Deep Learning on Massively Parallel Processing Databases

Frank McQuillan
Feb 2019

Greenplum

Keras

MADlib

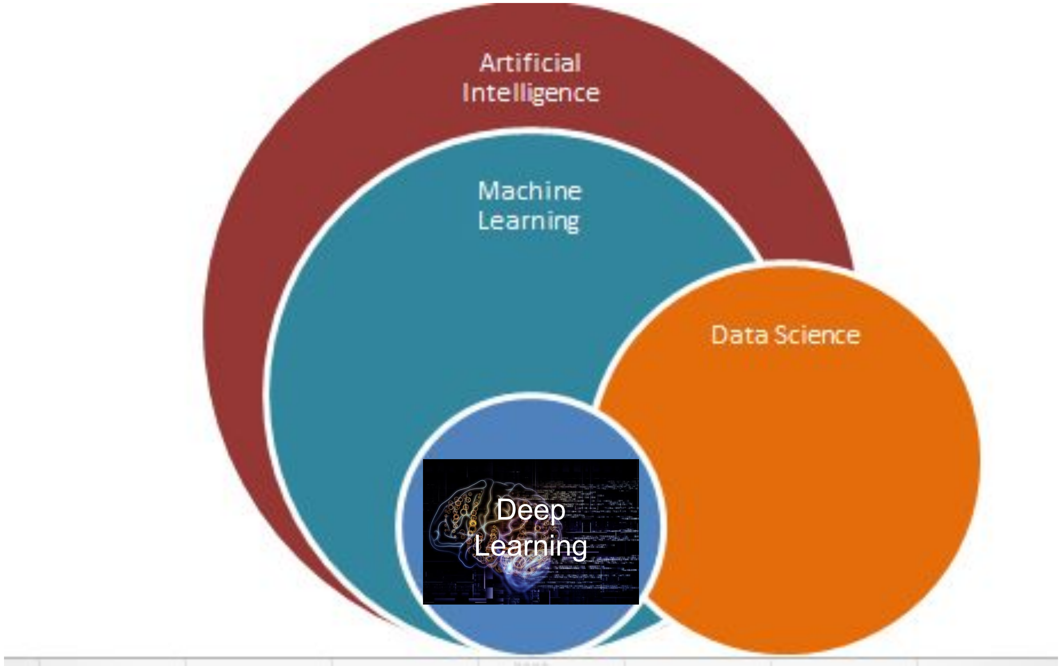
TensorFlow

python

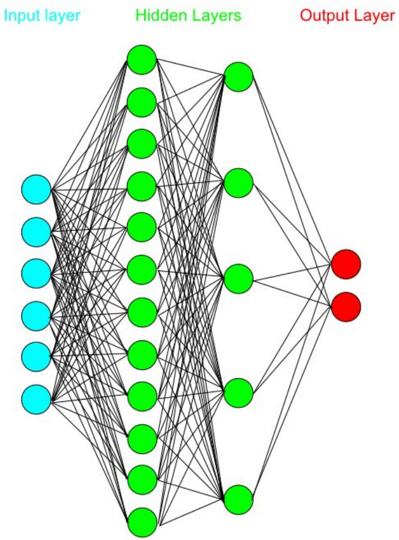
NumPy

A Brief Introduction to Deep Learning

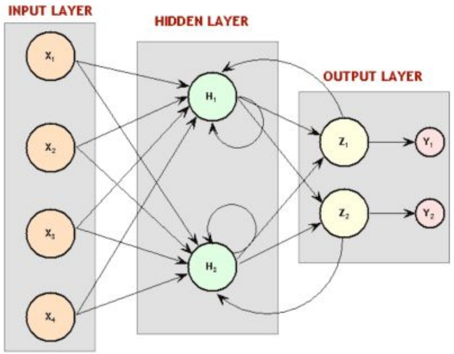
Artificial Intelligence Landscape



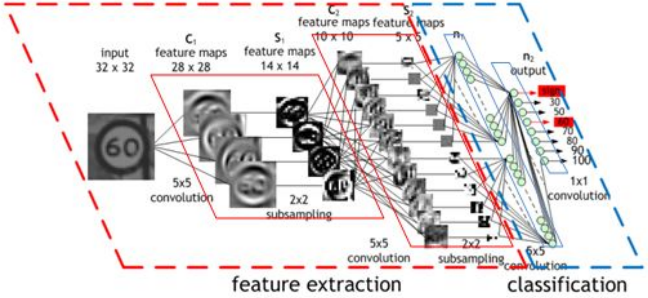
Example Deep Learning Algorithms



Multilayer
perceptron (MLP)



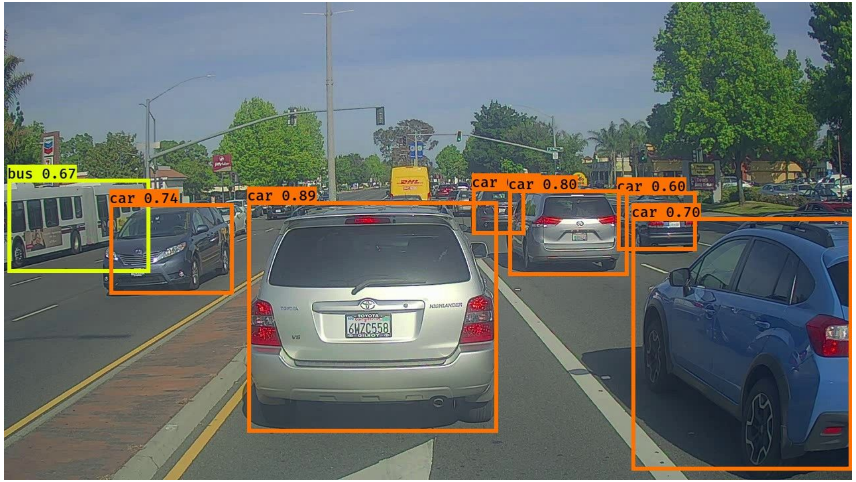
Recurrent
neural network (RNN)



Convolutional
neural network (CNN)

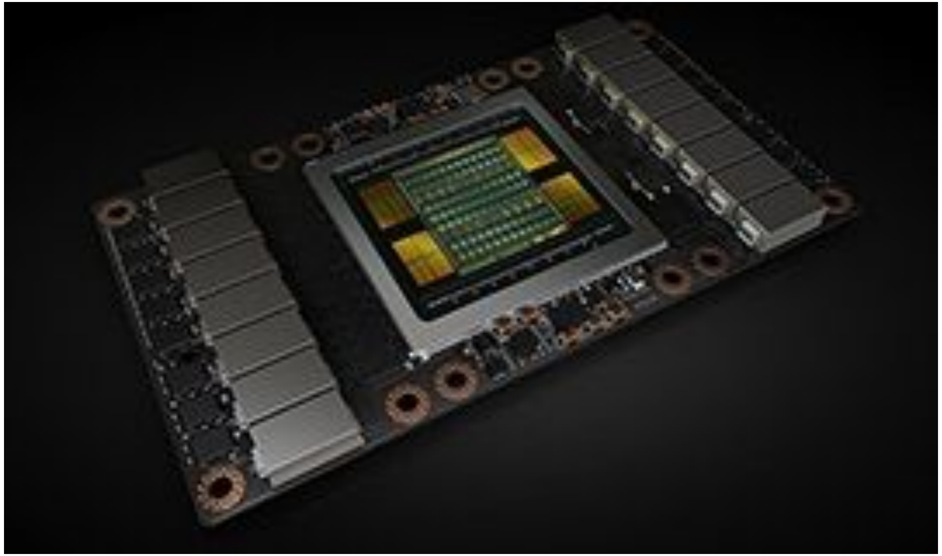
Convolutional Neural Networks (CNN)

- Effective for computer vision
- Fewer parameters than fully connected networks
- Translational invariance
- Classic networks: LeNet-5, AlexNet, VGG

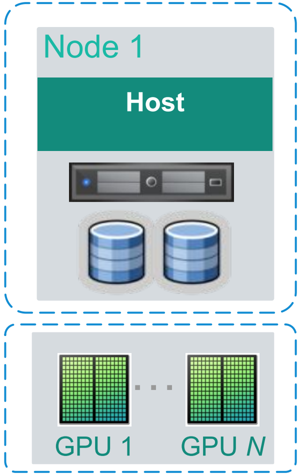


Graphics Processing Units (GPUs)

- Great at performing a lot of simple computations such as matrix operations
- Well suited to deep learning algorithms

