UW MLEARN 410: Applied Machine Learning

Advanced Topics: Big Data

UNIVERSITY of WASHINGTON

Requirements

- **Unix command line (sorry Windows folks)**
- **jq**
  https://stedolan.github.io/jq/download/
- **Vagrant, VirtualBox, Spark cluster from https://github.com/alexholmes/vagrant-hadoop-spark-hive**
- **Vowpal Wabbit**
  https://github.com/JohnLangford/vowpal_wabbit/wiki/Download
- **RCV1-V2 dataset**
  http://hunch.net/~vw/rcv1.tar.gz

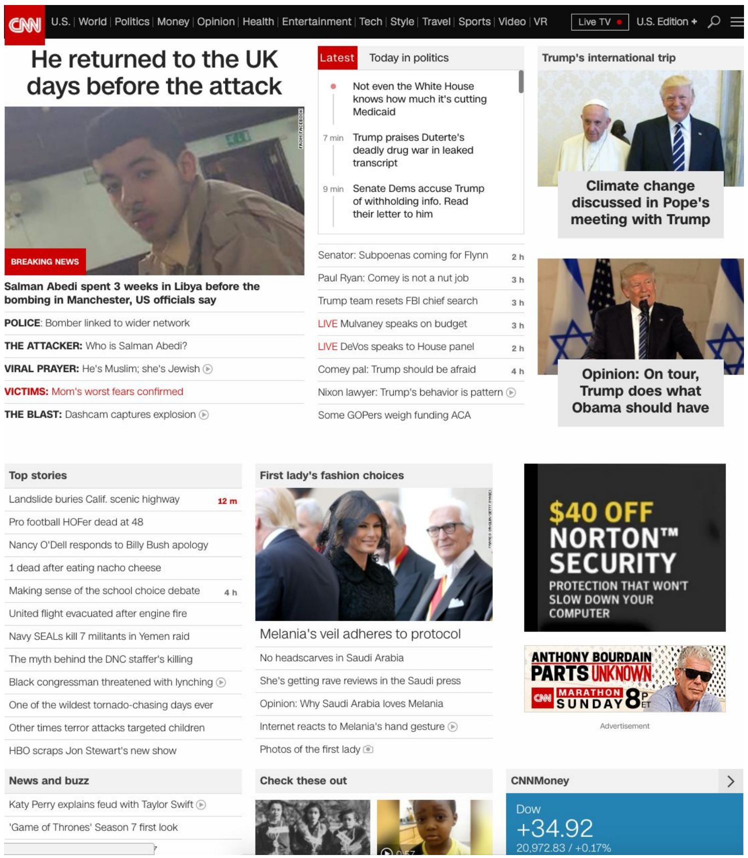
BIG DATA

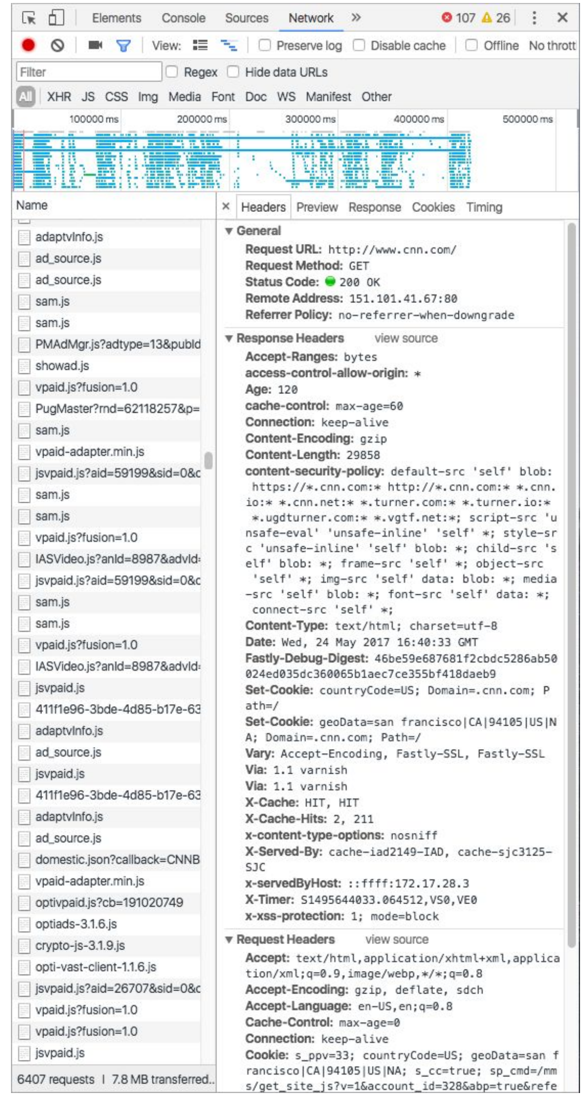
- **Our approach so far:** loading processed datasets into memory

- **Problems:**

  - **Useful data may be mixed in with other data**
  - **Data may need to be cleaned/formatted before using**
  - **Data may be too large to hold in memory**
    - **What does that mean? too many columns? too many rows?**
    - **Do we even need to use all the data?**

Website http logs





Site Logs Example

- **6400 requests made by one client over the course of ~5 minutes!**
- **Not all of these go to the CNN servers**
  - **Fun experiment - turn on an ad-blocker and visit the same sites**
- **Now think about how many requests the servers are *receiving***

Site Logs Example

- **Servers just dump all requests into log files and carry about their jobs**

- **Let's say we want to do some kind of ML with all the *GET* requests we sent out**

  - **An http *GET* request is basically asking a server to send some kind of information back to the client**

Site Logs Example

cat cnn.har | jq '.log.entries[] | .request.method, .serverIPAddress' | paste -d" " - - | grep GET | grep -v '\"\"' | cut -f2 -d' ' | sed 's/"//g' | sort | uniq -c | sort -k1,1nr

- **This processes the data line-by-line***
  - ***** ***jq*** **processes it chunk-by-chunk, but each chunk is not that huge**
  - ***** **The** ***sorts*** **are the only part that need the entire data**

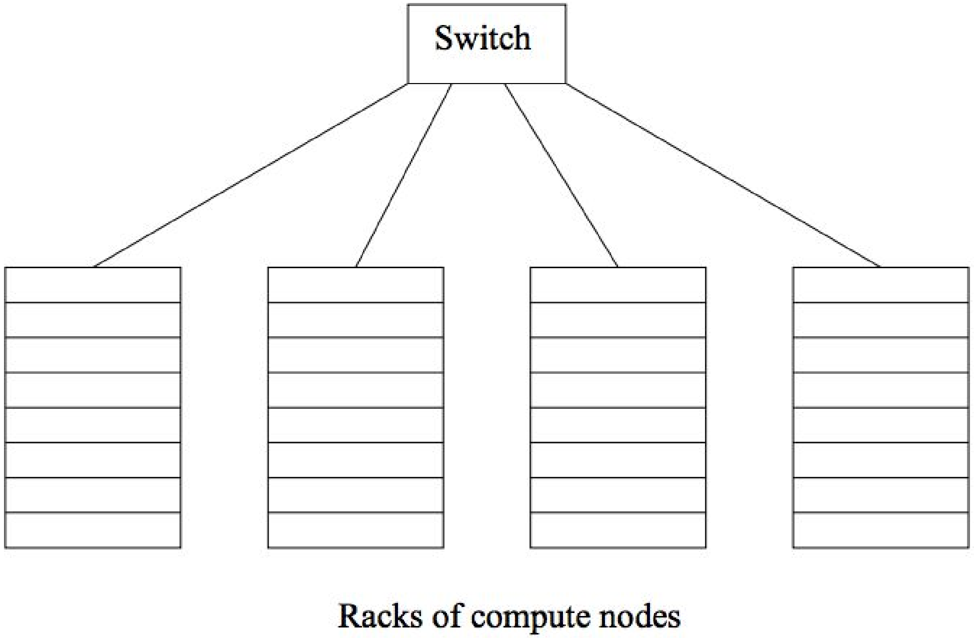
Bigger data? Fancier pre-processing?

