732A54/TDDE31 Big Data Analytics
Lecture 8: Machine Learning with MapReduce

Jose M. Peña
IDA, Linköping University, Sweden

Contents

Literature

- Main sources
  - Dean, J. and Ghemawat, S. MapReduce: Simplified Data Processing on Large Clusters. *Communications of the ACM*, 51(1):107-113, 2008.
  - Chu, C.-T. et al. Map-Reduce for Machine Learning on Multicore. In *Proceedings of the 19th International Conference on Neural Information Processing Systems*, 281-288, 2006.
- Additional sources
  - Dean, J. and Ghemawat, S. MapReduce: Simplified Data Processing on Large Clusters. In *Proceedings of the 6th Symposium on Operating Systems Design and Implementation*, 2004.
  - Gillick, D., Faria, A. and DeNero, J. MapReduce: Distributed Computing for Machine Learning. Technical Report, Berkley, 2006.
  - Slides for 732A99/TDDE01 Machine Learning.

MapReduce Framework

- Programming framework developed at Google to process large amounts of data by parallelizing computations across a cluster of computers.

MapReduce Framework

- Programming framework developed at Google to process large amounts of data by parallelizing computations across a cluster of computers.
- Easy to use, since the parallelization happens automatically.

MapReduce Framework

- Programming framework developed at Google to process large amounts of data by parallelizing computations across a cluster of computers.
- Easy to use, since the parallelization happens automatically.
- Easy to speed up by using/adding more computers to the cluster.

MapReduce Framework

- Programming framework developed at Google to process large amounts of data by parallelizing computations across a cluster of computers.
- Easy to use, since the parallelization happens automatically.
- Easy to speed up by using/adding more computers to the cluster.
- Typical uses at Google:
  - Large-scale machine learning problems, e.g. clustering documents from Google News.
  - Extracting properties of web pages, e.g. web access log data.
  - Large-scale graph computations, e.g. web link graph.
  - Statistical machine translation.
  - Processing satellite images.
  - Production of the indexing system used for Google's web search engine.

MapReduce Framework

- Programming framework developed at Google to process large amounts of data by parallelizing computations across a cluster of computers.
- Easy to use, since the parallelization happens automatically.
- Easy to speed up by using/adding more computers to the cluster.
- Typical uses at Google:
  - Large-scale machine learning problems, e.g. clustering documents from Google News.
  - Extracting properties of web pages, e.g. web access log data.
  - Large-scale graph computations, e.g. web link graph.
  - Statistical machine translation.
  - Processing satellite images.
  - Production of the indexing system used for Google's web search engine.
- Google replaced it with Cloud Dataflow/Dataproc/Platform, since it could not process the amount of data they produced.

MapReduce Framework

- Programming framework developed at Google to process large amounts of data by parallelizing computations across a cluster of computers.
- Easy to use, since the parallelization happens automatically.
- Easy to speed up by using/adding more computers to the cluster.
- Typical uses at Google:
  - Large-scale machine learning problems, e.g. clustering documents from Google News.
  - Extracting properties of web pages, e.g. web access log data.
  - Large-scale graph computations, e.g. web link graph.
  - Statistical machine translation.
  - Processing satellite images.
  - Production of the indexing system used for Google's web search engine.
- Google replaced it with Cloud Dataflow/Dataproc/Platform, since it could not process the amount of data they produced.
- However, it is still the processing core of Apache Hadoop, another framework for distributed storage and distributed processing of large datasets on computer clusters.

MapReduce Framework

- Programming framework developed at Google to process large amounts of data by parallelizing computations across a cluster of computers.
- Easy to use, since the parallelization happens automatically.
- Easy to speed up by using/adding more computers to the cluster.
- Typical uses at Google:
  - Large-scale machine learning problems, e.g. clustering documents from Google News.
  - Extracting properties of web pages, e.g. web access log data.
  - Large-scale graph computations, e.g. web link graph.
  - Statistical machine translation.
  - Processing satellite images.
  - Production of the indexing system used for Google's web search engine.
- Google replaced it with Cloud Dataflow/Dataproc/Platform, since it could not process the amount of data they produced.
- However, it is still the processing core of Apache Hadoop, another framework for distributed storage and distributed processing of large datasets on computer clusters.
- Moreover, it is a straightforward way to adapt some machine learning algorithms to cope with big data.

MapReduce Framework

- Programming framework developed at Google to process large amounts of data by parallelizing computations across a cluster of computers.
- Easy to use, since the parallelization happens automatically.
- Easy to speed up by using/adding more computers to the cluster.
- Typical uses at Google:
  - Large-scale machine learning problems, e.g. clustering documents from Google News.
  - Extracting properties of web pages, e.g. web access log data.
  - Large-scale graph computations, e.g. web link graph.
  - Statistical machine translation.
  - Processing satellite images.
  - Production of the indexing system used for Google's web search engine.
- Google replaced it with Cloud Dataflow/Dataproc/Platform, since it could not process the amount of data they produced.
- However, it is still the processing core of Apache Hadoop, another framework for distributed storage and distributed processing of large datasets on computer clusters.
- Moreover, it is a straightforward way to adapt some machine learning algorithms to cope with big data.
- Apache Mahout is a project to produce distributed implementations of machine learning algorithms. It builds on Hadoop's MapReduce. However, these implementations are now deprecated, in favor of Apache Spark.

MapReduce Framework

- The user only has to implement the following two functions:
  - Map function:
    - Input: A pair (in_key, in_value).
    - Output: A list $list(out\_key, intermediate\_value)$.
  - Reduce function:
    - Input: A pair $(out\_key, list(intermediate\_value))$.
    - Output: A list $list(out\_value)$.

MapReduce Framework

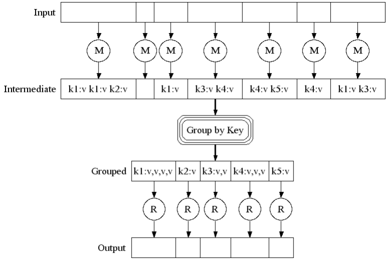
- The user only has to implement the following two functions:
  - Map function:
    - Input: A pair (in_key, in_value).
    - Output: A list $list(out\_key, intermediate\_value)$.
  - Reduce function:
    - Input: A pair $(out\_key, list(intermediate\_value))$.
    - Output: A list $list(out\_value)$.
  - All intermediate values associated with the same intermediate key are grouped together before passing them to the reduce function.

MapReduce Framework

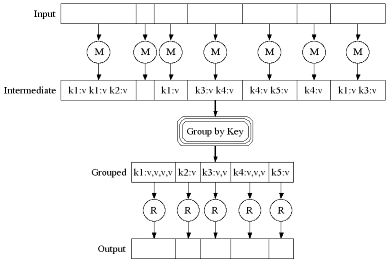
- The user only has to implement the following two functions:
  - Map function:
    - Input: A pair (in_key, in_value).
    - Output: A list $list(out\_key, intermediate\_value)$.
  - Reduce function:
    - Input: A pair $(out\_key, list(intermediate\_value))$.
    - Output: A list $list(out\_value)$.
  - All intermediate values associated with the same intermediate key are grouped together before passing them to the reduce function.
- Example for counting word occurrences in a collection of documents:

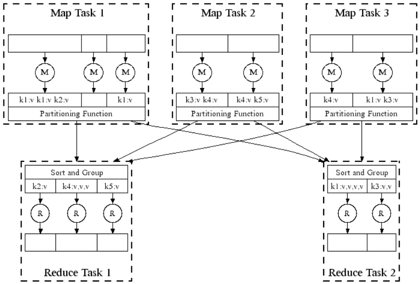
```
map(String key, String value):
  // key: document name
  // value: document contents
  for each word w in value:
    EmitIntermediate(w, "1");

reduce(String key, Iterator values):
  // key: a word
  // values: a list of counts
  int result = 0;
  for each v in values:
    result += ParseInt(v);
  Emit(AsString(result));
```

MapReduce Framework

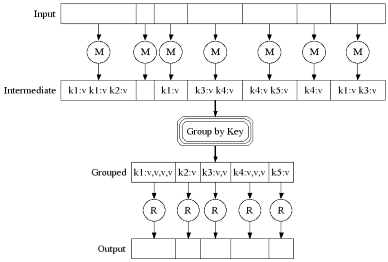


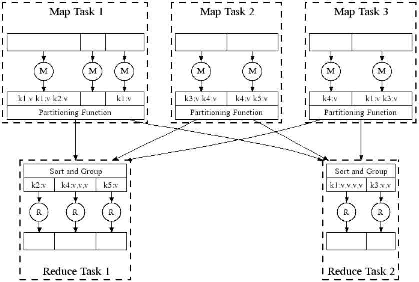
MapReduce Framework

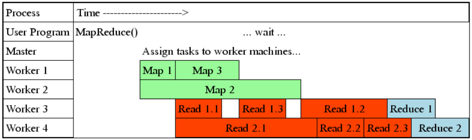




MapReduce Framework







MapReduce Framework

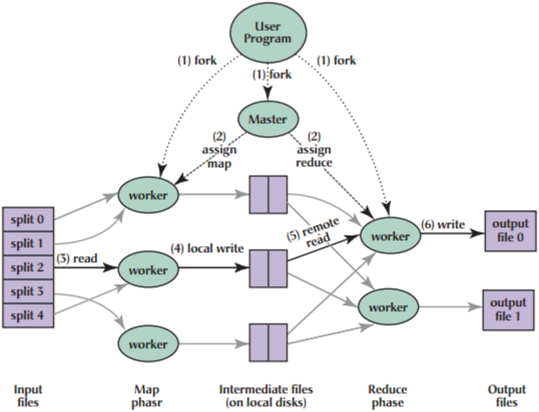


Fig. 1. Execution overview.

