Graduate Portfolio

By

[Nithin Kumar Guruswamy]

[8/13/2021]

Non-Thesis Portfolio

for the degree of

MASTER OF SCIENCE

in

Computer Science

Department of Electrical and Computer Engineering

Vanderbilt University

[Summer 2021]

[Advisor's Signature]

Academic Advisor:

[Dr. Akos Ledeczi, Professor and Directorate of Graduate Studies of Computer Science, Vanderbilt University]

1. SUMMARY DATA

a. Personal Information
- Name – Nithin kumar Guruswamy
- Program start Date - Spring 2020
- Status - Full time status
- Principal source of support –
    a. Topics in Big data Teaching Assistantship from EECS Department (Spring 2020).
    b. Research Assistantship funded by DARPA MiDAS project (Fall 2020).
    c. Topics in Big data Teaching Assistantship from EECS department (Spring 2021) and Research Assistantship funded by Cisco project (Spring 2021).
    d. Personal funds (Summer 2021).

b. Name of Academic advisor
- Dr. Akos Ledeczi

c. Course information:

Course Number, Title	Semester	Instructor	Grade Received
CS-6381-01, Distributed Systems Principles	Spring 2020	Dr. Aniruddha (Andy) S. Gokhale	A
CS-6377-01 Embedded Software & Sys	Spring 2020	Dr. Janos Sztipanovits	A
CS-5891-04 Special Topics: Reverse Engineering	Spring 2020	Dr. Daniel A. Balasubramanian	A
CS-5250-02, Algorithms	Fall 2020	Dr. Julie L. Johnson	B-
CS-5287-01, Principles of Cloud Computing	Fall 2020	Dr. Aniruddha (Andy) S. Gokhale	A
CS-6388-01 Model-Integrated Comp	Fall 2020	Dr. Janos Sztipanovits	A-
CS-6315-01 Automated Verification	Spring 2021	Dr. Taylor T. Johnson	A
CS-6360-01 Adv Artificial Intelligence	Spring 2021	Dr. Maithilee Kunda	A
CS-5892-01 Special Topics: Cyber Physical Systems	Spring 2021	Dr. Gabor Karsai	A-
EECE-8850-06 Independent Study	Summer 2021	Dr. Bharath L. Bhuva	A

2. Professional Goals:

a. **Statement**: To be a Lead software engineer/Architect/Leader, who specializes in Networked embedded systems with artificial intelligence. I want to lead, research, and individually contribute or develop software across various industry verticals with special interest in Networking/Distributed systems and Artificial Intelligence (AI).

b. **Achievements for the academic years:**
   - **2020(Spring and Fall):** 1) Achieved perfect 4.0 GPA for Spring 2020 semester. 2) Cisco summer internship for Summer 2021.
   - **2021(Spring and Summer):** 1) Worked as both RA (Cisco Project) and TA (big data class) for spring 2021 semester

c. **Interests:** Computer networking Distributed systems, cloud computing and Artificial Intelligence

3. CV:

Nithin Kumar Guruswamy

819 18th Avenue South Apartment #127 Nashville TN 37203
| Nithin.kumar.guruswamy@vanderbilt.edu;nithin.kumar.g@gmail.com |
+1 6156491337
https://www.linkedin.com/in/nithin-kumar-guruswamy-23302223/

EDUCATION:

M.S in Computer science, Vanderbilt university, Nashville, TN, USA GPA:3.81 Aug 2021

B.E in Electronics and Communication Engineering , National Institute of Engineering (NIE), VTU 71.4% (First Class with Distinction) Jul 2003

RESEARCH INTERESTS- Machine Learning (ML), Artificial Intelligence (AI), Computer security and Computer Networks.

MS COURSEWORKS TAKEN: Distributed systems, Reverse Engineering, Embedded software systems. Algorithms, Cloud computing, Model integrated computing, Cyber physical systems, Automated verification, Advanced artificial intelligence,

GITHUB LINK :https://github.com/nithinkumar030

Summer Intern, Cisco May 2021 to Aug 2021

Working on Cisco Hardware failure prediction project on Engine 2 timeseries analysis, Engine3 (Policy orchestration) module and Cisco Field/Failure rate prediction project.

Research assistant (RA) and Teaching Assistant (TA) Jan 2021 to May 2021

Performing Research in computer networks anomaly detection - especially on classical anomaly detection algorithms like CUSUM, EWMA and advanced ML techniques like Variational autoencoders (VAEs) and combination of LSTM and VAEs. Preparing and evaluating assignments for Bigdata class.

Research assistant (RA) Aug 2020 to Dec 2020

Worked on performance evaluation of a distributed File system like interplanetary file system (IPFS) and worked on defining adaptive software using a xtext (a textual Domain specific language).

Research assistant (RA) May 2020 to Aug 2020

I was also involved in leading a team to architect and develop mobile application for iOS and Android for Nashville Mass transit system using react-native framework during summer 2020.

Teaching assistant(TA) Jan 2020 to Apr 2020

Performed TA activities like assignment preparation, mentoring and grading for big data systems

PROFESSIONAL EXPERIENCE

Software Engineer 4, Cisco Systems Ltd Aug 2019 to Dec 2019

Cisco is a global IT & Networking provider housing 73k employees and generating annual revenue of USD 48 Bn

- Responsible for Data telemetry and Machine learning for the Proactive Hardware failure prediction research project using NCS540 edge router
- Responsible for embedded application development and debugging switch fabric driver & middleware for NCS6K core router.

Router Hardware Failure Prediction using HALT data **Aug 2019 to Dec 2019**

Cisco router Hardware accelerated Lifecycle testing (HALT) data Telemetry, collection, cleaning, pre-processing and exploratory data analysis of sensor data of CPU and Packet Processors using NCS540 router

Software Engineer 3, Cisco Systems Ltd. **Feb 2015 to July 2019**

Router Hardware Failure Prediction using MFG data (POC2) **Mar 2019 till July 2019**

- Cisco router Hardware Manufacturing (MFG) data Telemetry, collection, cleaning, pre-processing and exploratory data analysis of sensor data of CPU and Packet Processors

Router Hardware Failure Prediction using Cisco NCS540 router (POC1) **Sep 2018 till Feb 2019**

- Cisco Router Sensor Data Telemetry and Predicting failure of CPU, RAM, Packet Processors and switch fabric ASICs on Cisco routers using ML algorithms.

Power savings in Switched Fabric element (SFE) driver for Cisco Hackathon **Sep 2018- Oct 2018**

- Modelled traffic towards switch using logistic regression ML algorithm and saved 25% power (120W) during off peak loads by putting ASICs in low power state.(Sleep mode).

Fabric Software Development for NCS6k (v7.0.1 & v7.1.1) **Nov 2017 - Aug 2018**

Developed fabric interface ASIC (FIA) driver enhancement to support high scale (>4k) interface ports on NCS6k router

- Optimized FIA driver programming of ASIC resources by developing algorithm to parallelise execution using 1 thread/ASIC & port-locking mechanism; achieved least blocking time & saved interface creation time by 10%
- Debugged and implemented a logic fix for fabric going Multicast down during Multi Secure domain router (SDR) deletion case in fabric database application
- Designed and implemented software using ASIC recovery logic in C for Fabric FMEA for NCS6k router

In-service Software Upgrade (ISSU) Bug Fixes for NCS6k (v6.3.1 & v6.4.1) **Mar 2017 - Oct 2017**

- Fixed ISSU feature of FIA driver for 2Tbps Line Card by correcting logic and implemented ISSU debug enhancements (saving trace for V1 VM in v6.3.1 software) to achieve High Availability
- Ported NCS6k FIA driver to support new ISSU library by implementing new API software interface changes to simplify communication process with ISSU manager
- Hardened ISSU feature on 6.3.1 and 6.4.1 releases with 100% quality as per TL9000

FIA Driver Enhancements and Stability Fixes for NCS6K (v6.3.1 & v6.4.1) **Mar 2016 - Feb 2017**

- Implemented debug command-line interface (CLI) for FIA driver and modification for Collapsed Forwarding (CoFo) feature in C
- Enhanced line rate capacity for multicast traffic in FIA driver for 2Tbps Line Card by debugging via router CLI; improved FIA driver multi-threaded stability by 100% using mutexes
- Hardened FIA driver for 6.3.1 and 6.4.1 releases and delivered with 100% quality as per TL9000

NCS6k Router Switch Fabric Modules for Software v6.2.1 **Aug 2015 - Feb 2016**

- Debugged crash using GDB and improved multithreaded stability of switch fabric element driver by 100% using locks
- Corrected Multicast ID programming logic at S2 CLOS fabric stage in C and obtained 0% traffic drop in fabric multicast in middleware
- Hardened fabric modules for v6.2.1 release with 100% quality as per TL 9000

ISSU Enhancement for Software v6.1.1 Mar 2015 - Jul 2015

- Analyzed fabric switch database aggregator middleware to detect bugs; fixed router configuration handling logic in C

Lead Technologist, GE Transportation-Intelligent Transportation Systems (ITS) Jul 2014 - Jan 2015

-Product manufacturer for railroad, marine, and mining industries, housing 10k employees with revenue of USD 6 Bn

- Developed ElectrologIXS (wayside rail signaling product) with CENELEC safety features using embedded real-time software

Incremental Train Control System (ITCS) Protocol Stack for Wayside Rail Signaling Jul 2014 - Jan 2015

- Implemented Layer2, Layer3 ITCS real-time protocol stack using C on ucOS2 RTOS
- Implemented test software using MS visual studio .NET to test the protocol stack
- Sent Temporary Slow Orders (TSO) over Office Wayside Link (OWL) and Train to Wayside Communication (TWC) using C and over ElectrologIXS Radio Controller based on ColdFire Microcontroller platform

Honeywell Technology Solutions Lab Ltd. Nov 2008 - Jun 2014

-An aerospace technology manufacturer housing 130k employees with revenue of USD 40 Bn

- Developed device driver and embedded application software for Honeywell's Cabin Management Systems (CMS) using C on Embedded Linux 2, 6 and ARM Cortex-M3 platform with SAFERTOS

Tech Lead Jul 2013 - Jun 2014

Audio Video on Demand (AVOD) Client Application on iOS 5 Platform

- Designed and implemented client interface application, A/V player UI for Audio Video on Demand (AVOD) server on iOS devices, used in in-flight entertainment systems

5.1 Channel Downmixing Prototype App on IOS 5.0 Platform Dec 2012 - Jun 2013

Implemented an algorithm to check feasibility of using RTP instead of UDP over wireless channel

- Implemented 5.1 downmixing algorithm on iOS platform using Core Audio and UIKit framework of iOS 5 platform; used RTP over UDP protocol to receive AVOD streams
- Selected RTP over TCP as RTP over UDP showed 33% packet loss after performance testing

Ovation Select Cabin control App on iOS 5 Platform Nov 2011 - Nov 2012

Developed Cabin control application to control audio/video, lights, temperature, screens, etc. in Honeywell Ovation Select in-flight entertainment systems(https://itunes.apple.com/in/app/honeywell-cabin-control/id511323294?mt=8.)