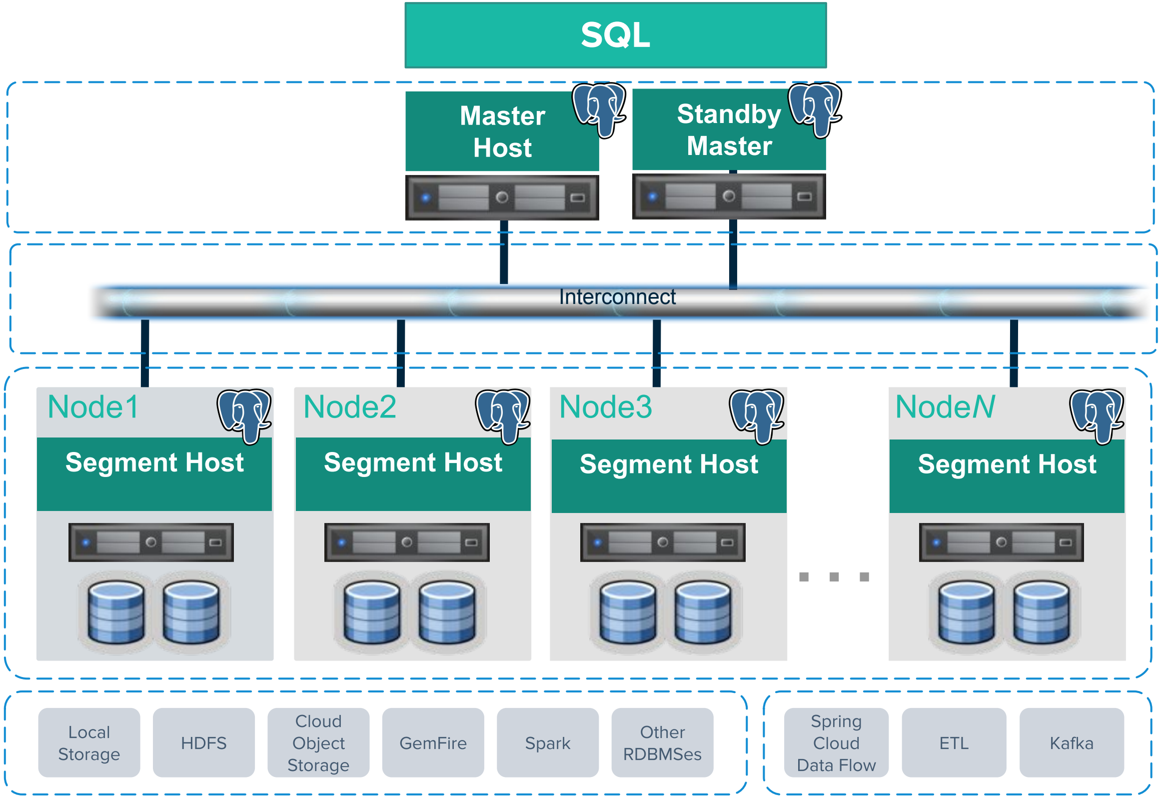


Single Node Multi-GPU

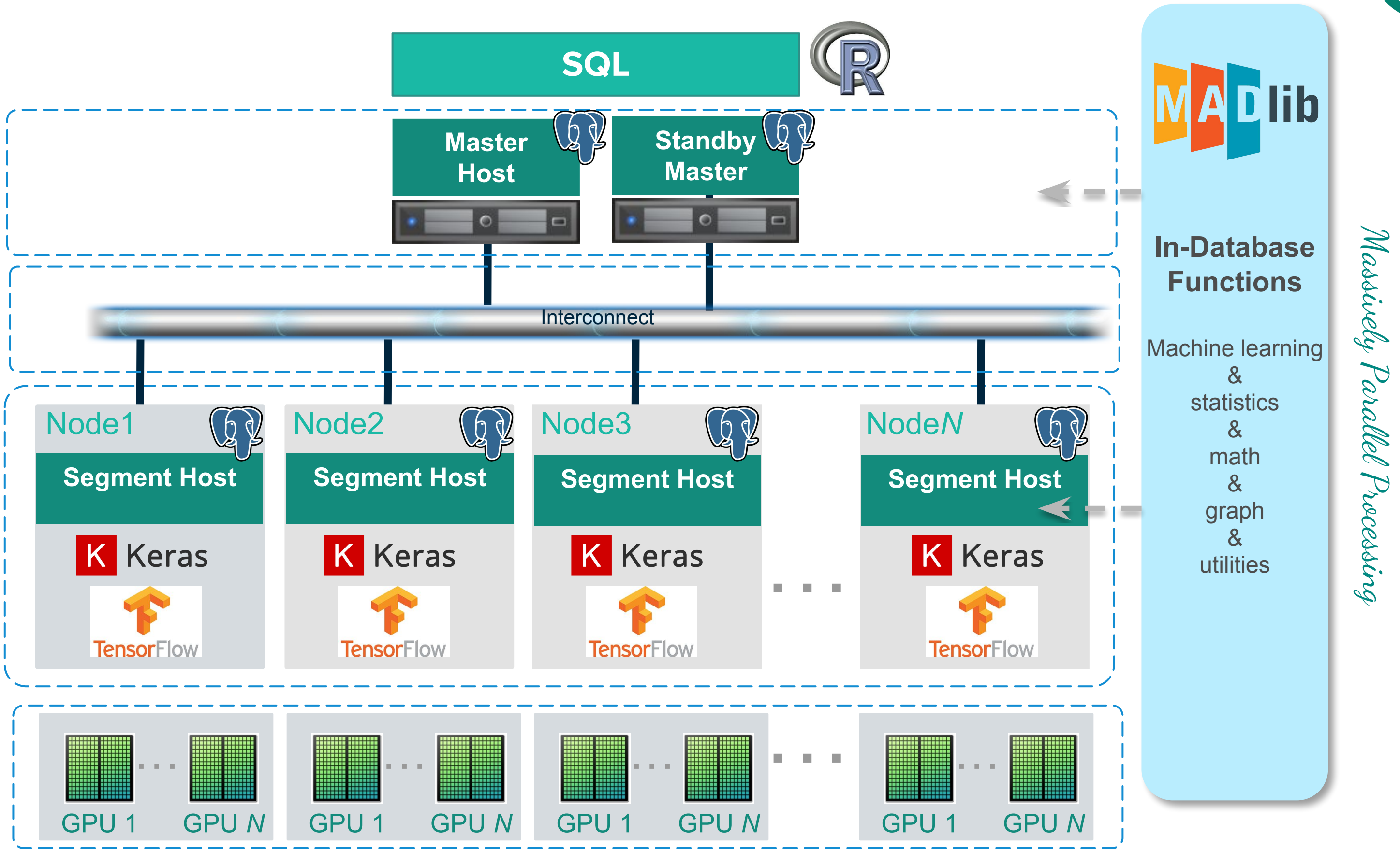


Greenplum Database and Apache MADlib

Greenplum Database



Multi-Node Multi-GPU



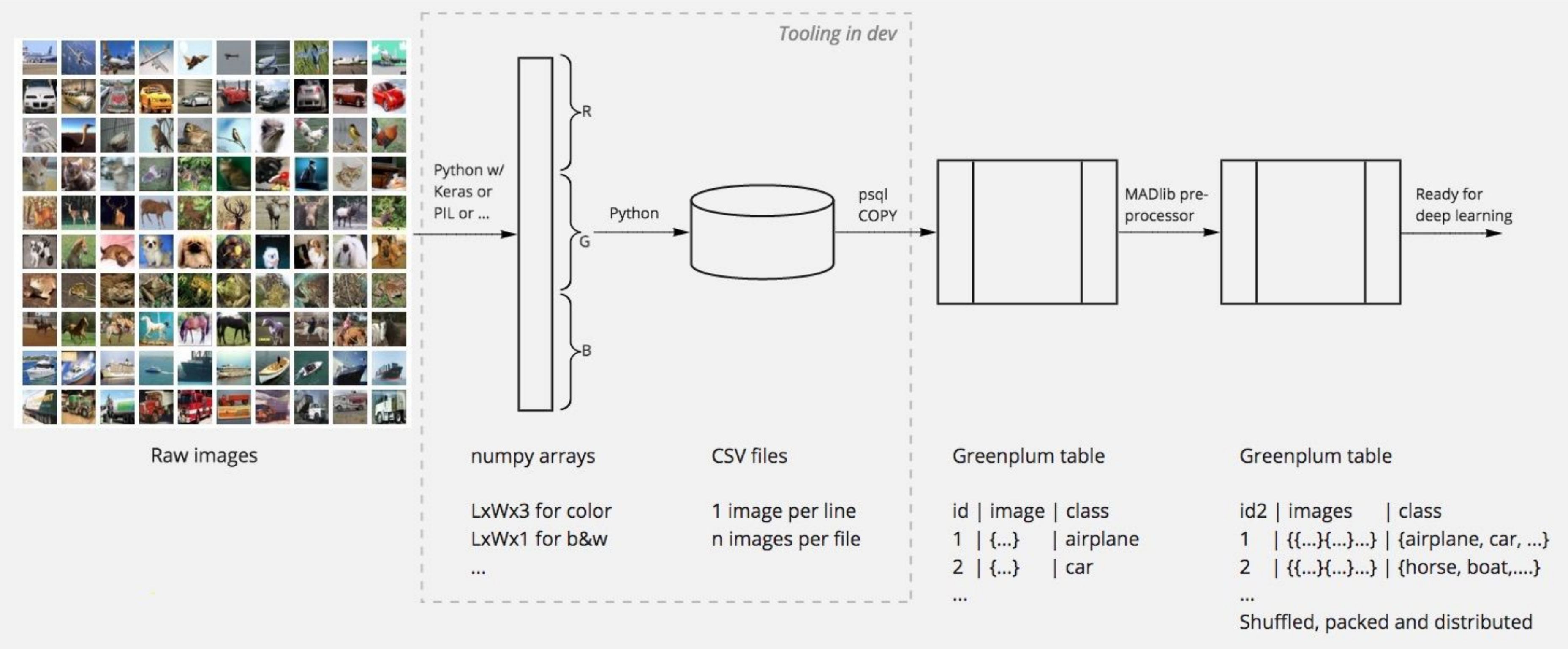
Deep Learning on a Cluster

Num	Approach	Description
1	Distributed deep learning	Train single model architecture across the cluster. Data distributed (usually randomly) across segments.
2	Data parallel models	Train same model architecture in parallel on different data groups (e.g., build separate models per country).
3	Hyperparameter tuning	Train same model architecture in parallel with different hyperparameter settings and incorporate cross validation. Same data on each segment.
4	Neural architecture search	Train different model architectures in parallel. Same data on each segment.

