Cuttlefish:
A Library For Building Elastic Distributed Neural Networks

Teresa Nicole Brooks, Rania Almajalid , Yu Hou and Abu Kamruzzaman

Computer Science, Pace University

Outline

INTRODUCTION

- Machine learning and more specifically the use of Neural Networks have many applications in both the research and commercial software.

- Though most machine learning techniques and algorithms employed today were developed over 20 years ago the rise of cheap, powerful processors (GPUs and CPUs) and higher capacity storage has allowed these techniques and algorithms to be used at scale.

- Our project about the implementation of Cuttlefish, a library for building configurable, fault tolerant, distributed neural networks.

BACKGROUND

- Deep Neural Networks
- Docker
- TensorFlow
- Amazon EC2 Container Service (ECS)

MOTIVATION

- The motivation for Cuttlefish was born from the observation that neural networks at their core are computational graphs and more specifically directed computational graphs, where each neuron in the graph is a single computational unit.
- Tasks for training models and performing inference in neural networks are inherently parallelizable. Hence, it should be possible to implement a library that allows users to create a fault tolerant, elastic distributed neural network, using configurable hyperparameters to drive the dynamic creation of a directed graph of neurons (a neural network).
- The neurons in the network would need an efficient mechanism to communicate and pass data from one neuron to another.

MOTIVATION

- The technologies and approaches needed to implement such a library:
  - Physical Representation of Neurons - Docker Containers

    The use of Docker to represent a physical neuron was an natural choice because Docker containers are self contained units that enable you to provide everything an application would need to run and nothing more; this includes an operating system, file system (volatile file system), and any needed software, frameworks or tools. Thus, allowing you to configure & tune resources such as CPU/GPUs, memory etc that are the most appropriate for the calculations on a per node basis.
  - Centralized Storage Of Weighted Model Parameters - Distributed NoSql Database.

    For our centralized parameter store, because the data is not highly relational and we would need a scalable solution distributed key, value stores such a Amazon Web Services' DynamoDB would be a good fit.

MOTIVATION

- Interneuron Communication.
  - For inter neural network communication we would need an inbound and output queue for each neuron in the network in order to pass data such as computed weights, and training model data from node to node.
  - To implement such communication fault tolerant message broker technologies such as rabbitmq or Apache Kafka are good choices.
- Automation of ”Spinning” Up Elastic Neural Networks.
  - Infrastructure orchestration software such as Kubernetes, Mesos and Amazon Web Service’s Elastic Container Service are tools used in both test and production environments to automate orchestrations for creating, configuring and managing docker containers, and hence makes them a natural fit for this task.

