Analytics and AI on Cray Systems

CSCS Container Workshop, Zurich June 2018

James Maltby, Charles Siegel and Alessandro Rigazzi

Tutorial Agenda

- **Session I**
  - Introduction/Urika-XC/Accounts
  - Python, Anaconda, and Dask
  - Deep Learning with TensorFlow
- **Session II**
  - Scaling Deep Learning with the Cray PE Machine Learning Plugin
  - HPC, AI, and Analytics with R and pbdR
- **Session III**
  - Spark and Alchemist
  - BigDL
- **Session IV**
  - Cray Graph Engine
  - AI and Analytics Examples
  - Wrap-up and questions

Analytics and AI on Cray Systems

Session I

Training Accounts on Piz Daint

- CSCS has generously provided temporary training accounts on Piz Daint
- Login through the front end, ela.cscs.ch, then can ssh daint
- Setup passwordless ssh. From daint:

```
user@daint102:~> ssh-keygen
user@daint102:~> ssh-copy-id -i .ssh/id_rsa.pub ela1
```

- Add to local .ssh/config to go directly to daint (simplifies notebook setup):

```
Host daint
 HostName daint.cscs.ch
 User your-daint-username
 ProxyCommand ssh -q -Y ela.cscs.ch -W %h:%p
```

- To run Urika-XC (please limit to at most 5 nodes each for now)

```
user@daint104:~> module use /apps/daint/system/cug
user@daint104:~> module load analytics
user@daint104:~> salloc -N 5 -C gpu --reservation=hpccrs02 start_analytics
```

Introduction to Urika-XC

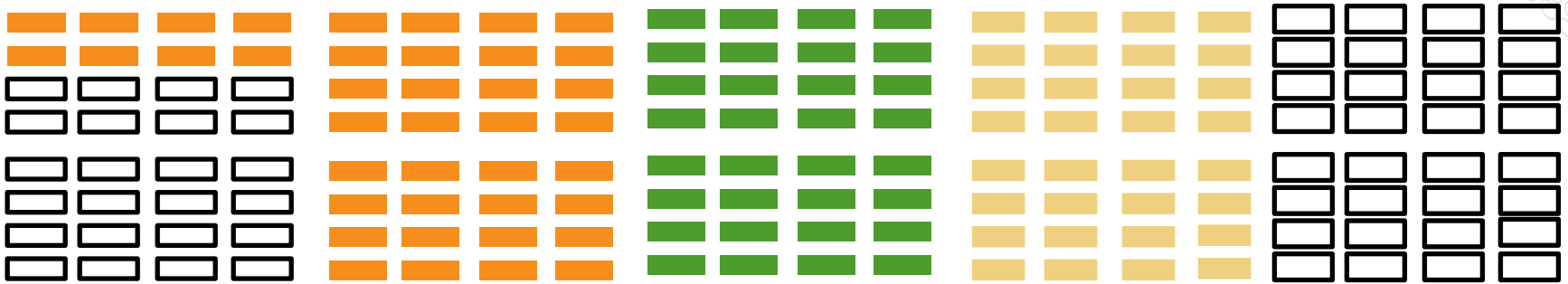
Session I

James Maltby, Cray Inc.

Urika-XC – AI and Analytics on XC

- **Supporting our AI and analytics capabilities on XC systems**
  - Simulation, AI and Data Analytics together on a single platform
  - Solve the “data gravity” problem
  - Enable mixed HPC / AI / Analytics workflows
- **Analytics applications run in the standard XC environment, with customer choice of workload manager**
- **Leverages the XC Shifter package for software distribution**

At The Core Of Urika-XC: Analytics At Supercomputer Scale



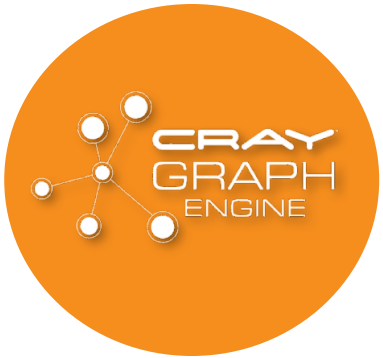
Cray Aries™

Run a mix of workloads on a shared system, each able to scale to >256 nodes leveraging the power of the Cray Aries network and Cray Linux Environment

- Cray Graph Engine
- HPC Simulation
- Python distributed Dask
- Apache Spark

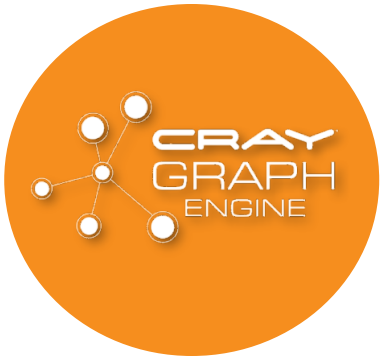
Urika-XC includes two core components

- A collection of scalable analytics frameworks and tools, enabling analytics, machine learning, and deep learning at scale



- A unique in-memory Semantic Graph database, implemented using HPC technology, designed to handle the largest and most demanding use cases where graph discovery and pattern matching are required

Urika-XC component packages



- Apache® Spark™
  - Intel BigDL
- TensorFlow
- Cray PE Machine Learning Plugin
- Anaconda Python
- Dask Distributed
- Jupyter Notebook
- Java
- R
- Scala
- Standard build tools

Runs inside a Container

- Cray Graph Engine (CGE)