- Designed application's renderer GUI using graphics from database and implemented pull-up menu using UIKit framework of iOS; hosted app on AppStore

Senior Engineer Nov 2008 - Oct 2011

Ovation Select iPod Media Dock

Designed and developed Apple iPod/iPhone USB HID class driver development on X86 Linux box and media dock PowerPC prototype hardware platforms

- Implemented factory acceptance test procedure software for media dock peripherals (USB host module, FRAM, and Apple authentication co-processor)
- Implemented embedded media dock application to store and display album art; certified as Level E under DO 178B avionics standards with 0 bugs

Sasken Ltd. Mar 2006 - Oct 2008

-A product engineering service provider housing 3.5k employees with revenue of USD 70 Mn

Senior Software Engineer (Client: Nortel Networks, France) Apr 2007 - Oct 2008

- Developed Board Support Package (BSP) and device driver for Nortel's S18k GSM base stations

Nortel IP-based BTS (Base Station) -Internet Protocol module (IPM)

- Implemented Nortel's patented PCM synchronization algorithm for IPM SYN driver; developed and tested BSP for PowerPC 8360 based IPM card on VxWorks RTOS in C

Software Engineer

Nortel S18000 RICAM GSM BTS Oct 2006 - Mar 2007

Developed test software to validate new BSP and QuadFALC Pulse Code Modulation (PCM) framer driver

- Debugged & stabilized BSP & PCM driver onsite and delivered software to Nortel with 100% quality

Nortel S18000 GSM BTS Mar 2006 - Mar 2007

- Conducted stability fix for SYN BTS PCM synchronization driver by offloading sync algorithm from Interrupt Service Routine (ISR) to task

Engineer, Tata Elxsi Ltd. May 2005 - Mar 2006

-A technology & design services provider housing 5k employees with revenue of USD 180 Mn

- Developed test strategy for TI OMAP camera products

Evaluation of Image and Video Quality for TI Camera May 2005 - Mar 2006

- Designed and delivered objective and subjective analysis test strategy to evaluate quality of image taken using Nokia 6600 phone

Mistral Solutions Ltd. May 2004 - Apr 2005

-A technology design and system engineering company housing 400 employees with revenue of USD 18Mn

- Developed embedded test software for verifying DY4 Champ AV3 IPC and DSP libraries

Field Application Engineer May 2004 - Apr 2005

Systems Integration for RADAR (Client: DRDO-LRDE)

- Developed test software in C on VxWorks 5.5 RTOS platform to verify inter-processor communication (IPC), DSP libs and inter-node communication for Champ Av4 DSP card

TECHNICAL SKILLS:

Programming Languages: C, C++, Java, Objective-C, Python and R.
Linuxdevice drivers, VxWorks, SAFERTOS, MicroCOS II, Cisco IOSXR, OOAD, UML2.0
Hardware: Broadcom Jericho, Arad packet processors, FE1600 and FE3600 switching ASICs and its SDK, PowerPC 8260/8360/8313e, ColdFire68k, ARM Cortex-M3.
Domain Knowledge: Machine learning, Deep learning,Computer Networks (core Routers), Avionics In-Flight entertainment system ,Railway Signalling, Mobile communications technologies.

CERTIFICATIONS- Six Sigma Green Belt, Honeywell, Nov 2009,Coursera Deep learning certification (2021), Data science MOOCs (Nov 2018), JHU, Machine learning MOOC from Stanford University(Aug 2019) and Data structures using Java and algorithms MOOC (Sep 2019), UC San Diego

AWARDS AND EXTRA-CURRICULAR ACTIVITIES

- Merit promotion to Software Engineer 4, effective from 1st Aug 2019, Cisco Systems recognizing the good work done at Cisco Systems.
- Cisco “Innovate Everywhere” and “You Amaze” award for MFG Hardware Failure prediction Data collection and cleaning (POC2),April 2019 and May 2019.
- Cisco “Innovate Everywhere – A needle inside Hardware” recognition among 100+ for CX Innovation using Machine Learning, Dec 2018
- Cisco “Benefit Everyone- Great willingness and contribution” Recognition among 100+ for The Proactive Hardware Failure Prediction research project(POC1) using Machine Learning, Nov 2018
- Received Cisco Recognition among 100+ teammates for “Connect Everything” for ISSU and Collapsed Forwarding feature support, Nov 2017
- Received Spot Award among 50 at Honeywell for completing Proof of Concepts(POC) for CMS, Jun 2014
- Won the Honeywell Engineer’s Week White Paper contest among 50 participants, Feb 2014

- Received Honeywell Partner award among 250+ teammates for media dock software development, Sep 2011.
- Received Spot Award among 250+ teammates at Honeywell for developing/debugging iPod media docking station software release for NBAA Show, Sarasota FL USA, Nov 2010
- Received Spot award among 100+ teammates in Sasken for securing the GSM BTS V15.1.1 CL2 and V16.1 CL1 software release versions, Apr 2007

VOLUNTEERING ACTIVITIES – Actively volunteered for 12 hours with NGOs like 'Youths for Seva','Children Love castles' and others via Cisco to build STEM models, Braille(Tactile) models for children with visual impairments

LANGUAGES- English, Kannada, Hindi (read, write, speak)

4. Knowledge and mastery of computer science concepts:

Please refer the Cloud computing (CS-5287-01) course project paper and artifacts below:

Source code and presentation artifacts: https://github.com/nithinkumar030/MS-portfolio/tree/main/cloud_computing

Pub-Sub In Network Layer using SDN

Nithin Kumar Guruswamy
EECS Department
Vanderbilt University

Ziqi Li
EECS Department
Vanderbilt University

***Abstract*—Pub-Sub is widely used for messaging purpose in IoT technology, distributed systems, and cloud computing. The traditional Pub-Sub system is implemented in the application layer with an intermediate broker. However, dynamically assuring the QoS requirements like throughput (bandwidth) among different topics under different network scenarios becomes a formidable challenge especially in cases of IoT and big data. Thus, [1] designed network-layer Pub-Sub, using multicast routing and SDN technology which allows programmatic network resource allocation on need. In SDN based pub sub, we completely eliminate the software based broker and make use of network layer multicast routing and programmable SDN technology to act as hardware based broker and also implement QoS in hardware switch thereby boosting the performance and enforcement of Dynamic QoS. Finally, we compare messaging time between traditional Pub-Sub like Kafka and network-layer/SDN based Pub-Sub, to see how faster the latter can be in terms of message forwarding. Furthermore, we also test QoS parameter like throughput is enforced dynamically using SDN based pub sub.**

Keywords—Pub-Sub, multicast, and Software defined networking (SDN)

I. Introduction

In software architecture, Pub-Sub is a messaging pattern where senders of messages, called publishers, do not program the messages to be sent directly to specific receivers, called subscribers, but instead categorize published messages into classes without the knowledge of which subscribers are going to receive them. Similarly, subscribers' express interest in one or more classes and only receive messages that are of interest to them, without knowledge of which publishers published them.

It is widely used for messaging purpose in IoT and big data. In the traditional topic-based Pub-Sub, a broker is used to receive messages from publishers and forward them to subscribers. However, this approach can be slow due to software-based packet processing/forwarding in the application layer. As a result, we want to realize Pub-Sub in the network layer, to make it faster and more reliable in terms of QoS management.

The basic idea comes from the paper [1]. Software broker in traditional pub sub can be replaced by the SDN L3 router routing/forwarding pipelines using topics to be encoded in IP addresses/ports. SDN allows dynamic configuration of router/ switch resources based on QoS parameters to be enforced on packet types using match action tables and allocating/modifying Queues with different QoS values.

In this design, each topic ID has 1-to-1 relationship with a multicast IP address. Then, the messaging for a one topic can be handled by one multicast group. With the help of SDN, we can realize some other functionalities, such as registration of one group. Also, in SDN, messaging and group management of multicast will be more efficient. So, we expect the network-layer Pub-Sub to be far faster than before.

The rest of this paper is organized as follow: In section II, we will discuss the design and architecture of the network-layer Pub-Sub; In section III, we will discuss implementation details; In section IV, we compare the speed of traditional Pub-Sub with network-layer Pub-Sub. Finally, we conclude out work and point out the future work.

II. Architecture

A. Traditional Pub-Sub

In many traditional Pub-Sub system, publishers post messages to an intermediary message broker, and subscribers register subscriptions with that broker, letting the broker perform the filtering. The broker normally performs a store and forward function to route messages from publishers to subscribers. In addition, the broker may prioritize messages in a queue before routing.

For the topic-based Pub-Sub, the broker uses topic to do the filtering. It can have a table, where key is topic and value is who subscribe this topic. When it receives a message with a certain topic from publisher, it queries the table, to find out to whom it should forward the message to. Finally, it sends the message to all the interested subscribers.

However, this software-based broker architecture can be slow due to packet processing in software/application layer. Packet processing involves pub/sub header parsing, matching, removal, subscriber lookup and encapsulation etc. These operations are sometimes in O(n^2) and don't scale well in software-based packet processing.

These operations are like network layer(L3) packet processing done by routers. Hence [1] and [2] both use network layer multicast routing/forwarding to replace the software-

based broker by encoding topic in multicast address. This entails that pub sub packet processing is now done in router by specialized ASIC based packet processors in hardware containing TCAMs which do packet processing very fast in order of ns thus improving QoS parameters like reducing latency and increasing throughput.

Furthermore, we need to maintain differentiated QoS for different topics and assure it for IoT applications for varying network conditions continuously. Example include throughput should be assured for High priority topics like stock market messages over low priority topic like weather messages. Low priority messages should not usurp the bandwidth of high priority messages thus causing them to drop. In traditional pub sub, topic based QoS are not enforced/assured.

Also, many problems can happen in this process. For example, the broker may fail. One solution is to have multiple replicas for the broker, so that once one broker fails, others will take on its work. However, this fail-over process needs some time. It may violate QoS requirement in cases of IoT and big data with stringent real-time needs.