this talk

Workflow

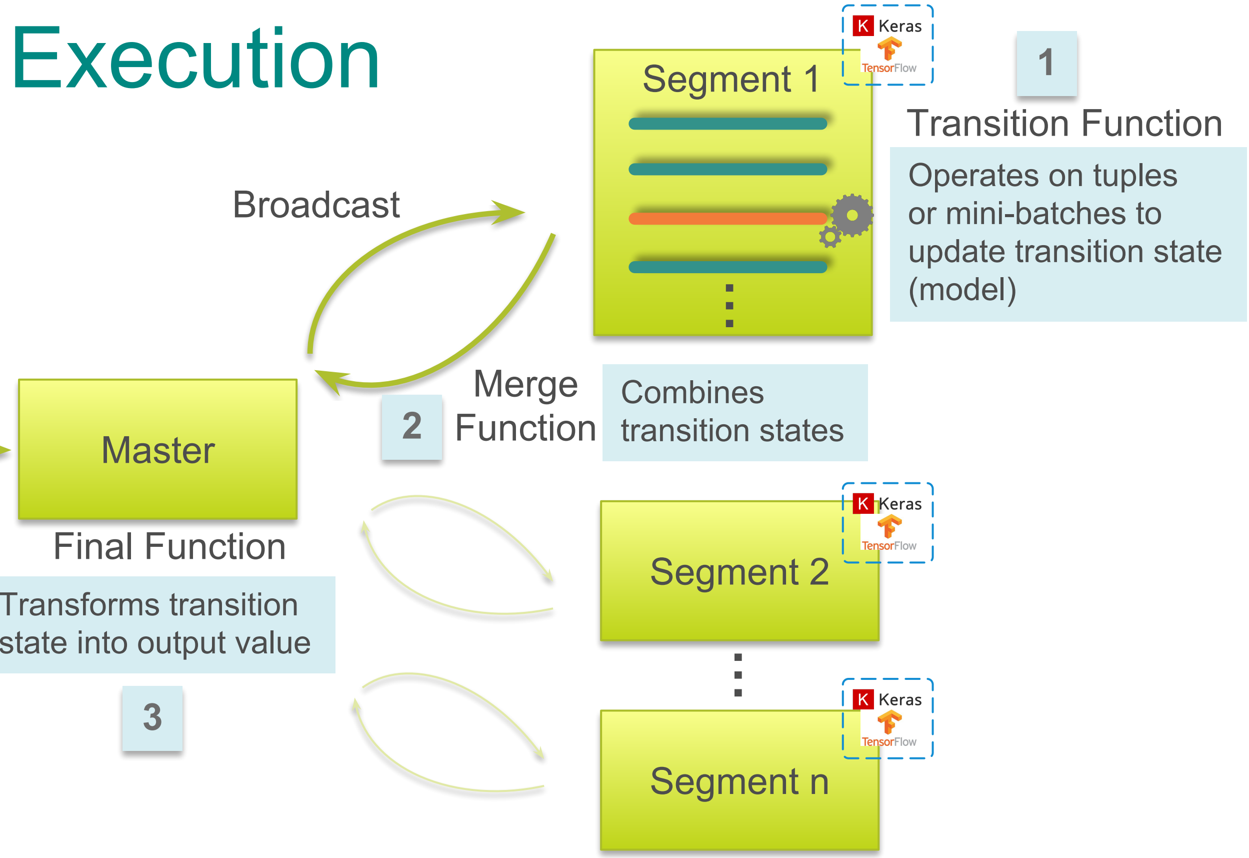
Data Loading and Formatting



Iterative Model Execution

Stored Procedure for Model

```
model = init(…)
WHILE model not converged
  model =
    SELECT
      model.aggregation(…)
    FROM
      data table
ENDWHILE
```



Distributed Deep Learning Methods

- Open area of research*
- Methods we have investigated so far:
  - Simple averaging
  - Ensembling
  - Elastic averaging stochastic gradient descent (EASGD)

* Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis
https://arxiv.org/pdf/1802.09941.pdf

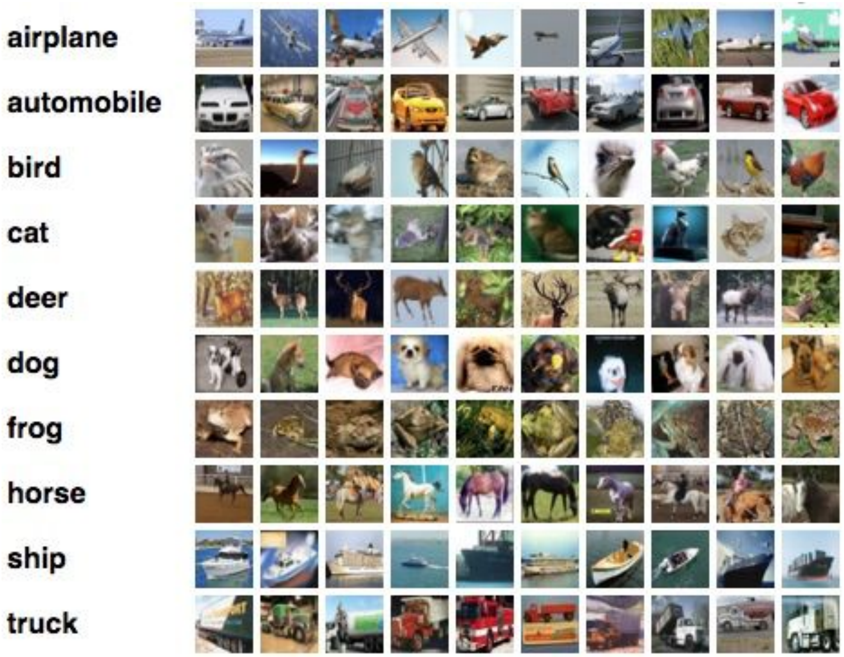
Some Results

Testing Infrastructure

- Google Cloud Platform (GCP)
- Type n1-highmem-32 (32 vCPUs, 208 GB memory)
- NVIDIA Tesla P100 GPUs
- Greenplum database config
  - Tested up to 20 segment (worker node) clusters
  - 1 GPU per segment

CIFAR-10

- 60k 32x32 color images in 10 classes, with 6k images per class
- 50k training images and 10k test images



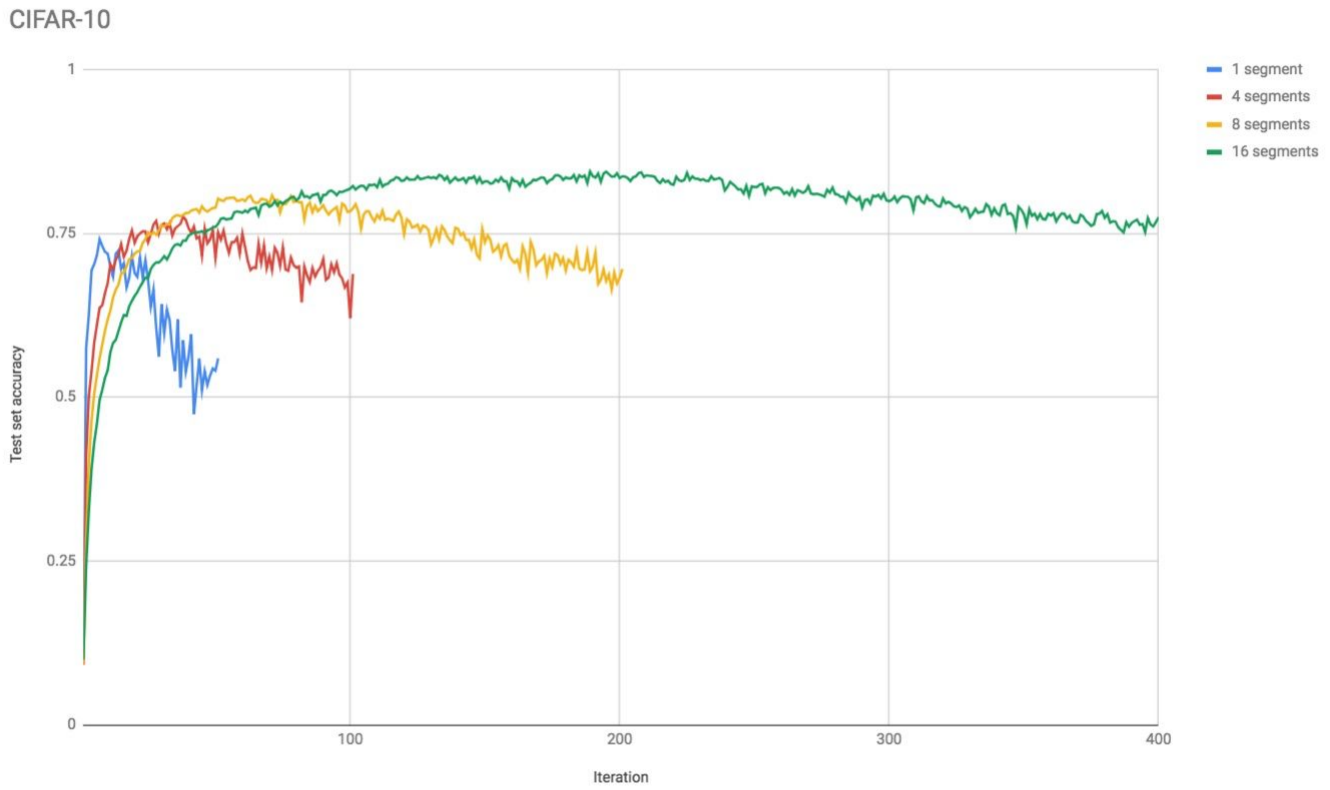
https://www.cs.toronto.edu/~kriz/cifar.html