OUR APPROACH

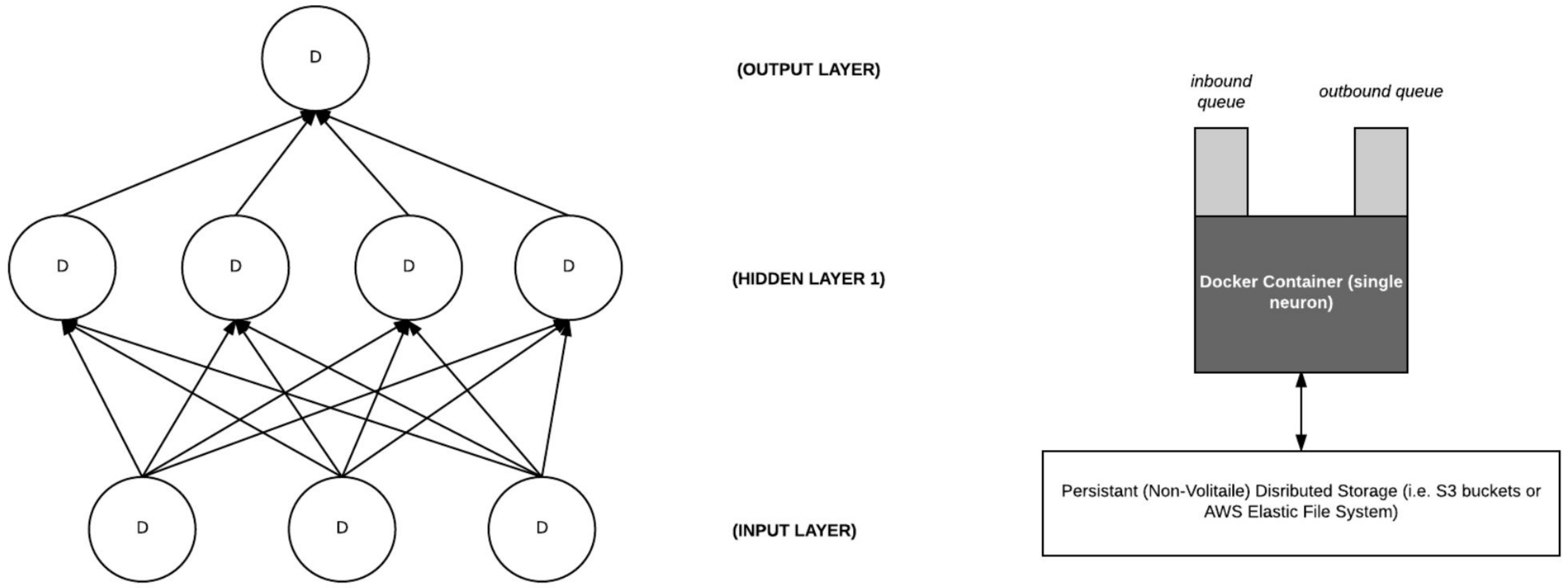

D
(OUTPUT LAYER)
(HIDDEN LAYER 1)
(INPUT LAYER)
inbound queue
outbound queue
Docker Container (single neuron)
Persistant (Non-Volitaile) Disributed Storage (i.e. S3 buckets or AWS Elastic File System)

Fig. 1. Diagram example neural network nodes as docker containers. Note, the persistent distributed storage represent non-volatile storage for localized data per docker container. Persistent storage is needed to provide fault tolerance for shared and unshared data.

