HPC and Data Analytics

Riccardo Zanella – `r.zanella@cineca.it`

SuperComputing Applications and Innovation Department

Table of Contents

Learning Algorithm

Definition of Learning Algorithm [Mitchell 1997][1]

A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

Learning Algorithm

Definition of Learning Algorithm [Mitchell 1997][1]

A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

So we need to identify:

- the class of tasks T

Learning Algorithm

Definition of Learning Algorithm [Mitchell 1997][1]

A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

So we need to identify:

- the class of tasks T
- the measure of performance P

Learning Algorithm

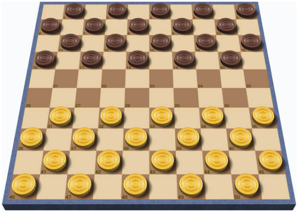
Definition of Learning Algorithm [Mitchell 1997][1]

A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

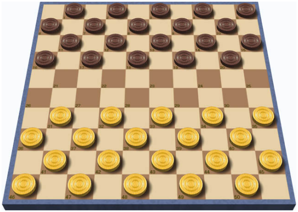
So we need to identify:

- the class of tasks T
- the measure of performance P
- the source of experience E

Example: checkers game

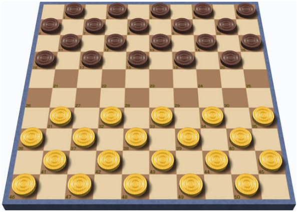


Example: checkers game



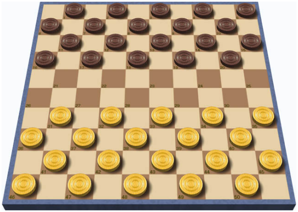
- **task class T:** playing checkers

Example: checkers game



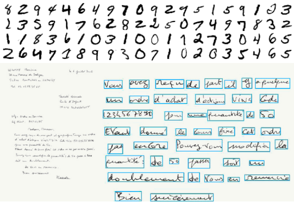
- **task class T:** playing checkers
- **performance measure P:** fraction of games won against opponents

Example: checkers game

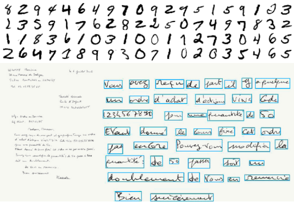


- **task class T:** playing checkers
- **performance measure P:** fraction of games won against opponents
- **training experience E:** playing practice games against itself

Example: handwritten characters recognition

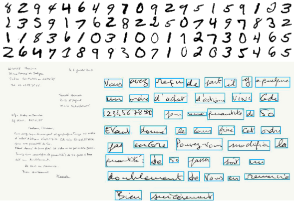


Example: handwritten characters recognition



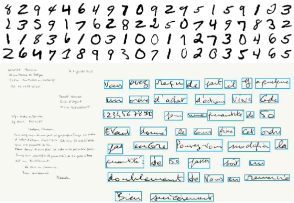
- **task class T:** recognizing and classifying handwritten characters within images

Example: handwritten characters recognition



- **task class T:** recognizing and classifying handwritten characters within images
- **performance measure P:** fraction of characters correctly classified

Example: handwritten characters recognition



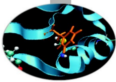
- **task class T:** recognizing and classifying handwritten characters within images
- **performance measure P:** fraction of characters correctly classified
- **training experience E:** a database of handwritten characters with given classifications

Example: supervised learning

- **training experience E:** a number of traininig examples $E = \{z_1, z_2, z_3 \dots\}$
  each example is a (input,target) pair: $Z_i = (X_i, Y_i)$

Example: supervised learning

- **training experience E:** a number of traininig examples $E = \{z_1, z_2, z_3 \dots\}$
  each example is a (input,target) pair: $Z_i = (X_i, Y_i)$
- **task class T:** a decision function f able to predict unknown Y from known X



Example: supervised learning

- **training experience E:** a number of traininig examples $E = \{z_1, z_2, z_3 \dots\}$

  each example is a (input,target) pair: $Z_i = (X_i, Y_i)$
- **task class T:** a decision function f able to predict unknown Y from known X
- **performance measure P:** a loss function L to measure the (non-symmetric) distance $L(f, Z_i) = d(f(X_i), Y_i)$

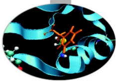
Example: supervised learning

- **training experience E:** a number of traininig examples $E = \{z_1, z_2, z_3 \dots\}$
  each example is a (input,target) pair: $Z_i = (X_i, Y_i)$
- **task class T:** a decision function f able to predict unknown Y from known X
- **performance measure P:** a loss function L to measure the (non-symmetric) distance $L(f, Z_i) = d(f(X_i), Y_i)$

Examples:

- regression
  - X is a real-valued scalar or vector
  - Y is a scalar real value
  - f is able to predict Y_i value from X_i
  - L is usually the euclidean norm

Example: supervised learning

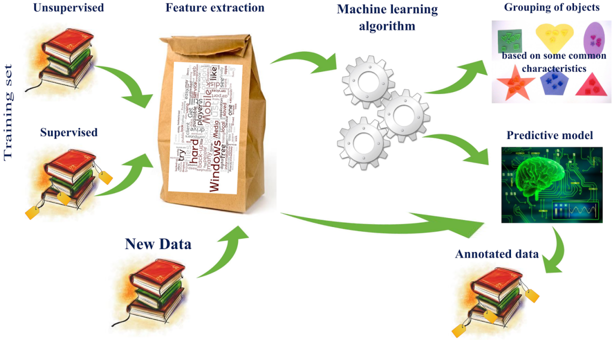


- **training experience E:** a number of traininig examples $E = \{z_1, z_2, z_3 \ldots\}$
  each example is a (input,target) pair: $Z_i = (X_i, Y_i)$
- **task class T:** a decision function f able to predict unknown Y from known X
- **performance measure P:** a loss function L to measure the (non-symmetric) distance $L(f, Z_i) = d(f(X_i), Y_i)$

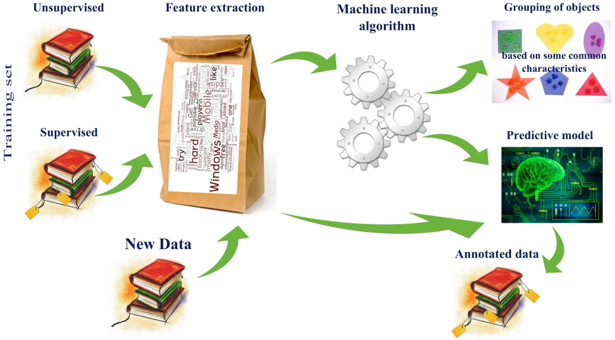
Examples:

- regression
  - X is a real-valued scalar or vector
  - Y is a scalar real value
  - f is able to predict Y_i value from X_i
  - L is usually the euclidean norm
- classification
  - X is a real-valued scalar or vector (features)
  - Y is an integer (label) corresponding to a class index
  - f is able to provide the probability of X_i being in class Y_i
  - L is usually the negative log-likelihood