B. Design of Network-layer Pub-Sub

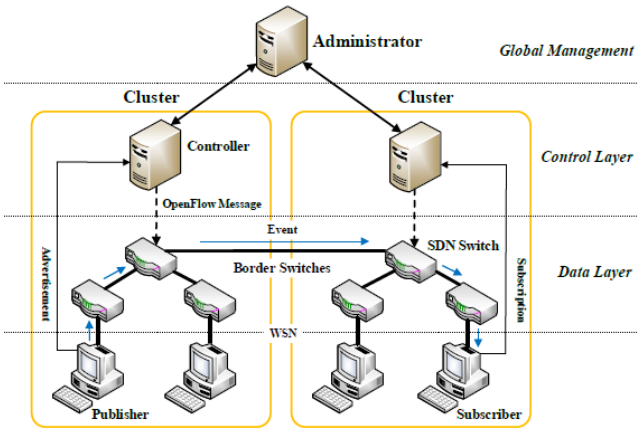
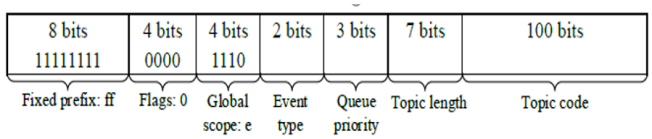


Fig. 1. SDN-based Pub-Sub

The idea comes from the paper [1]. To make Pub-Sub faster, we can encode each topic into a Multicast IP address. Then, we can create a group for multiple subscribers and publishers who are interested in one topic. Furthermore, we can use L3 multicast routing scheme to find the best route. We can use IGMP to manage the group, such as join and disjoin of subscribers. The SDN controller should be responsible for the work of topic registration, including management of multicast IP addresses.

Fig 1 shows the details of the IPv6 address. The first 8 bits indicate the range of multicast IP addresses. Here, we will not introduce the use of flags, global scope or event type. The queue priority is used for QoS purpose. A topic with higher priority should get more resources. The topic length indicates how many bits of topic code are valid. Topic code is similar to topic ID. 100 bits should be sufficient.



Event type	Code
subscription	00
advertisement	01
publication	10

Queue priority	Code
low	101
medium	110
high	111

e.g.
The code of topic "Free" is '000000100001'
The IPv6 code is :
ff0e:a8c0:2100:: for publication events with low priority
ff0e:38c0:2100:: for subscription events with high priority

Fig. 2. Details of IPv6 address for SDN-based Pub-Sub

The Pub Sub can also be designed by eliminating the software broker with L3 unicast routing/fowarding. Here a publisher can be a TCP server using zmq PUB socket with topic encoded as hash of IP address and port number.The subscriber can connect to the pub using SUB socket of zmq.The QoS can be enforced using s SDN router by matching and seperating the topics using port numbers at switch and assigning it to apropriate QoS queues in accordance with the topic priority. Also the QoS parameters like throughput are measured periodically and queue parameters are modified on the fly to enforce topic QoS dynamically according to the changing network behavior.

III. Implementation

We used Mininet to simulate the network topology. Once the switch in Mininet receives a packet with multicast IP address, it will send a request to the SDN controller to create a multicast group. Then, the controller will find a multicast routing scheme and install it in the related switches. IGMP is used to manage the group. Also, we used pyzmq radio-dish pattern to simulate the multicast messaging. The radio code connects to a multicast IP address and the dish code binds a multicast IP address. One IP address can be connected and bound by multiple radio and dish users.

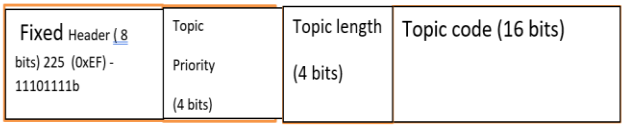


Fig. 3. Details of IPv4 address for SDN-based Pub-Sub

Since IPv6 is still not supported in the draft pyzmq radio-dish pattern, we only use IPv4 addresses for our experiments. The composition of IPv4 addresses is shown as Fig 2. The first 8 bits is fixed, indicating it is a multicast IP address. We have 4 bits for topic priority, 4 bits for topic length and the last 16 bits for topic code. Here, we ignore flags, global scope and event type to make it simple. We used a hashing algorithm to convert the

topic string into a topic code. The details of this algorithm can be found in our Python code.

IV. Evaluation

A. Speed of Network-layer Pub-Sub

The first experiment is to compare the packet transfer speed in network-layer Pub-Sub with the one in traditional Pub-Sub, to see how fast the former can be. We use Kafka server as the broker in the traditional Pub-Sub and use IPv4 unicast IP address for it. The architecture of both is shown as Fig 3 and Fig 4.

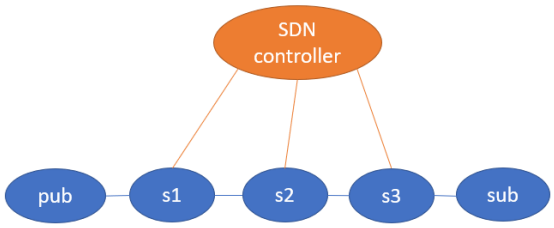


Fig. 4. Architecture of network-layer Pub-Sub

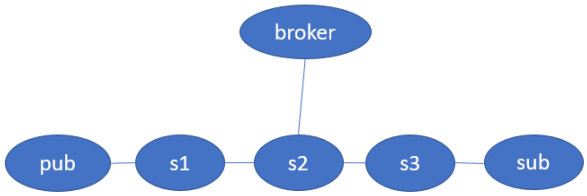


Fig. 5. Architecutre of tranditional Pub-Sub

The data flow for both architectures is as follow: for network-layer Pub-Sub, pub fist sends a packet with multicast address to s1, then the packet is sent to s2, then s3, finally sub. So, the data flow is pub -> s1 -> s2 -> s3 -> sub; for traditional Pub-Sub, once pub sends a packet with unicast IP address, it will be transferred to the Kafka broker first, then the broker will forward it to sub. So, the data flow is pub -> s1 -> s2 -> broker, broker -> s2 -> s3 -> sub. The messaging speed for network-layer Pub-Sub is expected to be faster than the broker-based Pub-Sub, since in the latter case, the routing path is longer than the former one and the broker needs time to receive and forward packets.

We define response time as the difference between the time when sub receives the packet and the time when pub sends the packet. We then compare the response time in both cases over 100 samples. Our results are as follow: the average response time for broker-based Pub-Sub is 0.0034s, while the one for network-layer Pub-Sub is 0.0036s. They are almost the same, which is out of our expectation. We think the reason is that the simulation tools are not perfect. The pyzmq radio-dish pattern written in Python is still a draft according to the official documentation, it may be slower compared to Kafka, which is written in Java. To exclude the simulation problem, we need to setup the experiment using hardware SDN switch rather than Mininet simulator which is also written in python. Another reason could be that switches take more time to look up the multicast flow table than the unicast flow table.

B. QoS requirements in Network-layer Pub-Sub

The second experiment checks the QoS enforcement between 2 topics. One is the high priority topic(stock) and other is low priority topic (weather). The high priority topic should be given more throughput to make its latency lesser. Also, the bandwidth is measured for all the topics and Queues in the SDN switch are modified dynamically using the POX SDN controller to enforce the QoS.

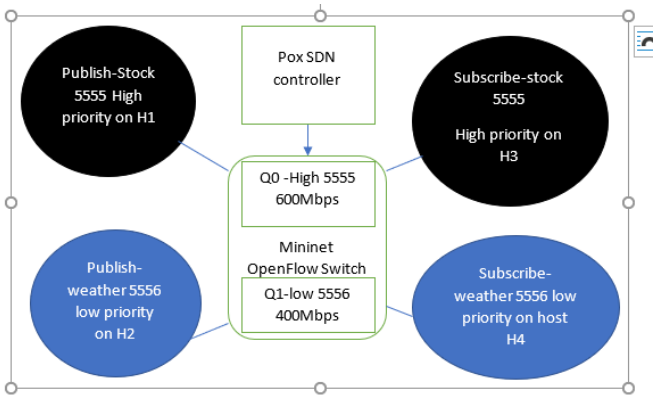


Fig. 6. Showing unicast Pox SDN based switch implementing assured QoS for SDN based pub sub

The test topology is as shown above which has a Mininet based SDN open flow switch simulator connected to a POX SDN controller and 4 hosts H1 runs the publish-stock, H3 runs subscribe-stock. H2 runs publish-weather and H4 runs subscribe-weather, respectively. Topic stock is having higher priority over weather topic. QoS is applied in the Mininet SDN Open flow switch by creating 2 queues Q0 having a bandwidth of 600 Mbps and Q1 having bandwidth of 400Mbps. Total bandwidth is restricted to 1Gbps for both queues. The OpenFlow switch match action table is modified by using POX SDN controller to match packets with port 5555(stock topic) to Q0 (600 Mbps queue) and with port 5556 (weather topic) to Q1(400Mbps queue). Results show that stock topic is given a higher bandwidth than weather topic as shown in fig 7.

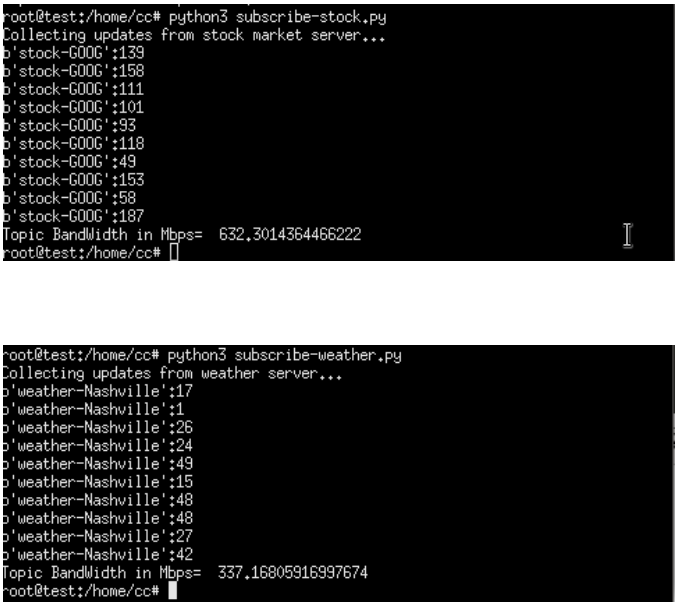


Fig. 7. Showing topic stock (high priority) is always allocated more throughput than for topic weather (low priority)

V. Teamwork

Ziqi's work: Read pyzmq documentations and found radio-dish pattern can be used for multicast; wrote sample pub and sub code with radio-dish pattern. Designed and setup the experiment to compare the speed of network-layer Pub-Sub and traditional Pub-Sub.

Nithin's work: Designed the topic to IPv4 address encoding. Created the multicast Publish middleware in pyzmq and python. I implemented and tested the unicast based SDN based pub sub middleware with 2 topics (1 high priority and 1 low priority) for QoS enforcement in Mininet Openflow switch and POX SDN controller.

VI. Conclusion

To make Pub-Sub faster and to assure QoS between topics, we have implemented it in the network-layer using multicast with SDN technology. We have shown in experiments that even in POX based controller with SDN switch simulator like Mininet results show that QoS is enforced for high priority topics showing high throughput values for high priority topics and it is sure to work on a real hardware SDN switch. This is a good result as Assured QoS is not supported by traditional pub sub like Kafka.