Bringing Open Source Frameworks to Supercomputing

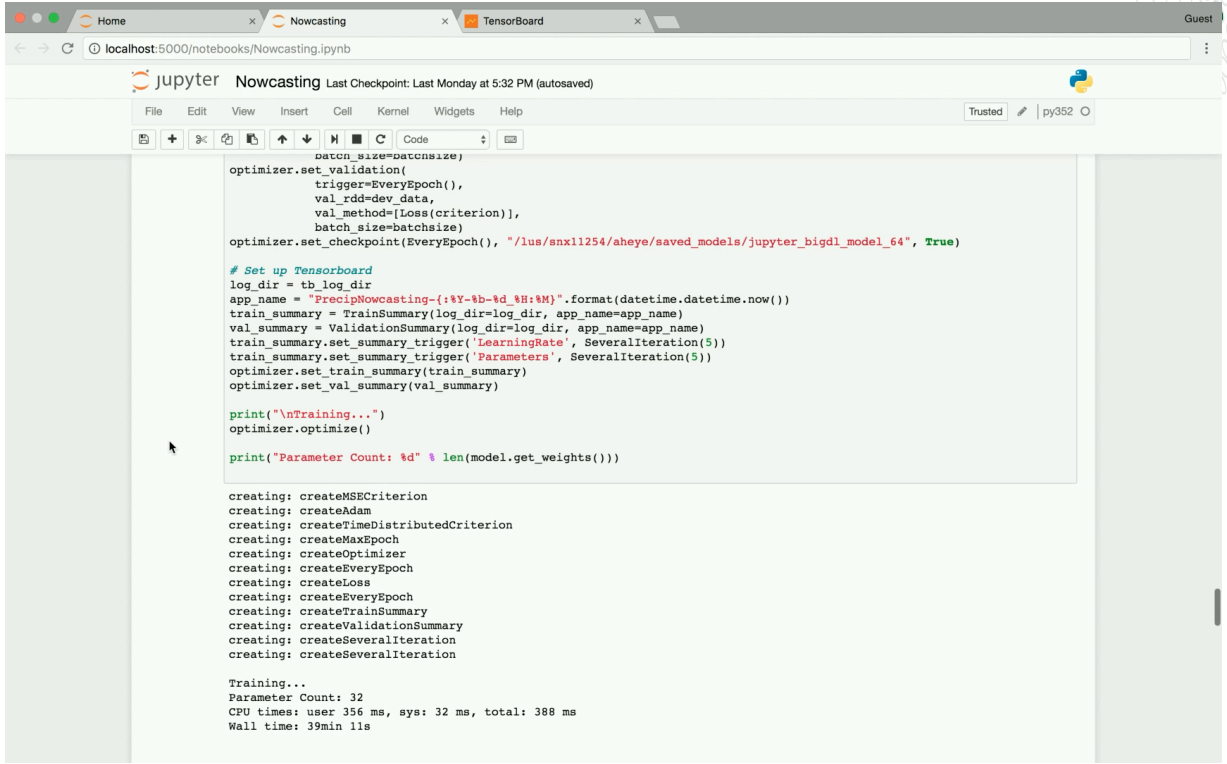
- Distributed in-memory processing for Big Data
  - Extensive set of libraries and packages for the Spark environment

- Anaconda open data science platform and Dask for Python-based distributed parallel computing
  - Well suited for large-scale data processing, predictive analytics, and scientific computing
  - Jupyter Notebooks for interactive supercomputing

- Integrated Deep Learning
  - TensorFlow popular deep learning framework
  - Native deep learning in Spark

Web UIs for Interactivity and Monitoring

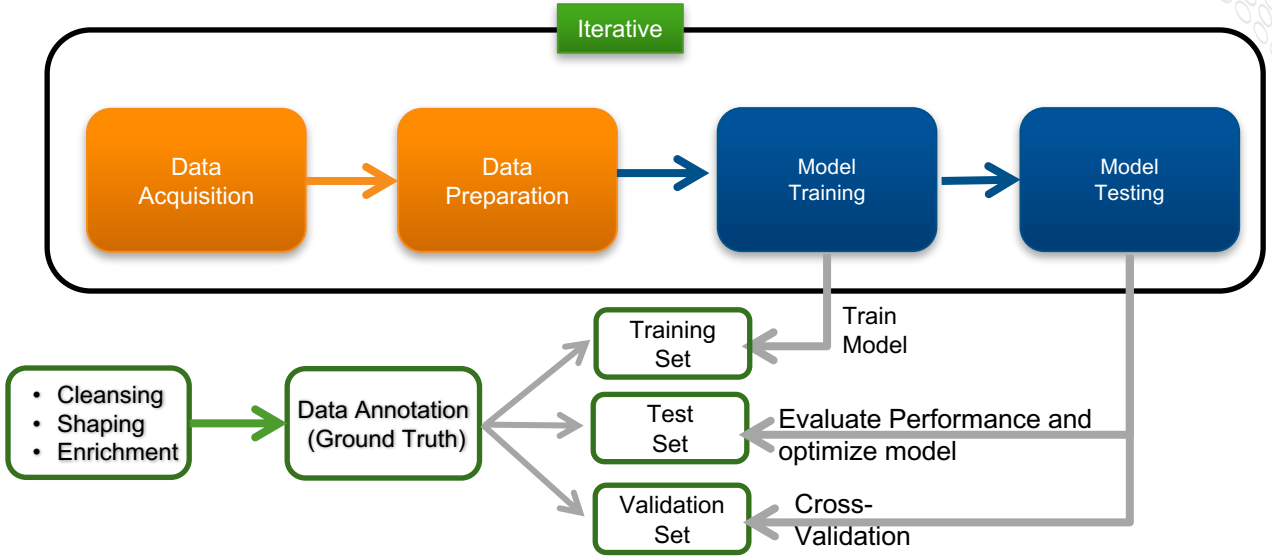
- Jupyter Notebooks
  - Interactive computing and visualization
  - Works with TensorFlow, Dask, PySpark, or other Python-based workflows
- TensorBoard
  - Visualize training
  - Examine DNN layers
  - Run live (monitor training) or after-the-fact