Machine Learning

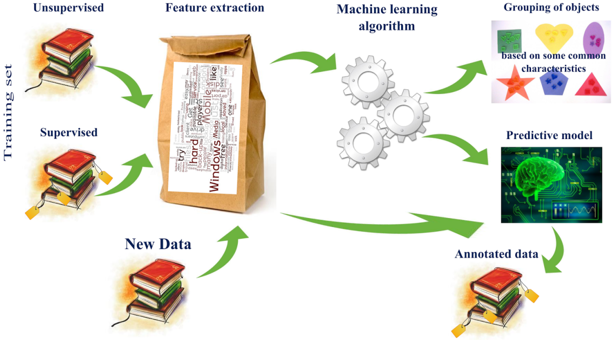


Machine Learning



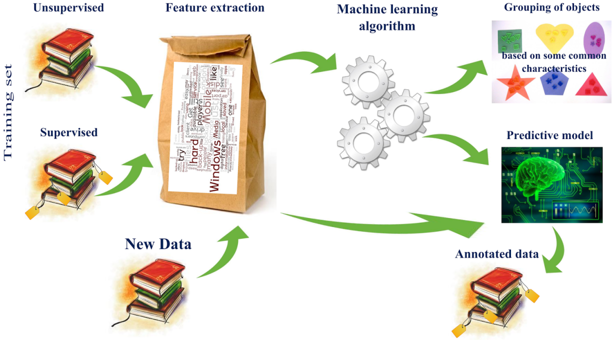
- Application of computer-enabled algorithm to a data set to find a pattern

Machine Learning



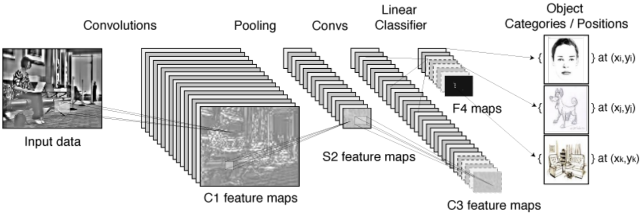
- Application of computer-enabled algorithm to a data set to find a pattern
- Wide range of tasks: segmentation. classification, clustering, supervised/unsupervised learning

Machine Learning

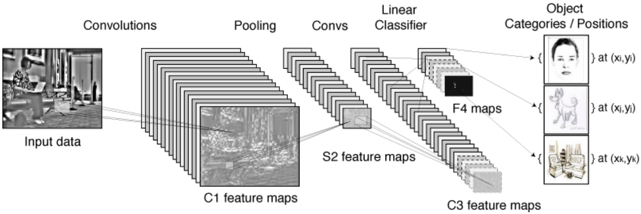


- Application of computer-enabled algorithm to a data set to find a pattern
- Wide range of tasks: segmentation. classification, clustering, supervised/unsupervised learning
- Various algorithms: association rules, decision trees, SVM

Deep Learning

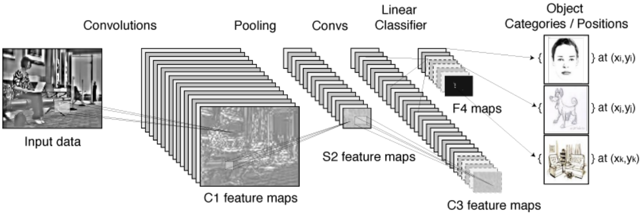


Deep Learning



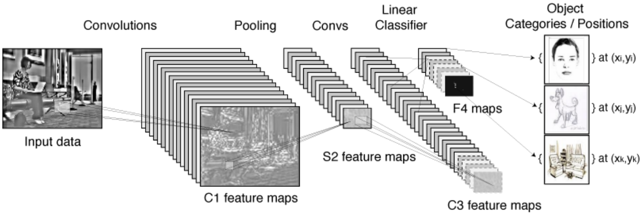
- Application of an Artificial Neural Network to a data set to find a pattern

Deep Learning



- Application of an Artificial Neural Network to a data set to find a pattern
- Multiple hidden layers (to mimic human brain processes associated to vision/hearing)

Deep Learning



- Application of an Artificial Neural Network to a data set to find a pattern
- Multiple hidden layers (to mimic human brain processes associated to vision/hearing)
- Big data sets and relevant number of variables

Framework desired features

We are interested in:

- classical machine learning algorithms
- deep learning approach (especially convolutional neural network)

Framework desired features

We are interested in:

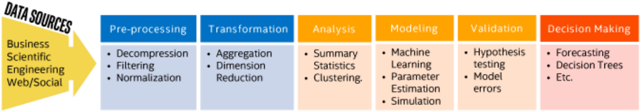
- classical machine learning algorithms
- deep learning approach (especially convolutional neural network)
- high level language (Python)
- little/no programming effort

Framework desired features

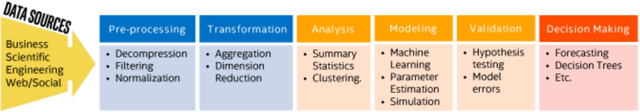
We are interested in:

- classical machine learning algorithms
- deep learning approach (especially convolutional neural network)
- high level language (Python)
- little/no programming effort
- integration with existing pipelines
- multi-core CPU and/or many-core GPU support

Intel Data Analytics Acceleration Library (DAAL)

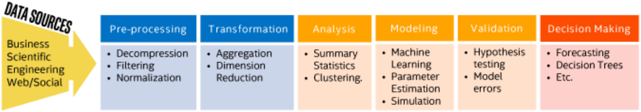


Intel Data Analytics Acceleration Library (DAAL)



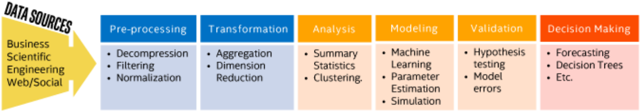
- Functions for machine learning, deep learning, data analytics

Intel Data Analytics Acceleration Library (DAAL)



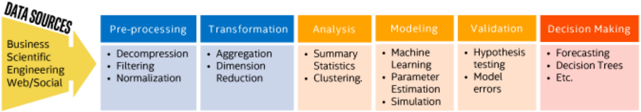
- Functions for machine learning, deep learning, data analytics
- Optimized for Intel architecture devices (processors, coprocessors, and compatibles)

Intel Data Analytics Acceleration Library (DAAL)



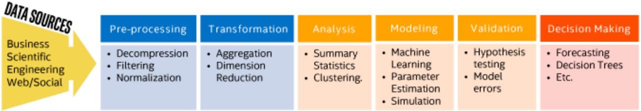
- Functions for machine learning, deep learning, data analytics
- Optimized for Intel architecture devices (processors, coprocessors, and compatibles)
- C++, Java and Python APIs

Intel Data Analytics Acceleration Library (DAAL)



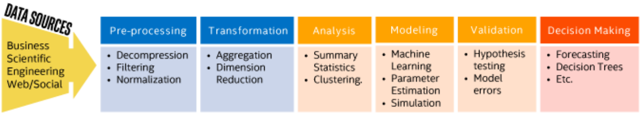
- Functions for machine learning, deep learning, data analytics
- Optimized for Intel architecture devices (processors, coprocessors, and compatibles)
- C++, Java and Python APIs
- Connectors to popular data sources including Spark and Hadoop

