Herbert Wertheim College of Engineering
UNIVERSITY of FLORIDA

# OptML: Fine Tuning Hyperparameters in the Cloud

Team: Deep Optimization

POWERING THE NEW ENGINEER TO TRANSFORM THE FUTURE

# Meet the Team

- Abhishek “The Snow” Parekh
  - Bringing the chill to Florida
- Ruben “The Hail” Vazquez {Team Coordinator}
  - The heavy hitter
- Justin “The Rain” Calean
  - Breathing life into this project

# Goals

- The creation of an tool for use in speed up the process of finding the right hyperparameter configuration.
- Using common frameworks, implement a hyperparameter tuning tool

# Deliverables

- A web application that shows an optimal configuration for hyperparameters.
  - Input: a model, evaluation metric, dataset, hyperparameters to test, and values
  - Output: The different Hyperparameter Configurations and their error value

# Machine Learning

- Uses statistical methods to develop and train models that can “learn” the relationship between a set of inputs and a set of outputs
- The parameters in a learning algorithm that are determined by the implementers of the learning algorithm
- Choosing a good set of hyperparameter values is not trivial

# Machine Learning Algorithms

- Least Mean Squares (LMS) algorithm
  - Linear search space
  - Gradient Descent
- Backpropagation algorithm
  - For artificial neural networks
  - Passes errors back to processing elements
    - Errors used to update the weights
  - Hyperparameters - network size, learning rate

# Optimization Approaches

- Three main approaches
  - Grid Search
  - Random Search
  - Bayesian
- Other ones have more specific usage and not enough support.