- **Even this data was relatively well structured (json with a schema)**

- **What if you have data scraped from the web?**
  - **Can be MASSIVE**

- **Need to parse the HTML/CSS/XML to get text (images/other media?) and then do NLP.**
  - **Way slower**
  - **Can't do on the command line**

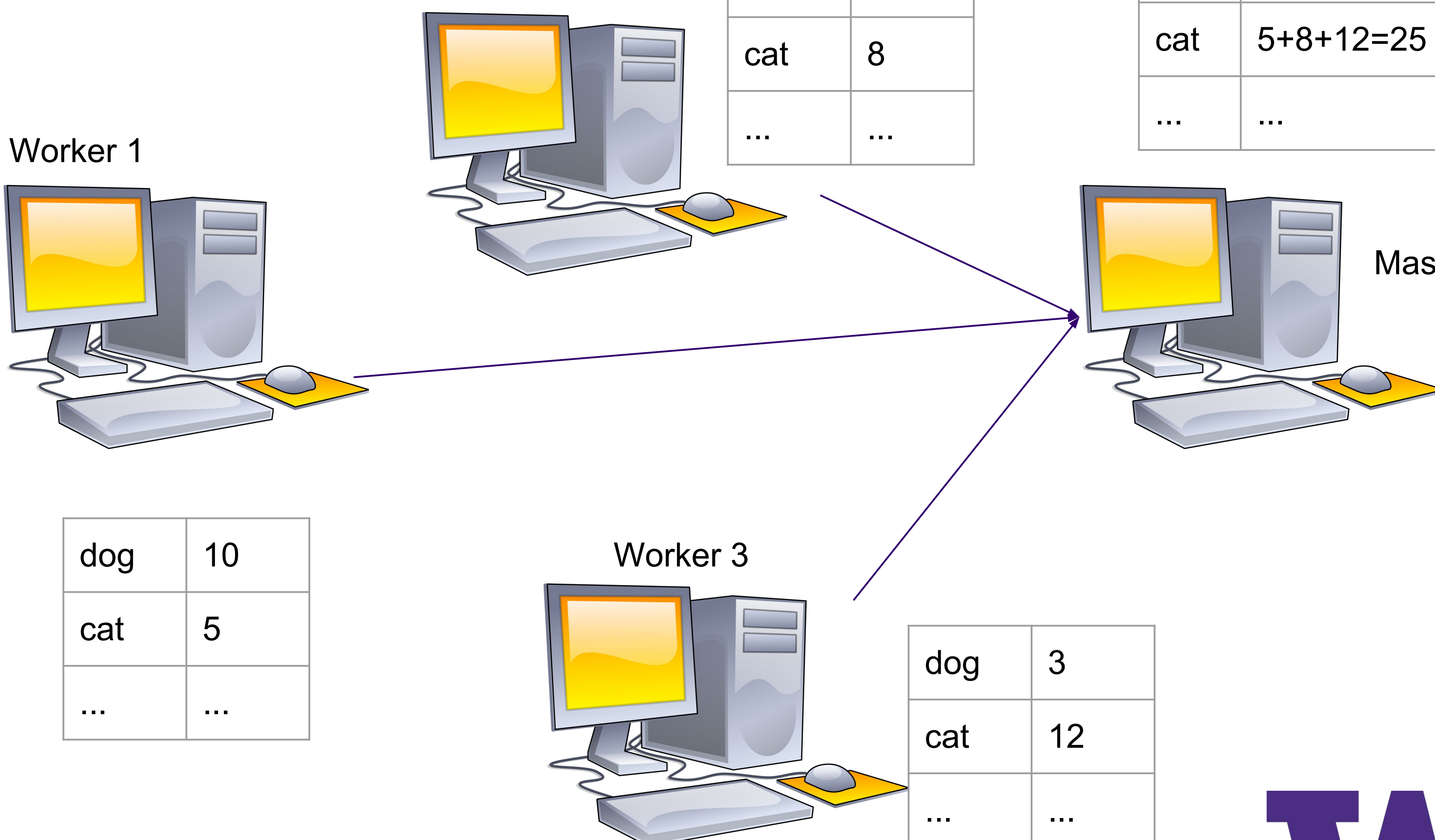
Cluster layout



Clusters!!!

- **Parsing one page is totally independent of the parsing every other page.**
- **In the previous example, we would not need to combine the data until the first *sort* step.**
- **If we had *k* computers, we could go *k* times faster!**
    - **modulo overhead in coordination**

Word counting - First attempt



Worker 1:

dog	10
cat	5
...	...

Worker 2:

dog	17
cat	8
...	...

Worker 3:

dog	3
cat	12
...	...

Master:

dog	10+17+3=30
cat	5+8+12=25
...	...

Problems with the first approach

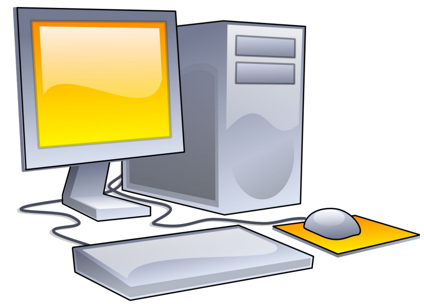
- **All computers are transferring data to the master at the same time**
  - **Bottleneck in data transfer**
- **Second step - only one computer is doing all the work**

Word counting - Step 1

Mapper 1

dog	10
cat	5
fish	12
...	...