Intel Data Analytics Acceleration Library (DAAL)



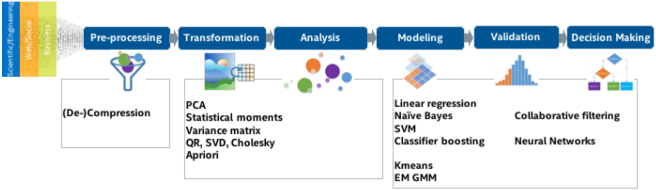
- Functions for machine learning, deep learning, data analytics
- Optimized for Intel architecture devices (processors, coprocessors, and compatibles)
- C++, Java and Python APIs
- Connectors to popular data sources including Spark and Hadoop
- Open source version under Apache 2.0 license

Intel Data Analytics Acceleration Library (DAAL)

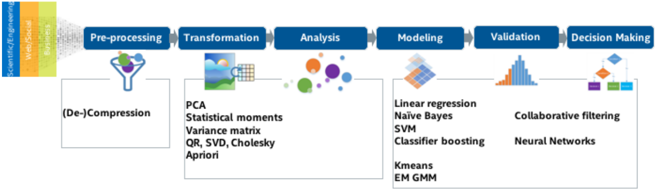


- Functions for machine learning, deep learning, data analytics
- Optimized for Intel architecture devices (processors, coprocessors, and compatibles)
- C++, Java and Python APIs
- Connectors to popular data sources including Spark and Hadoop
- Open source version under Apache 2.0 license
- Paid versions include premium support.

Algorithms

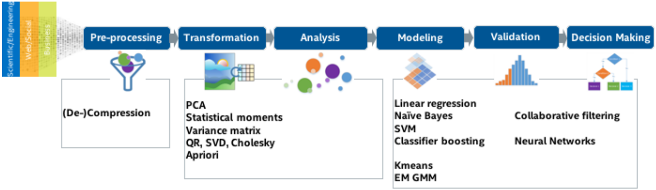

Scientific/Engineering
Web/Social
Business
Pre-processing
Transformation
Analysis
Modeling
Validation
Decision Making
(De-)Compression
PCA
Statistical moments
Variance matrix
QR, SVD, Cholesky
Apriori
Linear regression
Naïve Bayes
SVM
Classifier boosting
Kmeans
EM GMM
Collaborative filtering
Neural Networks

Algorithms



Statistics: min, max, mean, standard deviation, correlation, covariance matrix, correlation distance matrix, cosine distance matrix

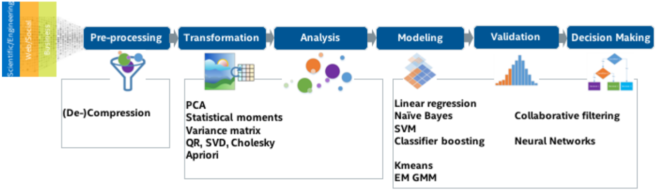
Algorithms



Statistics: min, max, mean, standard deviation, correlation, covariance matrix, correlation distance matrix, cosine distance matrix

Factorizations: Cholesky, QR, SVD

Algorithms

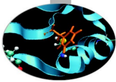


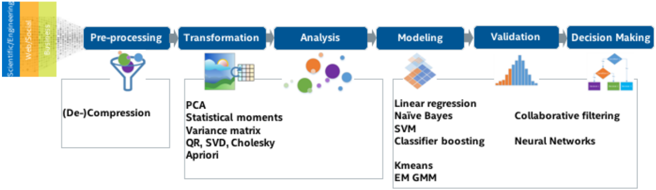
Statistics: min, max, mean, standard deviation, correlation, covariance matrix, correlation distance matrix, cosine distance matrix

Factorizations: Cholesky, QR, SVD

Dimensionality Reduction: PCA

Algorithms





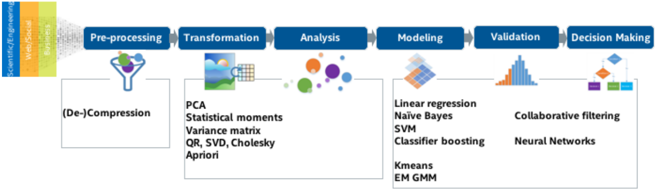
Statistics: min, max, mean, standard deviation, correlation, covariance matrix, correlation distance matrix, cosine distance matrix

Factorizations: Cholesky, QR, SVD

Dimensionality Reduction: PCA

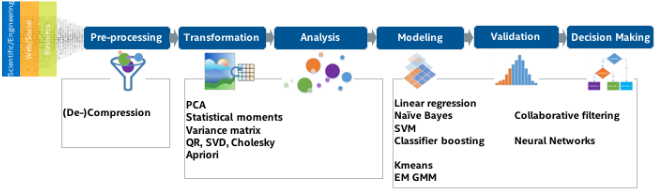
Classification: Naive Bayes, K-Nearest Neighbors, SVM, multiclass classification

Algorithms



- **Statistics:** min, max, mean, standard deviation, correlation, covariance matrix, correlation distance matrix, cosine distance matrix
- **Factorizations:** Cholesky, QR, SVD
- **Dimensionality Reduction:** PCA
- **Classification:** Naive Bayes, K-Nearest Neighbors, SVM, multiclass classification
- **Neural Networks:** layers of type: fully-connected, activation, convolutional, normalization, concat, split, softmax, loss function

Algorithms



Statistics: min, max, mean, standard deviation, correlation, covariance matrix, correlation distance matrix, cosine distance matrix

Factorizations: Cholesky, QR, SVD

Dimensionality Reduction: PCA

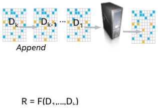
Classification: Naive Bayes, K-Nearest Neighbors, SVM, multiclass classification

Neural Networks: layers of type: fully-connected, activation, convolutional, normalization, concat, split, softmax, loss function

Clustering: K-Means, EM for GMM

Processing Modes

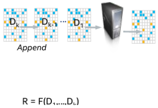
Processing Modes



Batch processing

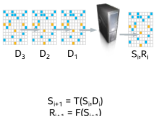
All data is stored in the memory of a single node. An Intel DAAL function is called to process the data all at once.

Processing Modes



Batch processing

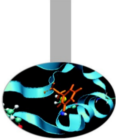
All data is stored in the memory of a single node. An Intel DAAL function is called to process the data all at once.

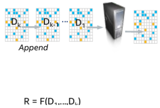


Streaming processing

All data does not fit in memory, or when data is arriving piece by piece. Intel DAAL can process data chunks individually and combine all partial results at the finalizing stage.

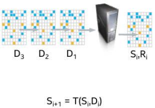
Processing Modes





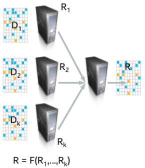
Batch processing

All data is stored in the memory of a single node. An Intel DAAL function is called to process the data all at once.



Streaming processing

All data does not fit in memory, or when data is arriving piece by piece. Intel DAAL can process data chunks individually and combine all partial results at the finalizing stage.