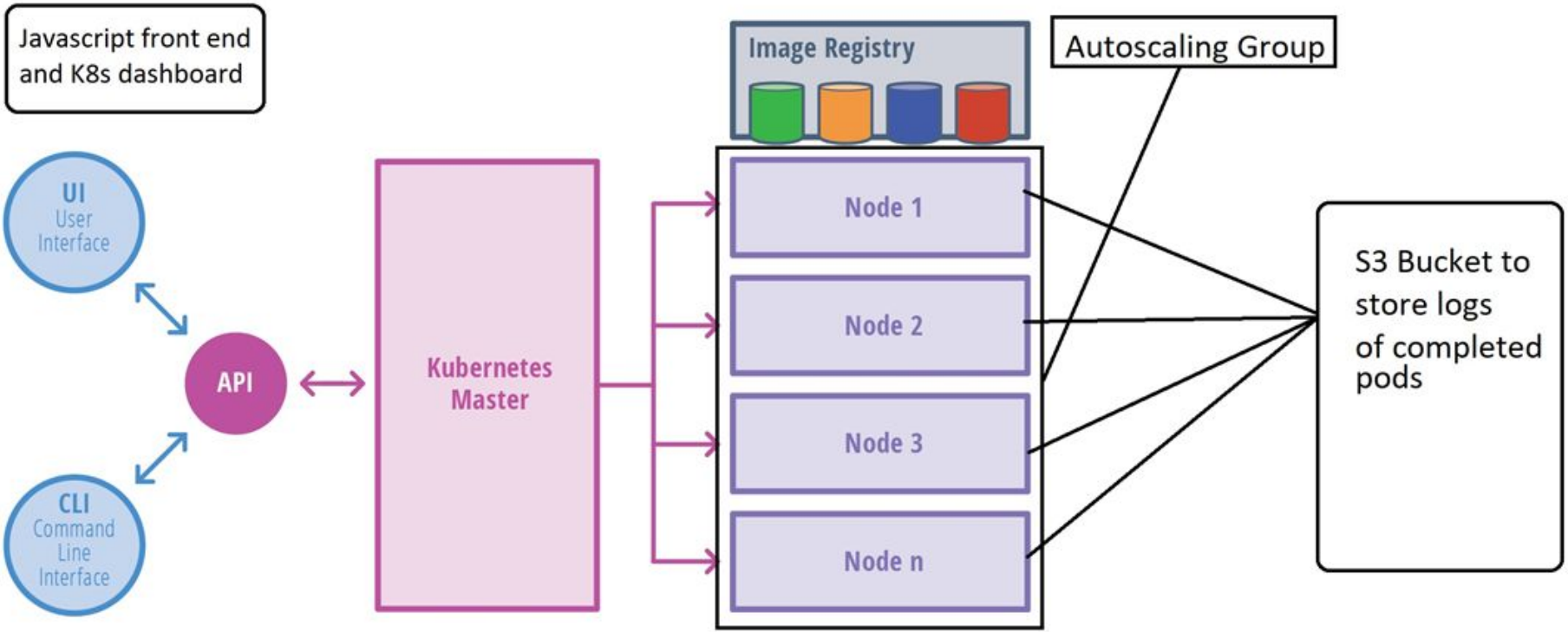
# Docker

- Setup the runtime environment.
  - Fast
    - Avoids setup time to install most of the software
  - Easy
- Open source images for containers.
- Programs must be pre-compiled to avoid licensing issues.

# Kubernetes Features

- Autoscaling
- Rolling Updates
  - Using Deployment Controller in Kubernetes
- Container Orchestration
  - YAML scripts to bring up or tear down the cluster
- Load Balancing
  - Using Amazon Elastic Load Balance and Kubernetes service
- High Availability
- Fault Tolerance
- Private Docker Image Registry

# Kubernetes Architecture



# Architecture

- Kubernetes Master EC2 instance starts with init script
- Autoscaling group spawns EC2 instances to act as Kubernetes Nodes which connect to this master via its public IP.
- The join token for the node is queried to the master apiserver using REST calls
- Expose the Kubernetes dashboard via Kubernetes service type NodePort

# Life Cycle

- We use the Job API in Kubernetes to run a ML algorithm
- The pods will be scheduled on the nodes with enough CPU and memory to the run the job
- The pods will terminate generating logs
- We push the logs to the S3 bucket and process them to generate the data

# Machine Learning Interface


- Driver file
  - Calls the ML functions
- Function file
  - generate_data()
    - Generate synthetic data or import real-world data
  - initialization()
    - Set up the data structures required for the algorithm
  - train()
    - Determine the values for the learning parameters
  - test_error()
    - Obtain the performance of the algorithm on a set of test data

# Machine Learning Interface (con't)

- Function file (con't)
  - Auxiliary functions (depends on algorithm)
    - Sigmoid and ReLU functions for the ANN
    - Kernel function for the KLMS algorithm

# Experiment Setup

- Create an EC2 Instance
- Run setup python file
- Kubernetes does the rest
- See logs in S3 Container
- Profit $$$

# Python Setup File

- SSH to the Kubernetes Master
- Deploy the ML Algorithm
- Deploy the hyperparameter file
- Deploy the job YAML file to Kubernetes Master
- Track time to finish job

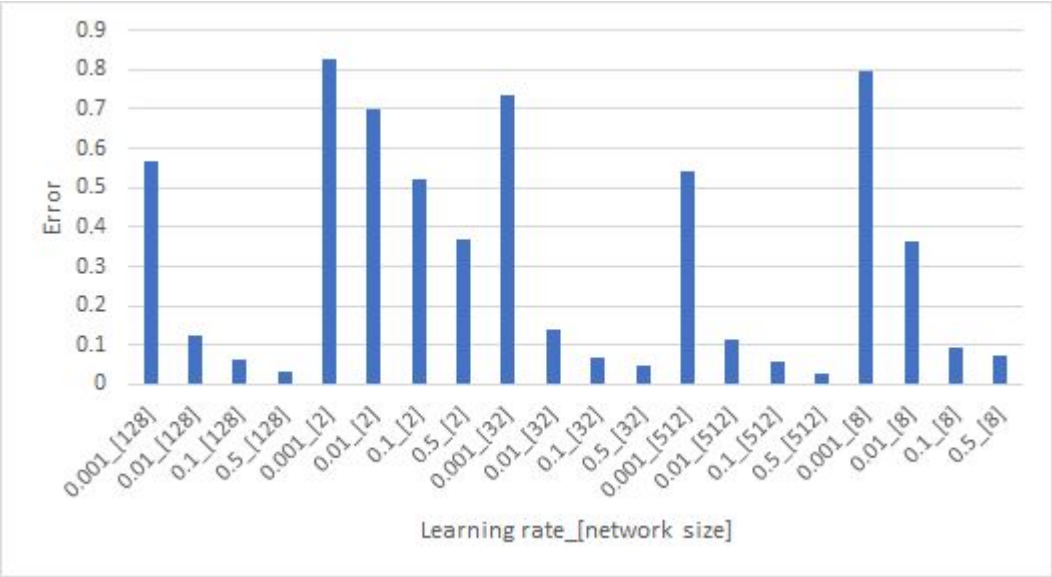
# Hyperparameter File

```
0.001 2
0.001 8
0.001 32
0.001 128
0.001 512
0.01 2
0.01 8
0.01 32
0.01 128
0.01 512
0.1 2
0.1 8
0.1 32
0.1 128
0.1 512
0.5 2
0.5 8
0.5 32
0.5 128
0.5 512
```