Urika-XC Focus: Support the Entire AI Workflow

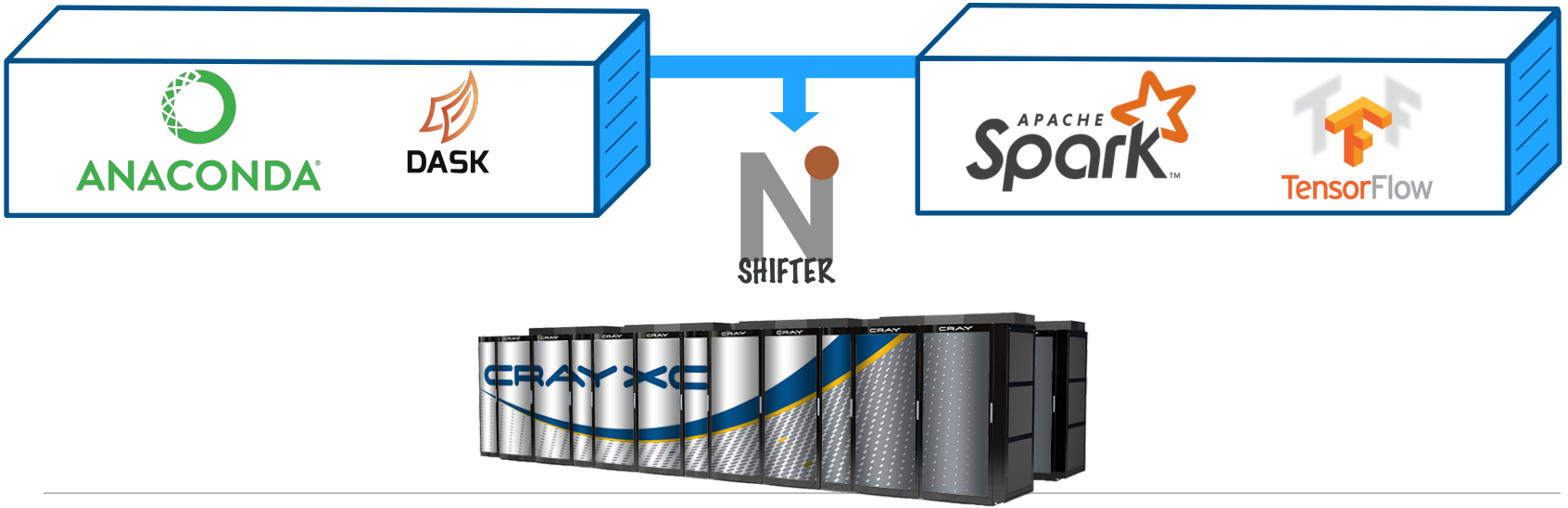
Deep Learning workflows are not limited to training

Similar to other HPC and analytics workloads, significant portions of DL jobs are devoted to data collection, preparation and management.