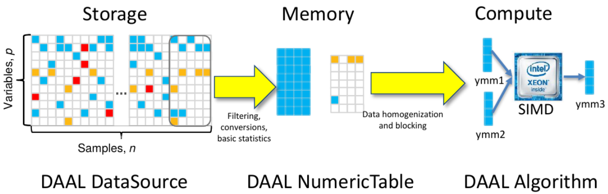


Distributed processing

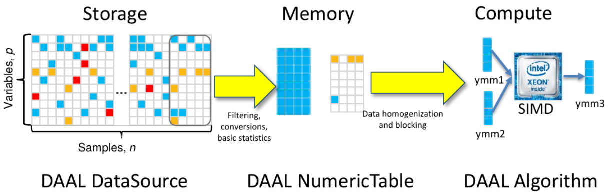
Intel DAAL supports a model similar to MapReduce. Slaves in a cluster process local data (map stage), and then the master process collects and combines partial results from slaves (reduce stage).

SuperComputing Applications and Innovation

DAAL data flow



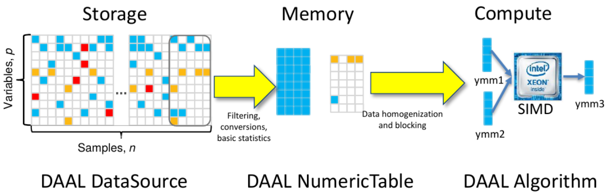
DAAL data flow



Data sources:

- file based (CSV, binary)
- database query (ODBC, SQL)
- Python: numpy array interoperability

DAAL data flow



Data sources:

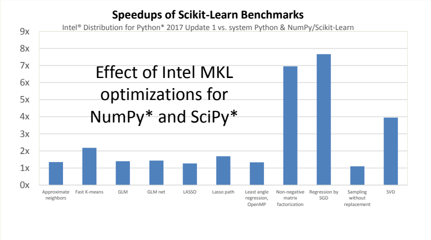
- file based (CSV, binary)
- database query (ODBC, SQL)
- Python: numpy array interoperability

Data structures:

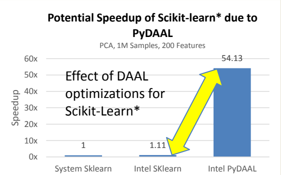
- numeric tables
  - homogeneous data: dense, sparse, packed, triangular matrix, symmetric matrix
  - heterogeneous data: SOA vs AOS
- tensors (n-dimensional matrices)

Official Intel benchmark results (I)

Skt-Learn* Optimizations With Intel® MKL… And Intel® DAAL



Intel® Distribution for Python* ships Intel® Data Analytics Acceleration Library with Python interfaces, a.k.a. pyDAAL



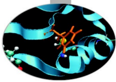
System info: 32x Intel® Xeon® CPU E5-2698 v3 @ 2.30GHz, disabled HT, 64GB RAM; Intel® Distribution for Python* 2017 Gold; Intel® MKL 2017.0.0; Ubuntu 14.04.4 LTS; Numpy 1.11.1; scikit-learn 0.17.1.

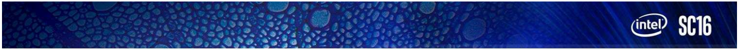
Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. * Other brands and names are the property of their respective owners. Benchmark Source: Intel Corporation

Optimization Notice: Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice revision #20110804 .

[Supercomputing 2016 (SC16), November 13-18, 2016, Salt Lake City]

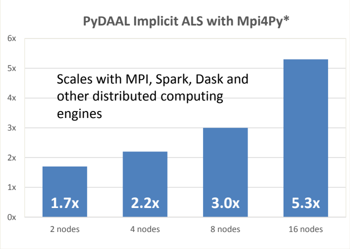
Official Intel benchmark results (II)





Distributed Parallelism

- Intel® MPI* accelerates Intel® Distribution for Python (mpi4py*, ipyparallel*)
- Intel Distribution for Python also supports
  - PySpark* - Python* interfaces for Spark*, an engine for large-scale data processing
  - Dask* - flexible parallel computing library for numerical computing



Configuration Info: Hardware (each node): Intel(R) Xeon(R) CPU E5-2697 v4 @ 2.30GHz, 2x18 cores, HT is ON, RAM 128GB; Versions: Oracle Linux Server 6.6, Intel® DAAL 2017 Gold, Intel® MPI 5.1.3; Interconnect: 1 GB Ethernet.

[Supercomputing 2016 (SC16), November 13-18, 2016, Salt Lake City]

pyDAAL installation

Requirements:

- Intel Math Kernel Library (MKL): for BLAS and LAPACK
- Integrated Performance Primitives (IPP) for data compression/decompression
- Threading Building Blocks (TBB) for multicore and many-core parallelism

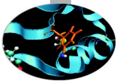
pyDAAL installation

Requirements:

- Intel Math Kernel Library (MKL): for BLAS and LAPACK
- Integrated Performance Primitives (IPP) for data compression/decompression
- Threading Building Blocks (TBB) for multicore and many-core parallelism

Installation methods:

1. anaconda Intel channel (Linux)
2. Intel distribution (Windows, Linux, OS X)
3. build from source



SVM multiclass classification in 11 steps

```
import numpy as np

# load digits dataset
from sklearn import datasets
digits = datasets.load_digits()

# define training set size
n_samples = len(digits.images)
n_training = int( 0.9 * n_samples )

data = np.ascontiguousarray( digits.data, dtype=np.double )
labels = np.ascontiguousarray( digits.target.reshape(n_samples, 1),
                               dtype=np.double )

from daal.data_management import HomogenNumericTable

train_data = HomogenNumericTable( data[:n_training] )
train_labels = HomogenNumericTable( labels[:n_training] )

test_data = HomogenNumericTable( data[n_training:] )
```

SVM multiclass classification in 11 steps

```
import numpy as np

# load digits dataset
from sklearn import datasets
digits = datasets.load_digits()

# define training set size
n_samples = len(digits.images)
n_training = int( 0.9 * n_samples )

data = np.ascontiguousarray( digits.data, dtype=np.double )
labels = np.ascontiguousarray( digits.target.reshape(n_samples, 1),
                               dtype=np.double )

from daal.data_management import HomogenNumericTable

train_data = HomogenNumericTable( data[:n_training] )
train_labels = HomogenNumericTable( labels[:n_training] )

test_data = HomogenNumericTable( data[n_training:] )
```

1. enjoy sklearn datasets import module

SVM multiclass classification in 11 steps

```
import numpy as np

# load digits dataset
from sklearn import datasets
digits = datasets.load_digits()

# define training set size
n_samples = len(digits.images)
n_training = int( 0.9 * n_samples )

data = np.ascontiguousarray( digits.data, dtype=np.double )
labels = np.ascontiguousarray( digits.target.reshape(n_samples, 1),
                              dtype=np.double )

from daal.data_management import HomogenNumericTable

train_data = HomogenNumericTable( data[:n_training] )
train_labels = HomogenNumericTable( labels[:n_training] )

test_data = HomogenNumericTable( data[n_training:] )
```

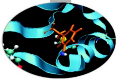
1. enjoy sklearn datasets import module
2. require a contiguous array

