CRAY

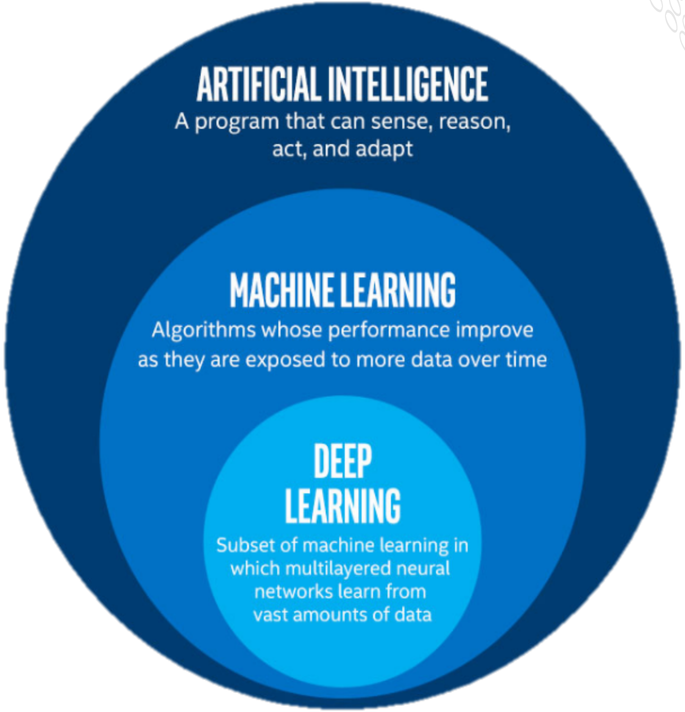
Precipitation Nowcasting Leveraging Deep Learning and HPC Systems to Optimize the Data Pipeline

Agenda

- Introduction
- Motivation
- Dataset
- Prediction Modeling
- Data Pipelines
- Results
- Q&A

Introduction

- **Nowcasting**
  - Predict precipitation locations and rates at a regional level over a short timeframe
  - Traditional Approach: Numerical Weather Prediction
  - Requires an extended lead time between newly acquired data and release of forecasts
- **Deep Learning**
  - Branch of Machine Learning based on Neural Networks
  - Deep implies multiple layers of computation between inputs and output
  - Pattern Matching



Motivation

- **Why is short-term nowcasting important?**
  - Provide reliable ride-to-work forecasts
  - Predict rapid formation of severe precipitation events: Flash Flood Warning!
- **Why Deep Learning?**
  - Traditional Nowcasting relies on NWP, slow to respond to new data
  - Deep Learning learns from past rainfall patterns
  - Trained models are computationally cheap to utilize
- **Why HPC?**
  - A lot of data! Training can utilize decades of observations
  - Models can be trained and inferred for small regions in parallel

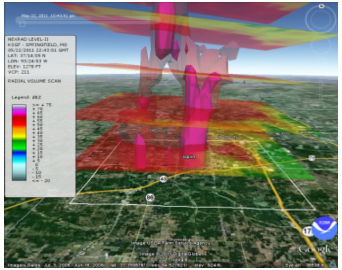
Prototype Nowcasting System

- **Small: 4 stations**
  - KATX (Seattle), KTLX (Oklahoma City), KTLH (Tallahassee), KBUF (Buffalo)
- **Variable sized dataset, as large as 7 years of historical rainfall data**
  - Total size (raw data): 4TB
  - Total size (processed): 684GB
- **Examine Pipeline Performance and Bottlenecks**
- **Explore Nowcasting Performance via Deep Learning**

Dataset Processing

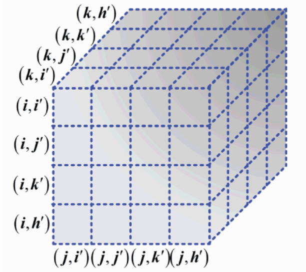
Data Collection

- Historical Radar Data (NETCDF)
- Geographical Region
- Days with over 0.1 inches of precipitation, info from NOAA – NCDC
- Radar scans every 5-10 minutes throughout the day



Transformation

- Raw radial data structure converted to evenly spaced Cartesian grid (Tensors with float 32)
- Resolution scaling and clipping
- Configure dimensionality
- Sequencing
- 2 channels – Reflectivity, Velocity
- Uses Py-ART package