MapReduce Framework

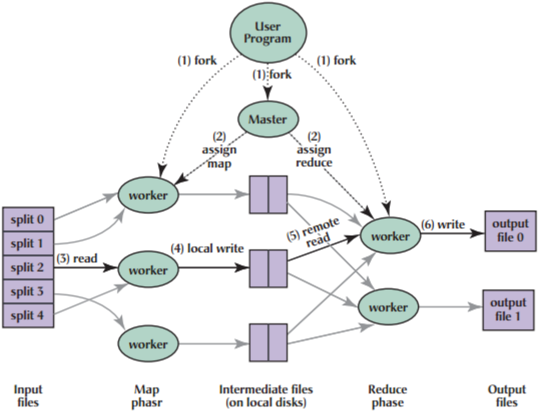


Fig. 1. Execution overview.

1. Split the input file in M pieces and store them on the local disks of the nodes of the cluster. Start up many copies of the user's program on the nodes.

MapReduce Framework

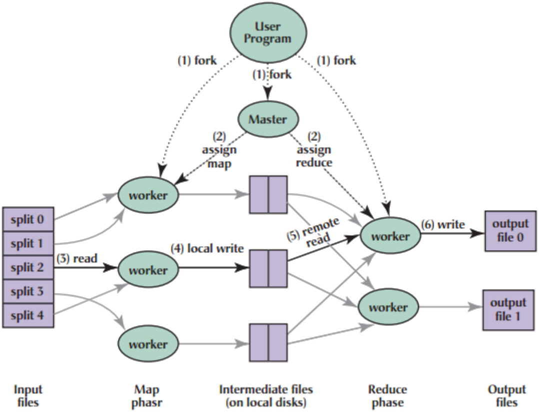


Fig. 1. Execution overview.

1. Split the input file in M pieces and store them on the local disks of the nodes of the cluster. Start up many copies of the user's program on the nodes.
2. One copy (the master) assigns tasks to the rest of the copies (the workers). To reduce communication, it tries to assign map workers to nodes with input data.

MapReduce Framework

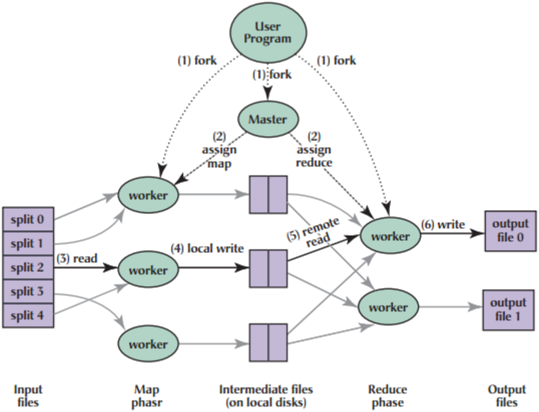


Fig. 1. Execution overview.

3. Each map worker processes a piece of input data by running the user's map function on each pair (key, value). The results are buffered in memory.

MapReduce Framework

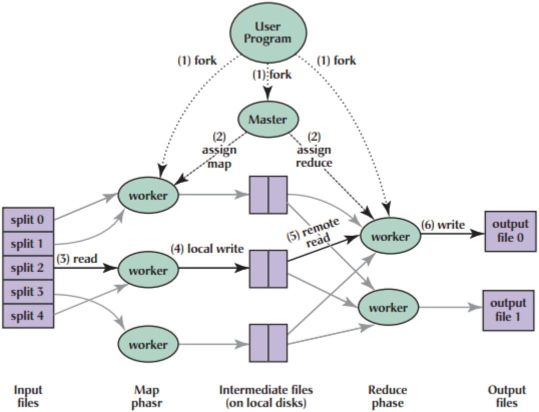


Fig. 1. Execution overview.

3. Each map worker processes a piece of input data by running the user's map function on each pair (key, value). The results are buffered in memory.
4. The buffered results are written to **local** disk. The disk is **partitioned** in R pieces, e.g. $hash(out\_key)\ mod\ R$. The location of the partitions on disk are passed back to the master so that they can be forwarded to the reduce workers.

MapReduce Framework

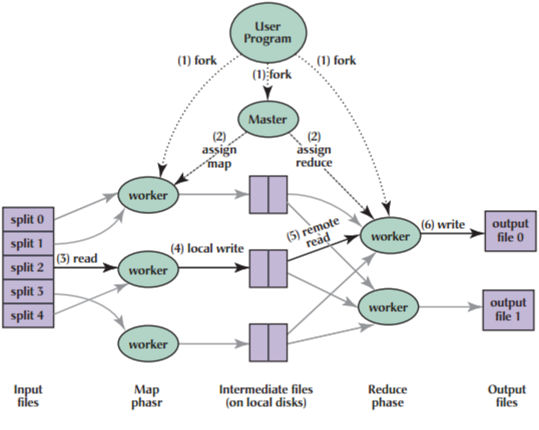


Fig. 1. Execution overview.

5. The reduce worker reads its partition remotely (a.k.a shuffle) and sorts it by key.

MapReduce Framework

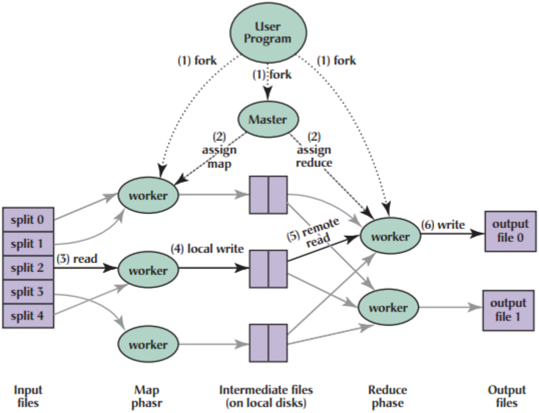


Fig. 1. Execution overview.

5. The reduce worker reads its partition remotely (a.k.a shuffle) and sorts it by key.
6. The reduce worker processes each key using the user's reduce function. The result is written to the **global** file system.

MapReduce Framework

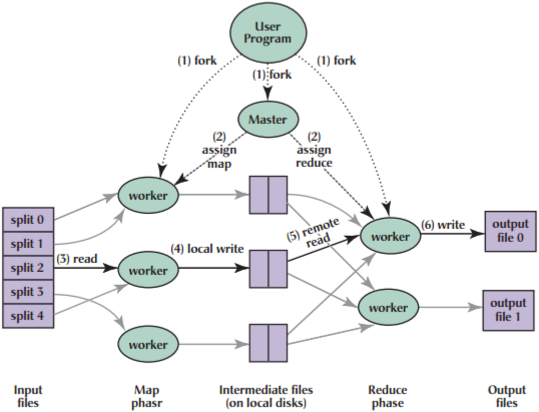


Fig. 1. Execution overview.

5. The reduce worker reads its partition remotely (a.k.a shuffle) and sorts it by key.
6. The reduce worker processes each key using the user's reduce function. The result is written to the **global** file system.
7. The output of a MapReduce call may be the input to another. Note that we have performed M map tasks and R reduce tasks.

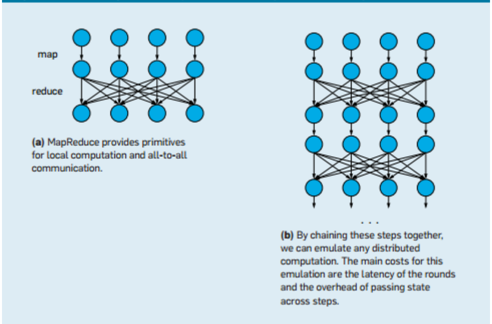
MapReduce Framework

- MapReduce can emulate any distributed computation, since this consists of nodes that perform local computations and occasionally exchange messages.

MapReduce Framework

- MapReduce can emulate any distributed computation, since this consists of nodes that perform local computations and occasionally exchange messages.
- Therefore, any distributed computation can be divided into a sequence of MapReduce calls:
  - First, nodes perform local computations (map), and
  - then, they exchange messages (reduce).