SVM multiclass classification in 11 steps

```
import numpy as np

# load digits dataset
from sklearn import datasets
digits = datasets.load_digits()

# define training set size
n_samples = len(digits.images)
n_training = int( 0.9 * n_samples )

data = np.ascontiguousarray( digits.data, dtype=np.double )
labels = np.ascontiguousarray( digits.target.reshape(n_samples, 1),
                               dtype=np.double )

from daal.data_management import HomogenNumericTable

train_data = HomogenNumericTable( data[:n_training] )
train_labels = HomogenNumericTable( labels[:n_training] )

test_data = HomogenNumericTable( data[n_training:] )
```

1. enjoy sklearn datasets import module
2. require a contiguous array
3. create instances of HomogenNumericTable

training algorithm setup



```
from daal.algorithms.svm import training as svm_training
from daal.algorithms.svm import prediction as svm_prediction
from daal.algorithms.multi_class_classifier import training as
     multiclass_training
from daal.algorithms.classifier import training as training_params

kernel = rbf.Batch_Float64DefaultDense()
kernel.parameter.sigma = 0.001

# Create two class svm classifier
# training alg
twoclass_train_alg = svm_training.Batch_Float64DefaultDense()
twoclass_train_alg.parameter.kernel = kernel
twoclass_train_alg.parameter.C = 1.0
# prediction alg
twoclass_predict_alg = svm_prediction.Batch_Float64DefaultDense()
twoclass_predict_alg.parameter.kernel = kernel

# Create a multiclass classifier object (training)
train_alg = multiclass_training.Batch_Float64OneAgainstOne()
train_alg.parameter.nClasses = 10
train_alg.parameter.training = twoclass_train_alg
train_alg.parameter.prediction = twoclass_predict_alg
```

training algorithm setup

```
from daal.algorithms.svm import training as svm_training
from daal.algorithms.svm import prediction as svm_prediction
from daal.algorithms.multi_class_classifier import training as
     multiclass_training
from daal.algorithms.classifier import training as training_params

kernel = rbf.Batch_Float64DefaultDense()
kernel.parameter.sigma = 0.001

# Create two class svm classifier
# training alg
twoclass_train_alg = svm_training.Batch_Float64DefaultDense()
twoclass_train_alg.parameter.kernel = kernel
twoclass_train_alg.parameter.C = 1.0
# prediction alg
twoclass_predict_alg = svm_prediction.Batch_Float64DefaultDense()
twoclass_predict_alg.parameter.kernel = kernel

# Create a multiclass classifier object (training)
train_alg = multiclass_training.Batch_Float64OneAgainstOne()
train_alg.parameter.nClasses = 10
train_alg.parameter.training = twoclass_train_alg
train_alg.parameter.prediction = twoclass_predict_alg
```

4. define kernel and kernel parameters

training algorithm setup

```
from daal.algorithms.svm import training as svm_training
from daal.algorithms.svm import prediction as svm_prediction
from daal.algorithms.multi_class_classifier import training as
     multiclass_training
from daal.algorithms.classifier import training as training_params

kernel = rbf.Batch_Float64DefaultDense()
kernel.parameter.sigma = 0.001

# Create two class svm classifier
# training alg
twoclass_train_alg = svm_training.Batch_Float64DefaultDense()
twoclass_train_alg.parameter.kernel = kernel
twoclass_train_alg.parameter.C = 1.0
# prediction alg
twoclass_predict_alg = svm_prediction.Batch_Float64DefaultDense()
twoclass_predict_alg.parameter.kernel = kernel

# Create a multiclass classifier object (training)
train_alg = multiclass_training.Batch_Float64OneAgainstOne()
train_alg.parameter.nClasses = 10
train_alg.parameter.training = twoclass_train_alg
train_alg.parameter.prediction = twoclass_predict_alg
```

4. define kernel and kernel parameters
5. create two class svm classifier

training algorithm setup

```python
from daal.algorithms.svm import training as svm_training
from daal.algorithms.svm import prediction as svm_prediction
from daal.algorithms.multi_class_classifier import training as
     multiclass_training
from daal.algorithms.classifier import training as training_params

kernel = rbf.Batch_Float64DefaultDense()
kernel.parameter.sigma = 0.001

# Create two class svm classifier
# training alg
twoclass_train_alg = svm_training.Batch_Float64DefaultDense()
twoclass_train_alg.parameter.kernel = kernel
twoclass_train_alg.parameter.C = 1.0
# prediction alg
twoclass_predict_alg = svm_prediction.Batch_Float64DefaultDense()
twoclass_predict_alg.parameter.kernel = kernel

# Create a multiclass classifier object (training)
train_alg = multiclass_training.Batch_Float64OneAgainstOne()
train_alg.parameter.nClasses = 10
train_alg.parameter.training = twoclass_train_alg
train_alg.parameter.prediction = twoclass_predict_alg
```

4. define kernel and kernel parameters
5. create two class svm classifier
6. create multi class svm classifier (training)

training phase

```
# Pass training data and labels
train_alg.input.set(training_params.data, train_data)
train_alg.input.set(training_params.labels, train_labels)

# training
model = train_alg.compute().get(training_params.model)
```

training phase

```
# Pass training data and labels
train_alg.input.set(training_params.data, train_data)
train_alg.input.set(training_params.labels, train_labels)

# training
model = train_alg.compute().get(training_params.model)
```

7. set input data and labels

training phase

```
# Pass training data and labels
train_alg.input.set(training_params.data, train_data)
train_alg.input.set(training_params.labels, train_labels)

# training
model = train_alg.compute().get(training_params.model)
```

7. set input data and labels
8. start training and get model

prediction algorithm setup

```
from daal.algorithms.multi_class_classifier import prediction as
     multiclass_prediction
from daal.algorithms.classifier import prediction as
     prediction_params

# Create a multiclass classifier object (prediction)
predict_alg = multiclass_prediction.
     Batch_Float64DefaultDenseOneAgainstOne()
predict_alg.parameter.nClasses = 10
predict_alg.parameter.training = twoclass_train_alg
predict_alg.parameter.prediction = twoclass_predict_alg
```

prediction algorithm setup

```
from daal.algorithms.multi_class_classifier import prediction as
     multiclass_prediction
from daal.algorithms.classifier import prediction as
     prediction_params

# Create a multiclass classifier object (prediction)
predict_alg = multiclass_prediction.
     Batch_Float64DefaultDenseOneAgainstOne()
predict_alg.parameter.nClasses = 10
predict_alg.parameter.training = twoclass_train_alg
predict_alg.parameter.prediction = twoclass_predict_alg
```

9. create multi class svm classifier (prediction)

prediction phase

```
# Pass a model and input data
predict_alg.input.setModel(prediction_params.model, model)
predict_alg.input.setTable(prediction_params.data, test_data)

# Compute and return prediction results
results = predict_alg.compute().get(prediction_params.prediction)
```

prediction phase

```
# Pass a model and input data
predict_alg.input.setModel(prediction_params.model, model)
predict_alg.input.setTable(prediction_params.data, test_data)

# Compute and return prediction results
results = predict_alg.compute().get(prediction_params.prediction)
```

10. set input model and data

prediction phase

```
# Pass a model and input data
predict_alg.input.setModel(prediction_params.model, model)
predict_alg.input.setTable(prediction_params.data, test_data)

# Compute and return prediction results
results = predict_alg.compute().get(prediction_params.prediction)
```

10. set input model and data
11. start prediction and get labels

Benchmark results

- Test description: 1797 samples total (90% training set, 10% test set), 64 features per sample
- Platform description: Intel Core i5-6300U CPU @ 2.40GHz

	sklearn	pyDAAL	speedup
training time [s]	0.161	0.018	8.9
test time [s]	0.017	0.004	4.3

Advantages and disadvantages

DAAL in general:

Advantages and disadvantages

DAAL in general:

- ✓ very simple installation/setup

Advantages and disadvantages

DAAL in general:

- ✓ very simple installation/setup
- ✓ wide range of algorithms (both for machine l. and for deep l.)