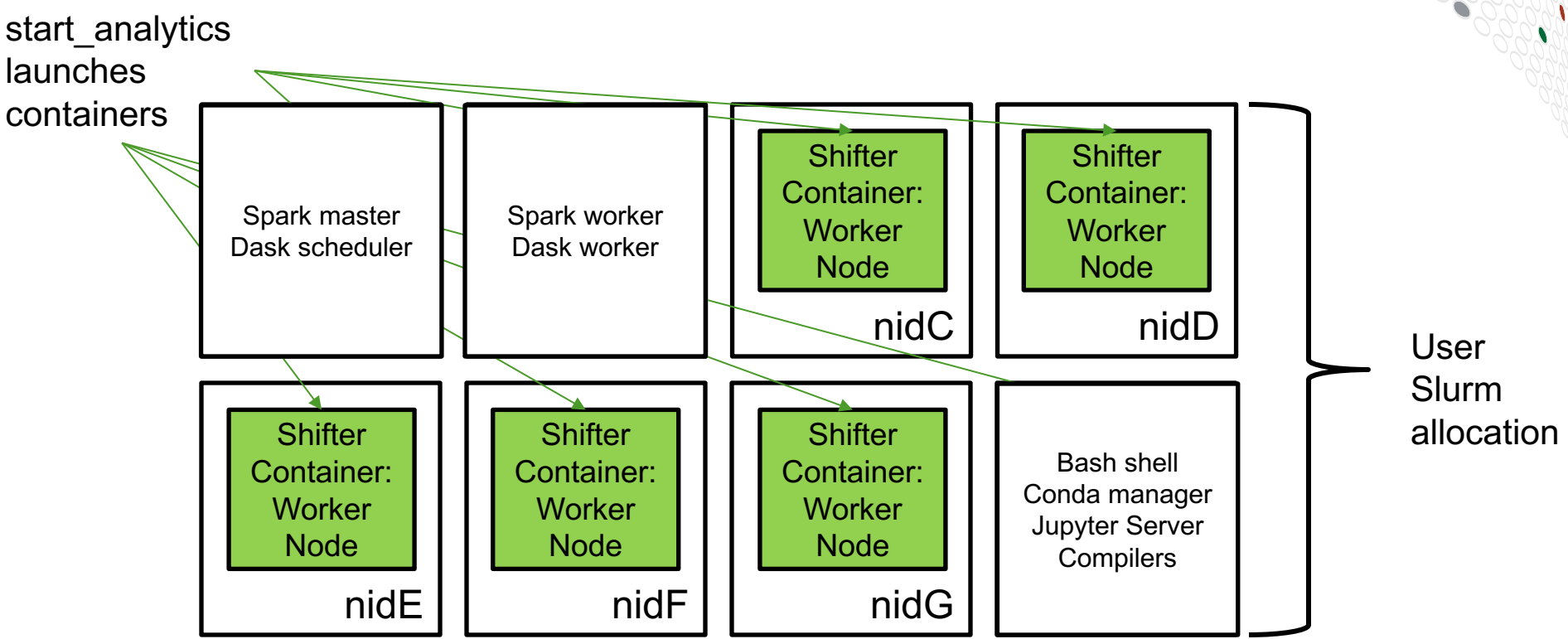


Delivered in a Shifter Container for Performance and Ease of Deployment

- **Shifter containers for HPC**
  - Github: https://github.com/NERSC/shifter



Shifter container execution model



Adding the power of a dedicated Graph-based analytics toolset

- **Urika-XC includes the Cray Graph Engine, an in-memory Semantic Graph Database, implemented using HPC technologies**
- **Based on W3C industry standards**
  - RDF graph data format (“Triple Store”)
  - SPARQL 1.1 query language

- **Includes Cray-specific extensions for demanding Graph problems**
  - Breadth-first search
  - Connected components
  - Community detection
  - …Anything else that entails an indefinite-length search of the graph

Q & A

James Maltby, Cray Inc.
jmaltby@cray.com

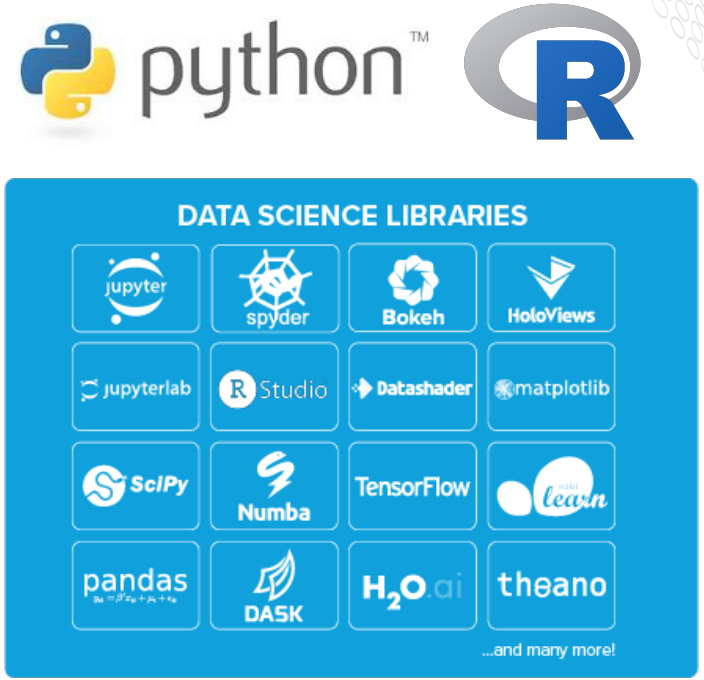
Anaconda Python and Dask Distributed

Session I

Charles Siegel, Cray Inc.

Anaconda Distribution of Python

- **Comes with large set of data science packages preinstalled**
  - 250+ Python and R packages preinstalled
  - >1000 available in repositories
  - Many optimized – e.g., work with Intel Python team
- **Conda environment manager**
  - Linked to conda repos for more Python and R packages
  - Ability to create, clone, share custom environments with your own python/package versions
  - Handles all dependencies
  - Allows sharing environments
- **Anaconda and conda built in to Urika-XC**



Using the Conda Environment Manager

- Create a new conda environment with conda create
  - E.g., create an environment with Python 3.6 and biopython:

```
conda create --name bio biopython python=3.6
```

- Activate your environment:

```
source activate bio
(bio) mikeri:~>
```

- Python 3.6 with biopython will now be your default python:

```
(bio) mikeri:~> python
Python 3.6.5 | Anaconda, Inc.
>>> import Bio
>>> from Bio.Seq import Seq
>>> my_seq = Seq('CATGTAGACTAG')
>>> my_seq.translate()
Seq('HVD*', HasStopCodon(ExtendedIUPACProtein(), '*'))
```

More Conda Commands

- Deactivate an environment: **source deactivate**
- Get rid of an environment: **conda remove**
- Clone an environment: **conda clone**
- List environments: **conda info --envs**
- Find available packages: **conda search**
- List packages: **conda list**
- Add package to current environment: **conda install**
  - Can even install pip, and use that to install in a conda environment!
- More in docs: https://conda.io/docs/index.html
- Conda Cheat Sheet: https://conda.io/docs/_downloads/conda-cheatsheet.pdf

Using Conda Environments with PySpark

- Start up Spark cluster (will discuss in Session III)
- Activate your Conda environment

```
source activate bio
```

- Set PYSPARK_PYTHON to point to environment python

```
export PYSPARK_PYTHON=$(which python)
```

- Run pyspark

```
pyspark
>>> import Bio
```

Let's try this now, in a notebook

- Make sure your local .ssh/config is set up for daint
- Set up a tunnel from your laptop to daint:

```
% ssh -L <local-port>:localhost:<login-port> daint
```

- Launch Urika-XC, pointing to the tunneled port on daint

```
% salloc -N 5 -C mc start_analytics --login-port <login-port> --ui-port <ui-port>
```