Figure 10. Emulating an arbitrary distributed computation with MapReduce.



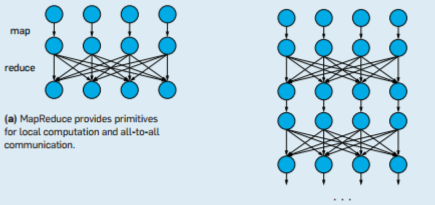
(a) MapReduce provides primitives for local computation and all-to-all communication.

(b) By chaining these steps together, we can emulate any distributed computation. The main costs for this emulation are the latency of the rounds and the overhead of passing state across steps.

MapReduce Framework

- MapReduce can emulate any distributed computation, since this consists of nodes that perform local computations and occasionally exchange messages.
- Therefore, any distributed computation can be divided into a sequence of MapReduce calls:
  - First, nodes perform local computations (map), and
  - then, they exchange messages (reduce).

Figure 10. Emulating an arbitrary distributed computation with MapReduce.



(a) MapReduce provides primitives for local computation and all-to-all communication.

(b) By chaining these steps together, we can emulate any distributed computation. The main costs for this emulation are the latency of the rounds and the overhead of passing state across steps.

- However, the emulation may be **inefficient** since the message exchange relies on external storage, e.g. disk.

MapReduce Framework

- Fault tolerance:
  - Necessary since thousands of nodes may be used.

MapReduce Framework

- Fault tolerance:
  - Necessary since thousands of nodes may be used.
  - The master pings the workers periodically. No answer means failure.

MapReduce Framework

- Fault tolerance:
  - Necessary since thousands of nodes may be used.
  - The master pings the workers periodically. No answer means failure.
  - If a worker fails then its **completed** and in-progress **map** tasks are re-executed, since its local disk is inaccessible.

MapReduce Framework

- Fault tolerance:
  - Necessary since thousands of nodes may be used.
  - The master pings the workers periodically. No answer means failure.
  - If a worker fails then its **completed** and in-progress **map** tasks are re-executed, since its local disk is inaccessible.
  - Note the importance of storing several copies (typically 3) of the input data on different nodes.

MapReduce Framework

- Fault tolerance:
  - Necessary since thousands of nodes may be used.
  - The master pings the workers periodically. No answer means failure.
  - If a worker fails then its **completed** and in-progress **map** tasks are re-executed, since its local disk is inaccessible.
  - Note the importance of storing several copies (typically 3) of the input data on different nodes.
  - If a worker fails then its in-progress **reduce** task is re-executed. The results of its completed reduce tasks are stored on the global file system and, thus, they are accessible.

MapReduce Framework

- Fault tolerance:
  - Necessary since thousands of nodes may be used.
  - The master pings the workers periodically. No answer means failure.
  - If a worker fails then its **completed** and in-progress **map** tasks are re-executed, since its local disk is inaccessible.
  - Note the importance of storing several copies (typically 3) of the input data on different nodes.
  - If a worker fails then its in-progress **reduce** task is re-executed. The results of its completed reduce tasks are stored on the global file system and, thus, they are accessible.
  - To be able to recover from the unlikely event of a master failure, the master periodically saves the state of the different tasks (idle, in-progress, completed) and the identity of the worker for the non-idle tasks.

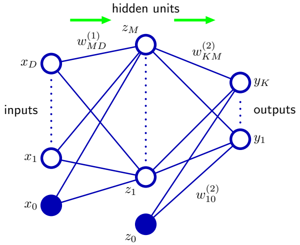
MapReduce Framework

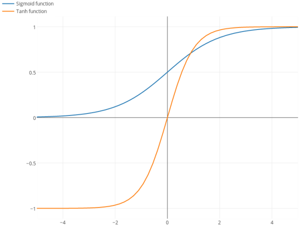
- Fault tolerance:
  - Necessary since thousands of nodes may be used.
  - The master pings the workers periodically. No answer means failure.
  - If a worker fails then its **completed** and in-progress **map** tasks are re-executed, since its local disk is inaccessible.
  - Note the importance of storing several copies (typically 3) of the input data on different nodes.
  - If a worker fails then its in-progress **reduce** task is re-executed. The results of its completed reduce tasks are stored on the global file system and, thus, they are accessible.
  - To be able to recover from the unlikely event of a master failure, the master periodically saves the state of the different tasks (idle, in-progress, completed) and the identity of the worker for the non-idle tasks.
- Task granularity:
  - M and R are larger than the number of nodes available.
  - Large M and R values benefit dynamic load balance and fast failure recovery.

MapReduce Framework

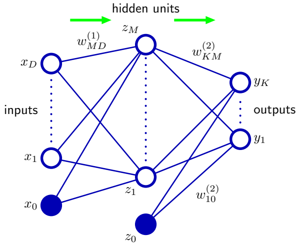
- Fault tolerance:
  - Necessary since thousands of nodes may be used.
  - The master pings the workers periodically. No answer means failure.
  - If a worker fails then its **completed** and in-progress **map** tasks are re-executed, since its local disk is inaccessible.
  - Note the importance of storing several copies (typically 3) of the input data on different nodes.
  - If a worker fails then its in-progress **reduce** task is re-executed. The results of its completed reduce tasks are stored on the global file system and, thus, they are accessible.
  - To be able to recover from the unlikely event of a master failure, the master periodically saves the state of the different tasks (idle, in-progress, completed) and the identity of the worker for the non-idle tasks.
- Task granularity:
  - M and R are larger than the number of nodes available.
  - Large M and R values benefit dynamic load balance and fast failure recovery.
  - Too large values may imply too many scheduling decisions, and too many output files.
  - For instance, $M = 200000$ and $R = 5000$ for 2000 available nodes.

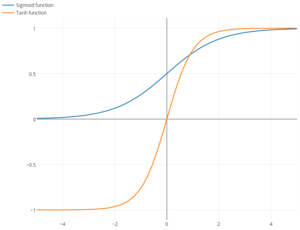
Machine Learning with MapReduce: Neural Networks





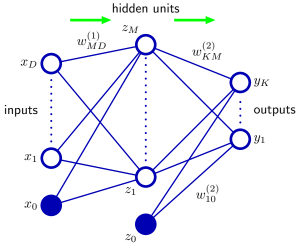
Machine Learning with MapReduce: Neural Networks

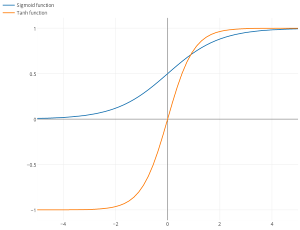




- Activations: $a_j = \sum_i w^{(1)}_{ji} x_i + w^{(1)}_{j0}$

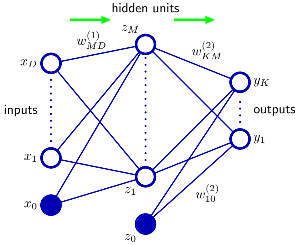
Machine Learning with MapReduce: Neural Networks

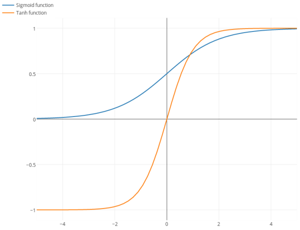




- Activations: $a_j = \sum_i w^{(1)}_{ji} x_i + w^{(1)}_{j0}$
- Hidden units and activation function: $z_j = h(a_j)$

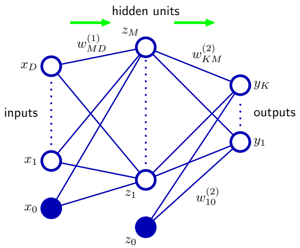
Machine Learning with MapReduce: Neural Networks

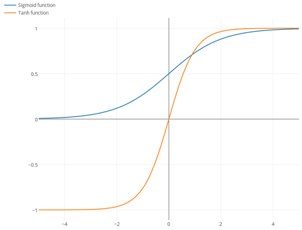




- Activations: $a_j = \sum_i w^{(1)}_{ji} x_i + w^{(1)}_{j0}$
- Hidden units and activation function: $z_j = h(a_j)$
- Output activations: $a_k = \sum_j w^{(2)}_{kj} z_j + w^{(2)}_{k0}$

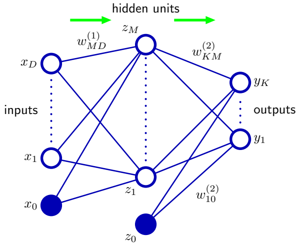
Machine Learning with MapReduce: Neural Networks