Sampling

- Time-series
- Inputs and Labels
- Random sampling

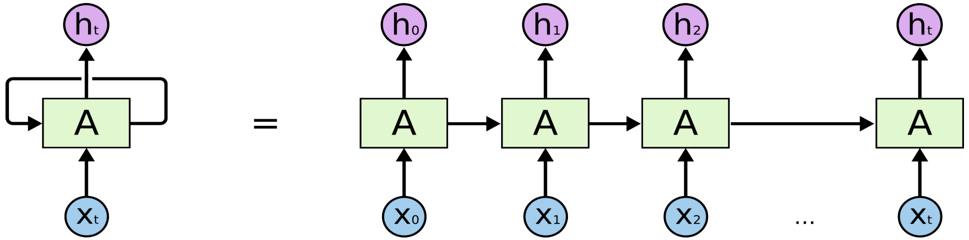
BigDL

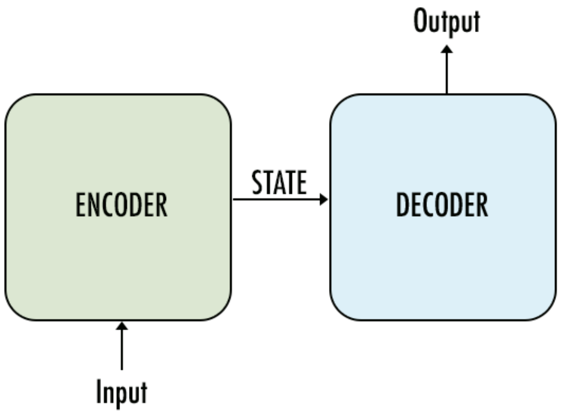
Framework

- Apache Spark on Urika-XC/GX
- Implemented in Jupyter notebooks and Python

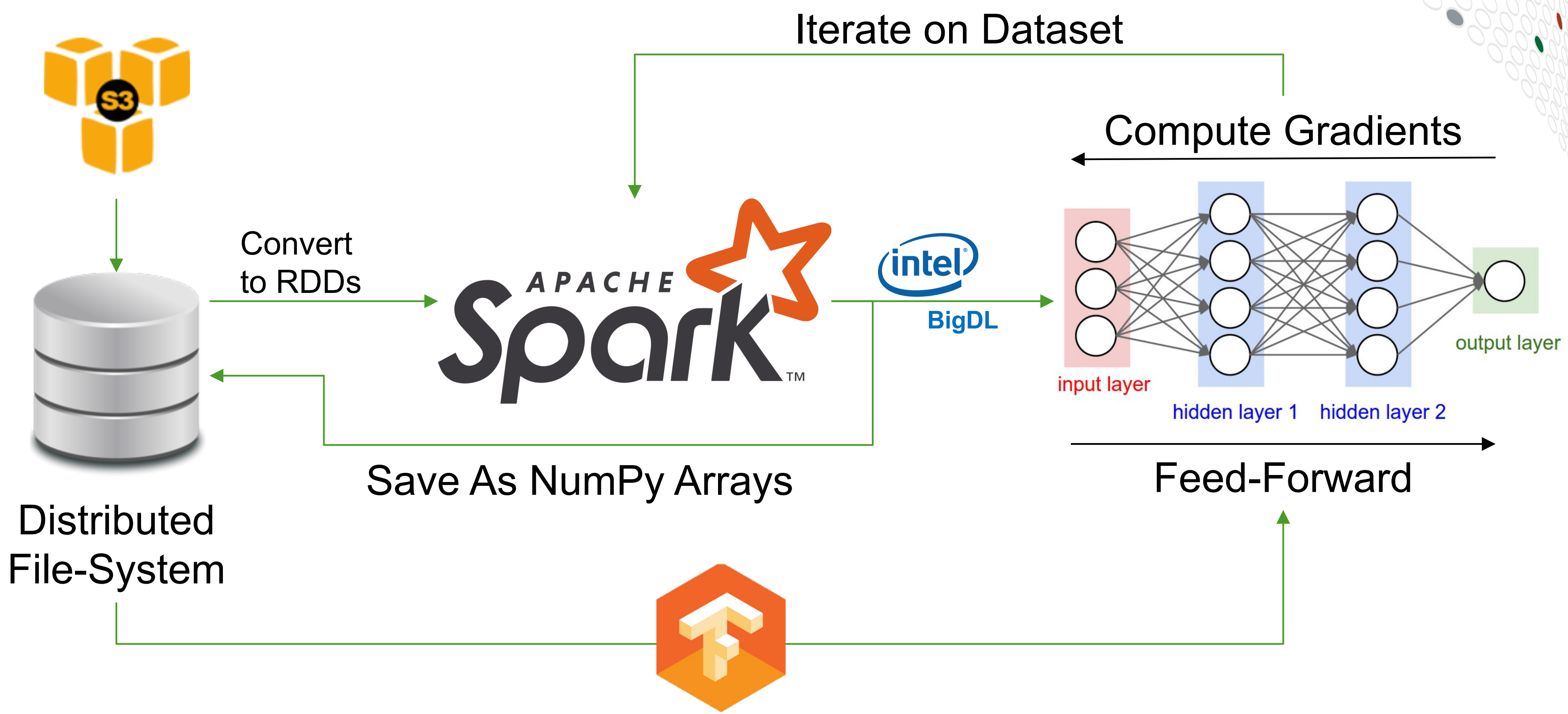
Prediction Modeling

- **Convolutional Recurrent Neural Network**
  - Convolutional Neural Network – Spatial Patterns
  - Recurrent Neural Network – Temporal Patterns
  - ConvLSTM – Convolutional Long Short-Term Memory Network
- **Sequence to Sequence**
  - Encoder Decoder
  - Use recent history to predict future changes