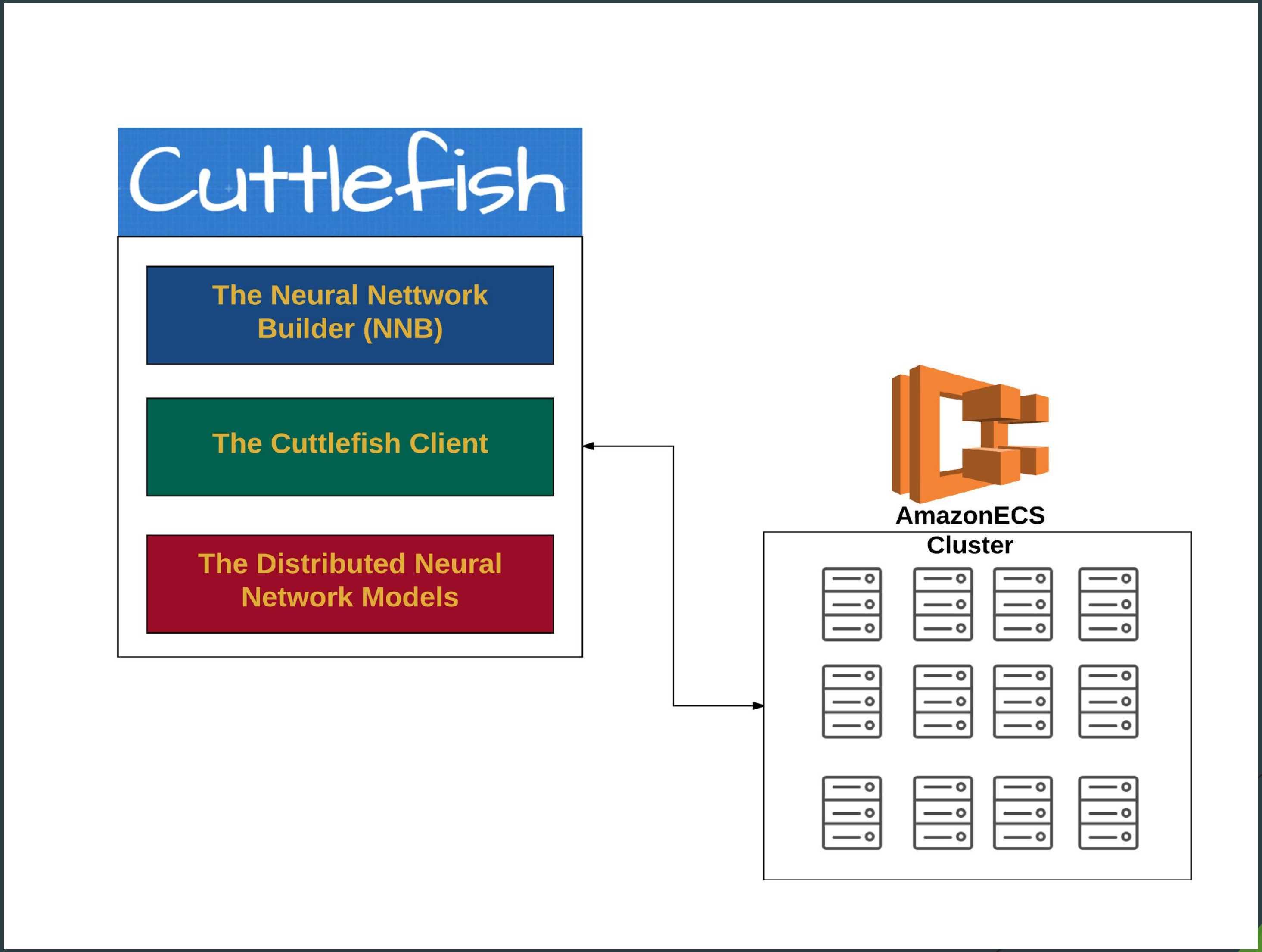
OUR APPROACH

- Computation Graph Distribution
  - Cuttlefish, will use the TensorFlow functionality along with the configuration of each docker container's system resources (memory, number of CPUs etc) to attempt to force TensorFlow's placement algorithm to map one node in the computation graph to one Docker container.
- Configuring Neural Network (Hyperparameters)
  - With Cuttlefish users define the configuration and shape of their neural network's hyperparameters in a yaml file.

OUR APPROACH

- Automation & Orchestration: Creating Docker Containers As Per Cuttlefish Configuration File
  - Cuttlefish’s ”build” functionality will use the Amazon Web Services’ Elastic Container Service’s (ECS) API [2] and it’s user defined elastic neural network configuration files to configure and build a distributed neural network using a cluster of docker containers (TensorFlow cluster).
- Data Set
  - We are using the MNIST data set for training distributed neural networks created by Cuttlefish. ”The MNIST database of handwritten digits has a training set of 60,000 examples, and a test set of 10,000 example.

SYSTEM ARCHITECTURE


Cuttlefish
The Neural Nettwork Builder (NNB)
The Cuttlefish Client
The Distributed Neural Network Models
AmazonECS Cluster

SYSTEM ARCHITECTURE

- Amazon EC2 Container Service (ECS)

  Amazon Web Services EC2 Container Service (ECS) is the heart of Cuttlefish’s system architecture. ECS enables users to easily automate orchestration of configuring and deploying a cluster of docker containers.

- Amazon Auto Scaling Groups

  Auto Scaling Groups allow you to treat a collection of EC2 instances as “a logical grouping for the purposes of instance scaling and management.”

- Logging & Monitoring

  By default AWS allows the use of the CloudWatch service to support logging and monitoring ECS containers and their corresponding EC2 instances.

SYSTEM ARCHITECTURE

- Amazon Elastic Compute (EC2) Virtual Machine Specifications

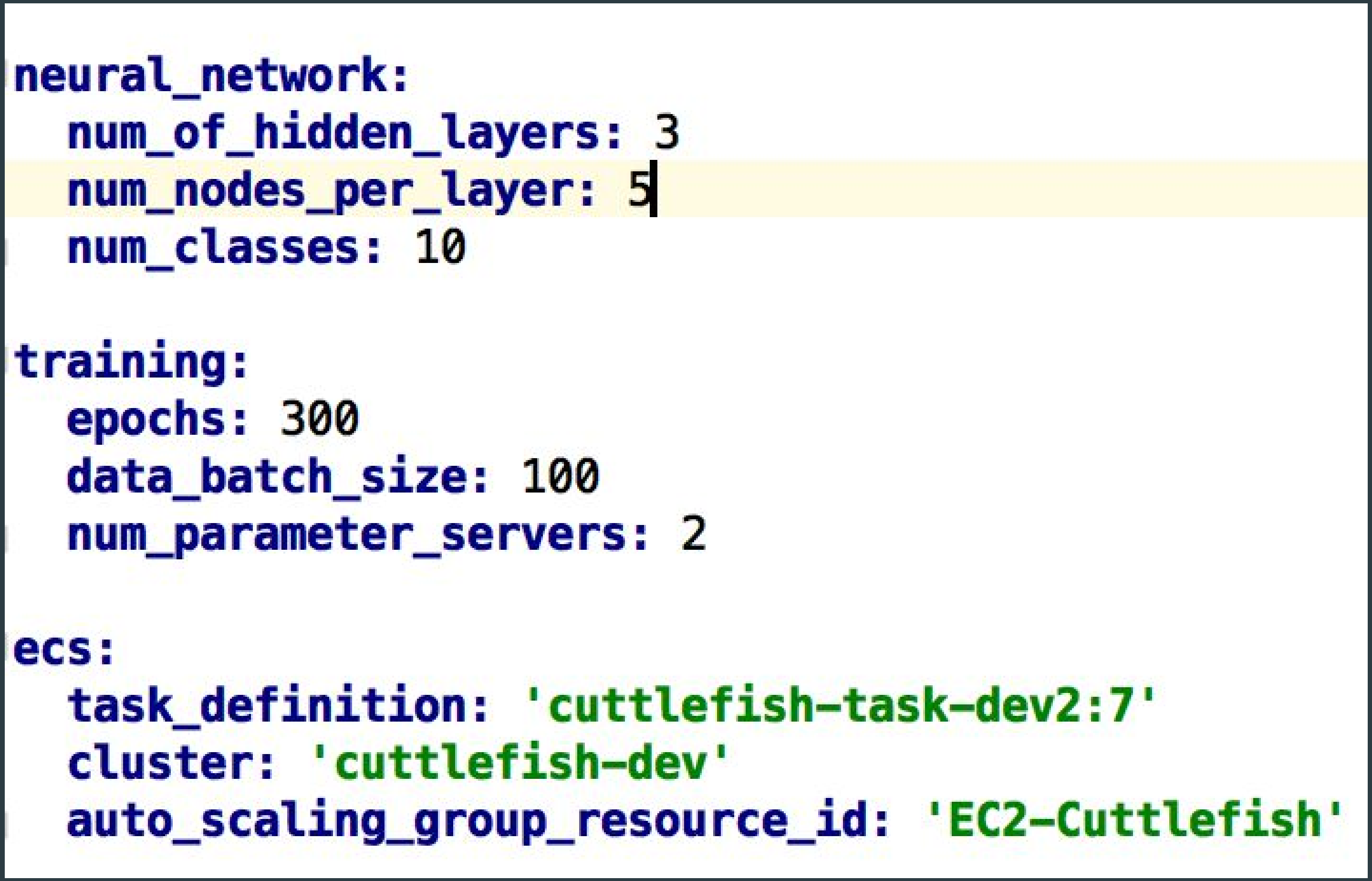
  For proof of concept testing we are using T2 EC2 instances to host the distributed TensorFlow Docker containers