Places

- Images comprising ~98% of the types of places in the world
- Places365-Standard: 1.8M images from 365 scene categories
- 256x256 color images with 50 images/category in validation set and 900 images/category in test set

http://places2.csail.mit.edu/index.html

6-layer CNN - Test Set Accuracy (CIFAR-10)

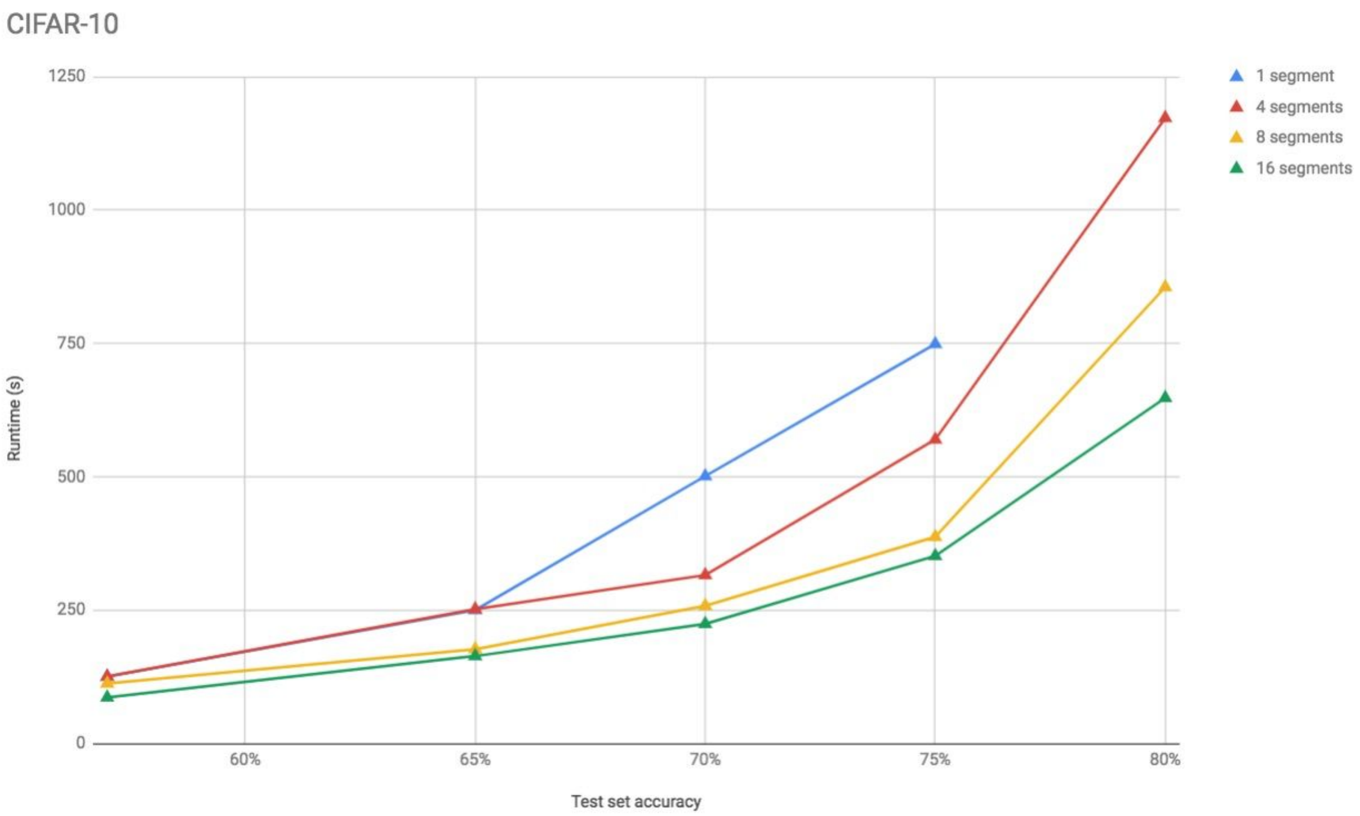



```
OPERATION           DATA DIMENSIONS   WEIGHTS(N)   WEIGHTS(%)

           Input   #####      3   32   32
          Conv2D    \|/  -------------------       896     0.0%
            relu   #####     32   32   32
         Dropout    | ||  -------------------         0     0.0%
                   #####     32   32   32
          Conv2D    \|/  -------------------      9248     0.4%
            relu   #####     32   32   32
    MaxPooling2D   Y max -------------------         0     0.0%
                   #####     32   16   16
          Conv2D    \|/  -------------------     18496     0.8%
            relu   #####     64   16   16
         Dropout    | ||  -------------------         0     0.0%
                   #####     64   16   16
          Conv2D    \|/  -------------------     36928     1.5%
            relu   #####     64   16   16
    MaxPooling2D   Y max -------------------         0     0.0%
                   #####     64    8    8
          Conv2D    \|/  -------------------     73856     3.1%
            relu   #####    128    8    8
         Dropout    | ||  -------------------         0     0.0%
                   #####    128    8    8
          Conv2D    \|/  -------------------    147584     6.2%
            relu   #####    128    8    8
    MaxPooling2D   Y max -------------------         0     0.0%
                   #####    128    4    4
         Flatten   ||||| -------------------         0     0.0%
                   #####       2048
         Dropout    | ||  -------------------         0     0.0%
                   #####       2048
           Dense   XXXXX -------------------   2098176    87.6%
            relu   #####       1024
         Dropout    | ||  -------------------         0     0.0%
                   #####       1024
           Dense   XXXXX -------------------     10250     0.4%
         softmax   #####         10
```

Method: Model weight averaging

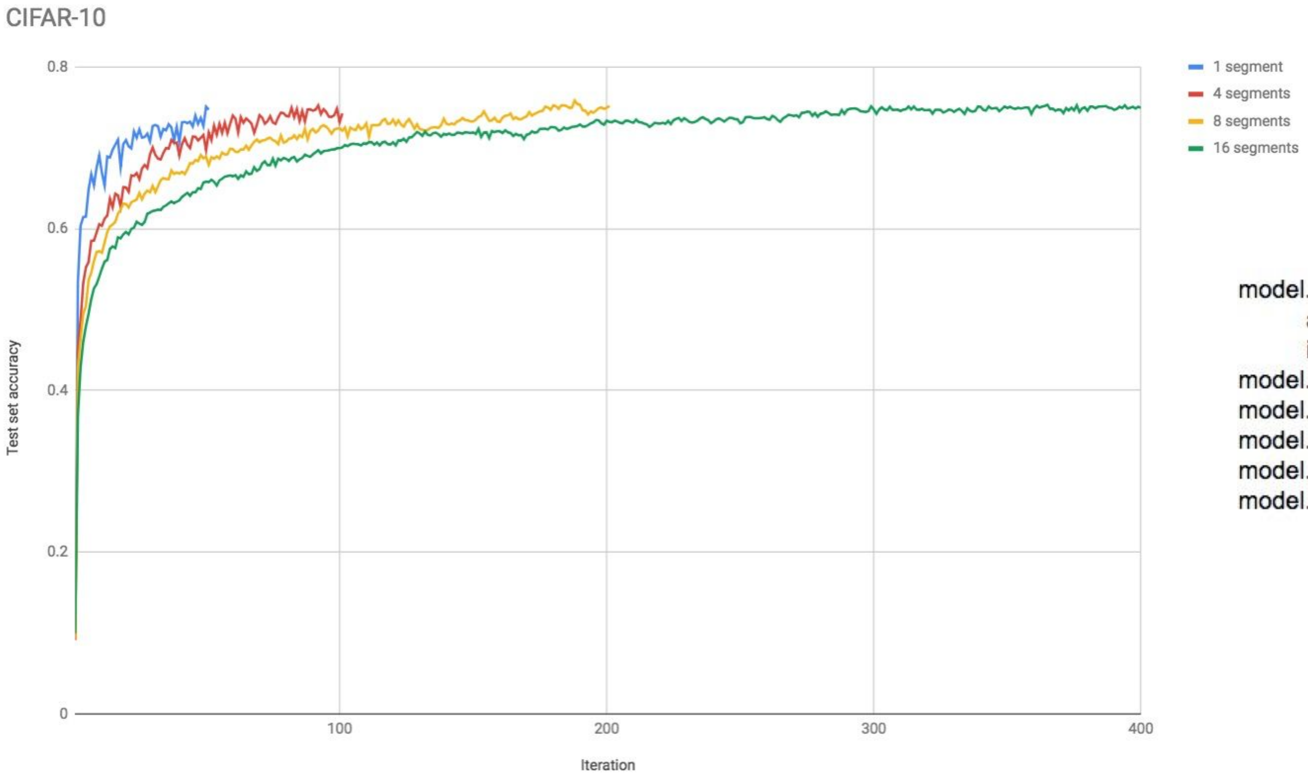
https://blog.plon.io/tutorials/cifar-10-classification-using-keras-tutorial/