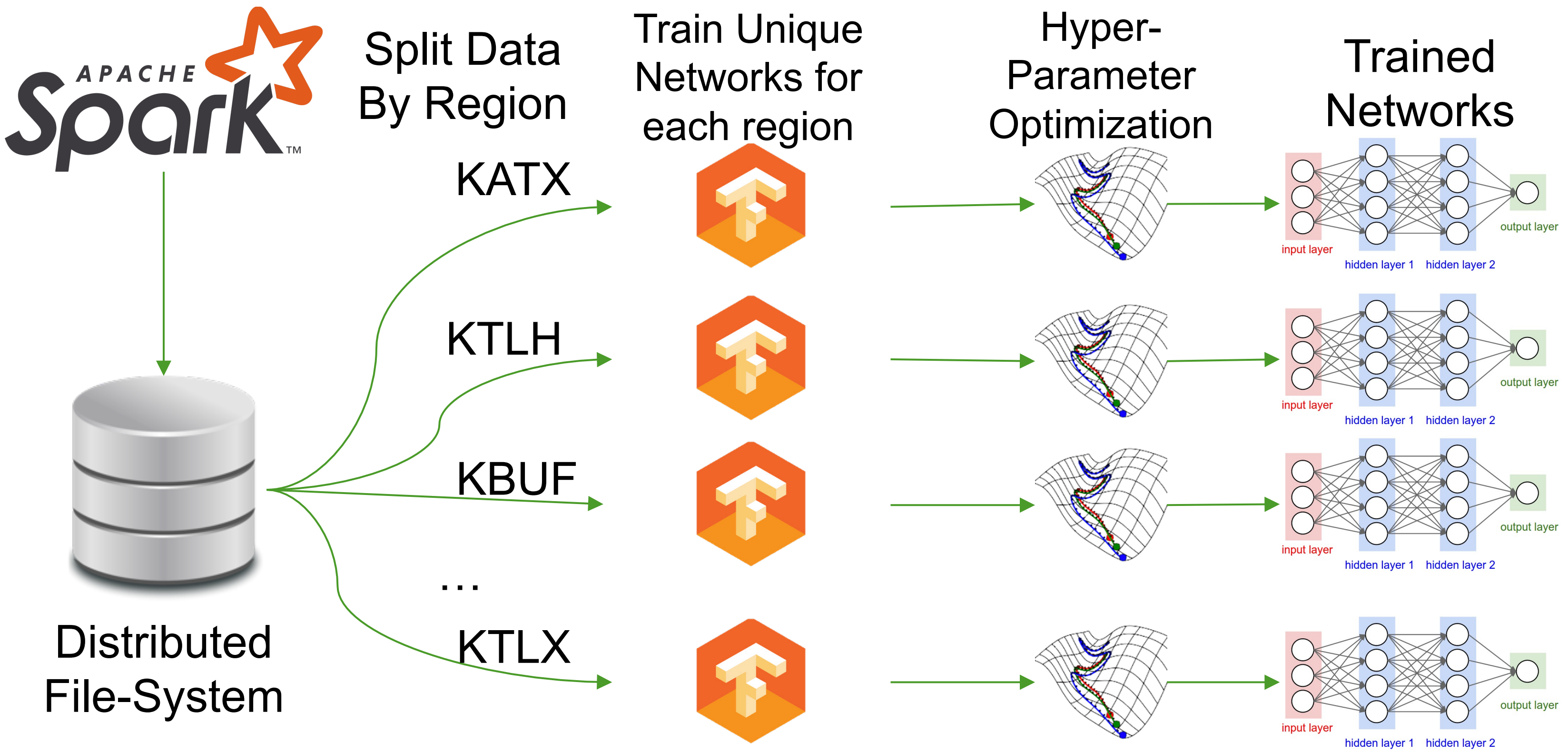




Pipeline: Data Processing



Pipeline: Distributed Training



Idealized Training Timeline

- **Station: KATX**
- **Dataset size:**
  - 118,342 Sequences
  - 101GB
- **Parameters**
- **Systems:**
  - Data processing: Cray Urika-GX – 1024 cores
  - Training: Cray CS-Storm – 8 Nvidia P100 GPUs

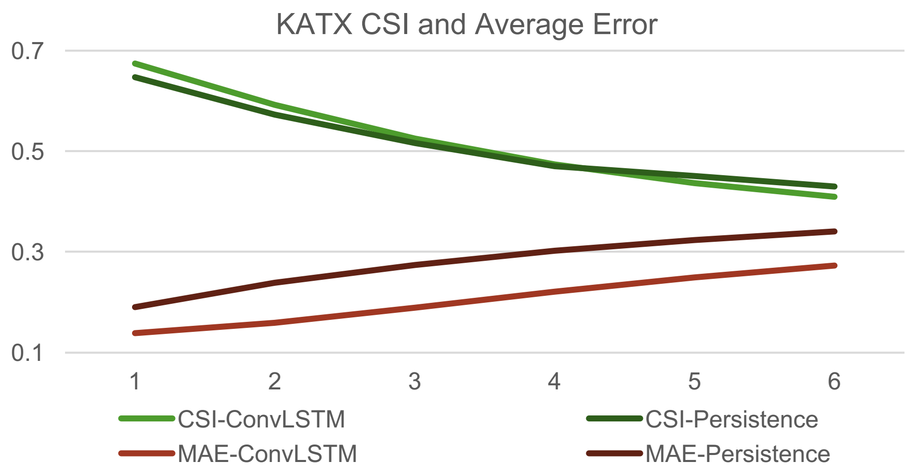
Process	Wall-Time	Proportion
Download	13 hours	32%
Spark	4 hours	10%
Training	24 hours	58%
Inference	10 seconds	0%

