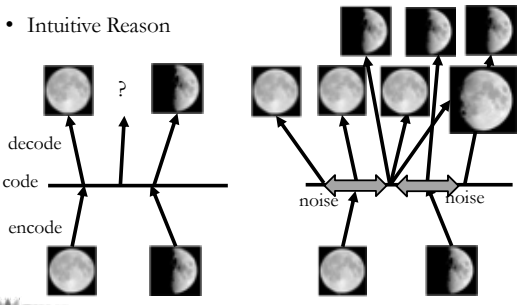


Variational Auto-Encoder

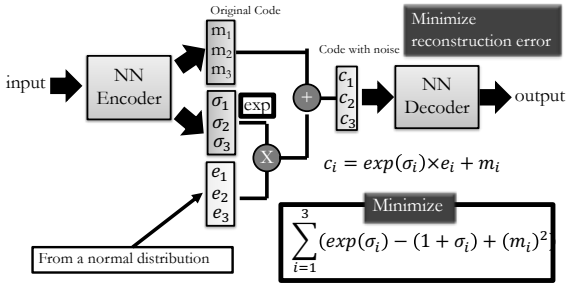
- Why VAE
- How VAE

Why VAE?

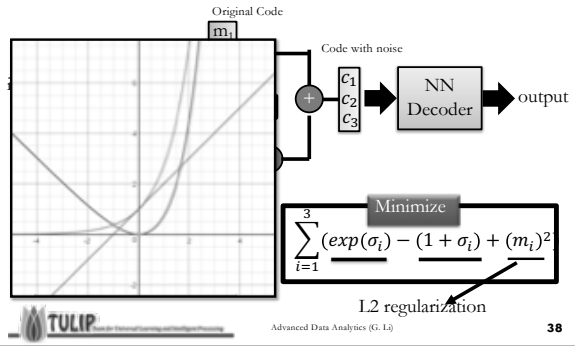
- Intuitive Reason



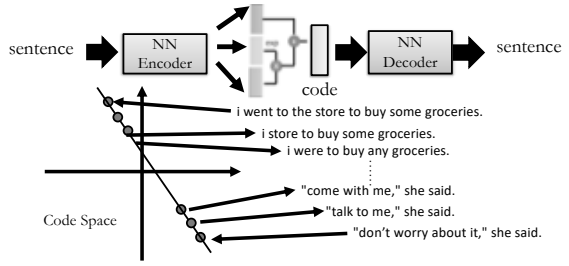
Variational Auto-Encoder



Variational Auto-Encoder



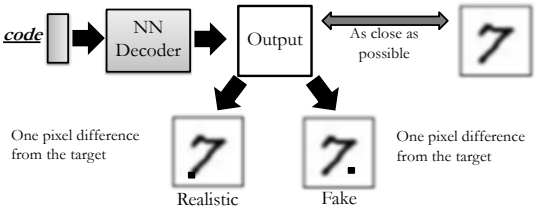
VAE Writing



- Ref: http://www.wired.co.uk/article/google-artificial-intelligence-poetry
- Samuel R. Bowman, Luke Vilnis, Oriol Vinyals, Andrew M. Dai, Rafal Jozefowicz, Samy Bengio, Generating Sentences from a Continuous Space, arXiv prepring, 2015

Problems of VAE

- It does not really try to simulate real images



- VAE may just memorize the existing images, instead of generating new images

Generative Adversarial Network

- Why GAN
- How GAN

Cifar-10

- Which one is machine-generated?

Ref: https://openai.com/blog/generative-models/

Yann LeCun's comment

What are some recent and potentially upcoming breakthroughs in deep learning?

Yann LeCun, Director of AI Research at Facebook and Professor at NYU
Written Jul 29 · Upvoted by Joaquin Quiñonero Candela, Director Applied Machine Learning at Facebook and Huang Xiao

Adversarial training is the coolest thing since sliced bread.

I've listed a bunch of relevant papers in a previous answer.

Expect more impressive results with this technique in the coming years.

What's missing at the moment is a good understanding of it so we can make it work reliably. It's very finicky. Sort of like ConvNet were in the 1990s, when I had the reputation of being the only person who could make them work (which wasn't true).

https://www.quora.com/What-are-some-recent-and-potentially-upcoming-breakthroughs-in-unsupervised-learning

Yann LeCun's comment

What are some recent and potentially upcoming breakthroughs in deep learning?