- Urika-XC will create a tunnel from *ui-port* on the compute node to *login-port* on daint
- Run your notebook, passing it *ui-port*

```
% export SHELL=$(which bash)
% jupyter notebook --port ui-port
```

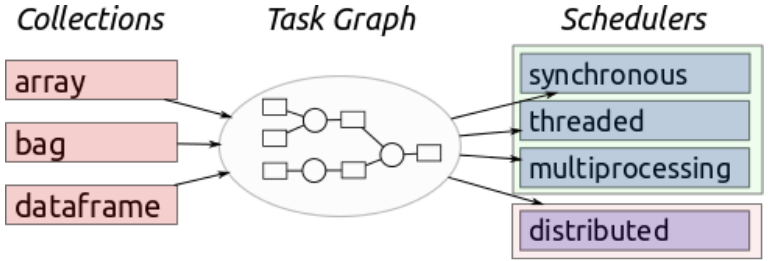
Optional, for terminal access

- Connect browser on laptop to localhost:*local-port*

Anaconda and PySpark on XC Demo

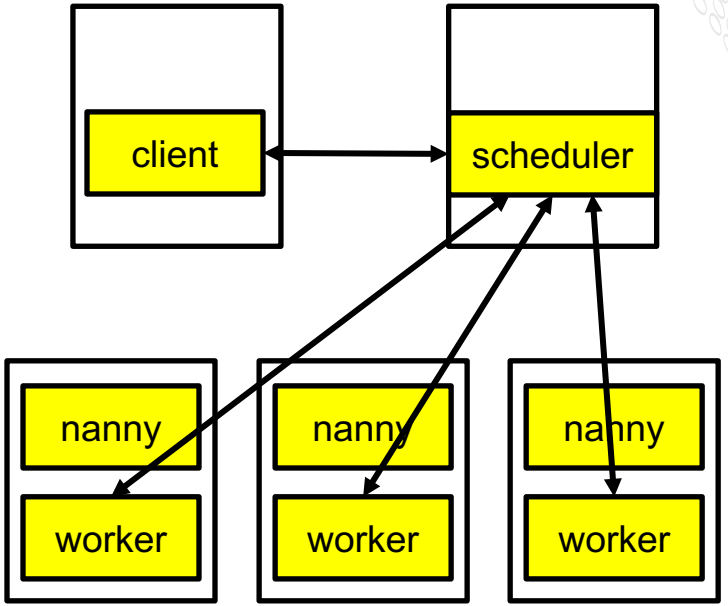
Dask and Dask Distributed

- **Dask**
  - Set of parallel collections and operations for Python
  - Integrated with most common packages, e.g., parallel version of numpy arrays
  - Supports multiple task schedulers
- **Threaded scheduler**
  - Backed by low-overhead thread pool
  - Subject to Python Global Interpreter Lock (GIL)
  - Best if application dominated by non-Python code
- **Multiprocess scheduler**
  - Tasks shipped to separate local processes
  - Not subject to Python GIL – allows true on-node parallelism
  - Low overhead to launch/utilize pool, but overhead of moving data
  - Best for mostly Python code (allows parallelism even with GIL)



Dask and Dask Distributed

- **Distributed scheduler**
  - Dask scheduler for multi-node parallelism
  - Runs a scheduler on one node, workers across allocated nodes
  - Nanny processes for fault tolerance
  - Supports distributed versions of all Dask data structures
  - Allows asynchronous execution (futures)



Setting Up a dask.distributed Cluster

- Set up a dask distributed environment in anaconda
  - `conda create --name mydask dask distributed`
- Get allocation
  - `salloc -N 4 -C mc`
- Activate dask distributed
  - `source activate mydask`
- Start scheduler on one node, start workers on rest
  - Urika-XC can do this automatically:
    - `start_analytics --dask-env mydask`
  - Otherwise can use ssh or srun/aprun (details will vary based on your system)

Demo: dask.distributed on XC

Q & A

Charles Siegel, Cray Inc.
csiegel@cray.com

Introduction to Deep Learning, TensorFlow, and Keras

Session I

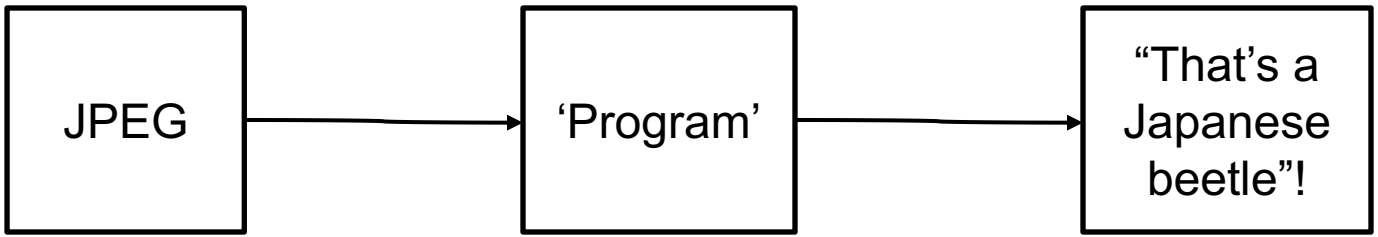
Charles Siegel, Cray Inc.

Outline

- Demystifying artificial intelligence / neural networks / deep learning
- What does the computational problem that is deep learning look like?
- Parallel DL – We will cover this in more depth
- What are TensorFlow and Keras?