However, the 2 experiments of measuring speed between SDN based pub sub on SDN switch simulator and traditional pub sub, the simulation does not show it is very faster. This is because the SDN simulator is Mininet and it is written in python and runs slower than Kafka which is native java application therefore, we believe that we can see very favorable results when real SDN switch is used rather than Mininet SDN switch simulator as our future work; second, giving a proper definition to Service-Level priority in multicast and implement it.

(1) References

[1] Shi, Yulong & Zhang, Yang & Jacobsen, Hans-Arno & Tang, Lulu & Elliott, Geoffrey & Zhang, Guanqun & Chen, Xiwei & Chen, Junliang. (2019). Using Machine Learning to Provide Reliable Differentiated Services for IoT in SDN-Like Publish/Subscribe Middleware. Sensors (Switzerland). 19. 10.3390/s19061449.

[2] Wang, Y.; Zhang, Y.; Chen, J. Pursuing Differentiated Services in a SDN-Based IoT-Oriented Pub/Sub System. In Proceedings of the 2017 IEEE International Conference on Web Services (ICWS), Honolulu, HI, USA,25–30 June 2017; pp. 906–909.

5. Communication Skills in computer science:

Please Refer to embedded software systems (CS-6377-01) project paper below:

https://github.com/nithinkumar030/MS-portfolio/tree/main/embedded_software_systems

Compositionality in Deep Learning Networks

***Abstract*—Compositionality is observed everywhere in nature. The PART-A of the paper explores the concept of compositionality in the context of Deep learning taking motivations from biology (Brain) and solves problems in deep learning computation efficiently. The PART-B explores problems in computer vision tasks such as object detection. Using the core idea of compositionality inspired from the biological visual cortex of the brain, solves the object detection task in Deep neural networks.**

Index: Modular Neural network. Deep neural network (DNN), Expectation maximization algorithm, Convolution neural network.

I. Introduction and motivation:

THere are several motivations for studying compositionality/modularity in case of Deep Neural network. It can be Computational, Biological, Psychological, and Implementational. In this paper I will mostly be exploring the biological motivations. Large scale neural networks or deep Neural networks give good prediction performance for big training data. As shown in the Figure 1 the more amount of training data you feed to the Deep neural networks, it provides proportional linear performance increase as amount of input training data increases. Whereas, other neural networks like medium and small neural networks does provide linear performance but, flattens out for much lesser sized training data. The other non-Neural network training algorithms like SVM, Logistic or linear regression gives even less performance as they flatten off much earlier ,thus deep neural networks are currently the best performing machine learning algorithms with the ability to train on large training data. However, this large capacity comes with a price of huge computing complexity/cost. The computation of the activation functions from its weights from all its deep layers and weight calculations using backpropagation etc., entails a huge compute cost. Also, the training time to train such complex algorithm is high.

We know that a Deep Neural network is a universal function approximator. For a given machine learning problem, we currently do not know how many layers of Deep neural network are enough to approximate it well, so that DNN performs well with less generalization error and with low compute cost and low training time.

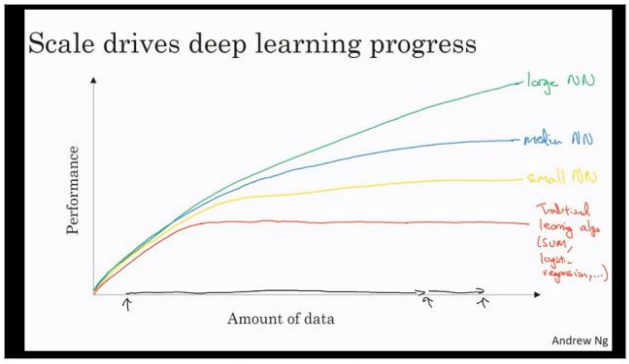


Fig. 1. Performance of some of the Machine learning algorithms vs. data size

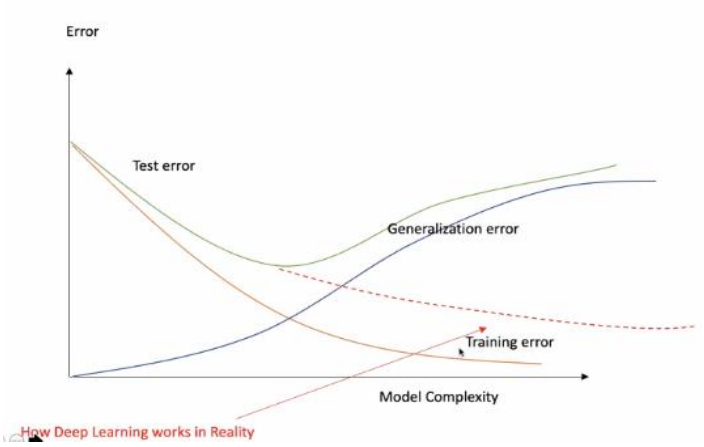


Fig. 2. Shows the Test and training errors decrease as more hidden layers are added into Deep neural network but it also the generalization errors.

As shown in the figure 2: the performance increase (test and training errors decrease) as the Neural network gets deeper with more complexity by adding more hidden layers. However, it also increases the generalization error.

In the PART-A of the paper, Modular neural network (MNN) tries to solve this problem by being inspired by the biological brain using compositionality. Deep neural network can be thought of as composed of multiple smaller neural networks modules. Modular neural networks decompose the complex monolithic neural network problem into modules like different regions of brain where, each region specializes in a diverse task. By decomposing the deep neural network into modules, it achieves many advantages.

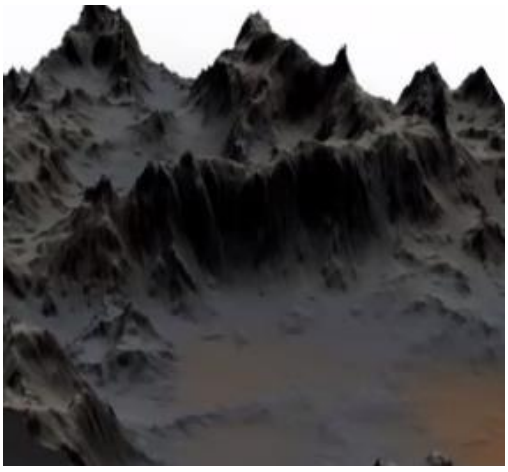


Fig. 3. 3-D visualization of a non-convex cost function of a Deep neural networks which requires complex optimizations which MNN avoids

First, the Modular neural network avoids the non-convex complex optimization problems as shown in Figure 2 encountered by monolithic neural network without decreasing the learning capacity and by using an ensemble of smaller Neural network modules. The smaller Neural network modules can be optimized easily as mostly it will be a convex function rather than a non-convex one for the monolithic deep neural network.

Secondly, MNNs enable to use a priory trained module to be used with other modules. This enables transfer learning and decreases the time taken to train a deep neural network. Also, the modular neural network is immune to temporal and spatial crosstalk – a problem faced by Deep neural network during training. Finally, theoretical and other empirical analysis shows MNN yields better generalization than its constituent modules.

Overall, modularization in MNNs leads to efficient and less complicated computation as MNNs do not suffer from high coupling burden as in monolithic deep neural network. This makes modular neural network scalable to implement a large deep neural network efficiently. Also, as an inspiration taken from biological human brain, only critical parts of the brain are active and unused regions are switched off to conserve energy due to modularity.

Taking inspiration from all the above observation, a new way of automatically decomposing the Monolithic modular neural network into reusable modules is being proposed. The choice of module selected is modeled as a latent variable in a probabilistic model. It learns the decomposition and module parameters end to end by maximizing a variational lower bound of the likelihood. A small fixed number out of a larger set of modules is selected to process a given input, and only the gradients of these selected modules need to be calculated during the backpropagation. This novel method is known as Expectation maximization (EM) algorithm and gives good module selection diversity compared to other modular Neural network algorithms. The latter part also explores the experimental verification of this method over other existing modular neural computation methods.

Fig. 4. Top half image is being seen as image in bottom half after being generalized by VGG-16 CNN after sufficient training.

The PART-B of the paper discusses the application of compositionality on Convolution Neural network (CNN) -which is a type of Deep neural network having 2-dimensional convolution as its activation function. We also observe the property of compositionality in nature especially in biological systems like in visual cortex of brain. The visual cortex of the brain processes the image information and constitutes biological vision. The area of the visual cortex that processes the object-scene interaction part exhibits non-linear compositionality. CNN is also able to model biological vision and have proved their competence in many visual processing tasks like image classification, object class detection, instance segmentation, image captioning and scene understanding. Since Convolution neural network is able to model biological vision shown in Figure 4., it also exhibits nonlinear compositionality and it is able to model human-object interaction well.

Fig. 5. CNN is not able to detect/select only object from its background scene like car or motorcycle.

However, CNN is not able to detect object from other non-contextual objects or background scene as shown in Figure 5. To improve the detection of a familiar object from a background context we need to inculcate linear compositionality to the CNN. It is a known fact that even the brain has a separate region for processing the object and for background scene. Also, the brain can exhibit linear compositionality. Linearity in CNNs can be inculcated by novel training method by applying separate masks to the objects and background and using separate CNNs for training and that will be discussed in much more detail later in the subsequent sections. Finally, an extensive experimental evaluation is performed to verify that this new novel training method does give an improved object detection from the scene by objective and subjective testing.

II. PART -A

1) Modular Neural network using Expectation Maximization (EM) algorithm:

2) Deep Neural networks:

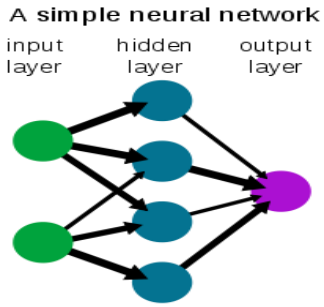


Fig. 6. Shows a Neural network with a single hidden layer.

The Neural network shown in figure 6 has only 1 hidden layer. It has input layers, output layer and hidden layers with many interconnections having weights. The input is multiplied by the weights and undergoes transformation as per the activation function. The output of one layer is the input to the subsequent layer. This flow from left to right is known as forward propagation.

During training phase, the output of the Neural network is calculated using forward propagation. Error or cost function is calculated by taking difference of actual output (in training set) to

the output of neural network. This error is minimized by iteratively adjusting the weights to reduce the cost or error known as backpropagation. A Simple neural network is able to approximate simple functions and its cost function will be mostly convex and many simple optimization techniques like Stochastic gradient descent will be able to able to able to optimize the cost function to its global minimum.

A Deep neural network contains many hidden layers as shown in Fig. 7. A deep neural network can approximate many complex functions. However, as stated previously it usually has complex non-convex optimization problem and error or cost function is not reduced fully thus, Learning may be not be efficient and generalization drops.