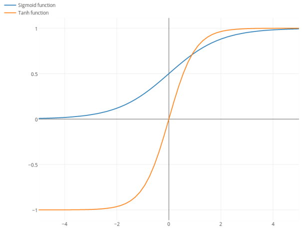




- Activations: $a_j = \sum_i w_{ji}^{(1)} x_i + w_{j0}^{(1)}$
- Hidden units and activation function: $z_j = h(a_j)$
- Output activations: $a_k = \sum_j w_{kj}^{(2)} z_j + w_{k0}^{(2)}$
- Output activation function for regression: $y_k(\mathrm{x}) = a_k$

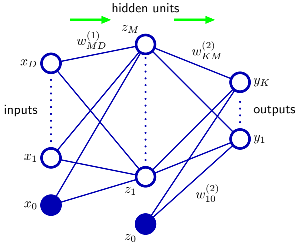
Machine Learning with MapReduce: Neural Networks

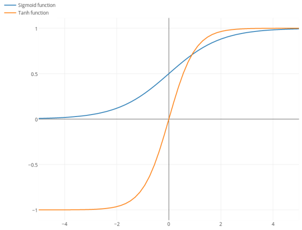




- Activations: $a_j = \sum_i w^{(1)}_{ji} x_i + w^{(1)}_{j0}$
- Hidden units and activation function: $z_j = h(a_j)$
- Output activations: $a_k = \sum_j w^{(2)}_{kj} z_j + w^{(2)}_{k0}$
- Output activation function for regression: $y_k(\mathrm{x}) = a_k$
- Output activation function for classification: $y_k(\mathrm{x}) = \sigma(a_k)$

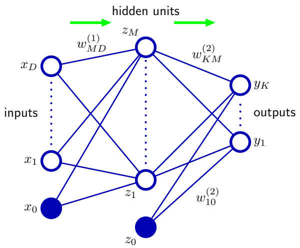
Machine Learning with MapReduce: Neural Networks

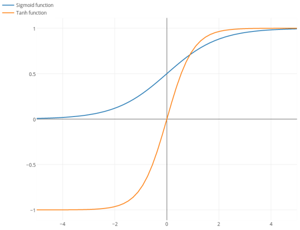




- Activations: $a_j = \sum_i w^{(1)}_{ji} x_i + w^{(1)}_{j0}$
- Hidden units and activation function: $z_j = h(a_j)$
- Output activations: $a_k = \sum_j w^{(2)}_{kj} z_j + w^{(2)}_{k0}$
- Output activation function for regression: $y_k(\mathrm{x}) = a_k$
- Output activation function for classification: $y_k(\mathrm{x}) = \sigma(a_k)$
- Sigmoid function: $\sigma(a) = \frac{1}{1+\exp(-a)}$

Machine Learning with MapReduce: Neural Networks

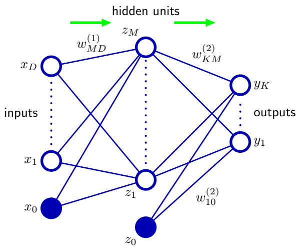


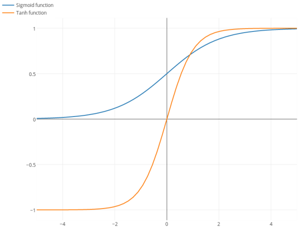


- Activations: $a_j = \sum_i w^{(1)}_{ji} x_i + w^{(1)}_{j0}$
- Hidden units and activation function: $z_j = h(a_j)$
- Output activations: $a_k = \sum_j w^{(2)}_{kj} z_j + w^{(2)}_{k0}$
- Output activation function for regression: $y_k(\mathrm{x}) = a_k$
- Output activation function for classification: $y_k(\mathrm{x}) = \sigma(a_k)$
- Sigmoid function: $\sigma(a) = \frac{1}{1+\exp(-a)}$
- Two-layer NN:

$$y_k(\mathrm{x}) = \sigma\Big(\sum_j w^{(2)}_{kj}\, h\Big(\sum_i w^{(1)}_{ji} x_i + w^{(1)}_{j0}\Big) + w^{(2)}_{k0}\Big)$$

Machine Learning with MapReduce: Neural Networks



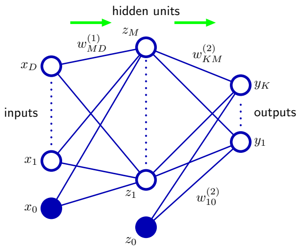


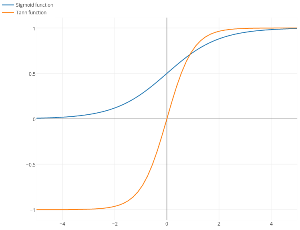
- Activations: $a_j = \sum_i w^{(1)}_{ji} x_i + w^{(1)}_{j0}$
- Hidden units and activation function: $z_j = h(a_j)$
- Output activations: $a_k = \sum_j w^{(2)}_{kj} z_j + w^{(2)}_{k0}$
- Output activation function for regression: $y_k(\mathrm{x}) = a_k$
- Output activation function for classification: $y_k(\mathrm{x}) = \sigma(a_k)$
- Sigmoid function: $\sigma(a) = \frac{1}{1+\exp(-a)}$
- Two-layer NN:

$$y_k(\mathrm{x}) = \sigma\Big(\sum_j w^{(2)}_{kj}\, h\Big(\sum_i w^{(1)}_{ji} x_i + w^{(1)}_{j0}\Big) + w^{(2)}_{k0}\Big)$$

- Evaluating the previous expression is known as forward propagation. The NN is said to have a feed-forward architecture.

Machine Learning with MapReduce: Neural Networks





- Activations: $a_j = \sum_i w_{ji}^{(1)} x_i + w_{j0}^{(1)}$
- Hidden units and activation function: $z_j = h(a_j)$
- Output activations: $a_k = \sum_j w_{kj}^{(2)} z_j + w_{k0}^{(2)}$
- Output activation function for regression: $y_k(\mathrm{x}) = a_k$
- Output activation function for classification: $y_k(\mathrm{x}) = \sigma(a_k)$
- Sigmoid function: $\sigma(a) = \frac{1}{1+\exp(-a)}$
- Two-layer NN:

$$y_k(\mathrm{x}) = \sigma\left(\sum_j w_{kj}^{(2)} h\left(\sum_i w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right)$$

- Evaluating the previous expression is known as forward propagation. The NN is said to have a feed-forward architecture.
- All the previous is, of course, generalizable to more layers.

Machine Learning with MapReduce: Neural Networks

- Consider regressing an K-dimensional continuous random variable on a D-dimensional continuous random variable.

Machine Learning with MapReduce: Neural Networks

- Consider regressing an K-dimensional continuous random variable on a D-dimensional continuous random variable.
- Consider a training set $\{(\boldsymbol{x}_n, \boldsymbol{t}_n)\}$ of size N. Consider minimizing the error function

$$E(\boldsymbol{w}) = \sum_n E_n(\boldsymbol{w}) = \sum_n \frac{1}{2}(\boldsymbol{y}(\boldsymbol{x}_n) - \boldsymbol{t}_n)^2 = \sum_n \sum_k \frac{1}{2}(y_k(\boldsymbol{x}_n) - t_{nk})^2$$

Machine Learning with MapReduce: Neural Networks

- Consider regressing an K-dimensional continuous random variable on a D-dimensional continuous random variable.
- Consider a training set $\{(\boldsymbol{x}_n, \boldsymbol{t}_n)\}$ of size N. Consider minimizing the error function

$$E(\boldsymbol{w}) = \sum_n E_n(\boldsymbol{w}) = \sum_n \frac{1}{2}(\boldsymbol{y}(\boldsymbol{x}_n) - \boldsymbol{t}_n)^2 = \sum_n \sum_k \frac{1}{2}(y_k(\boldsymbol{x}_n) - t_{nk})^2$$

- The weight space is highly multimodal and, thus, we have to resort to approximate iterative methods to minimize the previous expression.

Machine Learning with MapReduce: Neural Networks

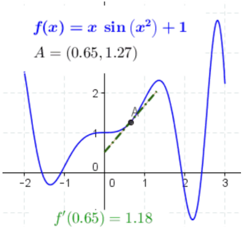
- Consider regressing an K-dimensional continuous random variable on a D-dimensional continuous random variable.
- Consider a training set $\{(\boldsymbol{x}_n, \boldsymbol{t}_n)\}$ of size N. Consider minimizing the error function

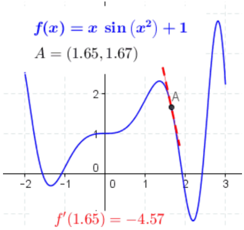
$$E(\boldsymbol{w}) = \sum_n E_n(\boldsymbol{w}) = \sum_n \frac{1}{2}(\boldsymbol{y}(\boldsymbol{x}_n) - \boldsymbol{t}_n)^2 = \sum_n \sum_k \frac{1}{2}(y_k(\boldsymbol{x}_n) - t_{nk})^2$$

- The weight space is highly multimodal and, thus, we have to resort to approximate iterative methods to minimize the previous expression.
- Batch gradient descent

$$\boldsymbol{w}^{t+1} = \boldsymbol{w}^t - \eta \nabla E(\boldsymbol{w}^t) = \boldsymbol{w}^t - \eta \sum_n \nabla E_n(\boldsymbol{w}^t)$$

where $\eta > 0$ is the learning rate, and $\nabla E_n(\boldsymbol{w}^t)$ can be computed efficiently thanks to the backpropagation algorithm.