Mapper 2

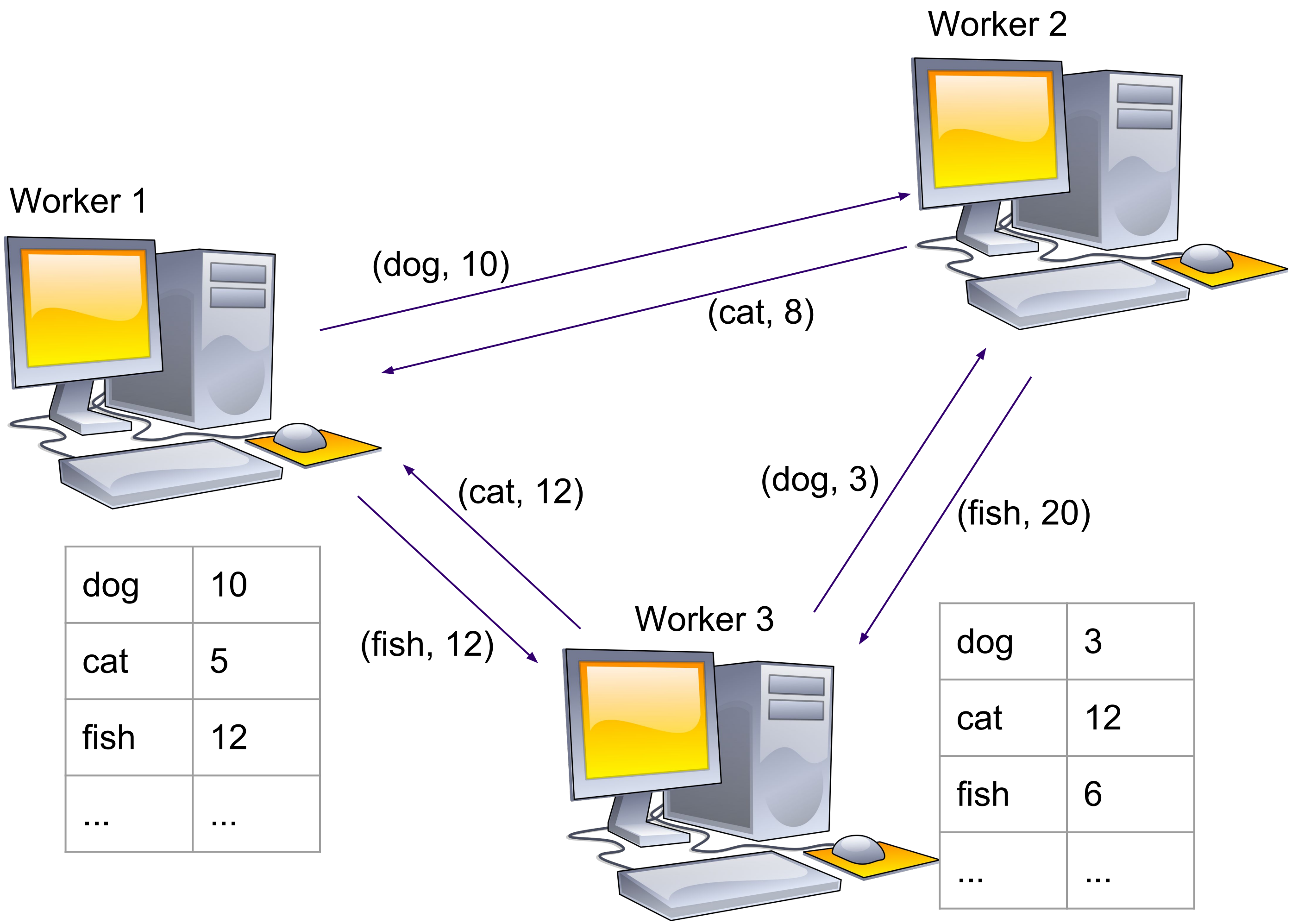


dog	17
cat	8
fish	20
...	...

Mapper 3

dog	3
cat	12
fish	6
...	...

Word counting - Transfer



Worker 1:

dog	10
cat	5
fish	12
...	...

Worker 2:

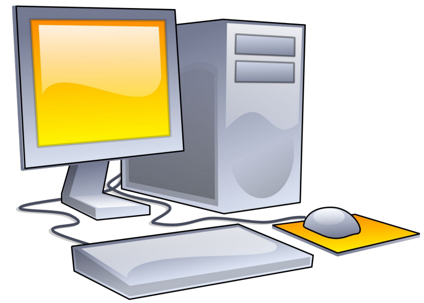
dog	17
cat	8
fish	20
...	...

Worker 3:

dog	3
cat	12
fish	6
...	...

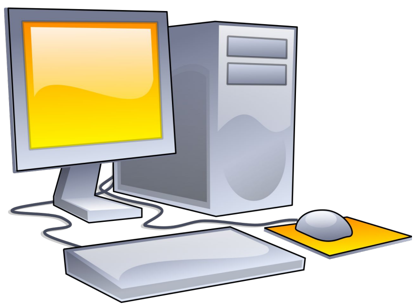
Word counting - Step 2

Reducer 2



dog	17+10+3
...	...

Reducer 1



cat	5+8+ 12
...	...

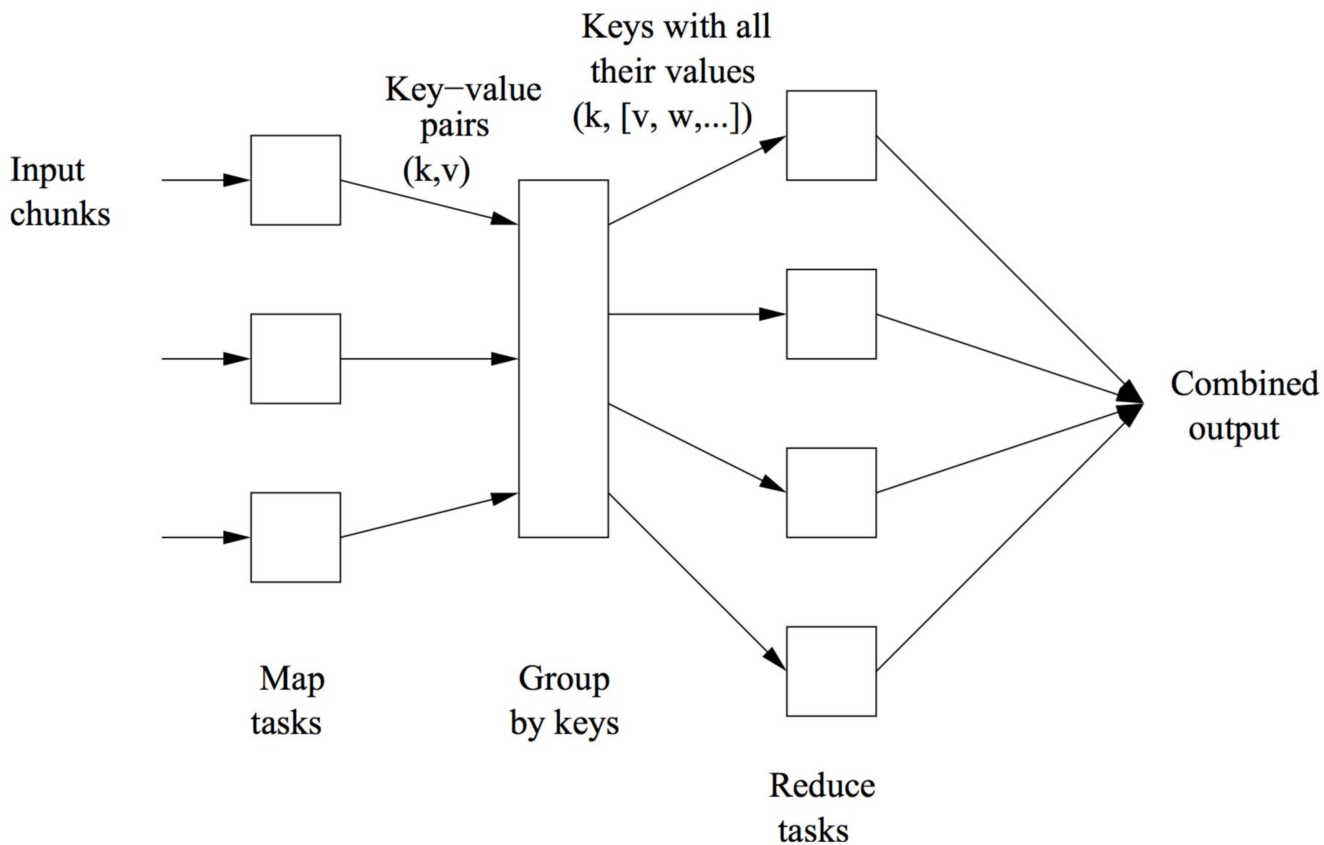
Reducer 3

fish	6+12+20
...	...