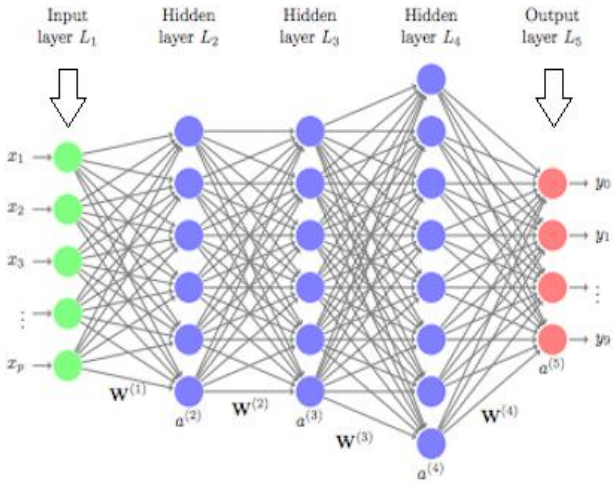


Fig. 7. Shows a deep neural network with many hidden layers which can be considered as composed of many simple neural network shown in Fig.6

Hence compositionality in deep learning networks by the way of Modular networks, which splits the deep networks into smaller ones, which has easier convex optimization functions can help here.

EM algorithm:

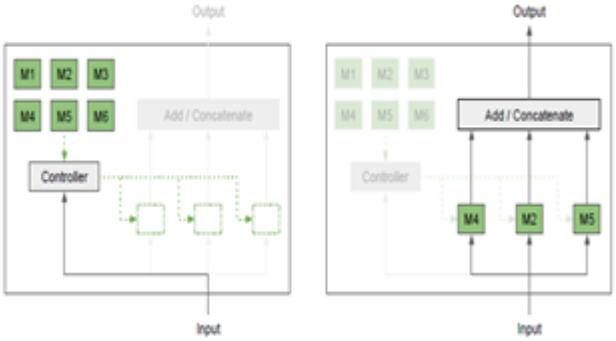


Fig. 7 . (a)Shows a Modular neural network of modules M=6;(b) shows K=3 modules are selected out of M=6 .

Consider a modular neural network layer of having M=6 modules at each hidden layer L and K=3 random modules are selected by a controller to process a given input. The backpropagation is applied to only these 3 modules only.

Let Modular Deep Neural Network be composed of *i* modules where $f_i(x, \theta_i)$ is the function modeling an *i*th module, which takes x as input vector and θ_i are the parameters for the *i*th module.

Let l be a modular layer,$l \in \{1, \ldots, L\}$, be composed of M modules. The controller selects k=3 modules out from M modules.

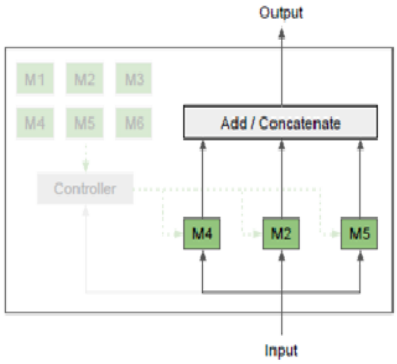


Fig. 8. Shows a Modular NN with k=3 modules (M4, M2 and M5) selected.

Let $a^{(l)}$ be the random variable denoting indices of module selected.

$$a^{(l)} \in \{1,2,..,M\}^k$$

If M=6 ; K=3 then

$$a^{(l)} \in \{1,..,215\}$$

Let the controller distribution at layer *l* be $p(a(l)|x(l), \emptyset(l))$

is parameterized by $\emptyset(l)$ where $\emptyset(l)$ is the parameter of distribution which is also dependent on input $x(l)$

The DNN output at level *l* from the module is given by summation method:

$$y^{(l)} = \sum_{i=1}^{k} \quad f_{a_i^{(l)}}\left(x^{(l)}; \theta_{a_i^{(l)}}\right) \rightarrow (1)$$

Where $y(l)$ can form input for +1 layer

The controller module selection at all layers has a joint distribution

$$P(a|x, \phi) = \prod_{l=1}^{L} P\left(a^{(1)}\middle|x^{(l)}, \emptyset^{(l)}\right) \qquad \rightarrow (2)$$

The entire DNN composed of 'a' module can be used to parameterize the distribution over the final Neural network output layer

$$y \sim P(y|x, a, \theta).$$

The joint distribution of MDNN over output y and module selection 'a' is

$$P(y|x, a, \theta) = P(y|x, a, \theta)P(a|x, \emptyset) \quad \rightarrow (3)$$

Selection of module is random so 'a' is stochastic.

Since Module selection by controller is stochastic, so 'a' is considered as a latent variable, output is defined as

$$P(y|x, a, \theta) = \sum . P(y|x, a, \theta)P(a|x, \emptyset) \rightarrow (4)$$

Selecting K modules at each of L layer makes the 'a' have M^{kL} states and summation of equation (4) is difficult.

For a given collection of input and output data $(x_n, y_n), n = I_1 \ldots, N$,from a probabilistic model, the training objective is maximum likelihood, that is we try to adjust module parameter θ and control parameter ϕ to maximize log likelihood.

$\mathcal{L}(\theta, \emptyset) = \sum_{n=1}^{N} log\ P(y_n|x_n, \theta, \phi)$ → (5)

Now to evaluate the summation over states of 'a' in Equation (4)

$$P(y|x, \theta, \emptyset) = \sum_{a} P(y|x, a, \theta)\, P(a|x, \phi)$$

we use generalized expectation maximization for single datapoint to ward off difficulty in calculating summation over latent variable 'a', when we use it in Equation (5) for a single data point we get

$log\ P(y|x, \theta, \emptyset)$ >= - $\sum_a q(a)\ log\ q(a)$ + $\sum_a q(a)\ log\big(p(y|x, a, \theta)p(a|x, \phi)\big)$ ≡ $L(q, \theta, \phi)$ → (6)

Where $q(a)$ is a variational distribution used to tighten the lower bound of likelihood.

We can compactly write Equation (6) as

$L(q, \theta, \emptyset) = E_{q(a)}[\log p(y, a|x, \theta, \emptyset]$ + H[q]

Where H[q] is the entropy of the distribution q . We try to adjust $q, \theta, \emptyset$ to maximize *L.*

The Partial M-step on $(\theta, \emptyset)$ is defined by taking multiple gradient descent steps where the gradient is

$\nabla_{\theta,\emptyset} L(q, \theta, \phi) = \nabla_{\theta,\emptyset} E_{q(a)}[log(p(y, a|x, \theta, \phi))]$ → (8)

In Practice we randomly select a minibatch of datapoints per iteration. Evaluating this gradient requires a summation over range of 'a'.

To avoid this, Viterbi EM approach effective in which q(a) is constraint to the form

$$q(a) = \delta(a, a^*) \quad \rightarrow (9)$$

Where $\delta(a_1 a^*)$ is the Kronecker delta function. This is 1 if x=y or 0 otherwise.

A full E-step (estimating latent variable 'a' for max likelihood) for a single datapoint would now update a^* to

$a^*_{new} = argmax_a\ P(y|x, a, \theta)p(a|x, \phi)$ → (10)

Where argmax is the maxima. For all datapoints, we make E-step partial in 2 ways for tractability.

1. We choose the best from only S samples $\widetilde{a_s} \sim P(a|x, \phi)$ for $s \in \{1, \dots, S\}$
   Keep the previous a^* if *L*(likelihood) decreases

$a^*_{new} = argmax\ _{(a)} P(y|x,a,\theta)p(a|x,\phi)$ → (11)

Where $a \in \{\tilde{a}_{s|}s \in \{1,\dots,S\}\} \cup \{a^*\}$

2. We apply this to a mini batch while keeping a^* associated with all other data points constant.

Algorithm 1 Training modular networks with generalized EM

Given dataset $D = \{(x_n, y_n)\,|n = 1,\dots,N\}$
Initialize a^*_n for all $n = 1,\dots,N$ by sampling uniformly from all possible module compositions
repeat
  Sample mini-batch of datapoint indices $I \subseteq \{1,\dots,N\}$ ▷ Partial E-step
  for each $n \in I$ **do**
    Sample module compositions $\tilde{A} = \{\tilde{a}_s \sim p(a_n|x_n,\phi)|s = 1,\dots,S\}$
    Update a^*_n to best value out of $\tilde{A} \cup \{a^*_n\}$ according to Equation 11
  end for
  repeat k times ▷ Partial M-step
    Sample mini-batch from dataset $B \subseteq D$
    Update θ and ϕ with gradient step according to Equation 8 on mini-batch B
until convergence

Fig. 9. Shows a summarized EM algorithm.

Alternative Modular Neural Network Algorithms:

1. Reinforce Method: This method maximizes the lower bound on log likelihood.

$B(\theta,\emptyset) \equiv \sum_a P(a|x,\phi)\, logP(y|x,a,\theta) \leq\ L(\theta,\phi)$

The gradients are obtained as below:

$$\nabla_\phi B(\theta,\emptyset) = E_{p(a|x,\phi)}(log\, P(y|x,a,\theta)\nabla_\phi logp(a|x,\phi))$$

$$\nabla_\theta B(\theta,\emptyset) = E_{p(a|x,\phi)}(\nabla_\phi logp(y|x,a,\theta))$$

The expectations are approximated by sampling from $p(a|x,\phi)$.

2. Noisy top-K mixture of experts:

This method is based on mixture of experts. Mixture of experts is a weighted sum of parameterized function each having certain assigned weights. A gating network is used to predict weight of each expert as shown in Fig. 9.

This can help model compositionality by splitting functionality into multiple separate components with different weights.

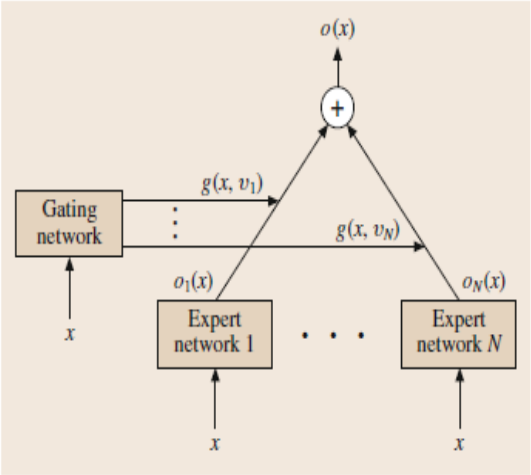


Fig. 9. Shows a mixture of experts

To disable any expert, set all non k weights to -infinity. Noise is added to the output of gating network. Only K units are evaluated and gradients backpropagated. However, there are many issues with these 2 alternate Modular neural network models like lack of diversity and high-test perplexity (lower is better).