Machine Learning with MapReduce: Neural Networks

- **Each iteration** of batch gradient descent can easily be casted into MapReduce terms:

Machine Learning with MapReduce: Neural Networks

- **Each iteration** of batch gradient descent can easily be casted into MapReduce terms:
  - Map function: Compute the gradient for a training point. Note that this implies forward and backward propagation.

Machine Learning with MapReduce: Neural Networks

- **Each iteration** of batch gradient descent can easily be casted into MapReduce terms:
  - Map function: Compute the gradient for a training point. Note that this implies forward and backward propagation.
  - Reduce function: Sum the partial gradients and update $\boldsymbol{w}$ accordingly.

Machine Learning with MapReduce: Neural Networks

- **Each iteration** of batch gradient descent can easily be casted into MapReduce terms:
  - Map function: Compute the gradient for a training point. Note that this implies forward and backward propagation.
  - Reduce function: Sum the partial gradients and update $\boldsymbol{w}$ accordingly.
- Note that $1 \leq M \leq N$, whereas $R = 1$.

Machine Learning with MapReduce: Neural Networks

- **Each iteration** of batch gradient descent can easily be casted into MapReduce terms:
  - Map function: Compute the gradient for a training point. Note that this implies forward and backward propagation.
  - Reduce function: Sum the partial gradients and update $\boldsymbol{w}$ accordingly.
- Note that $1 \leq M \leq N$, whereas $R = 1$.
- What is the key and what is the value ? What needs to be broadcasted ?

Machine Learning with MapReduce: Linear Support Vector Machines

- Consider binary classification with input space $\mathbb{R}^D$. Consider a training set $\{(\boldsymbol{x}_n, t_n)\}$ where $t_n \in \{-1, +1\}$. Consider using the linear model

$$y(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + b$$

so that a new point $\boldsymbol{x}$ is classified according to the sign of $y(\boldsymbol{x})$.

Machine Learning with MapReduce: Linear Support Vector Machines

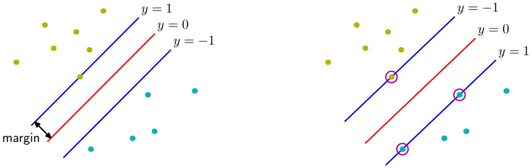
- Consider binary classification with input space $\mathbb{R}^D$. Consider a training set $\{(\boldsymbol{x}_n, t_n)\}$ where $t_n \in \{-1, +1\}$. Consider using the linear model

$$y(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + b$$

  so that a new point $\boldsymbol{x}$ is classified according to the sign of $y(\boldsymbol{x})$.

- If the training data is linearly separable, the separating hyperplane with the largest margin (i.e. the largest smallest perpendicular distance from any point to the hyperplane) is given by

$$\arg\min_{\boldsymbol{w}, b} \frac{1}{2} ||\boldsymbol{w}||^2$$



Machine Learning with MapReduce: Linear Support Vector Machines

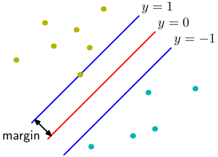
- Consider binary classification with input space $\mathbb{R}^D$. Consider a training set $\{(\boldsymbol{x}_n, t_n)\}$ where $t_n \in \{-1, +1\}$. Consider using the linear model

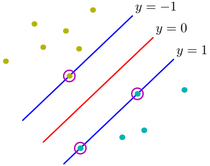
$$y(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + b$$

  so that a new point $\boldsymbol{x}$ is classified according to the sign of $y(\boldsymbol{x})$.

- If the training data is linearly separable, the separating hyperplane with the largest margin (i.e. the largest smallest perpendicular distance from any point to the hyperplane) is given by

$$\arg\min_{\boldsymbol{w}, b} \frac{1}{2} \|\boldsymbol{w}\|^2$$



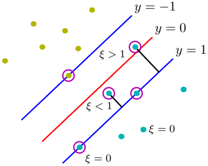


- The motivation is that the larger the margin, the smaller the generalization error.

Machine Learning with MapReduce: Linear Support Vector Machines

- Without the assumption of linearly separability and with a quadratic penalty for (almost-)misclassified points, the optimal separating hyperplane is given by

$$\arg\min_{\boldsymbol{w}} \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{n\in E}(\underbrace{\boldsymbol{w}^T\boldsymbol{x}_n - t_n}_{\xi_n})^2$$

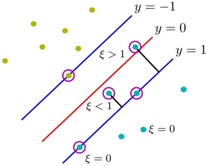


where C is a user-defined parameter, and $n \in E$ if and only if $t_n y(\boldsymbol{x}_n) < 1$.

Machine Learning with MapReduce: Linear Support Vector Machines

- Without the assumption of linearly separability and with a quadratic penalty for (almost-)misclassified points, the optimal separating hyperplane is given by

$$\arg\min_{\boldsymbol{w}} \frac{1}{2}||\boldsymbol{w}||^2 + C \sum_{n \in E} (\underbrace{\boldsymbol{w}^T \boldsymbol{x}_n - t_n}_{\xi_n})^2$$

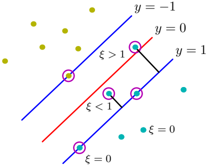


where C is a user-defined parameter, and $n \in E$ if and only if $t_n y(\boldsymbol{x}_n) < 1$.
- Note that the previous expression is a quadratic function and, thus, it is concave (up) and, thus, "easy" to minimize. For instance, we can use again batch gradient descent.

Machine Learning with MapReduce: Linear Support Vector Machines

- Without the assumption of linearly separability and with a quadratic penalty for (almost-)misclassified points, the optimal separating hyperplane is given by

$$\arg\min_{\boldsymbol{w}} \tfrac{1}{2}\|\boldsymbol{w}\|^2 + C \textstyle\sum_{n\in E}(\underbrace{\boldsymbol{w}^T\boldsymbol{x}_n - t_n}_{\xi_n})^2$$



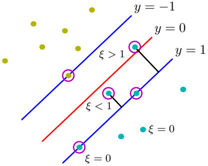
where C is a user-defined parameter, and $n \in E$ if and only if $t_n y(\boldsymbol{x}_n) < 1$.
- Note that the previous expression is a quadratic function and, thus, it is concave (up) and, thus, "easy" to minimize. For instance, we can use again batch gradient descent.
- The gradient is given by

$$\boldsymbol{w} + 2C\sum_{n\in E}(\boldsymbol{w}^T\boldsymbol{x}_n - t_n)\boldsymbol{x}_n$$

Machine Learning with MapReduce: Linear Support Vector Machines

- Without the assumption of linearly separability and with a quadratic penalty for (almost-)misclassified points, the optimal separating hyperplane is given by

$$\arg\min_{\boldsymbol{w}} \frac{1}{2}\|\boldsymbol{w}\|^2 + C \sum_{n \in E} (\underbrace{\boldsymbol{w}^T \boldsymbol{x}_n - t_n}_{\xi_n})^2$$



where C is a user-defined parameter, and $n \in E$ if and only if $t_n y(\boldsymbol{x}_n) < 1$.
- Note that the previous expression is a quadratic function and, thus, it is concave (up) and, thus, "easy" to minimize. For instance, we can use again batch gradient descent.
- The gradient is given by

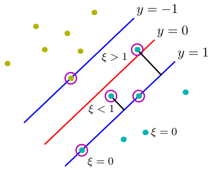
$$\boldsymbol{w} + 2C \sum_{n \in E} (\boldsymbol{w}^T \boldsymbol{x}_n - t_n) \boldsymbol{x}_n$$

- **Each iteration** of batch gradient descent can easily be casted into MapReduce terms:

Machine Learning with MapReduce: Linear Support Vector Machines

- Without the assumption of linearly separability and with a quadratic penalty for (almost-)misclassified points, the optimal separating hyperplane is given by

$$\arg\min_{\boldsymbol{w}} \frac{1}{2}||\boldsymbol{w}||^2 + C \sum_{n \in E} (\underbrace{\boldsymbol{w}^T \boldsymbol{x}_n - t_n}_{\xi_n})^2$$



  where C is a user-defined parameter, and $n \in E$ if and only if $t_n y(\boldsymbol{x}_n) < 1$.
- Note that the previous expression is a quadratic function and, thus, it is concave (up) and, thus, "easy" to minimize. For instance, we can use again batch gradient descent.
- The gradient is given by

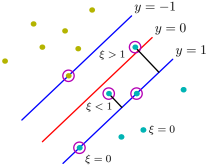
$$\boldsymbol{w} + 2C \sum_{n \in E} (\boldsymbol{w}^T \boldsymbol{x}_n - t_n) \boldsymbol{x}_n$$

- **Each iteration** of batch gradient descent can easily be casted into MapReduce terms:
  - Map function: Compute the gradient for a training point.