MapReduce

- **Step 1: *Map*   Step 2: *Reduce***
- **Balanced transfer of data and computation load in the Reduce step**
- **Important to ensure each word gets mapped to the same reducer node**
  - **Hash function f: <word> → {1,2, ..., #reducer nodes}**
  - **random assignment for load balancing**

MapReduce, formally



Map Step

- **Each map step iterates through each word and spits out the key, value pair: (<word>, 1)**
  - **The value is the constant 1 for each word**

- **E.g. Mapper 1 input: “The quick brown *dog* jumps over the lazy dog”**

- **Mapper 1 output: (the, 1), (quick, 1), (brown, 1), (dog, 1), (jumps, 1), (over, 1), (the, 1), (lazy, 1), (dog, 1)**

GroupBy Step

- **Each mapper sorts the pairs by the keys:**

(brown, 1), (dog, 1), (dog, 1), (jumps, 1), ...

- ***Optional step (combiner):*** **Combine pairs with the same keys at the mapper (usually using the same logic as the reducer)**

(brown, 1), (dog, 2), (jumps, 1)

Distribute to Reducers

- **Words are pseudo-randomly assigned to reducers using a hash function:**
  - **Important that all mappers use the same pseudo random hash function.**

- **Reducer 1 will see:**

(dog, 2), (jumps, 1), (lazy, 1), (dog, 4), (fish, 3), (lazy, 2)

from mapper 1, mapper 2, and so on...

Reducers

- **Reducer 1 input:**

(dog, 2), (jumps, 1), (lazy, 1), (dog, 4), (fish, 3), (lazy, 2)

- **Sort again, combine values with the same key:**

(dog, [2, 4]), (fish, [3]), (jumps, [1]), (lazy, [1, 2])

- **Sum values in list of each value**

Distributed File Systems

- **The input and output data were distributed across workers**

- **This is actually a feature of the file system**
  - **Based on Google File System (GFS)**
  - **Open source - Hadoop Distributed File System (HDFS)**

- **Files are split into chunks, replicated and stored on random nodes**

Distributed File Systems

- **When a map task comes in, each mapper takes the chunks on its local disk and works on those**

- **Also provides redundancy against failures**
  -
  - **If a machine goes down, all the data on it is stored on other nodes and can be re-processed as needed**

Hadoop versus Spark

- **Hadoop needs to do Map→Reduce→Map→Reduce**
- **Hadoop writes the output out to disk after every map and reduce step**
- **Spark can do Map→Reduce→Reduce→Reduce**
- **Spark holds everything in memory**
    - **Less File I/O speeds things up a lot**

Hadoop and Spark

- **Can use these paradigms to implement many kinds of algorithms on massive datasets**
    - **Numerical matrix algebra**
    - **Relational algebra type (SQL) operations - Joins, GroupBys, etc...**
    - **Machine Learning**
        - **PageRank**
        - **Random Forests**
- **Typically not great for algorithms that iteratively update parameters/state**

How to learn with big data

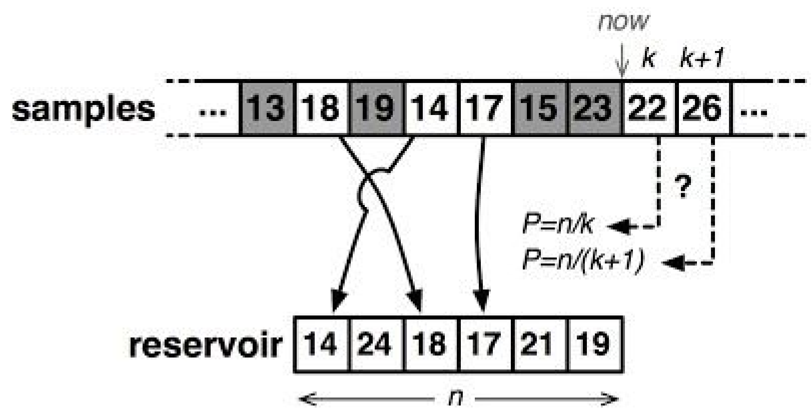
- **Large dataset processed - how to do ML?**
  - **possibly stored on a distributed file system**
- **Do you really need to use all the data to train?**
  - **Signal-to-noise level**
  - **Number of features**
  - **Complexity (#free parameters) of the model**

No? Sample!

- **Think before sampling:**
  - **Leakage of information between train and test splits?**
  - **Random sampling on <index> and selecting all rows for given values of <index>**
- **shuf -n N inputfile > outputfile**

Reservoir Sampling

- **Ongoing stream of data:**
  - ***n* points have passed by**
  - **want a uniform sample of *k* points such that**
  - **every point has probability of *k/n***



Reservoir Sampling Algorithm

- **Let the sample be $S[1], \ldots, S[k]$**
- **Store first k points in $S[1], \ldots,$ S[k], then**
- **Let i be the count of the current item**
- **Randomly draw an integer j from $[\,1, i\,]$**
- **If $j < k$, then overwrite $S[j] \leftarrow S[i]$**

Yes? Online learning!

- **Data too big to fit in memory - need to process in chunks**
  - **similar to pre-processing**
- **Added benefit → model parameters are continuously fit to newer data**
  - **If the underlying data distribution changes, the model will catch on automatically (eventually)**

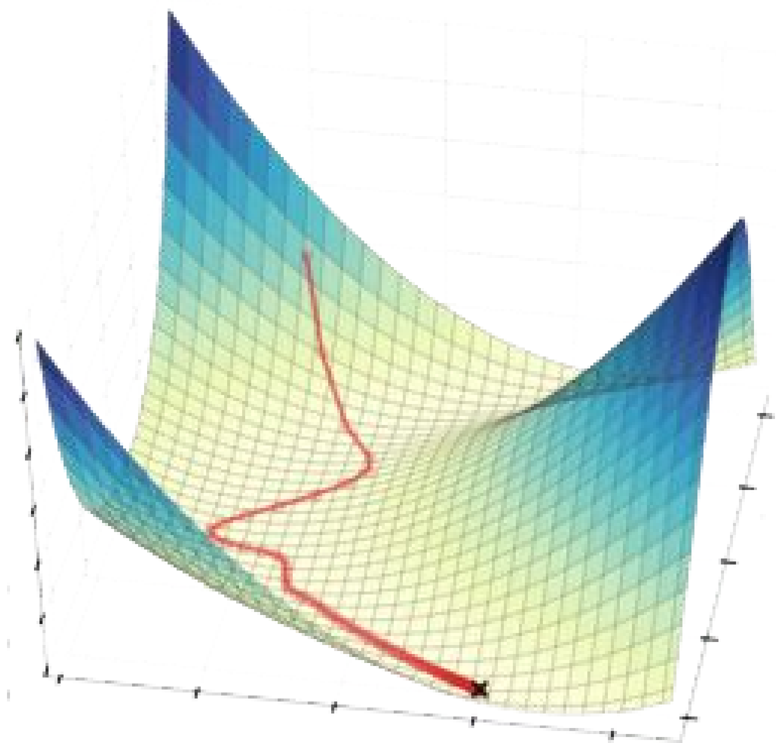
Gradient descent

- **Example: Logistic regression**
- **Minimize some loss function**
  - **Recall Lecture 1**

$$\mathcal{L}(X) := \frac{1}{N}\sum_{i=1}^{N} -y_i \log(f(\beta^T X_i)) - (1 - y_i)\log(1 - f(\beta^T X_i))$$