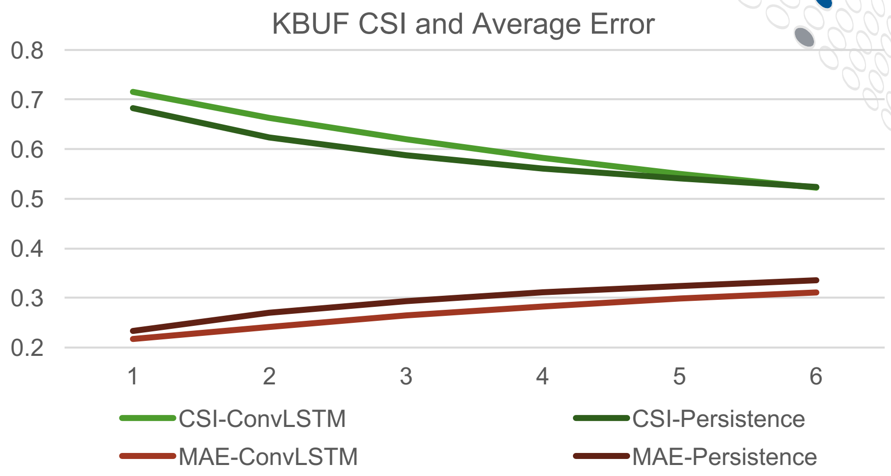
Scaling

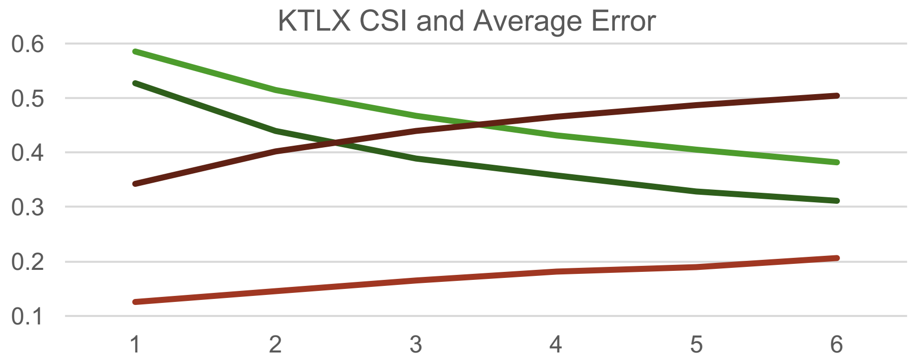
- Tensorflow via Cray MPI Com. Plugin
- Nvidia Tesla P100 GPUs
- Batchsize of 4 samples per device
- Throughput in Samples/Second

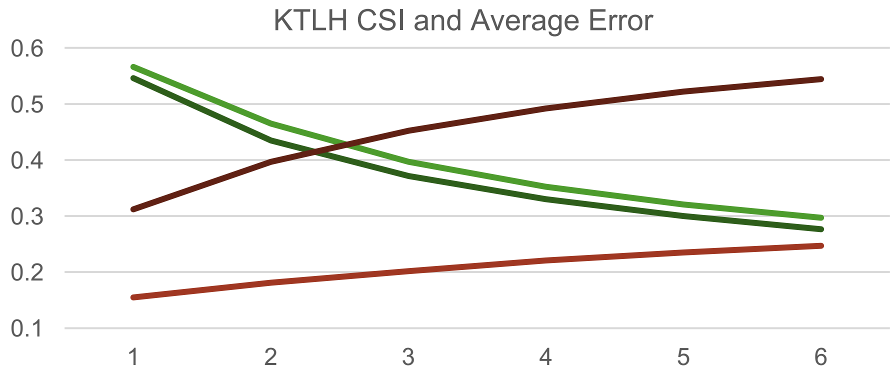
Device Count	Throughput	Scaling Efficiency
1	25.8	1.0
2	51.6	1.0