6-layer CNN - Runtime (CIFAR-10)



Method: Model weight averaging

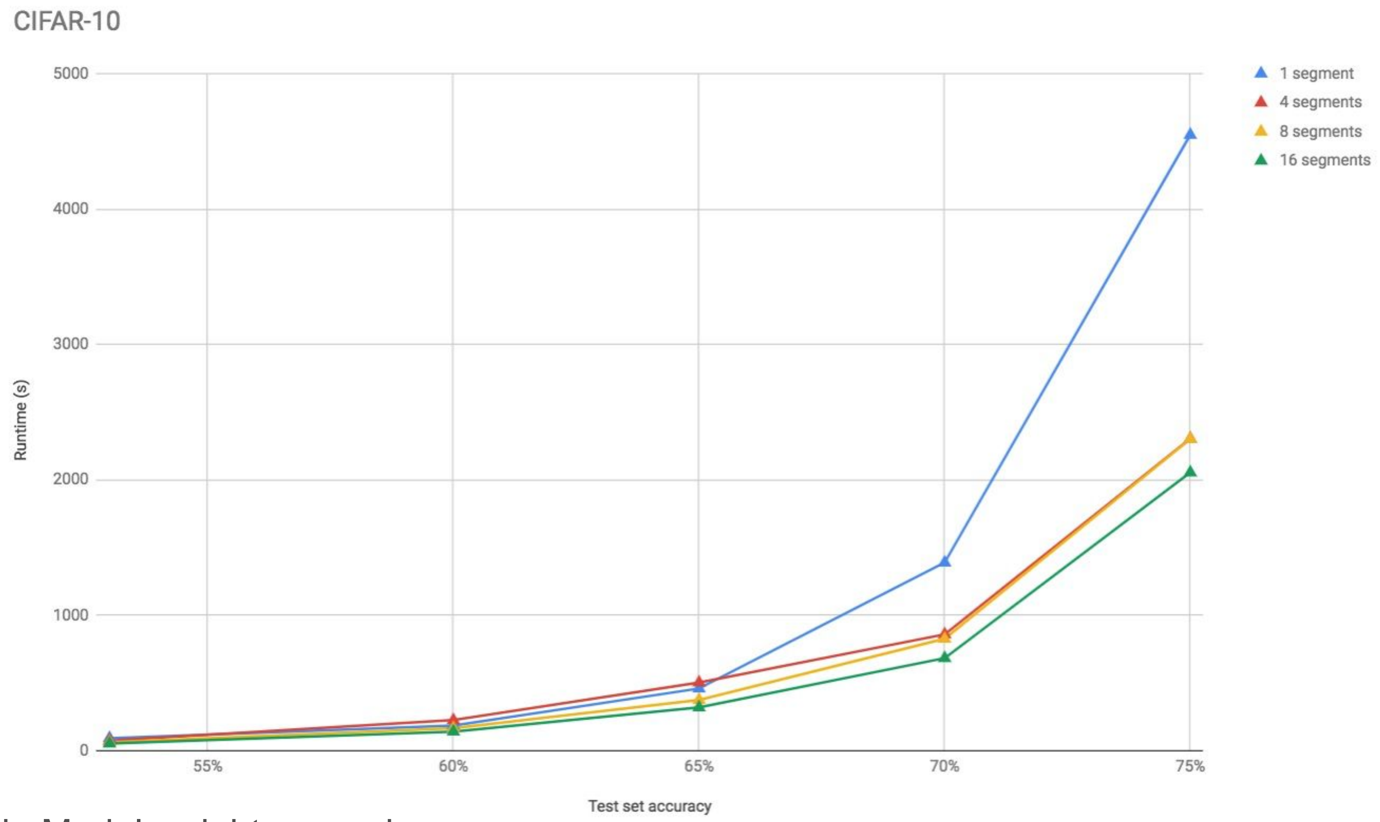
1-layer CNN - Test Set Accuracy (CIFAR-10)




```
model.add(Conv2D(32, kernel_size=(3, 3),
          activation='relu',
          input_shape=(32,32,3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(n_classes, activation='softmax'))
model.compile(loss='categorical_crossentropy',
            optimizer=Adam(),
            metrics=['categorical_accuracy'])
```

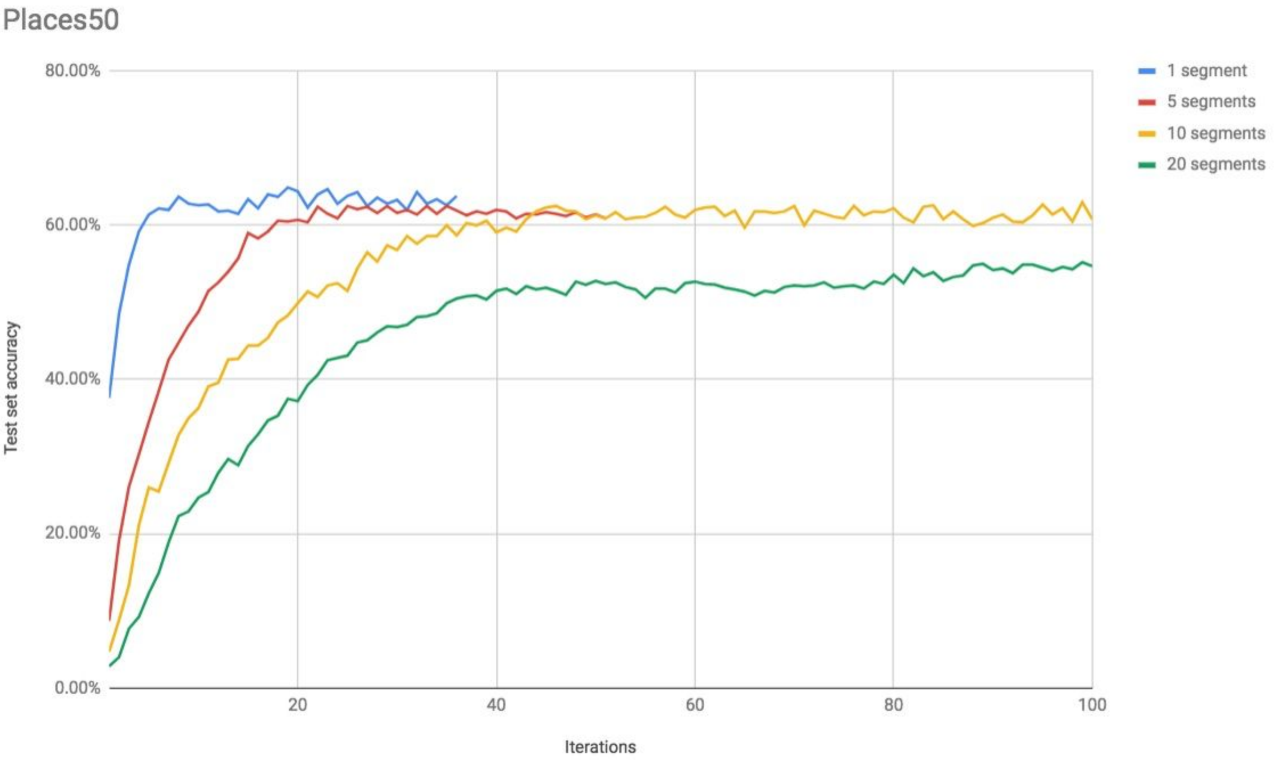
Method: Model weight averaging

1-layer CNN - Runtime (CIFAR-10)



Method: Model weight averaging

VGG-11 (Config A) CNN - Test Set Acc (Places50)