Gradient descent

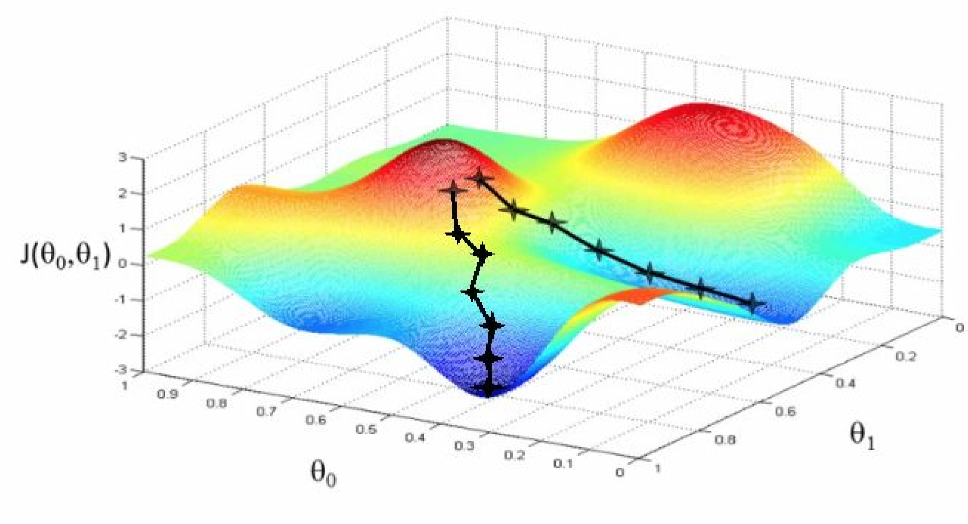
- **Intuition: "go downhill taking steps in the steepest direction"**



Gradient descent

- **The direction is given by the negative of the derivative (*gradient* in multiple dimensions)**

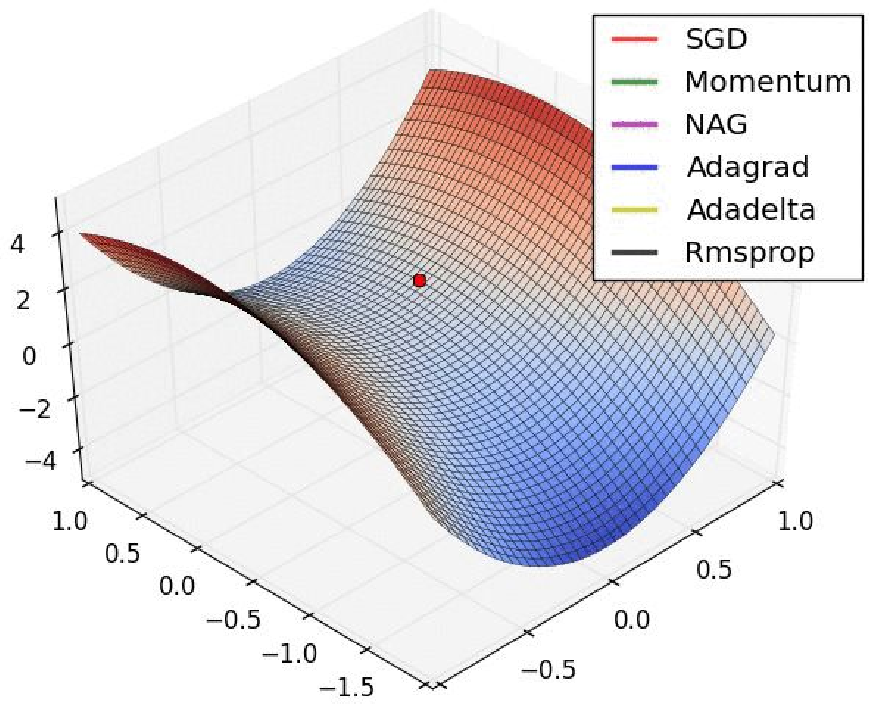
- **Issues**
  - **Local minima / Non-unique solutions**



Gradient descent

- **Issues**
  - **Saddle points**

(**http://sebastianruder.com/optimizing-gradient-descent/**)



Gradient descent

- **Calculates derivatives over all points**

$$\mathcal{L}(X) := \frac{1}{N}\sum_{i=1}^{N} -y_i \log(f(\beta^T X_i)) - (1 - y_i)\log(1 - f(\beta^T X_i))$$

- **Very slow if data not in memory**

Stochastic Gradient Descent

Problem: **It is expensive to use all the data at each step**

Solution: **Sample mini-batches of *m* << *N* points at each step**

- **i.i.d. assumption → create mini-batches of size *m* them as they come in, i.e.**

  average gradients over *i* = *k*, *k*+1, ..., *k*+*m*

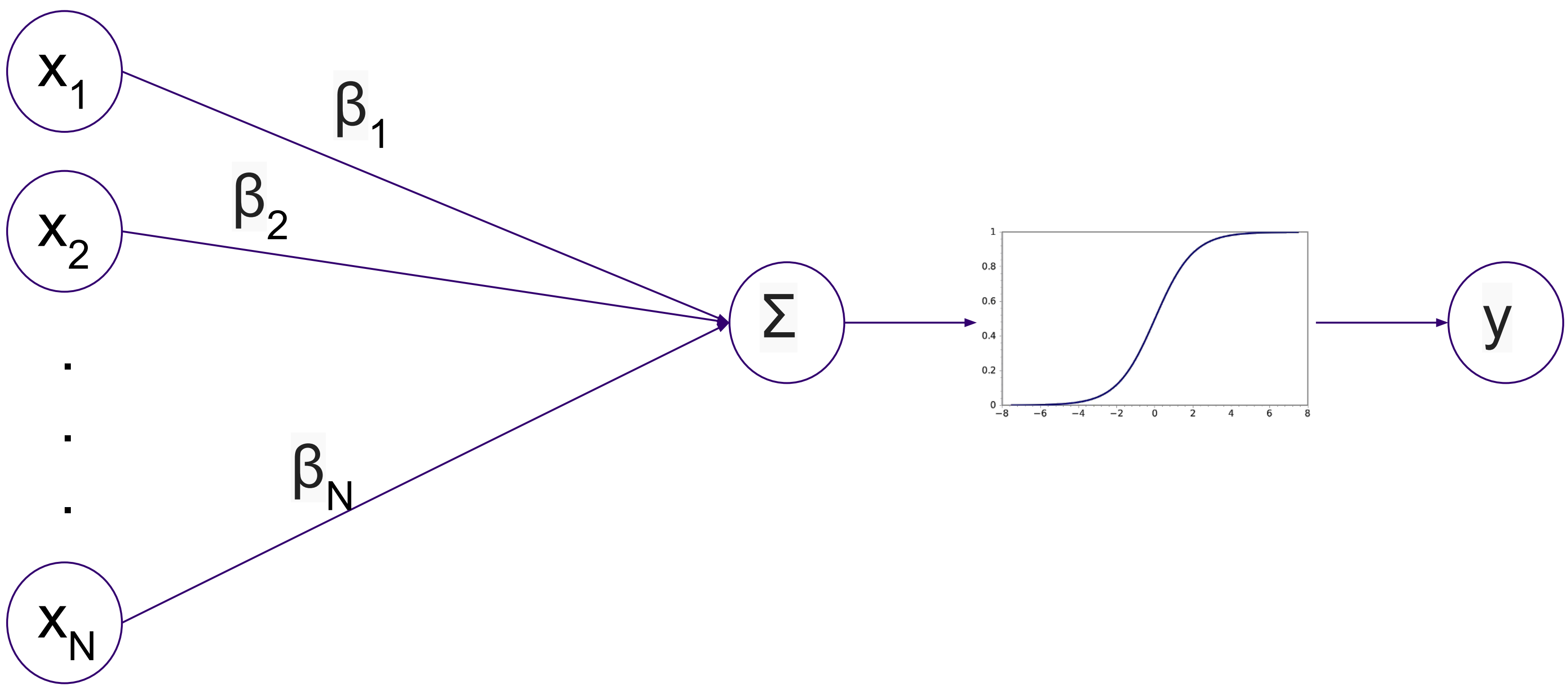
SGD is noisy...but...