- Below is a summary of EC2 instance specifications for machines used to house the Cuttlefish ECS cluster:
  - Box Type: t2.medium
  - AMI: Amazon Linux AMI 2016.09.a x86 64 ECS HVM GP2
  - Storage: EBS storage only
  - Memory: 4GB
  - CPU Units: 2

APPLICATION ARCHITECTURE

- Each neuron in the network is modeled as a single computational unit.
- Test automating spinning up distributed neural network as infrastructure using configuration files that describe the neural networks.
- The configuration includes hyperparameter values, TensorFlow and ECS related configuration values, hyperparameters, number of epochs etc.
- Cuttlefish is currently comprised of three major components, the Neural Network Builder (NNB), the Cuttlefish Client and the Cuttlefish neural network models.

APPLICATION ARCHITECTURE



```
neural_network:
  num_of_hidden_layers: 3
  num_nodes_per_layer: 5
  num_classes: 10

training:
  epochs: 300
  data_batch_size: 100
  num_parameter_servers: 2

ecs:
  task_definition: 'cuttlefish-task-dev2:7'
  cluster: 'cuttlefish-dev'
  auto_scaling_group_resource_id: 'EC2-Cuttlefish'
```

CHALLENGES AND LIMITATIONS USING DISTRIBUTED TENSORFLOW

- Distributed TensorFlow Manual Process For Cluster and Server Setup

  We encountered some challenges when trying to force TensorFlow’s placement algorithm to distributed the graphs using our approach. This challenge stems from the framework’s opinions on how graphs should be distributed.

- Limited Documentation

  Given that TensorFlow, though gaining in recent popularity is still a fairly new technology which limits documentation (blog posts, tutorials etc) found in the general development community

RESULTS

Single-machine Results			
# of Nodes	# of Layers	# of epochs	Accuracy
5	3	1000	88.61%
500	4	10	94.99%
50	3	2000	95.82%
784, 2500, 2000,1500,1000,500,10	5	30	96.95%

RESULTS

Distributed version Results				
# of Nodes	# of machines	# of Layers	# of epochs	Accuracy
5	2	3	1000	50.89%
500	2	4	10	
50	2	3	2000	85.75%

FUTURE WORK

- Implement Cuttlefish to Build Elastic Distributed Neural Networks Without TensorFlow
  - In order to fully explore our ideas regarding computational graph distribution we propose continued work on Cuttlefisth without leveraging TensorFlow. This will allow us to flexibility to explore all facets of the problem.
  - Experiment using Cuttlefish to build and distributed elastic neural networks across devices such as Raspberry Pis and docker containers.
- Implement A More Robust Orchestration and Automation Solution
  - We propose implementing a more robust orchestration and automation solution to replace our use of Amazon's Elastic Container Service (ECS). Although ECS is fairly easy to use, it has technical limitations that make it far less flexible and mature than competing products such as Kubernetes or Docker Swarm.

Code & Project Site

- Cuttlefish - GitHub
- Cuttlefish - Homepage

Questions?