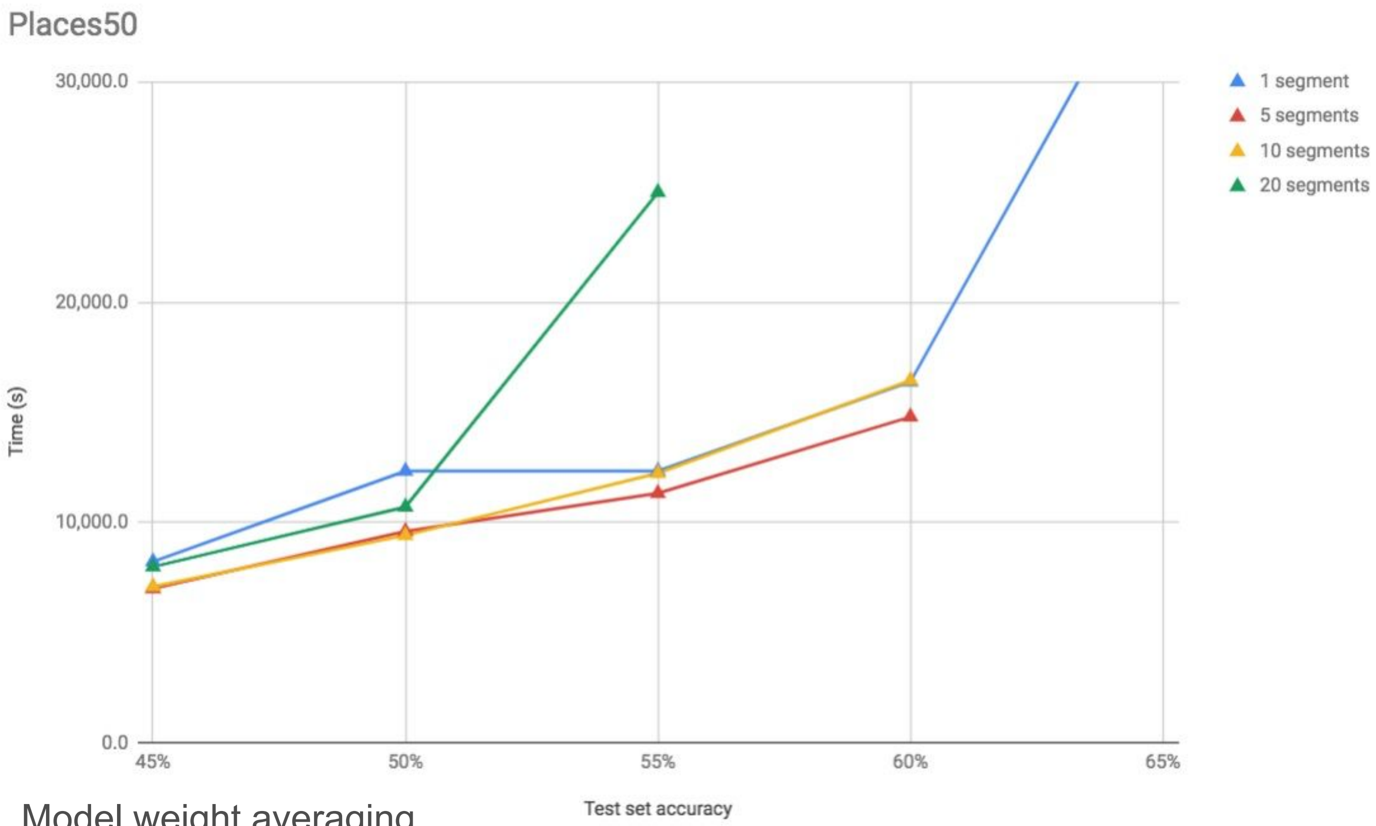
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

133M parameters

https://arxiv.org/pdf/1409.1556.pdf

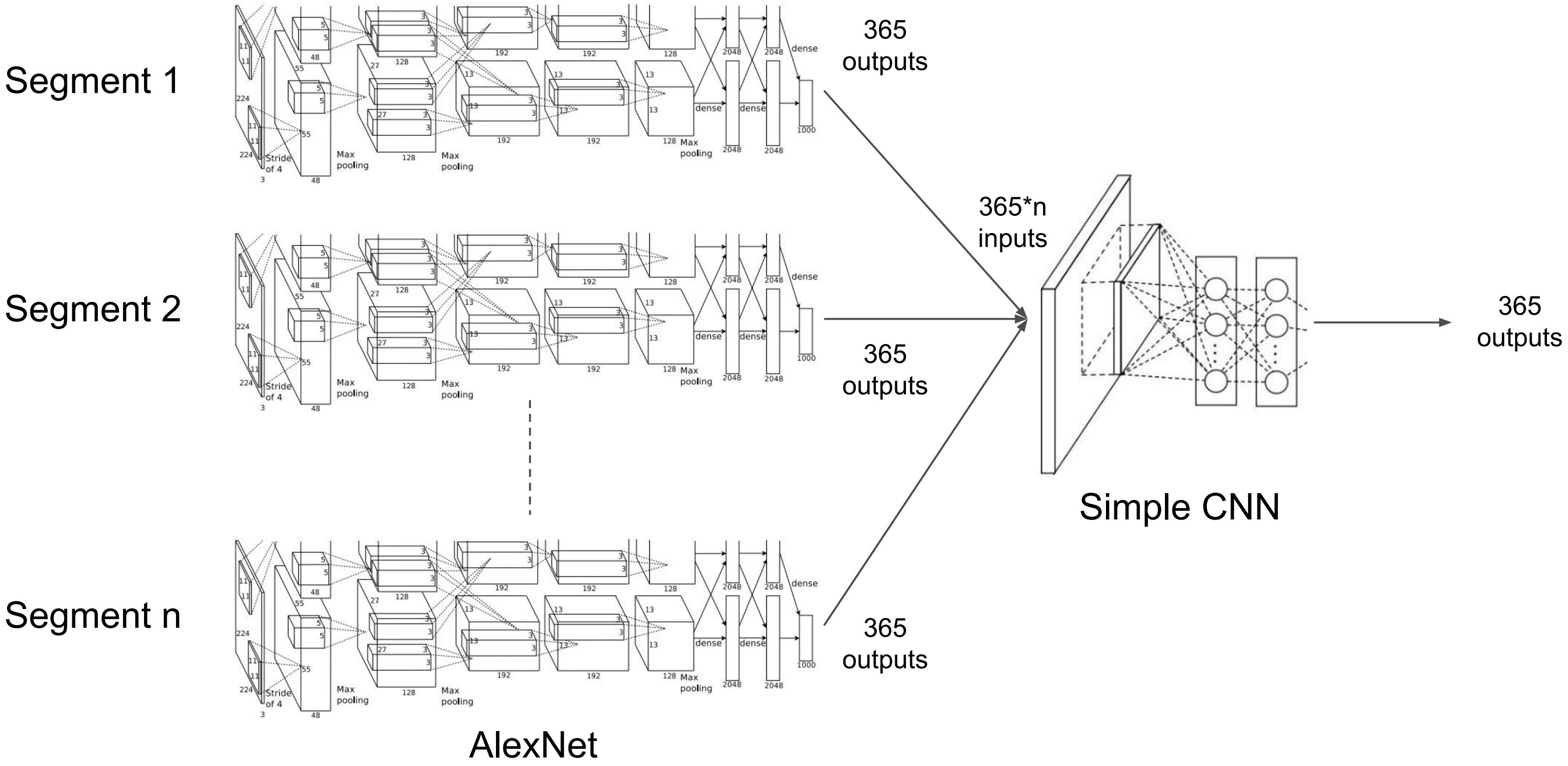
Method: Model weight averaging

VGG-11 (Config A) CNN - Runtime (Places50)