- **it allows us to use *much* more data to compensate/average out the noise**
- **the noise may actually help push us out of local minima and avoid saddles**

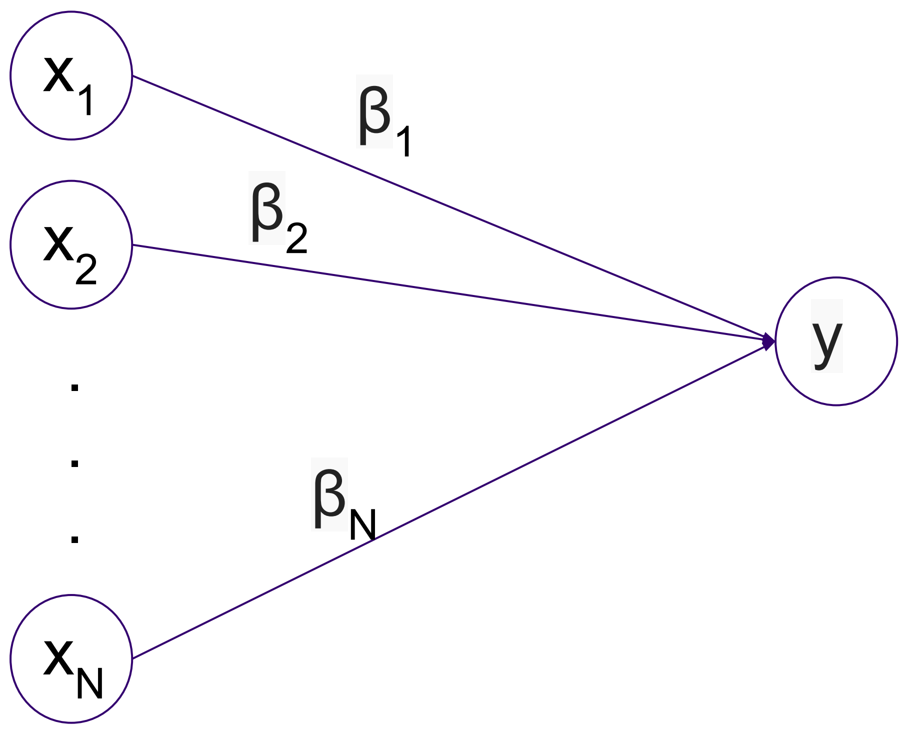
SGD free parameters

- Step size
  - constant? large in the beginning, and gets smaller?
- How many passes through the data?
- Sort the data if doing multiple passes?