Advantages and disadvantages

DAAL in general:

- ✓ very simple installation/setup
- ✓ wide range of algorithms (both for machine l. and for deep l.)
- ✓ good support through dedicated intel forum (even for non paid versions)

Advantages and disadvantages

DAAL in general:

- ✓ very simple installation/setup
- ✓ wide range of algorithms (both for machine l. and for deep l.)
- ✓ good support through dedicated intel forum (even for non paid versions)
- ✓ DAAL C++ can be called from R and Matlab (see how-to forum posts)

Advantages and disadvantages

DAAL in general:

- ✓ very simple installation/setup
- ✓ wide range of algorithms (both for machine l. and for deep l.)
- ✓ good support through dedicated intel forum (even for non paid versions)
- ✓ DAAL C++ can be called from R and Matlab (see how-to forum posts)
- ✗ documentation is sometimes not exhaustive

Advantages and disadvantages

DAAL in general:

- ✓ very simple installation/setup
- ✓ wide range of algorithms (both for machine l. and for deep l.)
- ✓ good support through dedicated intel forum (even for non paid versions)
- ✓ DAAL C++ can be called from R and Matlab (see how-to forum posts)
- ✗ documentation is sometimes not exhaustive
- ✗ examples cover very simple application cases

Advantages and disadvantages

DAAL in general:

- ✓ very simple installation/setup
- ✓ wide range of algorithms (both for machine l. and for deep l.)
- ✓ good support through dedicated intel forum (even for non paid versions)
- ✓ DAAL C++ can be called from R and Matlab (see how-to forum posts)
- ✗ documentation is sometimes not exhaustive
- ✗ examples cover very simple application cases

as a Python user:

- ✓ comes with Intel Python framework

Advantages and disadvantages

DAAL in general:

- ✓ very simple installation/setup
- ✓ wide range of algorithms (both for machine l. and for deep l.)
- ✓ good support through dedicated intel forum (even for non paid versions)
- ✓ DAAL C++ can be called from R and Matlab (see how-to forum posts)
- ✗ documentation is sometimes not exhaustive
- ✗ examples cover very simple application cases

as a Python user:

- ✓ comes with Intel Python framework
- ✓ faster alternative to scikit

Advantages and disadvantages

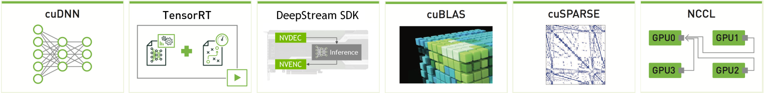
DAAL in general:

- ✓ very simple installation/setup
- ✓ wide range of algorithms (both for machine l. and for deep l.)
- ✓ good support through dedicated intel forum (even for non paid versions)
- ✓ DAAL C++ can be called from R and Matlab (see how-to forum posts)
- ✗ documentation is sometimes not exhaustive
- ✗ examples cover very simple application cases

as a Python user:

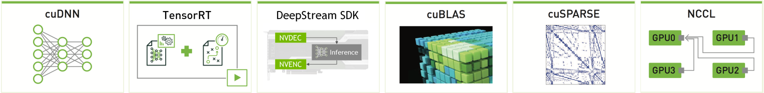
- ✓ comes with Intel Python framework
- ✓ faster alternative to scikit
- ✗ Python interface still in development phase
  - not all neural network layers parameters are accessible/modifiable

NVIDIA Deep Learning Software



Tools and libraries for designing and deploying GPU-accelerated deep learning applications.

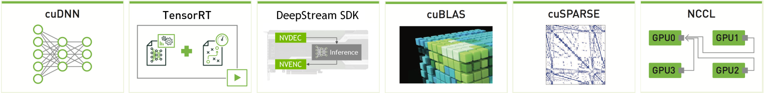
NVIDIA Deep Learning Software



Tools and libraries for designing and deploying GPU-accelerated deep learning applications.

- **Deep Learning Neural Network library (cuDNN)** forward and backward convolution, pooling, normalization, activation layers;

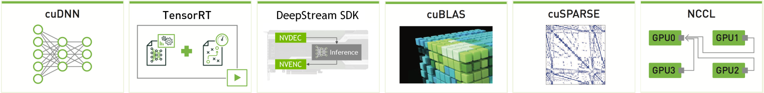
NVIDIA Deep Learning Software



Tools and libraries for designing and deploying GPU-accelerated deep learning applications.

- **Deep Learning Neural Network library (cuDNN)** forward and backward convolution, pooling, normalization, activation layers;
- **TensorRT** optimize, validate and deploy trained neural network for inference to hyperscale data centers, embedded, or automotive product platforms;

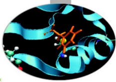
NVIDIA Deep Learning Software

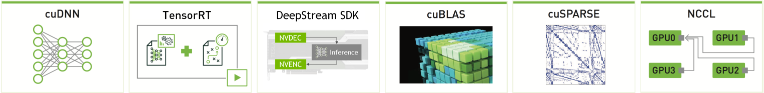


Tools and libraries for designing and deploying GPU-accelerated deep learning applications.

- **Deep Learning Neural Network library (cuDNN)** forward and backward convolution, pooling, normalization, activation layers;
- **TensorRT** optimize, validate and deploy trained neural network for inference to hyperscale data centers, embedded, or automotive product platforms;
- **DeepStream SDK** uses TensorRT to deliver fast INT8 precision inference for real-time video content analysis, supports also FP16 and FP32 precisions;

NVIDIA Deep Learning Software

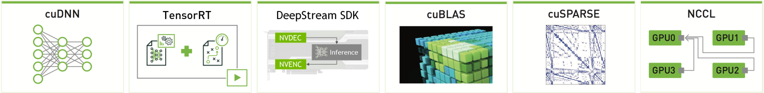




Tools and libraries for designing and deploying GPU-accelerated deep learning applications.