A Specific Example

- An organic gardener is building a robot in his garage to recognize the 10 insects found in his garden, and decide which ones to kill with a laser
- The robot will have a camera, and will capture JPEG files of the insects
- The robot needs a 'program' to classify each JPEG according to which of the 10 kinds of insect was photographed

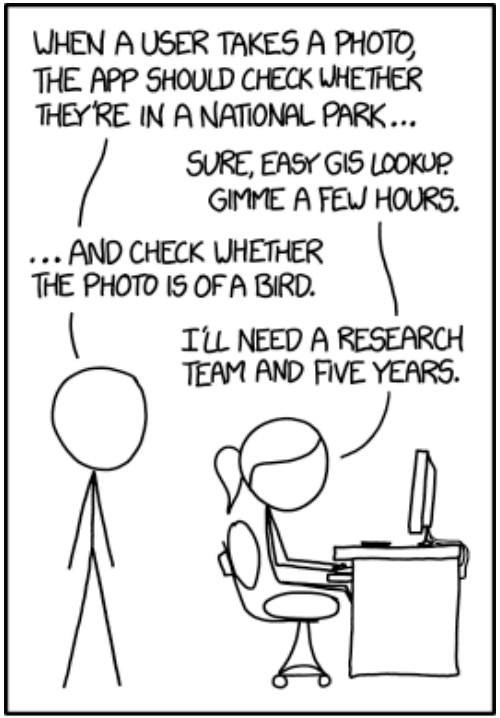


Inputs & Outputs

- **Our input is a JPEG**
  - 224x224 pixels, 3 colors → a 224x224x3 element vector of the pixel values
- **Our output is a classification**
  - One of 10 categories → a 10 element vector with a “1” in the position representing the category to which the image belongs

How many “IF” statements will we need to figure out that a bunch of pixel values is a Japanese beetle?

This is an Artificial Intelligence Problem



- If you can't get the output from the input with a bunch of loops and conditionals, it's AI
- But, if that won't work, how can we do it?

- Hint #1: Any mapping of inputs to outputs is a function
- Hint #2: A function can be approximated using a (good) approximating function

An Approximating Function

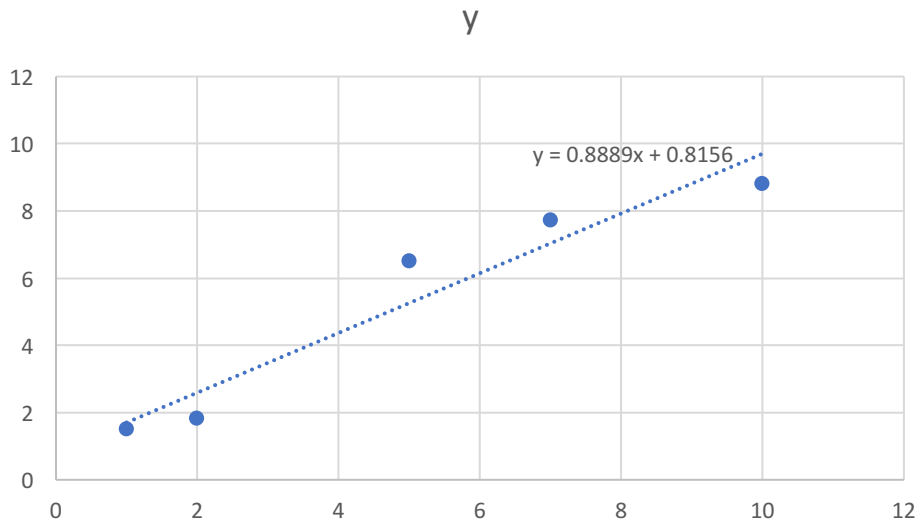
- **How can we determine a good approximating function?**
  - Choose its form (linear, polynomial, …)
  - Minimize the overall error at a finite number of inputs with known outputs - - **fit the curve**
    - We have to find the values of the free parameters of the function that minimize the error – it doesn't matter how we do it

Fitting the curve is a lot like *training* the function to know the answer for arbitrary inputs

Training via Gradient Descent

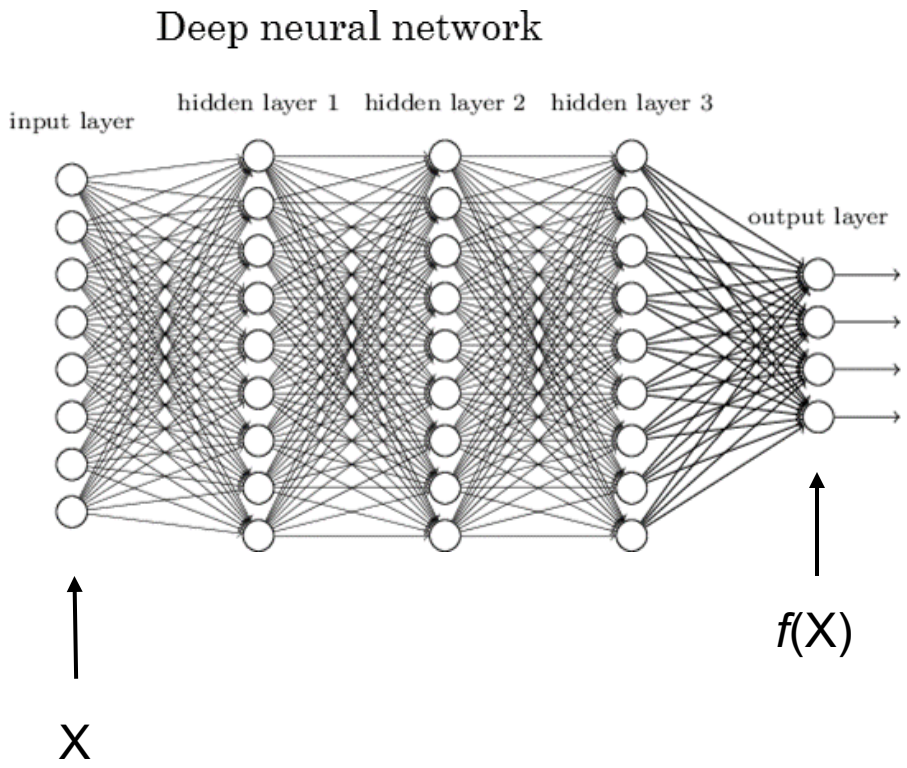
- **We want to approximate y=*f*(x)**
  - Really, we want to find a function that maps a set of inputs to a set of outputs, to some level of accuracy
- **We know $y_i=f(x_i)$, for i=1,N**
- **Iterate:**
  - First iteration only: initialize the free parameters of *f*
  - Calculate error (over our N known points)
  - Calculate gradient of error, as a function of the free parameters of function *f*
  - Adjust the free parameters of function *f* a 'small' distance in the direction of the negative of the error gradient
  - Assess convergence & stop when 'good enough'