4	102.7	.995
8	205.4	.995
16	410.5	.994

Model Performance





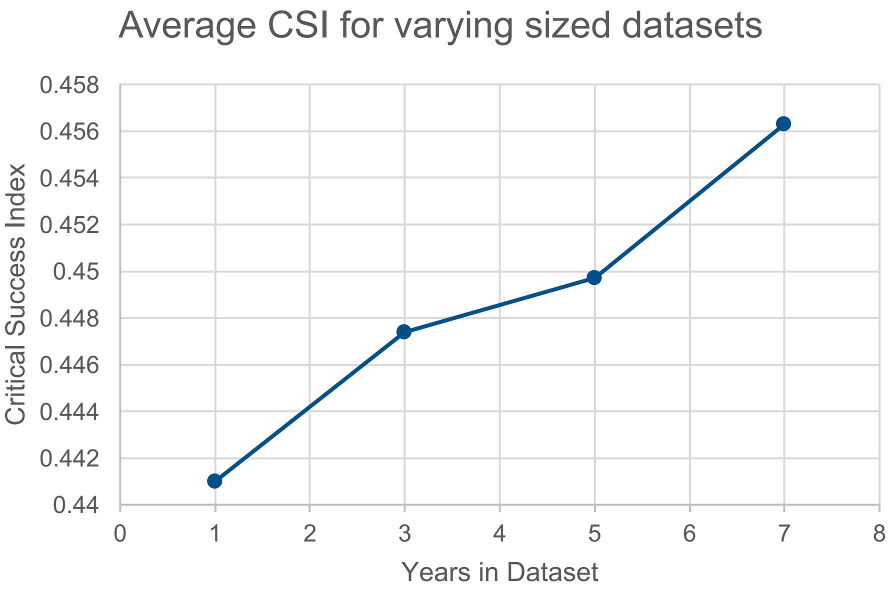




Effect of dataset size

- More data is correlated to higher performing prediction model
- Station: KATX (Seattle)

Years	Size (GB)	Sequences
1	40	43,171
3	109	118,342
5	143	155,337
7	217	235,301



Sample Prediction + Q/A