3. Module collapse:

This is a phenomenon wherein, same modules are selected for different batches of Input B. This occurs because due to self-reinforcing effect, wherein favored modules are trained rapidly. Most of the Alternate modular methods like Reinforce and Noisy top-k mixture of experts suffered from this effect and used regularization. In regularization, the modular algorithm is forced to select different module within an input mini-batch. But regularization has its own set of performance problems. The EM algorithm does not suffer from Module collapse and no regularization is required.

Testing methodology – test metrics

Two test metrics are defined

I. Module selection entropy:

$$H_a = \frac{1}{BL}\sum_{l=1}^{L}\sum_{n=1}^{B}\mathbb{H}\left[p(a_n^{(l)}|x_n,\phi)\right]$$

Where B is the batch size. The module selection entropy should be small to achieve convergence. That is a given particular input batch being assigned to a module which gives high likelihood.

II. Batch module selection Entropy:

This is Module selection is done over entire batch.

$$H_b = \frac{1}{L}\sum_{l=1}^{L}\mathbb{H}\left[\frac{1}{B}\sum_{n=1}^{B}p(a_n^{(l)}|x_n,\phi)\right]$$

This metric should be high to achieve high diversity so that All different input batches are not assigned a single module.

Module collapse will have low Hb.

Application on EM algorithms:

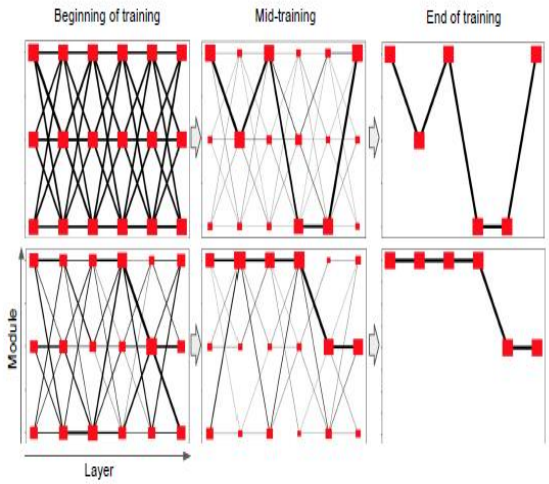


Fig. 10. Shows a summarized EM algorithm.

Consider a DNN with layers L=6, each layer has Modules M=3. K out of M modules are selected where K=1.

The Input task is image classification. Classification task begin with arbitrary assignment of modules for each input. Both upper and lower sections of the Figure 9, depict different subsets of input data points. Algorithm clusters similar inputs by assigning them same modules.

In each partial E-step we use controller $P(a^{(l)}|x^{(1)}, \emptyset^{(l)})$ as a guide to reassign modules to input. In each partial M-step module params θ and controller parameters ϕ are adjusted.

The EM algorithm assigns similar inputs to a common group of modules as shown in top and bottom parts of Fig 9

Language Modelling using EM Algorithm:

Modularization can be used in recurrent neural networks (RNNs) also. It is a known fact that the RNNs are good in modeling sequence to sequence tasks, that is tasks having temporal variation characteristics like speech. The modularization can be used to update state in RNN. This enables to predict modules based on context.

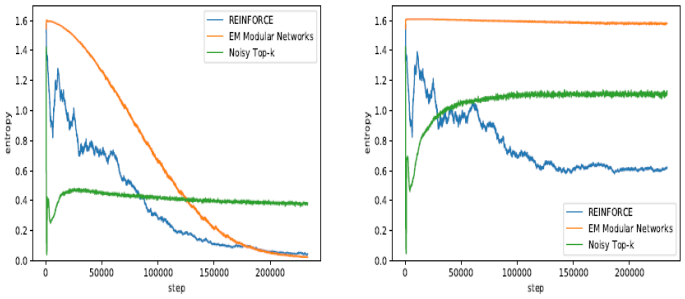
The modularization experiment is carried out on Gated recurrent unit (GRU) – which is a type of long short-Term Memory (LSTM) with no output gate. GRUs have input and forget gates only. The state

update operation is modularized with only K modules are selected and gradients applied at each step. Penn treebank dataset is used which has around a million words with 10k vocabulary. Four module configurations are tested, where K/M = 1/5,3/5,1/15 and 3/15.

EM-approach is compared with unregularized reinforce method, noisy-top k and baseline Gated Recurrent unit. We can see that in Fig EM algorithm approach has lowest test perplexity (accuracy in predicting probability distribution) among other modularizing algorithms.

Table 1: Test perplexity after 50,000 steps on Penn Treebank

Type	#modules (M)	#parallel modules (K)	test perplexity
EM Modular Networks	15	1	**229.651**
EM Modular Networks	5	1	236.809
EM Modular Networks	15	3	246.493
EM Modular Networks	5	3	236.314
REINFORCE	15	1	240.760
REINFORCE	5	1	240.450
REINFORCE	15	3	274.060
REINFORCE	5	3	267.585
Noisy Top-k ($k = 4$)	15	1	422.636
Noisy Top-k ($k = 4$)	5	1	338.275
Baseline	1	1	247.408
Baseline	3	3	241.294



(a) Module selection entropy H_a (b) Batch module selection entropy H_b

Fig. 11. Shows Ha and Hb for Language modelling using Penn tree bank for EM, Reinforce and noisy top-k Mixture of experts

Image classification using EM algorithm in Convolutional neural network (CNN):

It is a known fact that CNN can perform well in computer vision tasks. Here modularization is applied in the context on CNN. Comparison is done for EM algorithm enabled feed forward neural network, EM modularized CNN (Both 2 fully connected layers and 3 convolution layers are modularized) and baseline CNN. For testing CIFAR-10 dataset is used.

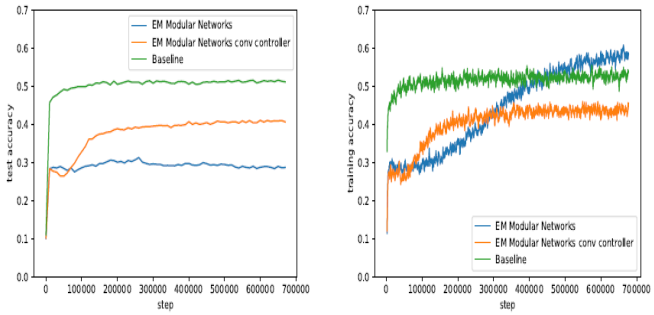


Fig. 11. Shows the testing and training accuracy vs step size

We can see that Modularized approach gives higher training accuracy and generalizes well due to low test perplexity scores.

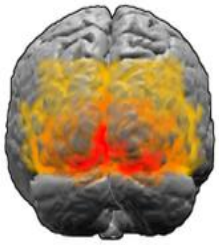
III. PART -B

Compositionality in Deep neural network in computer vision application:

1) Introduction to Convolutional neural network (CNN):

CNN is a type of neural network which uses non-linear activation function such as 2D-convolution. Convolution Neural Network (CNN) is a type of edge detector and gives good performance in many computer visions tasks such as image classification, scene understanding and object detection. CNN models object-scene interaction as non-linear compositionality like brain.

2) CNN relation to visual cortex:



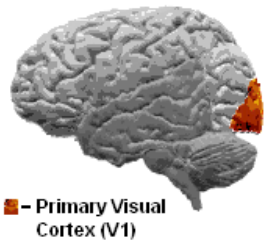


Fig. 12. Shows the Visual cortex region of the brain

The CNNs do take a biological inspiration from the Visual cortex. We know that Visual cortex is part of brain that processes vision. The visual cortex has small regions of cells that are sensitive to specific regions of the visual field. This idea was expanded upon by a fascinating experiment by Hubel et al in 1962 where they showed that some individual neuronal cells in the brain responded or fired only in the presence of edges of a certain orientation. We know that the CNN also is sensitive to edges in the images. Hence there is a similarity between CNN and biological vision processing part of the brain.

3) Convolution Neural network structure:

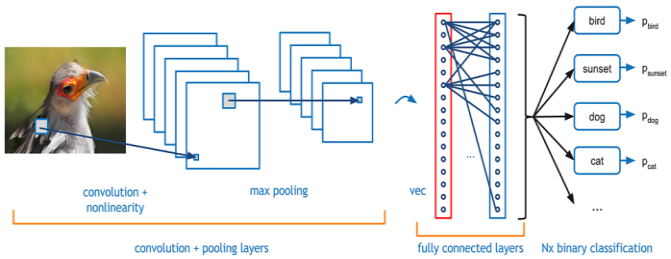
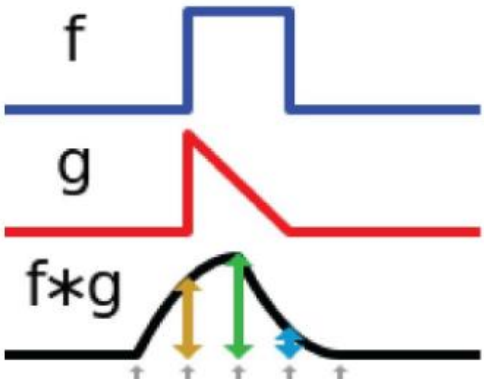


Fig. 13. Shows the Visual cortex region of the brain

As shown in the figure 13. CNN Consists of many alternate layers of Convolution and sub sampling (Pooling). Convolutions are done with a customized edge detecting kernel. Last layer is a fully connected DNN which does N-ary classification.



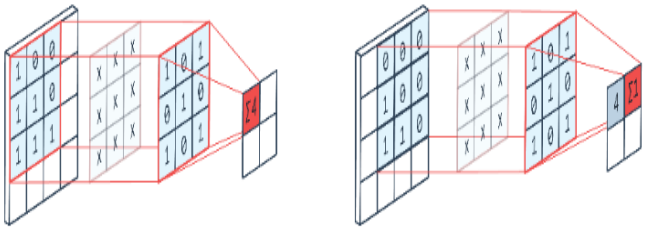


Fig. 14. Shows the 1D, 2D convolution operation

Convolution operation as shown in Fig 14. Looking at the 1D convolution we can see that it spreads the input signal based on the other signal.2D convolution in CNN is done with a specialized edge detecting kernel. After convolution, the dimension of output changes due spreading nature of the convolution operation.

4) Visual cortex shows non-linear compositionality:

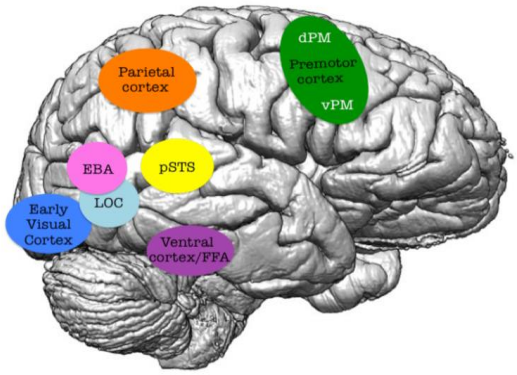


Fig. 14. Shows different parts of the visual cortex

The Downstream regions of the visual cortex process high-level visual information. There are specialized regions for processing which responds well to certain visuals stimulus .The LOC (Lateral Occipital cortex) responds on Objects, The EBA (Extrastriate body area) on Human poses and The pSTS (posterior Superior temporal Sulcus) responds well on human interactions with objects. Certain areas in the brain like pSTS which are highly sensitive specifically to human-object interactions.