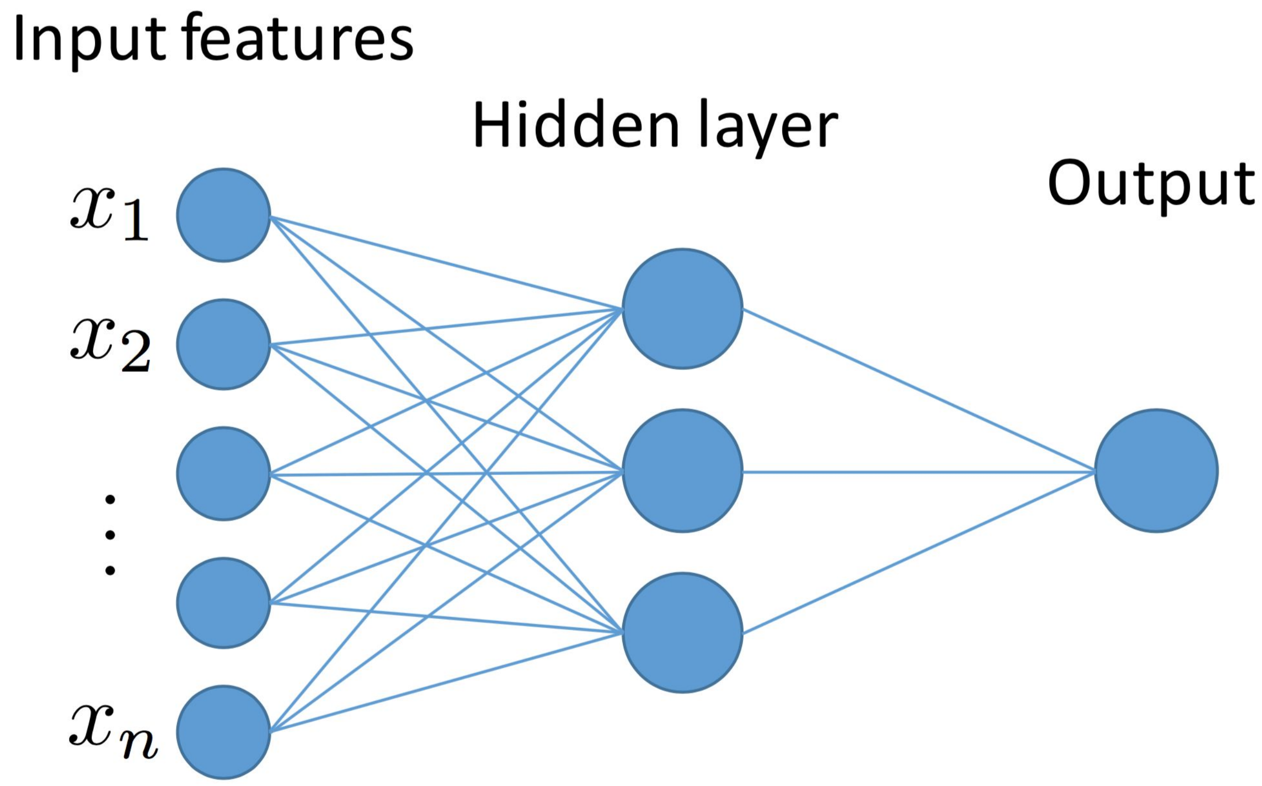
Neural Network - Logistic Regression



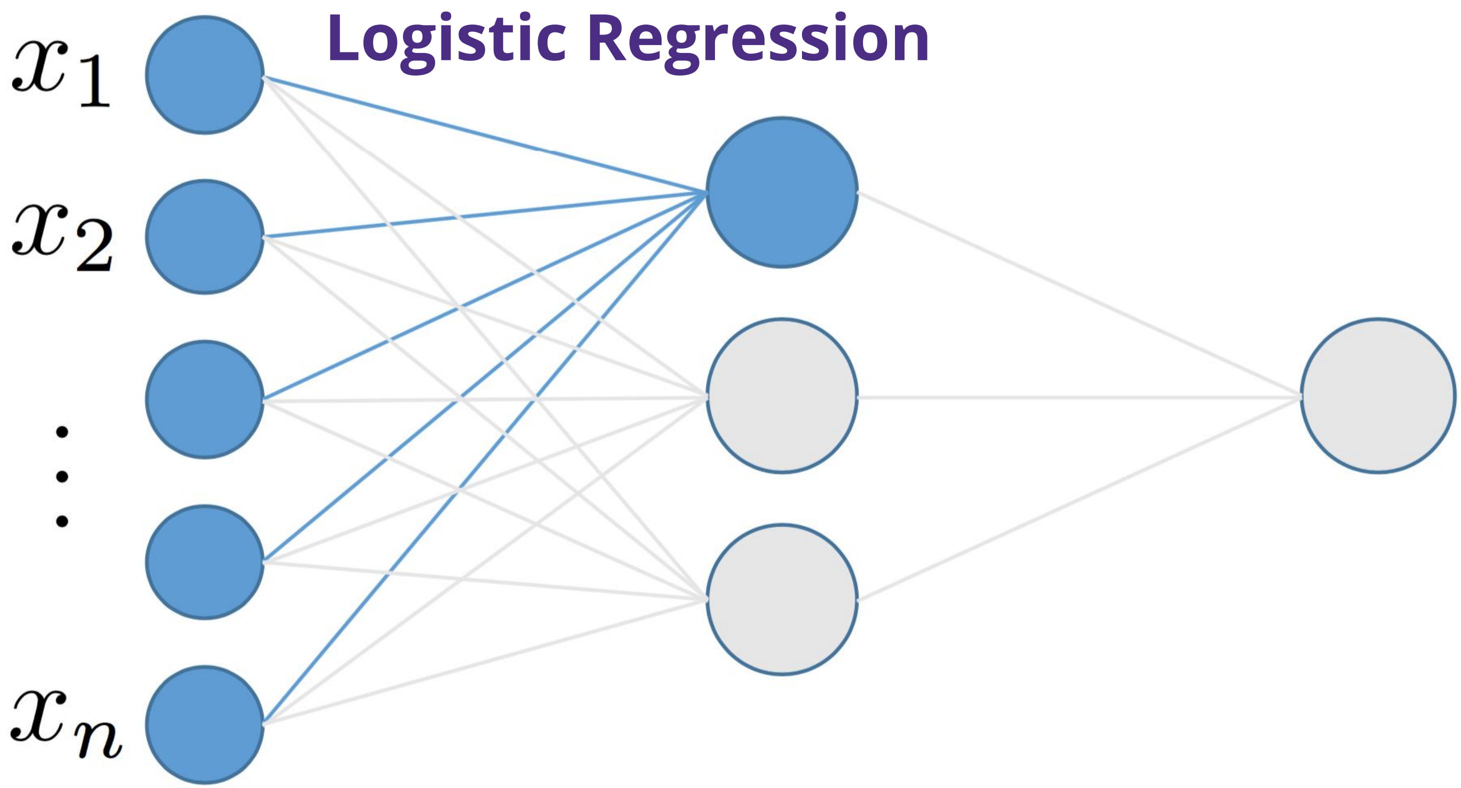
Typically represented as...