Machine Learning with MapReduce: Linear Support Vector Machines

- Without the assumption of linearly separability and with a quadratic penalty for (almost-)misclassified points, the optimal separating hyperplane is given by

$$\arg\min_{\boldsymbol{w}} \tfrac{1}{2}\|\boldsymbol{w}\|^2 + C \textstyle\sum_{n \in E} (\underbrace{\boldsymbol{w}^T \boldsymbol{x}_n - t_n}_{\xi_n})^2$$



  where C is a user-defined parameter, and $n \in E$ if and only if $t_n y(\boldsymbol{x}_n) < 1$.
- Note that the previous expression is a quadratic function and, thus, it is concave (up) and, thus, "easy" to minimize. For instance, we can use again batch gradient descent.
- The gradient is given by

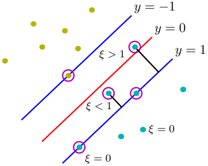
$$\boldsymbol{w} + 2C \sum_{n \in E} (\boldsymbol{w}^T \boldsymbol{x}_n - t_n)\boldsymbol{x}_n$$

- **Each iteration** of batch gradient descent can easily be casted into MapReduce terms:
  - Map function: Compute the gradient for a training point.
  - Reduce function: Sum the partial gradients and update $\boldsymbol{w}$ accordingly.

Machine Learning with MapReduce: Linear Support Vector Machines

- Without the assumption of linearly separability and with a quadratic penalty for (almost-)misclassified points, the optimal separating hyperplane is given by

$$\arg\min_{\boldsymbol{w}} \frac{1}{2}\|\boldsymbol{w}\|^2 + C \sum_{n \in E} (\underbrace{\boldsymbol{w}^T \boldsymbol{x}_n - t_n}_{\xi_n})^2$$



where C is a user-defined parameter, and $n \in E$ if and only if $t_n y(\boldsymbol{x}_n) < 1$.

- Note that the previous expression is a quadratic function and, thus, it is concave (up) and, thus, "easy" to minimize. For instance, we can use again batch gradient descent.
- The gradient is given by

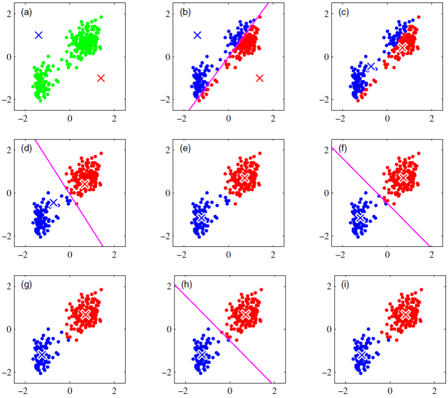
$$\boldsymbol{w} + 2C \sum_{n \in E} (\boldsymbol{w}^T \boldsymbol{x}_n - t_n) \boldsymbol{x}_n$$

- **Each iteration** of batch gradient descent can easily be casted into MapReduce terms:
  - Map function: Compute the gradient for a training point.
  - Reduce function: Sum the partial gradients and update $\boldsymbol{w}$ accordingly.
- Note that $1 \leq M \leq N$, whereas $R = 1$. What is the key and what is the value ? What needs to be broadcasted ?

Machine Learning with MapReduce: K-Means Algorithm

- Consider data clustering (a.k.a. unsupervised learning) via the K-means algorithm.

```
Assign each point to a cluster (a.k.a. subpopulation) at random
Compute the cluster centroids as the averages of the points assigned to each cluster
Repeat until the centroids do not change
    Assign each point to the cluster with the closest centroid
    Update the cluster centroids as the averages of the points assigned to each cluster
```



Machine Learning with MapReduce: K-Means Algorithm

```
Assign each point to a cluster (a.k.a. subpopulation) at random
Compute the cluster centroids as the averages of the points assigned to each cluster
Repeat until the centroids do not change
    Assign each point to the cluster with the closest centroid
    Update the cluster centroids as the averages of the points assigned to each cluster
```

- **Each iteration** of the K-means algorithm can easily be casted into MapReduce terms:

Machine Learning with MapReduce: K-Means Algorithm

```
Assign each point to a cluster (a.k.a. subpopulation) at random
Compute the cluster centroids as the averages of the points assigned to each cluster
Repeat until the centroids do not change
    Assign each point to the cluster with the closest centroid
    Update the cluster centroids as the averages of the points assigned to each cluster
```

- **Each iteration** of the K-means algorithm can easily be casted into MapReduce terms:
  - Map function: Assign a training point to the cluster with the closest centroid.

Machine Learning with MapReduce: K-Means Algorithm

```
Assign each point to a cluster (a.k.a. subpopulation) at random
Compute the cluster centroids as the averages of the points assigned to each cluster
Repeat until the centroids do not change
    Assign each point to the cluster with the closest centroid
    Update the cluster centroids as the averages of the points assigned to each cluster
```

- **Each iteration** of the K-means algorithm can easily be casted into MapReduce terms:
  - Map function: Assign a training point to the cluster with the closest centroid.
  - Reduce function: Recalculate the cluster centroids from the assignments of the map tasks.

Machine Learning with MapReduce: K-Means Algorithm

```
Assign each point to a cluster (a.k.a. subpopulation) at random
Compute the cluster centroids as the averages of the points assigned to each cluster
Repeat until the centroids do not change
    Assign each point to the cluster with the closest centroid
    Update the cluster centroids as the averages of the points assigned to each cluster
```

- **Each iteration** of the K-means algorithm can easily be casted into MapReduce terms:
  - Map function: Assign a training point to the cluster with the closest centroid.
  - Reduce function: Recalculate the cluster centroids from the assignments of the map tasks.
- Note that $1 \leq M \leq N$, whereas $R = 1$ or $\boldsymbol{R = K}$ depending on whether we decide to use the cluster assignment as intermediate key or not.

Machine Learning with MapReduce: K-Means Algorithm

```
Assign each point to a cluster (a.k.a. subpopulation) at random
Compute the cluster centroids as the averages of the points assigned to each cluster
Repeat until the centroids do not change
    Assign each point to the cluster with the closest centroid
    Update the cluster centroids as the averages of the points assigned to each cluster
```

- **Each iteration** of the K-means algorithm can easily be casted into MapReduce terms:
  - Map function: Assign a training point to the cluster with the closest centroid.
  - Reduce function: Recalculate the cluster centroids from the assignments of the map tasks.
- Note that $1 \leq M \leq N$, whereas $R = 1$ or $\boldsymbol{R = K}$ depending on whether we decide to use the cluster assignment as intermediate key or not.
- What is the key and what is the value ? What needs to be broadcasted ?

Machine Learning with MapReduce: EM Algorithm

- The K-means algorithm partitions the data, i.e. it hard-assigns instances to subpopulations. Model-based clustering on the other hand aims to soft-assign instances to the subpopulations by applying Bayes theorem as follows:

$$p(k|\boldsymbol{x}, \boldsymbol{\theta}, \boldsymbol{\pi}) = \frac{\pi_k p(\boldsymbol{x}|\boldsymbol{\theta}_k)}{\sum_k \pi_k p(\boldsymbol{x}|\boldsymbol{\theta}_k)}$$

  where $p(\boldsymbol{x}|\boldsymbol{\theta}_k)$ are called mixture components, and $\pi_k = p(k)$ are called mixing coefficients. A component models the data distribution for a chosen subpopulation, and a coefficient represents the probability of a subpopulation being chosen.

Machine Learning with MapReduce: EM Algorithm

- The K-means algorithm partitions the data, i.e. it hard-assigns instances to subpopulations. Model-based clustering on the other hand aims to soft-assign instances to the subpopulations by applying Bayes theorem as follows:

$$p(k|\boldsymbol{x},\boldsymbol{\theta},\boldsymbol{\pi}) = \frac{\pi_k p(\boldsymbol{x}|\boldsymbol{\theta}_k)}{\sum_k \pi_k p(\boldsymbol{x}|\boldsymbol{\theta}_k)}$$

  where $p(\boldsymbol{x}|\boldsymbol{\theta}_k)$ are called mixture components, and $\pi_k = p(k)$ are called mixing coefficients. A component models the data distribution for a chosen subpopulation, and a coefficient represents the probability of a subpopulation being chosen.
- More specifically, for components modeled as multivariate Gaussian distributions, we have that:

$$p(\boldsymbol{x}|\boldsymbol{\theta}_k) = \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k) = \frac{1}{2\pi^{D/2}}\frac{1}{|\boldsymbol{\Sigma}_k|^{1/2}} e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_k)^T\boldsymbol{\Sigma}_k^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_k)}.$$

- To solve model-based clustering, we have to estimate the model parameters $(\boldsymbol{\theta},\boldsymbol{\pi})$ from data. To this end, we use the EM algorithm.