The neural representations of interactions are not a linear sum of the human and object representations. This is as per the inference of experiments done by Baldassano et al.

Experiment to check if CNN mimics visual cortex:

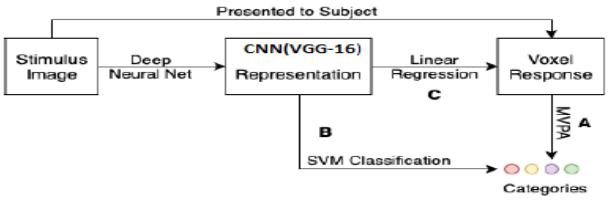


Fig. 15. Shows block diagram of the qualitative experiment

Four classes of images showing humans interacting with objects are taken like typing carrying objects etc. Test humans are asked to see the images and their brain functional Magnetic resonance Imaging(fMRI) is taken. The fMRI's Voxel (Volume equivalent of pixel) response are analyzed using Multivoxel pattern analysis (MVPA) which gives 4 distinct classes. This is the same experimental procedure done by Baldassano et al.

Same Images are fed to CNN and later classified by a SVM classifier as done by Aditi Jha et al. results obtained were Interesting. CNN is also able to classify into 4 categories like humans and much more as explained next.

Results

Direct DNN-based classification

Task	Accuracy
Object classification	0.90
Human pose classification	0.74
Interaction classification	0.86

Table 2: Direct classification performance of VGG-16 representations on the Baldassano et al. (2016) stimuli.

The set of CNNs trained on objects alone were able to predict objects well as in LOC part of visual cortex. Another set of CNNs that were trained on human poses was able to predict human poses as EBA in visual cortex. Also, CNNs trained on human-object interactions able to predict interactions as pSTS part in visual cortex. But CNNs trained to predict human object interactions failed to predict accurately the objects alone or human poses only. This was also the same case in pSTS part in visual cortex. So, this implies that the human object interaction is not a simple linear sum of human and object interaction but has nonlinear relation.

In many real-life applications such as object detection with different background scene, autonomous vehicles and industrial automation, we may have to detect humans or objects from the background. So, we need to inculcate linear compositionality in CNN. This can be done by a novel training method proposed in next section and an exhaustive experimental evaluation follows.

Algorithm for linear compositionality in CNN:

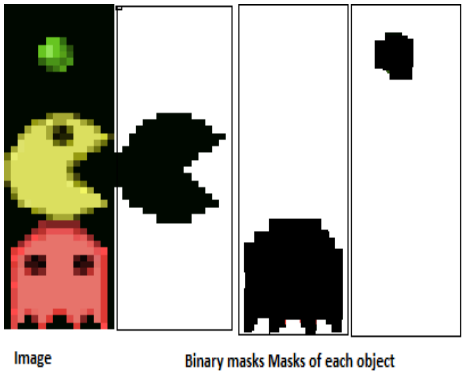


Fig. 16. Shows object on left most side and its binary binary mask m

Let X be an input Image with (x, y, c) dimensions. Let 'm' be a binary mask of an object in X.

m is a tensor with same shape of X with 1s indicating mask of an object. Let 'm' is of same size of X. Let $\emptyset$ be the 2D convolution activation function or a mapping between Image layer in a CNN. Let $\phi(X)$ is the 2D convolved output with a kernel having (h, w, c) dimension.

We define $\emptyset$ to be compositional if and only if

$$\emptyset(m \cdot X) = P(m) \cdot \emptyset(X) \quad \rightarrow (1)$$

Where . Operator is an element-wise multiplication (not matrix multiplication).

Let p be a projection operator.

The projection p(m) down samples object mask m (x,y,c) to the size of output of $\phi(X)$ ie (h,w,c)

For Example: if $\phi(X)$ is the activations of size (h,w,c); where h,w are spatial dimensions and c is the feature (RGB-color)channels p(m) will downsample the object-mask m to size (h, w) and then stack c copies of the down-sampled object mask on top of each other to produce a mask of size (h, w, c).

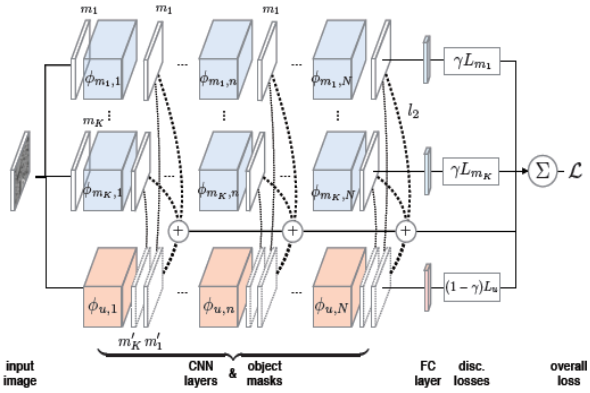


Fig. 17. Linearized CNN architecture for K objects

This architecture follows Equation (1). If input Image X has k objects then as per Equation (1) we will have k masked CNN and 1 unmasked CNNs .The (k+1)th CNN receives unmasked input.

The training method tries to reduce 2 types of losses.

1. *Discriminative Loss (Ld.):* For correct discrimination of objects we add this for all K masked and 1 unmasked CNN.

$$\mathcal{L}_d = \frac{1}{K}\left(\sum_k \gamma L_{m_k}\right) + (1-\gamma)L_u. \qquad \text{-->(12)}$$

Where, L_{mk} is discriminative loss for K masked CNNs, *Lu* is the discriminative loss for 1 unmasked CNN and $\gamma \epsilon \{0,1\}$ is the Hyperparameter of CNN.

2. *Compositional Loss (Lc):* It is a l2 difference on activation of masked CNN and Activation of the unmasked function

$$\mathcal{L}_c = \frac{1}{K}\sum_k \sum_n \lambda_n ||\phi_{m_k,n} - \phi_{u,n} m'_k||_2^2. \qquad \text{--> (13)}$$

Where m'_k is a tensor of 1s for all k Objects or 0 otherwise.

λ_n is the layer specific penalty hyperparameter.

Final objective is L = Lc+Ld. We must reduce this L loss during the training.

Simple experimental verification with single object and mask:

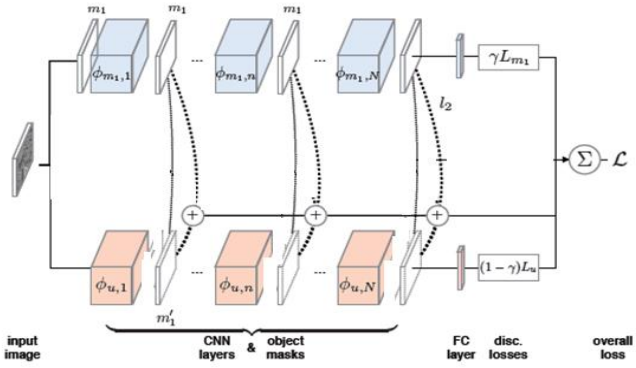


Fig. 18. Linearized CNN architecture for 1 object (K=1)

In this simplified experiment to inculcate linearity, we implement K =1, So we have 2 CNNs. We select 1 object from scene (accounts for masked CNN). We have 1 unmasked CNN which sees the entire scene. The Parameter (input features) space is shared by the 2 CNNs. We select the hyperparameter γ = 5. This model is 50% slower to train than a single CNN as there are 2 compositional CNNs.

To Minimize the loss function L, where L = lc + ld, the following method is used. Ld. is composed of L_{mk} and Lu are instantiated as softmax function and cross entropy is used to minimize it. Lc. is of standard Mean squared error (MSE) form(convex) and Stochastic Gradient Descent (SGD) can be used to minimize.

Experimental verification:

Experiments are performed on 3 types of image types.

I. 3D single and 3D multi images:

Fig. 19. Shows the test images of 3D single and 3D multi objects

Twelve -3D object classes (example: Guns, Bus etc.) containing 20 instance per class with 50 different perspective and with 20 different backgrounds. 3D-single **(**1600 images) shows single object in front of random backgrounds.3D-Multi (800 images) shows multi objects of various degrees of occlusion. We distinguish category level setting and between 3D-single and 3D-Multi images. There is an 80% training and 20% test set division among the images.

II. MNIST dataset:

Fig. 20. Shows the MNIST images

These are images of handwriting of numbers. We divide this dataset as 3D-single and 3D-multi as per previous general rules.

III. MS-COCO dataset:

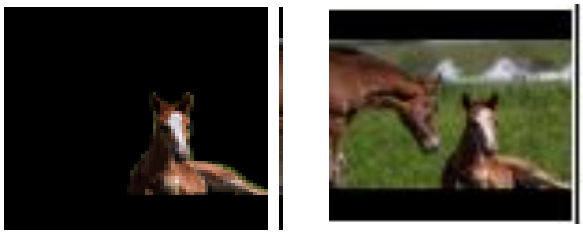


Fig. 21. Shows the MS-COCO images

These are images of objects in their natural contexts.1st test - we test classification performance on size of objects. 2nd Test – we test the classification for objects in and out of contexts.

Test Metrics:

For 3D Single,3D Multi and MNIST**,** performance is the average fraction of correctly predicted object classes among top-k scoring network predictions.

Where, k is the number of objects in the image.

For MS-COCO dataset: we treat object classes separately and report average precision (AP**).**

For all cases performance is monitored on a test over different epochs as training progresses.

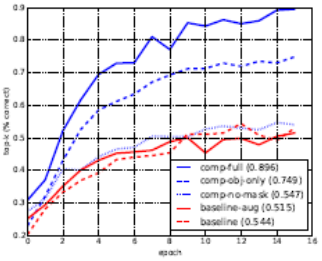
Test comparison strategy:

Below are the versions of CNN versions tested.

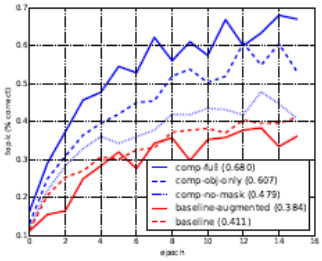
- COMP-FULL: Full compositional model.
- COMP-OBJ-ONLY: COMP-FULL model but with only mask equal to 1s for k objects, no mask for background activation.
- COMP-NO_MASK: COMP-FULL with no masks.
- BASELINE: A standard CNN.
- BASELINE-AUG: it splits each image into 2 equal parts and 1 part is put in black background and other half is put in cluttered background for each batch.
- BASELINE-REG**:** BASELINE with regularization.