Training Error and Validation Error



- Here, we chose the function y=ax+b, with “a” and “b” as the free parameters
- “a” and “b” were chosen to minimize the *training error*, using the 5 points shown
- If we test this function against a distinct set of known data points, we could determine the *validation error*

A Really Useful Kind of Function



- **This image shows a *deep neural network***
  - An approximating function, with free parameters called *weights* and *biases*
  - Deep networks have been found to be especially powerful
  - Neural networks can approximate any continuous function arbitrarily well

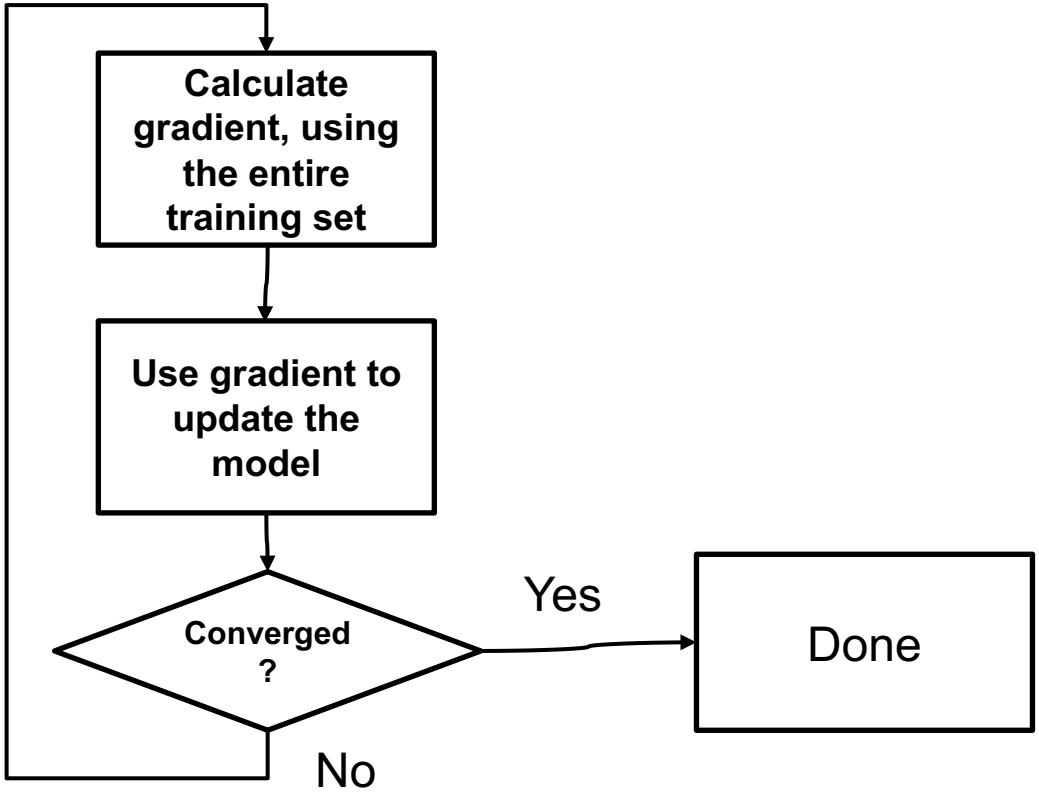
The Big Picture

- **Training a "sufficiently complex" neural network on a "large" and "representative" data set should allow it to "know" about novel data**
  - If we show the neural network 1,000,000 pictures of cats, it should recognize new pictures of cats
  - If we only show the network pictures of black cats, it might not recognize white cats
  - If the network only has 4 "neurons", it probably can't learn to recognize cats

Some Terminology

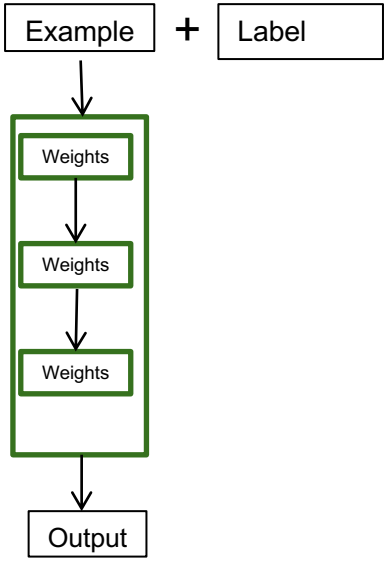
- **The training data consists of training *examples***
  - Each example is an input with a known correct output, called a *label*
  - Having labeled examples is a special but common case, and we won't go deeper on this topic today
- **A subset of the training data is often called a *minibatch***
- **One 'trip' through the whole training set is called an *epoch***
  - Often, bookkeeping, convergence testing, checkpointing, etc. are done after each epoch