Machine Learning with MapReduce: EM Algorithm

- Given a sample $\{\boldsymbol{x}_n\}$ of size N from a mixture of multivariate Gaussian distributions, the expected log likelihood function is maximized when

$$
\begin{aligned}
\pi_k^{ML} &= \frac{\sum_n p(z_{nk}|\boldsymbol{x}_n, \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma})}{N} \\
\boldsymbol{\mu}_k^{ML} &= \frac{\sum_n \boldsymbol{x}_n p(z_{nk}|\boldsymbol{x}_n, \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma})}{\sum_n p(z_{nk}|\boldsymbol{x}_n, \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma})} \\
\boldsymbol{\Sigma}_k^{ML} &= \frac{\sum_n (\boldsymbol{x}_n - \boldsymbol{\mu}_k^{ML})(\boldsymbol{x}_n - \boldsymbol{\mu}_k^{ML})^T p(z_{nk}|\boldsymbol{x}_n, \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma})}{\sum_n p(z_{nk}|\boldsymbol{x}_n, \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma})}
\end{aligned}
$$

  where $\boldsymbol{z}_n$ is a K-dimensional binary vector indicating component memberships (one-hot encoding):

$$
p(z_{nk}|\boldsymbol{x}_n, \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{p(\boldsymbol{x}_n|z_{nk}, \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) p(z_{nk}|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma})}{\sum_k p(\boldsymbol{x}_n|z_{nk}, \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) p(z_{nk}|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma})} = \frac{\pi_k p(\boldsymbol{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_k \pi_k p(\boldsymbol{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}
$$

Machine Learning with MapReduce: EM Algorithm

- Given a sample $\{\boldsymbol{x}_n\}$ of size N from a mixture of multivariate Gaussian distributions, the expected log likelihood function is maximized when

$$\begin{aligned}
\pi_k^{ML} &= \frac{\sum_n p(z_{nk}|\boldsymbol{x}_n, \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma})}{N} \\
\boldsymbol{\mu}_k^{ML} &= \frac{\sum_n \boldsymbol{x}_n p(z_{nk}|\boldsymbol{x}_n, \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma})}{\sum_n p(z_{nk}|\boldsymbol{x}_n, \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma})} \\
\boldsymbol{\Sigma}_k^{ML} &= \frac{\sum_n (\boldsymbol{x}_n - \boldsymbol{\mu}_k^{ML})(\boldsymbol{x}_n - \boldsymbol{\mu}_k^{ML})^T p(z_{nk}|\boldsymbol{x}_n, \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma})}{\sum_n p(z_{nk}|\boldsymbol{x}_n, \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma})}
\end{aligned}$$

  where $\boldsymbol{z}_n$ is a K-dimensional binary vector indicating component memberships (one-hot encoding):

$$p(z_{nk}|\boldsymbol{x}_n, \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{p(\boldsymbol{x}_n|z_{nk}, \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) p(z_{nk}|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma})}{\sum_k p(\boldsymbol{x}_n|z_{nk}, \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) p(z_{nk}|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma})} = \frac{\pi_k p(\boldsymbol{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_k \pi_k p(\boldsymbol{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}$$

- This is not a closed form solution, but it suggests the following algorithm.

Machine Learning with MapReduce: EM Algorithm

- Given a sample $\{\boldsymbol{x}_n\}$ of size N from a mixture of multivariate Gaussian distributions, the expected log likelihood function is maximized when

$$
\begin{aligned}
\pi_k^{ML} &= \frac{\sum_n p(z_{nk}|\boldsymbol{x}_n, \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma})}{N} \\
\boldsymbol{\mu}_k^{ML} &= \frac{\sum_n \boldsymbol{x}_n p(z_{nk}|\boldsymbol{x}_n, \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma})}{\sum_n p(z_{nk}|\boldsymbol{x}_n, \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma})} \\
\boldsymbol{\Sigma}_k^{ML} &= \frac{\sum_n (\boldsymbol{x}_n - \boldsymbol{\mu}_k^{ML})(\boldsymbol{x}_n - \boldsymbol{\mu}_k^{ML})^T p(z_{nk}|\boldsymbol{x}_n, \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma})}{\sum_n p(z_{nk}|\boldsymbol{x}_n, \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma})}
\end{aligned}
$$

  where $\boldsymbol{z}_n$ is a K-dimensional binary vector indicating component memberships (one-hot encoding):

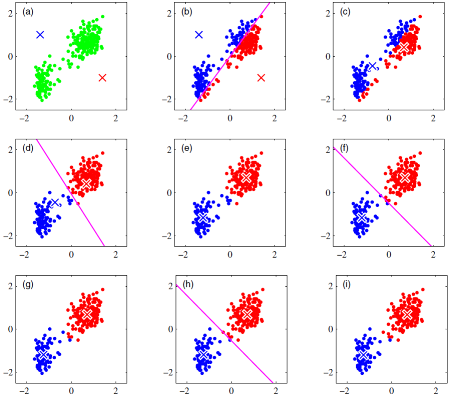
$$
p(z_{nk}|\boldsymbol{x}_n, \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{p(\boldsymbol{x}_n|z_{nk}, \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) p(z_{nk}|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma})}{\sum_k p(\boldsymbol{x}_n|z_{nk}, \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) p(z_{nk}|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma})} = \frac{\pi_k p(\boldsymbol{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_k \pi_k p(\boldsymbol{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}
$$

- This is not a closed form solution, but it suggests the following algorithm.

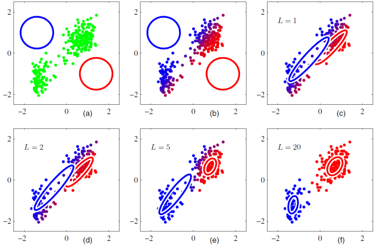
EM algorithm

Set $\boldsymbol{\pi}$, $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ to some initial values
Repeat until $\boldsymbol{\pi}$, $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ do not change
    Compute $p(z_{nk}|\boldsymbol{x}_n, \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma})$ for all n /* E step */
    Set π_k to π_k^{ML}, $\boldsymbol{\mu}_k$ to $\boldsymbol{\mu}_k^{ML}$, and $\boldsymbol{\Sigma}_k$ to $\boldsymbol{\Sigma}_k^{ML}$ for all k /* M step */

Machine Learning with MapReduce: EM Algorithm



K-means algorithm



EM algorithm

Machine Learning with MapReduce: EM Algorithm

- **Each iteration** of the EM algorithm can easily be casted into **two chained** MapReduce jobs:

Machine Learning with MapReduce: EM Algorithm

- **Each iteration** of the EM algorithm can easily be casted into **two chained** MapReduce jobs:
  - Map function I: For the n-th training point, compute

$$p(z_{nk}|\boldsymbol{x}_n, \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) \tag{1}$$

and

$$\boldsymbol{x}_n p(z_{nk}|\boldsymbol{x}_n, \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}). \tag{2}$$

Machine Learning with MapReduce: EM Algorithm

- **Each iteration** of the EM algorithm can easily be casted into **two chained** MapReduce jobs:
  - Map function I: For the *n*-th training point, compute

    $$p(z_{nk}|\boldsymbol{x}_n, \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) \tag{1}$$

    and

    $$\boldsymbol{x}_n p(z_{nk}|\boldsymbol{x}_n, \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}). \tag{2}$$

  - Reduce function I: Sum up the results (1) of the map tasks and divide it by N. Sum up the results (2) of the map tasks and divide it by the sum of the results (1). This gives $\boldsymbol{\pi}_k^{ML}$ and $\boldsymbol{\mu}_k^{ML}$.