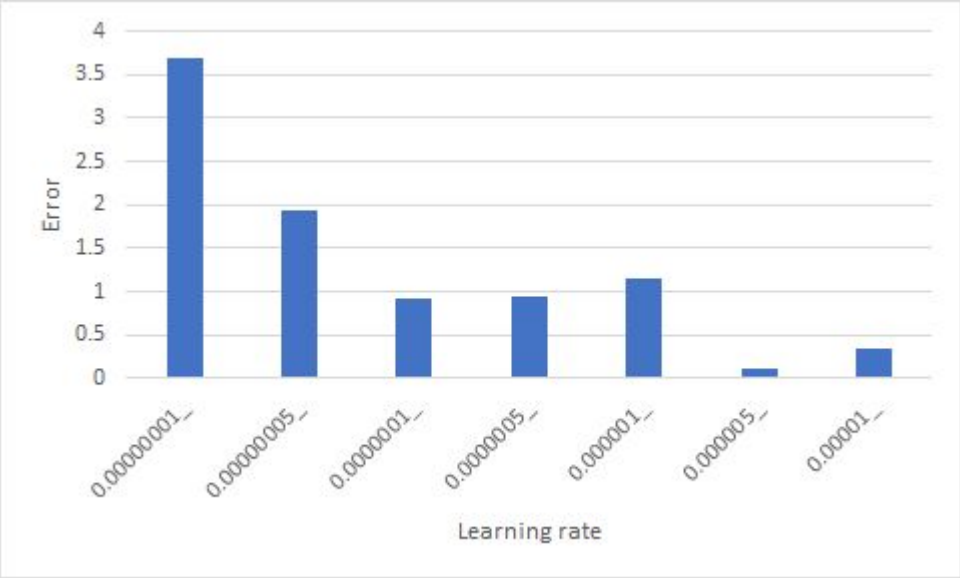
# Job YAML file

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: deepoptimization1
spec:
  template:
    spec:
      containers:
      - name: optml
        image: abhishekparekh1/deepoptml:deadpool
        command: ["python3","/app/EC2_Kubernetes/ML_Algorithms/ANN/ANN_TF_Driver.py","--DEBUG 0","--learning_rate 0.001",--network_size 2]
      restartPolicy: Never
  backoffLimit: 1
```

# Results


0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
Error
0.001_[128]
0.01_[128]
0.1_[128]
0.5_[128]
0.001_[2]
0.01_[2]
0.1_[2]
0.5_[2]
0.001_[32]
0.01_[32]
0.1_[32]
0.5_[32]
0.001_[512]
0.01_[512]
0.1_[512]
0.5_[512]
0.001_[8]
0.01_[8]
0.1_[8]
0.5_[8]
Learning rate_[network size]

# Results


4
3.5
3
2.5
2
1.5
1
0.5
0
Error
0.00000001_
0.00000005_
0.0000001_
0.0000005_
0.000001_
0.000005_
0.00001_
Learning rate

# Speedup Calculation

| | Time | | |
|---|---|---|---|
| | Serial | Parallel | Speedup |
| ANN | 1126.58 | 60.89622 | 18.5 |

| | Time | | |
|---|---|---|---|
| | Serial | Parallel | Speedup |
| LMS | 34.60083 | 2.601566 | 13.3 |

# Conclusion

- Prototype is working as intended
- Grid search is embarrassingly parallel
- Greater than 10x speedup for algorithms

# Future Work

- Improving job creation
- Improving the user interface for the web application
- Adding a asynchronous queue service
- Testing with a larger cluster
- Adding support for other frameworks
- Testing other optimization techniques
- Include Support for Neural Architecture Search

# Stop…Demo Time