- **Deep Learning Neural Network library (cuDNN)** forward and backward convolution, pooling, normalization, activation layers;
- **TensorRT** optimize, validate and deploy trained neural network for inference to hyperscale data centers, embedded, or automotive product platforms;
- **DeepStream SDK** uses TensorRT to deliver fast INT8 precision inference for real-time video content analysis, supports also FP16 and FP32 precisions;
- **Linear Algebra (cuBLAS and cuBLAS-XT)** accelerated BLAS subroutines for single and multi-GPU acceleration;

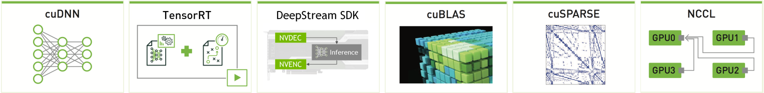
NVIDIA Deep Learning Software



Tools and libraries for designing and deploying GPU-accelerated deep learning applications.

- **Deep Learning Neural Network library (cuDNN)** forward and backward convolution, pooling, normalization, activation layers;
- **TensorRT** optimize, validate and deploy trained neural network for inference to hyperscale data centers, embedded, or automotive product platforms;
- **DeepStream SDK** uses TensorRT to deliver fast INT8 precision inference for real-time video content analysis, supports also FP16 and FP32 precisions;
- **Linear Algebra (cuBLAS and cuBLAS-XT)** accelerated BLAS subroutines for single and multi-GPU acceleration;
- **Sparse Linear Algebra (cuSPARSE)** supports dense, COO, CSR, CSC, ELL/HYB and Blocked CSR sparse matrix formats, Level 1,2,3 routines, sparse triangular solver, sparse tridiagonal solver;

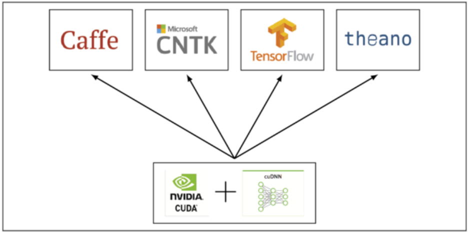
NVIDIA Deep Learning Software



Tools and libraries for designing and deploying GPU-accelerated deep learning applications.

- **Deep Learning Neural Network library (cuDNN)** forward and backward convolution, pooling, normalization, activation layers;
- **TensorRT** optimize, validate and deploy trained neural network for inference to hyperscale data centers, embedded, or automotive product platforms;
- **DeepStream SDK** uses TensorRT to deliver fast INT8 precision inference for real-time video content analysis, supports also FP16 and FP32 precisions;
- **Linear Algebra (cuBLAS and cuBLAS-XT)** accelerated BLAS subroutines for single and multi-GPU acceleration;
- **Sparse Linear Algebra (cuSPARSE)** supports dense, COO, CSR, CSC, ELL/HYB and Blocked CSR sparse matrix formats, Level 1,2,3 routines, sparse triangular solver, sparse tridiagonal solver;
- **Multi-GPU Communications (NCCL, pronounced "Nickel")** optimized primitives for collective multi-GPU communication;

NVIDIA based frameworks



TensorFlow

- Google Brain's second generation machine learning system

TensorFlow

- Google Brain's second generation machine learning system
- computations are expressed as stateful dataflow graphs

TensorFlow

- Google Brain's second generation machine learning system
- computations are expressed as stateful dataflow graphs
- automatic differentiation capabilities

TensorFlow

- Google Brain's second generation machine learning system
- computations are expressed as stateful dataflow graphs
- automatic differentiation capabilities
- optimization algorithms: gradient and proximal gradient based

TensorFlow

- Google Brain's second generation machine learning system
- computations are expressed as stateful dataflow graphs
- automatic differentiation capabilities
- optimization algorithms: gradient and proximal gradient based
- code portability (CPUs, GPUs, on desktop, server, or mobile computing platforms)

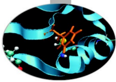
TensorFlow

- Google Brain's second generation machine learning system
- computations are expressed as stateful dataflow graphs
- automatic differentiation capabilities
- optimization algorithms: gradient and proximal gradient based
- code portability (CPUs, GPUs, on desktop, server, or mobile computing platforms)
- Python interface is the preferred one (Java and C++also exist)

TensorFlow

- Google Brain's second generation machine learning system
- computations are expressed as stateful dataflow graphs
- automatic differentiation capabilities
- optimization algorithms: gradient and proximal gradient based
- code portability (CPUs, GPUs, on desktop, server, or mobile computing platforms)
- Python interface is the preferred one (Java and C++also exist)
- installation through: virtualenv, pip, Docker, Anaconda, from sources

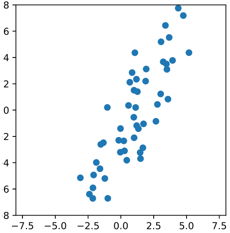
Linear regression (I)



```python
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

# Set up the data with a noisy linear relationship between X and Y.
num_examples = 50
X = np.array([np.linspace(-2, 4, num_examples), np.linspace(-6, 6,
     num_examples)])
X += np.random.randn(2, num_examples)
x, y = X
x_with_bias = np.array([(1., a) for a in x]).astype(np.float32)

losses = []
training_steps = 50
learning_rate = 0.002
```

Linear regression (I)

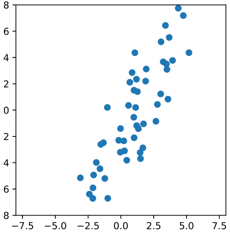
```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

# Set up the data with a noisy linear relationship between X and Y.
num_examples = 50
X = np.array([np.linspace(-2, 4, num_examples), np.linspace(-6, 6,
     num_examples)])
X += np.random.randn(2, num_examples)
x, y = X
x_with_bias = np.array([(1., a) for a in x]).astype(np.float32)

losses = []
training_steps = 50
learning_rate = 0.002
```



1. generate noisy input data

Linear regression (I)

```python
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

# Set up the data with a noisy linear relationship between X and Y.
num_examples = 50
X = np.array([np.linspace(-2, 4, num_examples), np.linspace(-6, 6,
     num_examples)])
X += np.random.randn(2, num_examples)
x, y = X
x_with_bias = np.array([(1., a) for a in x]).astype(np.float32)

losses = []
training_steps = 50
learning_rate = 0.002
```



1. generate noisy input data
2. set slack variables and fix algorithm parameters

Linear regression (II)

```
# Start of graph description
# Set up all the tensors, variables, and operations.
A = tf.constant(x_with_bias)
target = tf.constant(np.transpose([y]).astype(np.float32))
weights = tf.Variable(tf.random_normal([2, 1], 0, 0.1))

yhat = tf.matmul(A, weights)
yerror = tf.sub(yhat, target)
loss = tf.nn.l2_loss(yerror)

update_weights =
  tf.train.GradientDescentOptimizer(learning_rate).minimize(loss)

sess = tf.Session()
sess.run( tf.global_variables_initializer() )
for _ in range(training_steps):
  # Repeatedly run the operations, updating variables
  sess.run( update_weights )
  losses.append( sess.run( loss ) )
```