Gradient Descent Algorithm



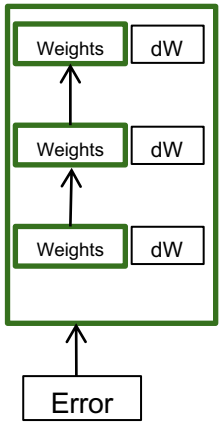
Training Schematic

Feedforward



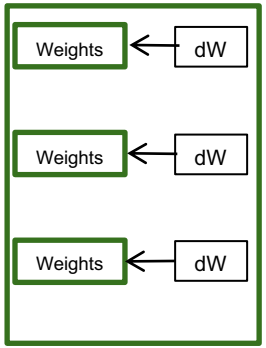
One or more training examples *feedforward* through the layers of *weights*, producing an output

Backpropagate



The error, which is the difference between the label and the output, is *backpropagated* through the layers, producing the *gradients*

Update



The weights are updated by adding the gradients (scaled by a multiplier) to them

Feedforward and backpropagate are much more expensive than update (>100X)

Variations on the Gradient Descent Algorithm

- **Stochastic Gradient Descent**
  - A gradient is calculated, and the model is updated, for each training example
- **Batch Gradient Descent**
  - The training examples are divided into minibatches
  - A gradient is calculated and the model is updated for each minibatch
- **Strict Gradient Descent is seldom if ever used**
- **Strict Stochastic Descent is seldom if ever used**
- **Batch Gradient Descent is almost always used**
  - And, everyone calls it Stochastic Gradient Descent (SGD)

Parallelizing SGD

- **Data parallel methods**
  - “Minibatch Parallel”
  - Every worker independently calculates a “local gradient” using a “local minibatch”
  - All workers participate in an allreduce, or communicate with a parameter server, to average all the gradients and synchronize with other workers
- **Model parallel methods**
  - Break the neural network up – different layers on different nodes
  - Useful if the model is too large for a single node
  - But often more communication that data parallel methods

For More Information…

A good overview:

Efficient Processing of Deep Neural Networks: A Tutorial and Survey

https://arxiv.org/abs/1703.09039

TensorFlow

- Developed by Google
- Most popular DL framework
- Large open source community
- APIs for
  - Python
  - C++
  - Go
  - Java
- Optimized for CPU and GPU architectures
- Ships with Urika-XC
- Learn TensorFlow
  - Docs: https://www.tensorflow.org/get_started/
  - Programmer's Guide: https://www.tensorflow.org/programmers_guide/
  - Tutorials: https://www.tensorflow.org/tutorials/

Keras

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3,3),activation='relu',input_shape=ish))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(), metrics=['accuracy'])
model.fit(x_train, y_train, batch_size=batch_size,
          epochs=epochs, verbose=1, validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
```

- High-level neural networks API – just add layers!
- Compatible with TensorFlow, Theano, CNTK (and others)

Keras

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3,3),activation='relu',input_shape=ish))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(), metrics=['accuracy'])
model.fit(x_train, y_train, batch_size=batch_size,
          epochs=epochs, verbose=1, validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
```

Construct Model

- High-level neural networks API – just add layers!
- Compatible with TensorFlow, Theano, CNTK (and others)

Keras

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3,3),activation='relu',input_shape=ish))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(), metrics=['accuracy'])
model.fit(x_train, y_train, batch_size=batch_size,
          epochs=epochs, verbose=1, validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
```

Configure Model for Training

- High-level neural networks API – just add layers!
- Compatible with TensorFlow, Theano, CNTK (and others)

Keras

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3,3),activation='relu',input_shape=ish))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(), metrics=['accuracy'])
model.fit(x_train, y_train, batch_size=batch_size,
          epochs=epochs, verbose=1, validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
```

Train Model

- High-level neural networks API – just add layers!
- Compatible with TensorFlow, Theano, CNTK (and others)

Keras

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3,3),activation='relu',input_shape=ish))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(), metrics=['accuracy'])
model.fit(x_train, y_train, batch_size=batch_size,
          epochs=epochs, verbose=1, validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
```

Evaluate Model

- High-level neural networks API – just add layers!
- Compatible with TensorFlow, Theano, CNTK (and others)

Demo: Keras MNIST

Legal Disclaimer

Information in this document is provided in connection with Cray Inc. products. No license, express or implied, to any intellectual property rights is granted by this document.

Cray Inc. may make changes to specifications and product descriptions at any time, without notice.

All products, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

Cray hardware and software products may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Cray uses codenames internally to identify products that are in development and not yet publicly announced for release. Customers and other third parties are not authorized by Cray Inc. to use codenames in advertising, promotion or marketing and any use of Cray Inc. internal codenames is at the sole risk of the user.

Performance tests and ratings are measured using specific systems and/or components and reflect the approximate performance of Cray Inc. products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance.

The following are trademarks of Cray Inc. and are registered in the United States and other countries: CRAY and design, SONEXION, URIKA and YARCDATA. The following are trademarks of Cray Inc.: CHAPEL, CLUSTER CONNECT, CLUSTERSTOR, CRAYDOC, CRAYPAT, CRAYPORT, DATAWARP, ECOPHLEX, LIBSCI, NODEKARE, REVEAL. The following system family marks, and associated model number marks, are trademarks of Cray Inc.: CS, CX, XC, XE, XK, XMT and XT. The registered trademark LINUX is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis. Other trademarks used on this website are the property of their respective owners.

Q & A

Charles Siegel, Cray Inc.
csiegel@cray.com