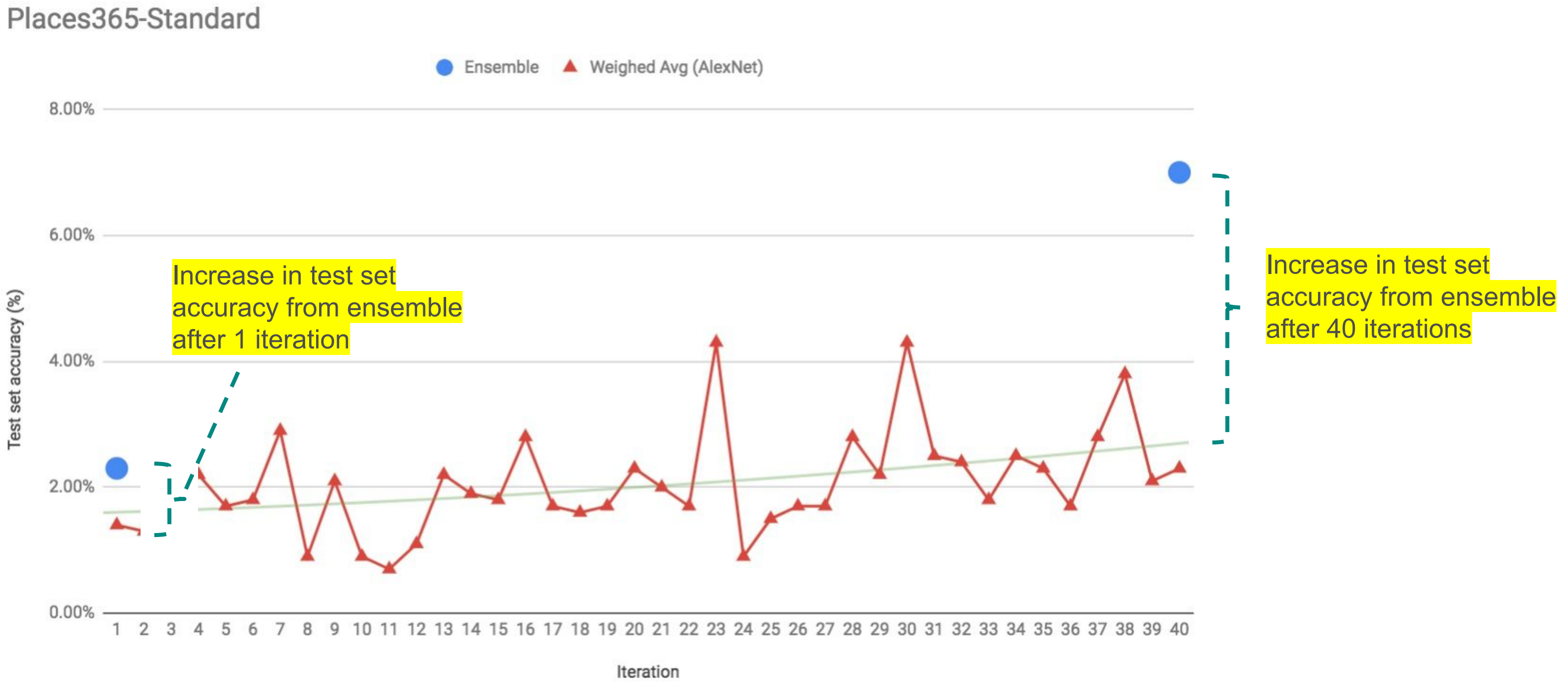
Method: Model weight averaging

Ensemble with Places365



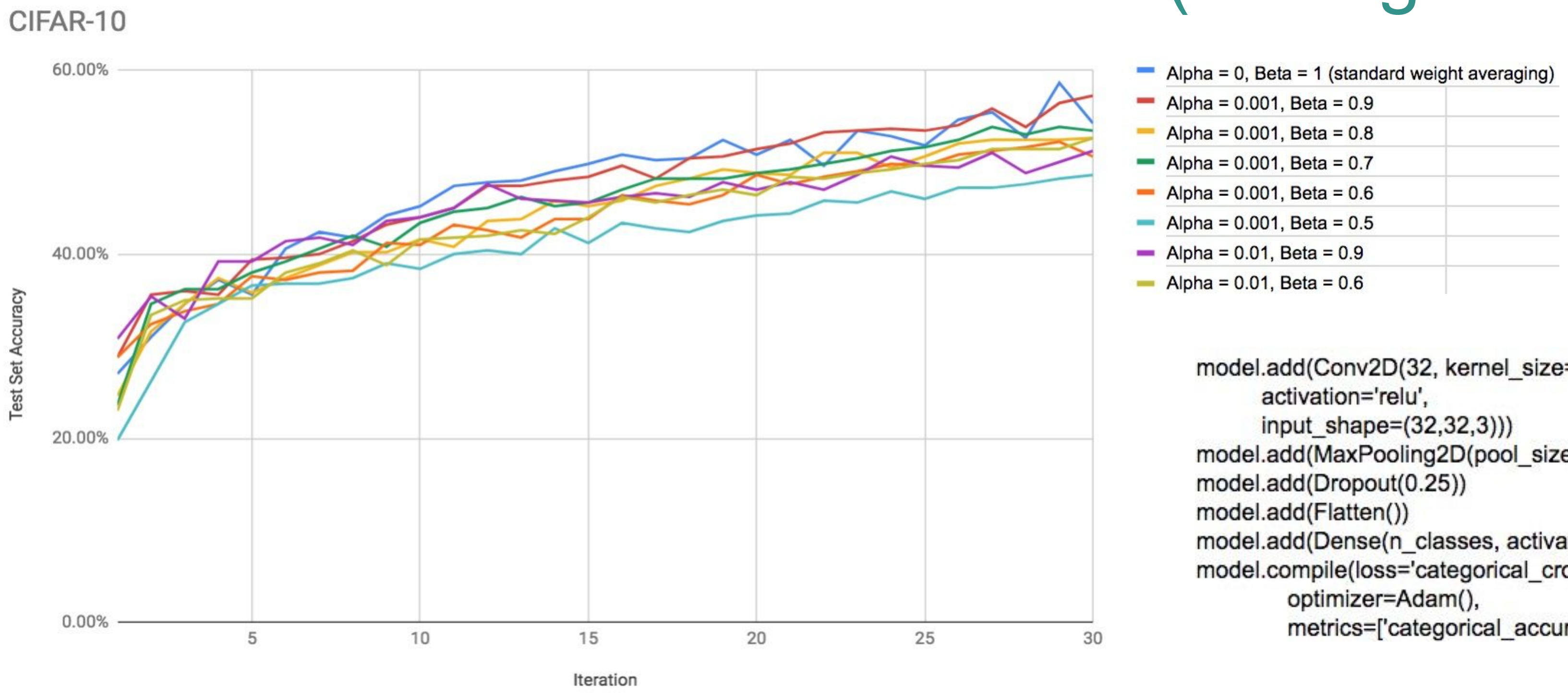
https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

AlexNet+Ensemble CNN - Test Set Acc (Places 365) (20 segments)



Method: Model weight averaging with simple ensemble CNN
https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

1-layer CNN - Test Set Accuracy (Places365) (20 segments)




```
model.add(Conv2D(32, kernel_size=(3, 3),
        activation='relu',
        input_shape=(32,32,3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(n_classes, activation='softmax'))
model.compile(loss='categorical_crossentropy',
          optimizer=Adam(),
          metrics=['categorical_accuracy'])
```

Method: Elastic averaging stochastic gradient descent (EASGD)
https://arxiv.org/pdf/1412.6651.pdf

Lessons Learned and Next Steps

Lessons Learned

- Distributed deep learning can potentially run faster than single node, to achieve a given accuracy
- Deep learning in a distributed system is challenging (but fun!)
- Database architecture imposes some limitations compared to Linux cluster

Infrastructure Lessons Learned

- Beware the cost of GPUs on public cloud!
- Memory management can be finicky
  - GPU initialization settings and freeing TensorFlow memory
- GPU configuration
  - Not all GPUs available in all regions (e.g., Tesla P100 avail in us-east but not us-west on GCP)
  - More GPUs does not necessarily mean better performance
- Library dependencies important (e.g., cuDNN, CUDA and Tensorflow)

Future Deep Learning Work*

- 1.16 (Q1 2019)
  - Initial release of distributed deep learning models using Keras with TensorFlow backend, including GPU support
- 2.0 (Q2 2019)
  - Model versioning and model management
- 2.x (2H 2019)
  - More distributed deep learning methods
  - Massively parallel hyperparameter tuning
  - Support more deep learning frameworks
  - Data parallel models

*Subject to community interest and contribution, and subject to change at any time without notice.

Thank you!

Backup Slides

Apache MADlib Resources

- Web site
  - http://madlib.apache.org/
- Wiki
  - https://cwiki.apache.org/confluence/display/MADLIB/Apache+MADlib
- User docs
  - http://madlib.apache.org/docs/latest/index.html
- Jupyter notebooks
  - https://github.com/apache/madlib-site/tree/asf-site/community-artifacts
- Technical docs
  - http://madlib.apache.org/design.pdf
- Pivotal commercial site
  - http://pivotal.io/madlib
- Mailing lists and JIRAs
  - https://mail-archives.apache.org/mod_mbox/incubator-madlib-dev/
  - http://mail-archives.apache.org/mod_mbox/incubator-madlib-user/
  - https://issues.apache.org/jira/browse/MADLIB
- PivotalR
  - https://cran.r-project.org/web/packages/PivotalR/index.html
- Github
  - https://github.com/apache/madlib
  - https://github.com/pivotalsoftware/PivotalR

Infrastructure Lessons Learned (Details)

Num	Category	Lessons learned	Notes
1	Cost	Beware the cost of scale testing with GPUs	Easy to spend $30K/month on scale testing on 2-3 clusters from 1-20 segments (worker nodes) on GCP with Tesla P100 GPUs.
2	GPU memory management	During initialization, set gpu_options.allow_growth=False	There are a lot of blog posts and forum answers where people recommend setting this to True. We tried that at first, but concluded it's not the best idea for our purposes. Setting it to True means it only requests a small amount of memory when you initialize the tensorflow session, but then every time you perform any operations (fit, evaluate, etc.) it will try to allocate more memory as needed. This is dangerous because you never know when you will run out of memory, especially if multiple GPU's are sharing. It's unpredictable and the errors are very difficult to sort out after it happens. Much cleaner to diagnose is setting it to False, where a fixed fraction of the GPU memory (see next option: per_process_gpu_memory_fraction) is allocated up front, and never grown after that. Any issues will come up right at the beginning, not suddenly after you've trained half the dataset!
3	GPU memory management	During initialization, set gpu_options.per_process_gpu_memory_fraction	This tells each segment what fraction of the GPU memory to use. If every segment has it's own GPU, then pick something close to 1.0. We have been using 0.9 just in case there are some small things that need to run (such as nvidia-smi tool, used for monitoring GPU memory usage.) If you want more than one segment to share a GPU, this has to be less than 0.5.
4	CPU memory management	CPU memory needs not always crystal clear.	All the hosts on our two gpu clusters are high memory machines (208 GB). We've never seen it use more than 50GB or so in htop, but at some point we were getting crashes with only 120GB and raising it to 208GB.
6	TensorFlow memory management	Be very careful about freeing TensorFlow memory.	The only guaranteed solution that works is to make sure the process that runs TensorFlow dies before any new TF session is started (killing the process will free the memory). Our current solution includes creating a TF session for each iteration and closing it at the end of the iteration.
5	Multi-GPU	Increasing GPUs per segment did not help beyond 2.	We've tried running Keras on only 1 segment per host with 1, 2, 4, and 8 GPU's per segment, and compared performance. 2 was a nice improvement from 1, but beyond that 4 and 8 didn't seem to add much.
7	GPU selection	Certain GPUs n/a in certain zones	Availability of a certain type of GPU may depend on the region in GCP. For ex Tesla P100 is not available in us-east but is available in us-west.
8	GPU selection	We are currenlty using the Tesla P100 GPU which is more expensive but way faster than Tesla K80.	
9	Library dependencies	CUDA, cuDNN and TensorFlow versions must be in sync.	We saw segmentation faults with the GPU setup if the TensorFlow version is newer than what is supported by cuDNN library. One solution is to downgrade the TF version or alternatively upgrade the cuDNN version The problem was that the cuDNN library was older than what TF 1.11 expected. We had installed tensorflow using pip install tensorflow-gpu which always gives us the latest TF. I downgraded the TF version to 1.9 so that it matches the cuDNN library pip install --upgrade --force-reinstall tensorflow-gpu==1.9.0 --user