Yann LeCun, Director of AI Research at Facebook and Professor at NYU
Written Jul 29 · Upvoted by Joaquin Quiñonero Candela, Director Applied Machine Learning at Facebook and Nikhil Garg, I lead a team of Quora engineers working on ML/NLP problems

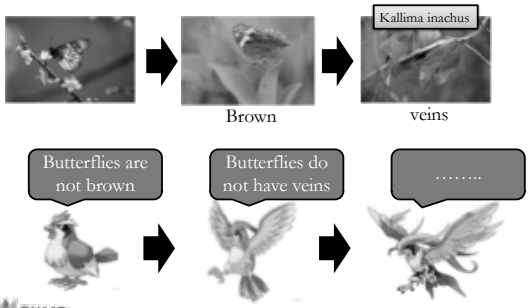
……

The most important one, in my opinion, is adversarial training (also called GAN for Generative Adversarial Networks). This is an idea that was originally proposed by Ian Goodfellow when he was a student with Yoshua Bengio at the University of Montreal (he since moved to Google Brain and recently to OpenAI).

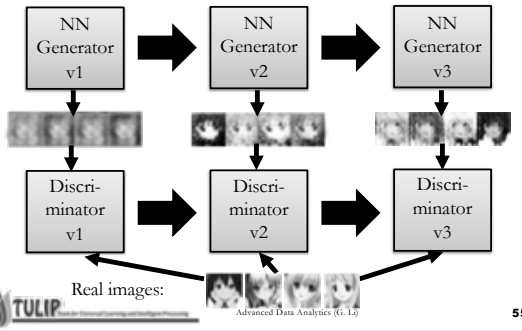
This, and the variations that are now being proposed is the most interesting idea in the last 10 years in ML, in my opinion.

https://www.quora.com/What-are-some-recent-and-potentially-upcoming-breakthroughs-in-deep-learning

Evolution

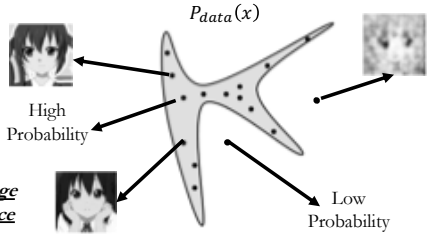


The evolution of generation



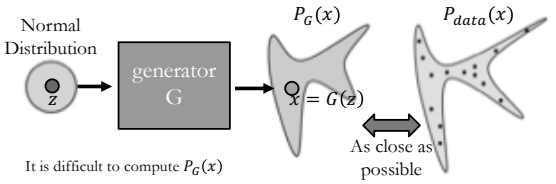
Basic Idea of GAN

- The data we want to generate has a distribution $P_{data}(x)$



Basic Idea of GAN

- A generator G is a network. The network defines a probability distribution.

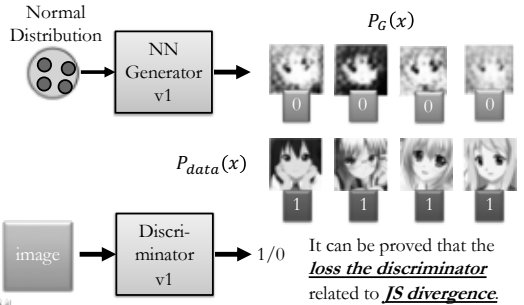


It is difficult to compute $P_G(x)$

We do not know what the distribution looks like.

https://blog.openai.com/generative-models/

Basic Idea of GAN



It can be proved that the ***loss the discriminator*** related to ***JS divergence***.

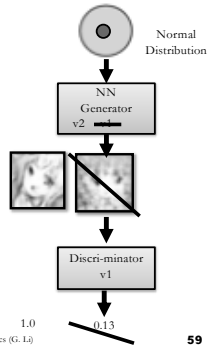
Basic Idea of GAN

- ***Next step***:
  - Updating the parameters of generator
  - To minimize the JS divergence

➡ The output be classified as "real" (as close to 1 as possible)

Generator + Discriminator = a network

Using gradient descent to update the parameters in the generator, but fix the discriminator