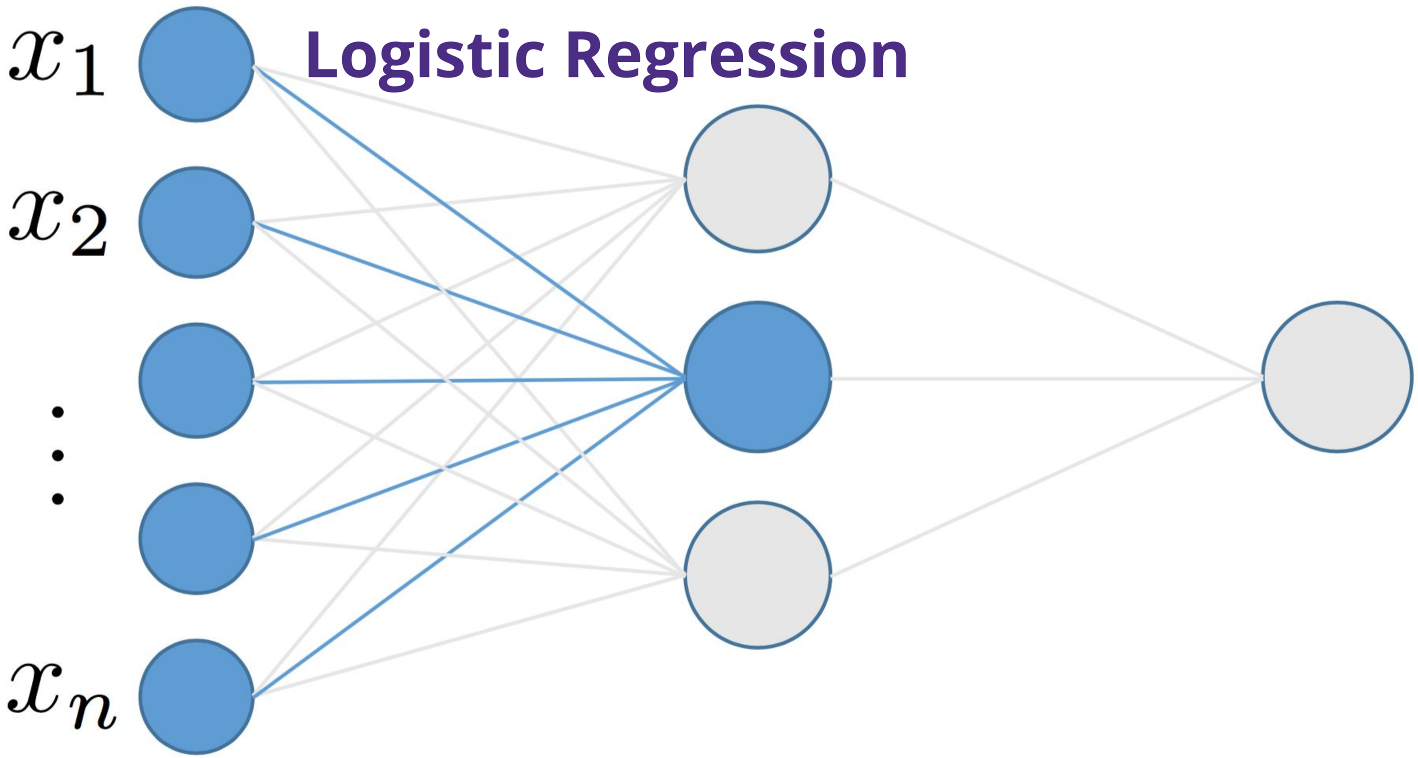
1 Hidden layer NN



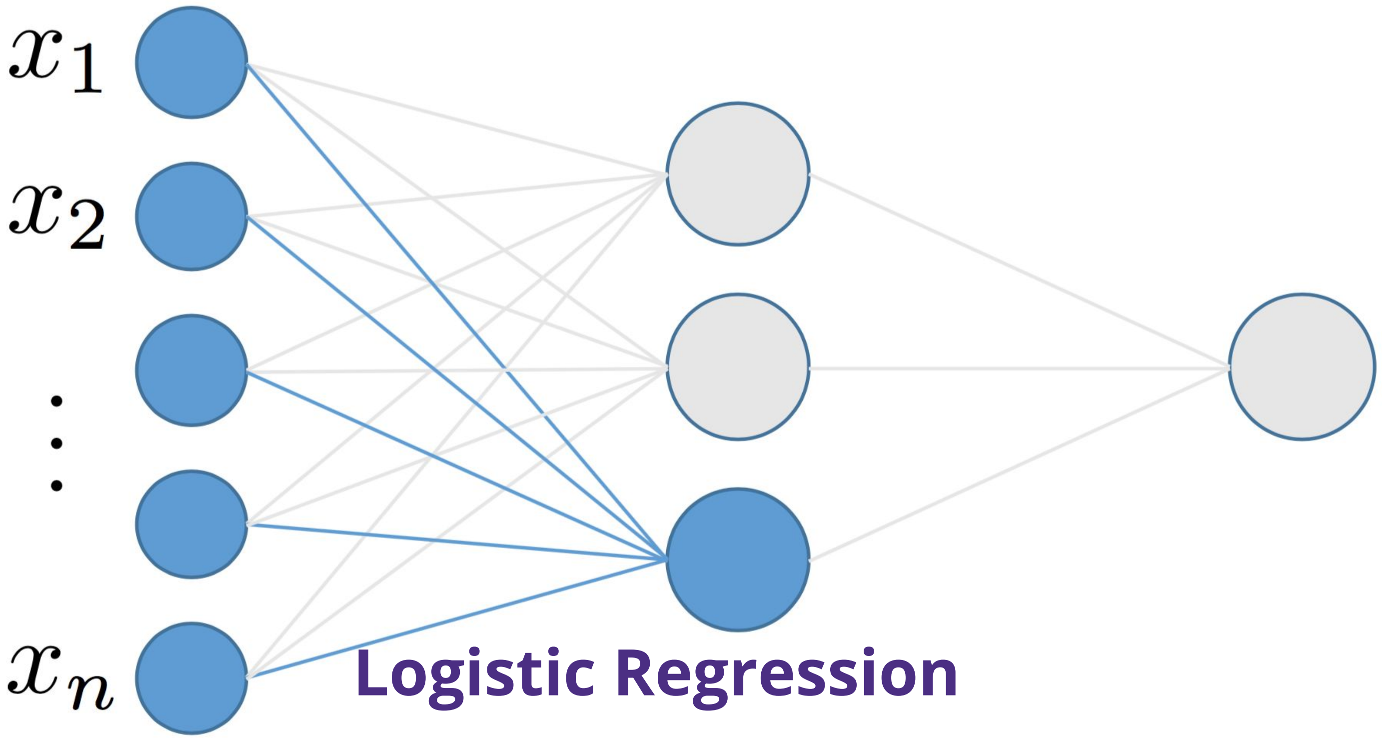
Decomposing the NN



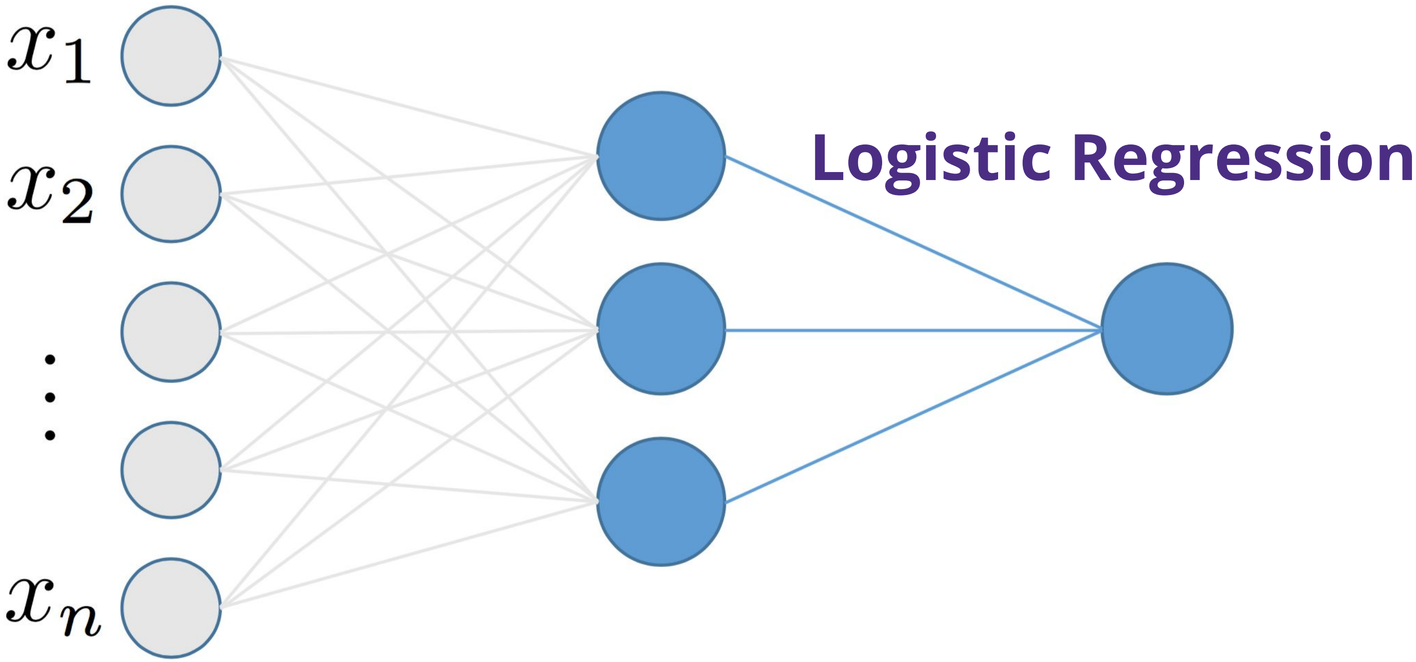
Decomposing the NN



Decomposing the NN



Decomposing the NN



Deep Neural Networks - Gradients

- $y_i = f_{1,i}(x_1, x_2, x_N)$
- $z_j = f_{2,j}(y_1, y_2, y_M)$

- Compose layers as follows:

  $\underline{z} = \underline{f}_2(\ \underline{f}_1(x_1, x_2, x_N)\ ),$

  where $\underline{f}_1 = (f_{1,1}, f_{1,2}, \ldots, f_{1,M})$ and $\underline{f}_2 = (f_{2,1}, f_{2,2}, \ldots, f_{2,P})$

Deep Neural Networks - Gradients

$\underline{z} = \underline{f}_2(\underline{f}_1(x_1, x_2, x_N)),$

where $\underline{f}_1 = (f_{1,1}, f_{1,2}, \ldots, f_{1,M})$ and $\underline{f}_2 = (f_{2,1}, f_{2,2}, \ldots, f_{2,P})$

Compute gradients of the error at each layer

Errors are then be composed using the chain rule.

This is called *backpropagation*.

Computed automatically on Tensorflow, Torch, etc.

Deep Neural Networks

- **Simplest type of neural network - *feedforward neural network***
- **Add more hidden layers to make it *deeper***
- **Deeper networks can learn more complicated transformations**

Deep Neural Networks

- **A sufficiently deep and wide neural network can approximate *ANY* function**
  - ***Universal function approximation* property**

- **More nodes/layers → more parameters to infer**

- **More parameters require more data!**