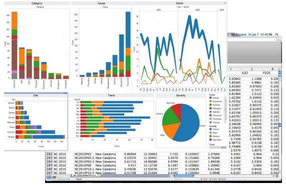
SQL Interface

```sql
-- fit
SELECT madlib_keras_fit (
                                        -- mandatory params
          source_table,                 -- source table
          model,                        -- model output
          dependent_varname,            -- dependent var cols/expr in source table
          independent_varname,          -- independent var col in source table
          model_arch_table,             -- model architecture
          model_arch_id,                -- model architecture id
          compile_params,               -- model compile params in a string
          fit_params,                   -- fit params as named params in a string
          num_iterations,               -- number of training iterations
          num_classes,                  -- number of classes

                                        -- optional params
          use_gpu,                      -- select GPU or CPU
          validation_table,             -- data to validate,
          name,                         -- user entered name
          description,                  -- user entered description
          initial_weights,              -- weights to initialize for training
          distributed                           -- run model centralized on one node or distributed
          );


-- predict
SELECT madlib_keras_predict (
          model,                         -- model to predict
          model_id                       -- id if model repository used
          data_table,                    -- table to predict on
          output_table,                  -- table to write predictions
          id_col_name                    -- row id in test table
          );
```

Greenplum Integrated Analytics

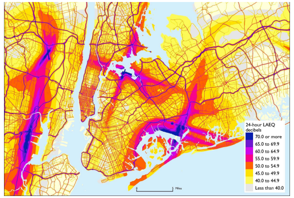
Data Transformation

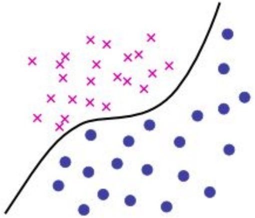
Text



Traditional BI

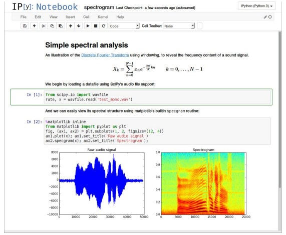
Geospatial

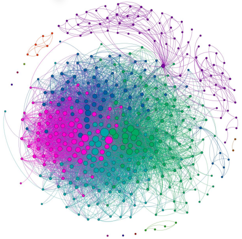




Machine Learning

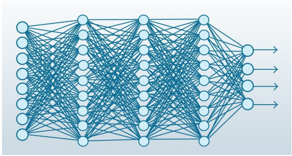
Data Science Productivity Tools





Graph

Deep Learning



Scalable, In-Database Machine Learning

Apache MADlib: Big Data Machine Learning in SQL

Open source, top level Apache project

For PostgreSQL and Greenplum Database

Powerful machine learning, graph, statistics and analytics for data scientists

- Open source https://github.com/apache/madlib
- Downloads and docs http://madlib.apache.org/
- Wiki https://cwiki.apache.org/confluence/display/MADLIB/

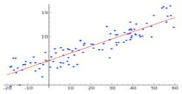
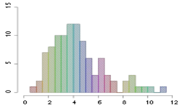
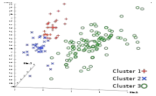
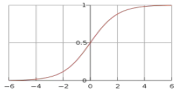
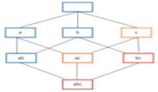
History

MADlib project was initiated in 2011 by EMC/Greenplum architects and Professor Joe Hellerstein from University of California, Berkeley.

UrbanDictionary.com:
mad *(adj.): an adjective used to enhance a noun.*

1- dude, you got skills.
2- dude, you got ***mad*** *skills.*

MADlib Functions

Supervised Learning
Neural Networks
Support Vector Machines (SVM)
Conditional Random Field (CRF)
Regression Models
- Clustered Variance
- Cox-Proportional Hazards Regression
- Elastic Net Regularization
- Generalized Linear Models
- Linear Regression
- Logistic Regression
- Marginal Effects
- Multinomial Regression
- Naïve Bayes
- Ordinal Regression
- Robust Variance

Tree Methods
- Decision Tree
- Random Forest

Unsupervised Learning
Association Rules (Apriori)
Clustering (k-Means)
Principal Component Analysis (PCA)
Topic Modelling (Latent Dirichlet Allocation)

Nearest Neighbors
- k-Nearest Neighbors

Graph
All Pairs Shortest Path (APSP)
Breadth-First Search
Hyperlink-Induced Topic Search (HITS)
Average Path Length
Closeness Centrality
Graph Diameter
In-Out Degree
PageRank and Personalized PageRank
Single Source Shortest Path (SSSP)
Weakly Connected Components

Utility Functions
Columns to Vector
Conjugate Gradient
Linear Solvers
- Dense Linear Systems
- Sparse Linear Systems

Mini-Batching
PMML Export
Term Frequency for Text
Vector to Columns

Sampling
Balanced
Random
Stratified

Time Series Analysis
- ARIMA

Data Types and Transformations
Array and Matrix Operations
Matrix Factorization
- Low Rank
- Singular Value Decomposition (SVD)

Norms and Distance Functions
Sparse Vectors
Encoding Categorical Variables
Path Functions
Pivot
Sessionize
Stemming

Statistics
Descriptive Statistics
- Cardinality Estimators
- Correlation and Covariance
- Summary

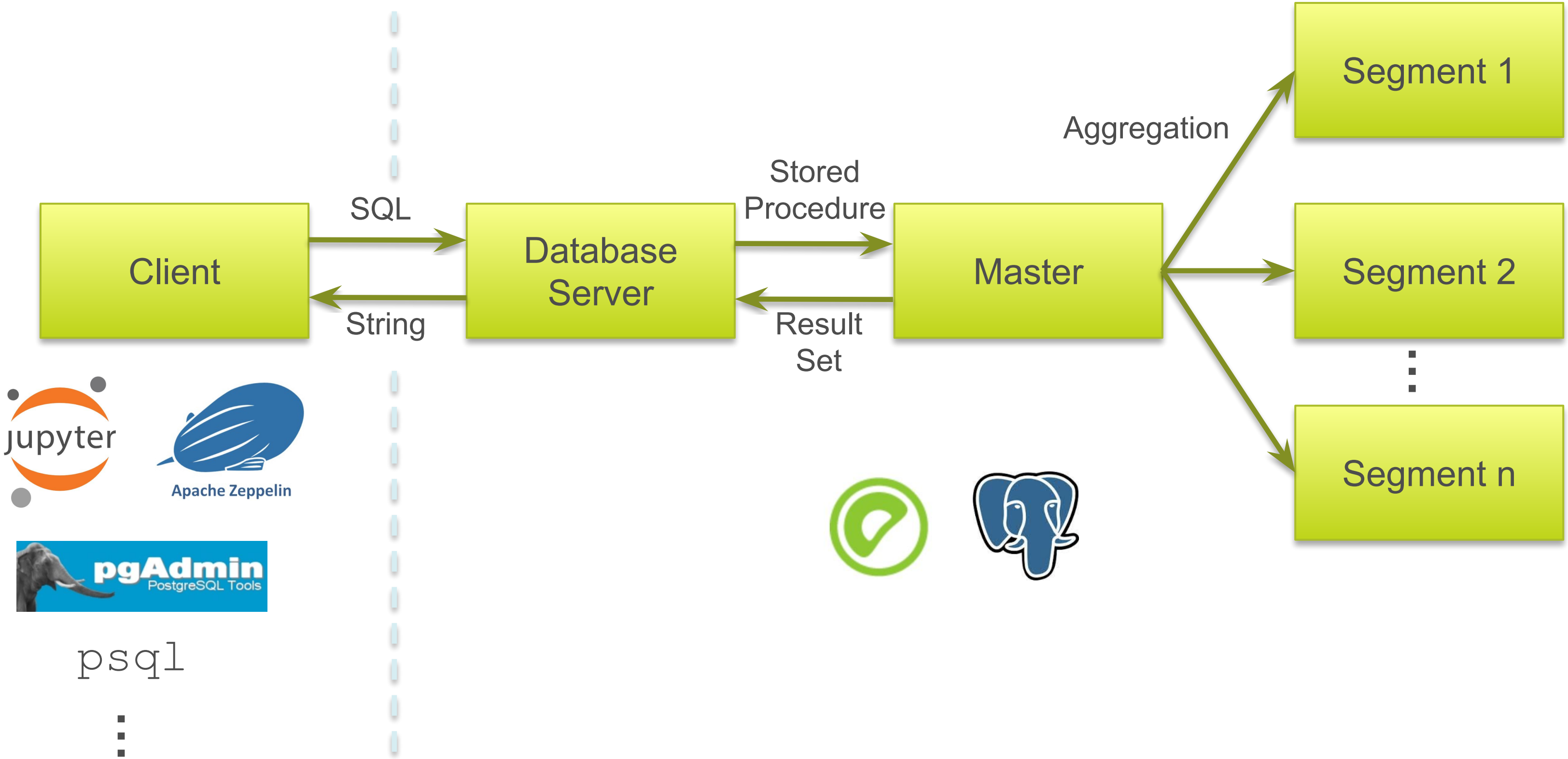
Inferential Statistics
- Hypothesis Tests

Probability Functions

Model Selection
Cross Validation
Prediction Metrics
Train-Test Split

Aug 2018

Execution Flow



Architecture

User Interface — SQL

High-Level Iteration Layer
(iteration controller) — Python

Functions for Inner Loops
(implements ML logic)

Low-level Abstraction Layer
(array operations,
C++ to DB type-bridge, …)

C++

Eigen

boost
C++ LIBRARIES

RDBMS Built-in Functions

C API
(Greenplum, PostgreSQL, HAWQ)