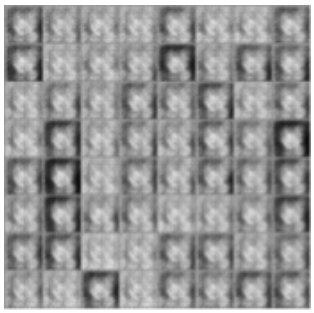


GAN- DCGAN

Source of images: https://zhuanlan.zhihu.com/p/24767059

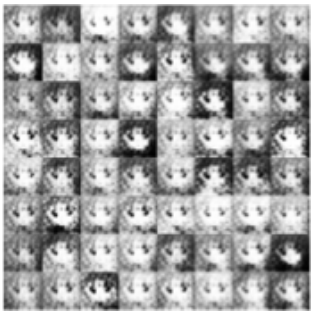
DCGAN: https://github.com/carpedm20/DCGAN-tensorflow

GAN- DCGAN



100 rounds

GAN- DCGAN



1000 rounds

GAN- DCGAN

2000 rounds

GAN- DCGAN

5000 rounds

GAN- DCGAN

10,000 rounds

GAN- DCGAN

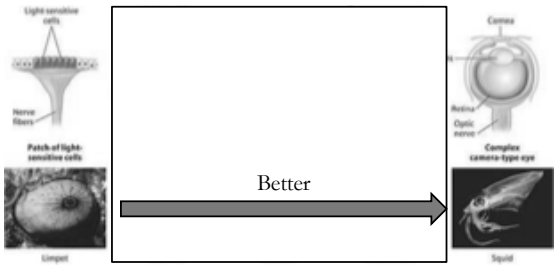
20,000 rounds

GAN- DCGAN

50,000 rounds

http://www.guokr.com/post/773890/

Why GAN is hard to train?



Why GAN is hard to train?

?

$P_{G_0}(x)$ $P_{data}(x)$ $JS(P_{G_1}||P_{data}) = log2$

$P_{G_{50}}(x)$ $P_{data}(x)$ $JS(P_{G_2}||P_{data}) = log2$

Not really better ……

$P_{G_{100}}(x)$ $P_{data}(x)$ $JS(P_{G_2}||P_{data}) = 0$

WGAN (Wasserstein distance)

Better

$P_{G_0}(x)$ d_0 $P_{data}(x)$ $W(P_{G_1}||P_{data}) = d_0$

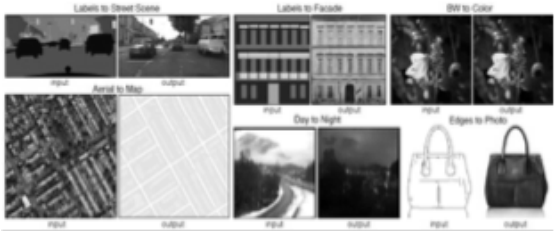
$P_{G_{50}}(x)$ d_{50} $P_{data}(x)$ $W(P_{G_2}||P_{data}) = d_{50}$

$P_{G_{100}}(x)$ $P_{data}(x)$ $W(P_{G_2}||P_{data}) = 0$

Moving on the code space

- Alec Radford, Luke Metz, Soumith Chintala, Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks, ICLR, 2016

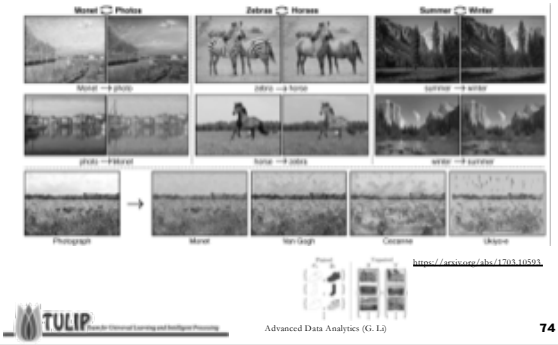
Image-to-image Translation



- Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, Alexei A. Efros, "Image-to-Image Translation with Conditional Adversarial Networks", arXiv preprint, 2016

Image-to-image Translation - Results

Cycle GAN



https://arxiv.org/abs/1703.10593

Variants of GAN

Modifying the Optimization of GAN	Different Structure from the Original GAN
fGAN	Conditional GAN
WGAN	Semi-supervised GAN
Least-square GAN	InfoGAN
Loss Sensitive GAN	BiGAN
Energy-based GAN	Cycle GAN
Boundary-seeking GAN	Disco GAN
Unroll GAN	VAE-GAN
……	……

Questions?