Linear regression (II)

```
# Start of graph description
# Set up all the tensors, variables, and operations.
A = tf.constant(x_with_bias)
target = tf.constant(np.transpose([y]).astype(np.float32))
weights = tf.Variable(tf.random_normal([2, 1], 0, 0.1))

yhat = tf.matmul(A, weights)
yerror = tf.sub(yhat, target)
loss = tf.nn.l2_loss(yerror)

update_weights =
  tf.train.GradientDescentOptimizer(learning_rate).minimize(loss)

sess = tf.Session()
sess.run( tf.global_variables_initializer() )
for _ in range(training_steps):
  # Repeatedly run the operations, updating variables
  sess.run( update_weights )
  losses.append( sess.run( loss ) )
```

3. define tensorflow constants and variables

Linear regression (II)

```
# Start of graph description
# Set up all the tensors, variables, and operations.
A = tf.constant(x_with_bias)
target = tf.constant(np.transpose([y]).astype(np.float32))
weights = tf.Variable(tf.random_normal([2, 1], 0, 0.1))

yhat = tf.matmul(A, weights)
yerror = tf.sub(yhat, target)
loss = tf.nn.l2_loss(yerror)

update_weights =
  tf.train.GradientDescentOptimizer(learning_rate).minimize(loss)

sess = tf.Session()
sess.run( tf.global_variables_initializer() )
for _ in range(training_steps):
  # Repeatedly run the operations, updating variables
  sess.run( update_weights )
  losses.append( sess.run( loss ) )
```

3. define tensorflow constants and variables
4. define nodes

Linear regression (II)

```
# Start of graph description
# Set up all the tensors, variables, and operations.
A = tf.constant(x_with_bias)
target = tf.constant(np.transpose([y]).astype(np.float32))
weights = tf.Variable(tf.random_normal([2, 1], 0, 0.1))

yhat = tf.matmul(A, weights)
yerror = tf.sub(yhat, target)
loss = tf.nn.l2_loss(yerror)

update_weights =
  tf.train.GradientDescentOptimizer(learning_rate).minimize(loss)

sess = tf.Session()
sess.run( tf.global_variables_initializer() )
for _ in range(training_steps):
  # Repeatedly run the operations, updating variables
  sess.run( update_weights )
  losses.append( sess.run( loss ) )
```

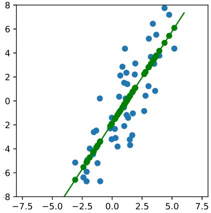
3. define tensorflow constants and variables
4. define nodes
5. start evaluation

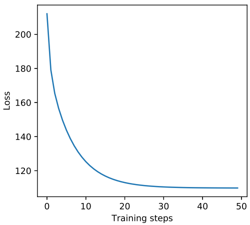
Linear regression (III)

```
# Training is done, get the final values
betas = sess.run( weights )
yhat = sess.run( yhat )
```

Linear regression (III)

```python
# Training is done, get the final values
betas = sess.run( weights )
yhat = sess.run( yhat )
```



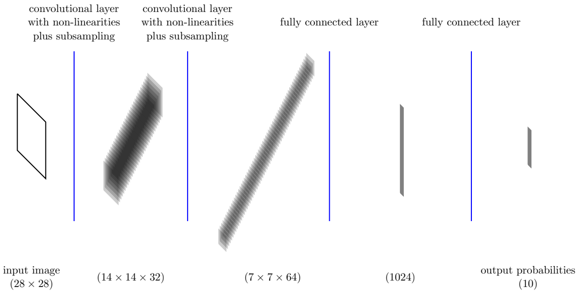


Benchmark results

- MNIST dataset of handwritten digits:
  - training set: 60k samples, 784 features per sample (MNIST)
  - test set: 10k samples, 784 features per sample (MNIST)

Benchmark results

- MNIST dataset of handwritten digits:
  - training set: 60k samples, 784 features per sample (MNIST)
  - test set: 10k samples, 784 features per sample (MNIST)
- Convolutional NN: two conv. layers, two fully conn. layers (plus reg.) ≈ 3m variables



Benchmark results (II)

Optimization method:

- stochastic gradient descent (batch size: 50 examples)
- fixed learning rate
- 2000 iterations

Benchmark results (II)

Optimization method:

- stochastic gradient descent (batch size: 50 examples)
- fixed learning rate
- 2000 iterations

Platforms:

- (1) Intel Core i5-6300U CPU @2.4GHz
- (2) 2 x Intel Xeon 2630 v3 @2.4GHz
- (3) Nvidia Tesla K40

Benchmark results (II)

Optimization method:

- stochastic gradient descent (batch size: 50 examples)
- fixed learning rate
- 2000 iterations

Platforms:

- (1) Intel Core i5-6300U CPU @2.4GHz
- (2) 2 x Intel Xeon 2630 v3 @2.4GHz
- (3) Nvidia Tesla K40

	PL (1)	PL (2)	PL (3)
training time [s]	3307.2	866.1	191.8
test time [s]	11.9	1.7	1.2

Benchmark results (II)

Optimization method:

- stochastic gradient descent (batch size: 50 examples)
- fixed learning rate
- 2000 iterations

Platforms:

- (1) Intel Core i5-6300U CPU @2.4GHz
- (2) 2 x Intel Xeon 2630 v3 @2.4GHz
- (3) Nvidia Tesla K40

	PL (1)	PL (2)	PL (3)
training time [s]	3307.2	866.1	191.8
test time [s]	11.9	1.7	1.2

Same code achieves 3.8x when running in 1 GALILEO node and 17.2x on a single GPU (training phase).

Advantages and disadvantages

TensorFlow in general:

Advantages and disadvantages

TensorFlow in general:

✓ very simple installation/setup

Advantages and disadvantages

TensorFlow in general:

- ✓ very simple installation/setup
- ✓ plenty of tutorials, exhaustive documentation

SuperComputing Applications and Innovation

Advantages and disadvantages

TensorFlow in general:

- ✓ very simple installation/setup
- ✓ plenty of tutorials, exhaustive documentation
- ✓ tools for exporting (partially) trained graphs (see MetaGraph)

Advantages and disadvantages

TensorFlow in general:

- ✓ very simple installation/setup
- ✓ plenty of tutorials, exhaustive documentation
- ✓ tools for exporting (partially) trained graphs (see MetaGraph)
- ✓ debugger, graph flow visualization

Advantages and disadvantages

TensorFlow in general:

- ✓ very simple installation/setup
- ✓ plenty of tutorials, exhaustive documentation
- ✓ tools for exporting (partially) trained graphs (see MetaGraph)
- ✓ debugger, graph flow visualization

as a Python user:

- ✓ “Python API is the most complete and the easiest to use”

Advantages and disadvantages

TensorFlow in general:

- ✓ very simple installation/setup
- ✓ plenty of tutorials, exhaustive documentation
- ✓ tools for exporting (partially) trained graphs (see MetaGraph)
- ✓ debugger, graph flow visualization

as a Python user:

- ✓ “Python API is the most complete and the easiest to use”
- ✓ numpy interoperability

Advantages and disadvantages

TensorFlow in general:

- ✓ very simple installation/setup
- ✓ plenty of tutorials, exhaustive documentation
- ✓ tools for exporting (partially) trained graphs (see MetaGraph)
- ✓ debugger, graph flow visualization

as a Python user:

- ✓ "Python API is the most complete and the easiest to use"
- ✓ numpy interoperability
- ✗ lower level than pyDAAL (?)