Machine Learning with MapReduce: EM Algorithm

- **Each iteration** of the EM algorithm can easily be casted into **two chained** MapReduce jobs:
  - Map function I: For the n-th training point, compute
$$p(z_{nk}|\boldsymbol{x}_n, \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) \tag{1}$$
    and
$$\boldsymbol{x}_n p(z_{nk}|\boldsymbol{x}_n, \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}). \tag{2}$$
  - Reduce function I: Sum up the results (1) of the map tasks and divide it by N. Sum up the results (2) of the map tasks and divide it by the sum of the results (1). This gives $\boldsymbol{\pi}_k^{ML}$ and $\boldsymbol{\mu}_k^{ML}$.
  - Map function II: For the n-th training point, compute
$$p(z_{nk}|\boldsymbol{x}_n, \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) \tag{3}$$
    and
$$(\boldsymbol{x}_n - \boldsymbol{\mu}_k^{ML})(\boldsymbol{x}_n - \boldsymbol{\mu}_k^{ML})^T p(z_{nk}|\boldsymbol{x}_n, \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}). \tag{4}$$

Machine Learning with MapReduce: EM Algorithm

- **Each iteration** of the EM algorithm can easily be casted into **two chained** MapReduce jobs:
  - Map function I: For the n-th training point, compute

$$p(z_{nk}|\boldsymbol{x}_n, \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) \tag{1}$$

    and

$$\boldsymbol{x}_n p(z_{nk}|\boldsymbol{x}_n, \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}). \tag{2}$$

  - Reduce function I: Sum up the results (1) of the map tasks and divide it by N. Sum up the results (2) of the map tasks and divide it by the sum of the results (1). This gives $\boldsymbol{\pi}_k^{ML}$ and $\boldsymbol{\mu}_k^{ML}$.
  - Map function II: For the n-th training point, compute

$$p(z_{nk}|\boldsymbol{x}_n, \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) \tag{3}$$

    and

$$(\boldsymbol{x}_n - \boldsymbol{\mu}_k^{ML})(\boldsymbol{x}_n - \boldsymbol{\mu}_k^{ML})^T p(z_{nk}|\boldsymbol{x}_n, \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}). \tag{4}$$

  - Reduce function II: Sum up the results (4) of the map tasks and divide it by the sum of the results (3). This gives $\boldsymbol{\Sigma}_k^{ML}$.

Machine Learning with MapReduce: EM Algorithm

- **Each iteration** of the EM algorithm can easily be casted into **two chained** MapReduce jobs:
  - Map function I: For the n-th training point, compute
$$p(z_{nk}|\boldsymbol{x}_n, \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) \tag{1}$$
    and
$$\boldsymbol{x}_n p(z_{nk}|\boldsymbol{x}_n, \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}). \tag{2}$$
  - Reduce function I: Sum up the results (1) of the map tasks and divide it by N. Sum up the results (2) of the map tasks and divide it by the sum of the results (1). This gives $\boldsymbol{\pi}_k^{ML}$ and $\boldsymbol{\mu}_k^{ML}$.
  - Map function II: For the n-th training point, compute
$$p(z_{nk}|\boldsymbol{x}_n, \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) \tag{3}$$
    and
$$(\boldsymbol{x}_n - \boldsymbol{\mu}_k^{ML})(\boldsymbol{x}_n - \boldsymbol{\mu}_k^{ML})^T p(z_{nk}|\boldsymbol{x}_n, \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}). \tag{4}$$
  - Reduce function II: Sum up the results (4) of the map tasks and divide it by the sum of the results (3). This gives $\boldsymbol{\Sigma}_k^{ML}$.
- Note that $1 \leq M \leq N$, whereas $R = 1$ or $\boldsymbol{R} = \boldsymbol{K}$ in both jobs, depending on whether we decide to use the component index as intermediate key or not. What is the key and what is the value ? What needs to be broadcasted ?

Machine Learning with MapReduce

Data Sets	samples (m)	features (n)
Adult	30162	14
Helicopter Control	44170	21
Corel Image Features	68040	32
IPUMS Census	88443	61
Synthetic Time Series	100001	10
Census Income	199523	40
ACIP Sensor	229564	8
KDD Cup 99	494021	41
Forest Cover Type	581012	55
1990 US Census	2458285	68

	lwlr	gda	nb	logistic	pca	ica	svm	nn	kmeans	em
Adult	1.922	1.801	1.844	1.962	1.809	1.857	1.643	1.825	1.947	1.854
Helicopter	1.93	2.155	1.924	1.92	1.791	1.856	1.744	1.847	1.857	1.86
Corel Image	1.96	1.876	2.002	1.929	1.97	1.936	1.754	2.018	1.921	1.832
IPUMS	1.963	2.23	1.965	1.938	1.965	2.025	1.799	1.974	1.957	1.984
Synthetic	1.909	1.964	1.972	1.92	1.842	1.907	1.76	1.902	1.888	1.804
Census Income	1.975	2.179	1.967	1.941	2.019	1.941	1.88	1.896	1.961	1.99
Sensor	1.927	1.853	2.01	1.913	1.955	1.893	1.803	1.914	1.953	1.949
KDD	1.969	2.216	1.848	1.927	2.012	1.998	1.946	1.899	1.973	1.979
Cover Type	1.961	2.232	1.951	1.935	2.007	2.029	1.906	1.887	1.963	1.991
Census	2.327	2.292	2.008	1.906	1.997	2.001	1.959	1.883	1.946	1.977
avg.	1.985	2.080	1.950	1.930	1.937	1.944	1.819	1.905	1.937	1.922

Table 3: Speedups achieved on a dual core processor, without load time. Numbers reported are dual-core time / single-core time. Super linear speedup sometimes occurs due to a reduction in processor idle time with multiple threads.

Machine Learning with MapReduce

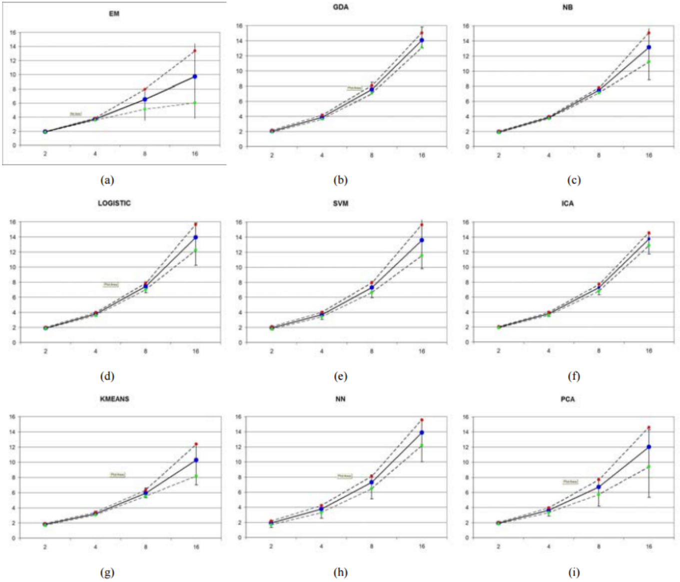


Figure 2: (a)-(i) show the speedup from 1 to 16 processors of all the algorithms over all the data sets. The Bold line is the average, error bars are the max and min speedups and the dashed lines are the variance.

Summary

- MapReduce is a framework to process large datasets by parallelizing computations.

Summary

- MapReduce is a framework to process large datasets by parallelizing computations.
- The user only has to specify the map and reduce functions, and parallelization happens automatically.

Summary

- MapReduce is a framework to process large datasets by parallelizing computations.
- The user only has to specify the map and reduce functions, and parallelization happens automatically.
- Many machine learning algorithms (e.g. SVMs, NNs, MMs, K-means and EM algorithms) can easily be reformulated in terms of such functions.

Summary

- MapReduce is a framework to process large datasets by parallelizing computations.
- The user only has to specify the map and reduce functions, and parallelization happens automatically.
- Many machine learning algorithms (e.g. SVMs, NNs, MMs, K-means and EM algorithms) can easily be reformulated in terms of such functions.
- This does not apply for algorithms based on stochastic gradient descent.

Summary

- MapReduce is a framework to process large datasets by parallelizing computations.
- The user only has to specify the map and reduce functions, and parallelization happens automatically.
- Many machine learning algorithms (e.g. SVMs, NNs, MMs, K-means and EM algorithms) can easily be reformulated in terms of such functions.
- This does not apply for algorithms based on stochastic gradient descent.
- Moreover, MapReduce is **inefficient** for iterative tasks on the same dataset: Each iteration is a MapReduce call that loads the data anew from disk.

Summary

- MapReduce is a framework to process large datasets by parallelizing computations.
- The user only has to specify the map and reduce functions, and parallelization happens automatically.
- Many machine learning algorithms (e.g. SVMs, NNs, MMs, K-means and EM algorithms) can easily be reformulated in terms of such functions.
- This does not apply for algorithms based on stochastic gradient descent.
- Moreover, MapReduce is **inefficient** for iterative tasks on the same dataset: Each iteration is a MapReduce call that loads the data anew from disk.
- Such iterative tasks are common in many machine learning algorithms, e.g. gradient descent, K-means and EM algorithms.

Summary

- MapReduce is a framework to process large datasets by parallelizing computations.
- The user only has to specify the map and reduce functions, and parallelization happens automatically.
- Many machine learning algorithms (e.g. SVMs, NNs, MMs, K-means and EM algorithms) can easily be reformulated in terms of such functions.
- This does not apply for algorithms based on stochastic gradient descent.
- Moreover, MapReduce is **inefficient** for iterative tasks on the same dataset: Each iteration is a MapReduce call that loads the data anew from disk.
- Such iterative tasks are common in many machine learning algorithms, e.g. gradient descent, K-means and EM algorithms.
- Solution: Spark framework, in the next lecture.