- BASELINE-AUG-REG: BASELINE-AUG with regularization.

Results:

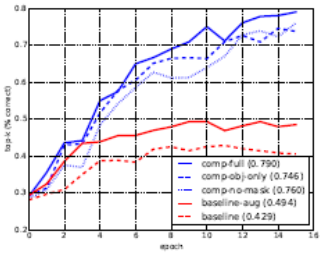


(a) 3D-Single-Inst

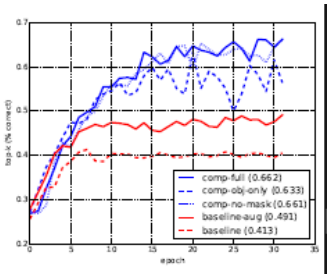


(b) 3D-Single

X- axis is epoch and y-axis are percentage accuracy of top-k. All variants of Compositional CNN perform well (Blue curves) esp. FULL_COMP. All baselines models show less accuracy esp. Baseline CNN.

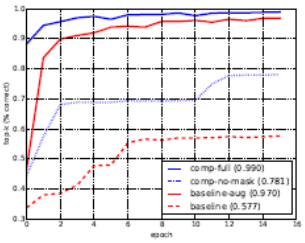


(c) 3D-Multi-Inst

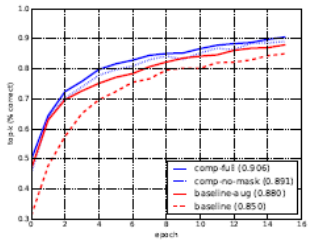


(d) 3D-Multi

For MNIST (handwritten Number images): The accuracy is almost same for compositional vs baseline, with COMP-FULL still beats baseline with small margin

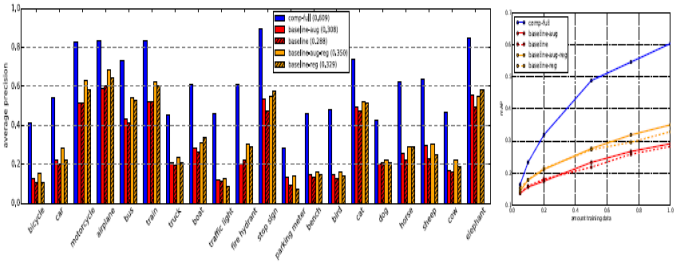


(e) MNIST-Single



(f) MNIST-Multi

MS-COCO (objects in their natural settings) : The accuracy for COMP-FULL is higher by 25.5% than baseline CNN.

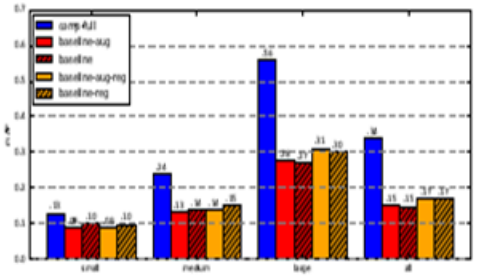


(a) MS-COCO-sub test performance (AP) per object category (corresponding to the last epoch for COMP-FULL / best case performance for baselines reported)

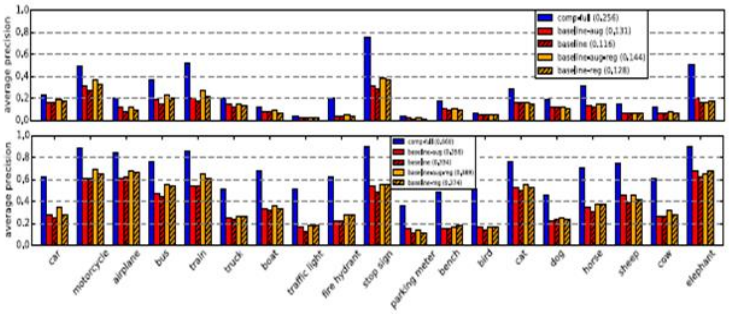
(b) Performance (mAP) for fractions of training data.

(a)For MS-COCO dataset, the FULL composition model's performance exceeds by 32% for various image types

(b)Gives Average precision for different sizes of training data –Here also the COMP-FULL exceeds the others.



(c) MS-COCO-sub performance for objects of different sizes



(d) MS-COCO-sub performance for objects in-context (bottom) and out-of-context (top).

COMP-FULL performance exceeds by 25% over others for MS-COCO-sub.

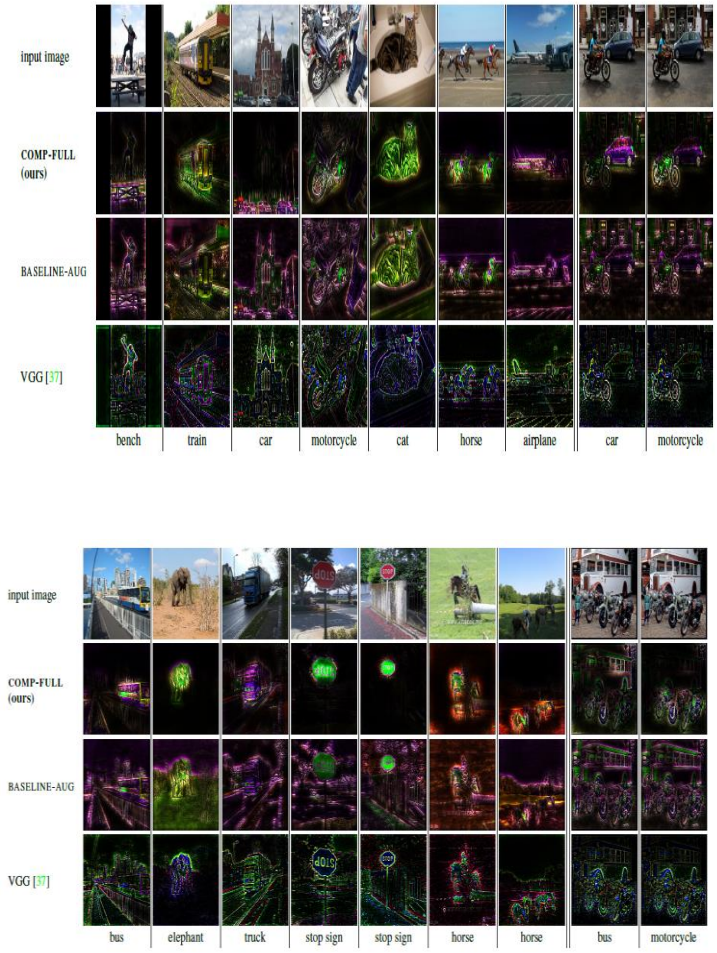


Fig. 22. Shows the MS-COCO dataset and various other CNN algorithms processed output images for qualitative analysis.

By the qualitative analysis of the images, it shows that COMP-FULL algorithm can detect the object from its surroundings well compared to baseline CNN and other algorithms.

IV. Conclusion:

The Subject of Compositionality in Modular Deep Neural networks inspired from biological brain were explored in depth using 2 instances. PART-A was modularization using EM algorithm which dynamically selects the best performing module for an input batch and assigns them thereby automatically selecting the best size of the deep neural network. This solves computation complexity by splitting big problem into smaller problem and time to train which are the 2 most important problem in deep networks. This also solves the problem of manually selecting the size (Layers) of DNN for a given task which can be inefficient and helps in transfer learning where pre trained modules can be mixed with new modules to reduce the training time.

In PART-B, we explored how the topic of computer vision takes inspiration from the biological visual cortex of the brain to do both linear and non-linear compositionality using CNN. We saw the baseline CNN exhibits non-linear compositionality and is not able to detect an object from its surroundings and required linear compositionality. We saw an algorithm that inculcated linearity by a modified training method, thereby making it detect an object from its surroundings more efficiently.

Going forward further study can be done to find the effects of EM Algorithm for large scale problems like RESNET and for small size datasets where It shows overfitting. Also the linearized CNN computation complexity is high as it employs k CNNs.

V. REFERENCES

[1] 7508 Modular networks learning to decompose neural computation – Louis Kirsch et al.

[2] Teaching compositionality to CNNs – Austin Stone et al.

[3] Do Deep Neural networks model Nonlinear compositionality in neural representations of human object interactions – Aditi Jha et al.

[4] https://machinelearningmastery.com/expectation-maximization-em-algorithm/

[5] Deep learning by Andrew Ng – Coursera

[6] http://blog.ivank.net/viterbi-algorithm-clarified.html

[7] Wikipedia

[8] https://en.wikipedia.org/wiki/Convolutional_neural_network

[9] https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/

[10] 2015 Springer Handbook of computations

6. Conduct Independent enquiry in computer science:

Please refer to the paper and artifacts in the below link for Automated Verification (CS-6315) course project below:

Source code, presentation slides and datasets are located in the link:

https://github.com/nithinkumar030/MS-portfolio/tree/main/automated_verification

Project presentation Video link:

https://youtu.be/lUWBeyL4Jf0

Project Demo Video link:

https://youtu.be/pINNMoGJuCg

Safety and Robustness Verification of Autoencoder based regression models using NNV tool

Nithin Kumar Guruswamy and Neelanjana Pal
Vanderbilt University, Nashville TN USA
Nithin.kumar.guruswamy@vanderbilt.edu
Neelanjana.pal@vanderbilt.edu

INTRODUCTION

A. **Autoencoder:** Autoencoders are a subset of Artificial Neural Networks (ANN) which try to reconstruct the input at the output. In general, an autoencoder model is composed of two main components namely encoder and decoder. The encoder part compresses the input into a latent space representation and the decoder tries to recreate the input from the output provided by the encoder. In other terms, the encoder reduces the feature dimensions of the input (Similar as PCA) and can thus be used for the data preparation steps for other machine learning models. Typical applications of autoencoders lies in data-denoising [1]–[3], high quality non-linear feature detection[4], anomaly detection [5], [6], fault classification [7], [8],imbalanced data classification [9] etc. Based on the learning objective all these applications can be broadly classified in two main types:

1) **Regression Task:** Here the autoencoder is basically used to recreate the input at the output.

2) **Classification Task:** In this application, the complete autoencoder model is first trained as a regression model. Then the decoder part is removed from the model and only the encoder part is used for Classification purposes. A SoftMax classification layer is added after the bottleneck and the input-labels are one hot coded and passed along with the input. These slightly modified model is then trained again to generate a autoencoder based classification model.

B. **Neural Network Verification Tool [10]:** Neural Network Verification Tool or NNV is a set based framework for neural network verification, developed by the VeriVital lab of Vanderbilt University. It supports several reachability algorithms for safety verification and robustness analysis of several types of deep neural networks(NN). In general, for reachability analysis of a NN, all reachable sets are calculated from an input and updated after each layers of the network and after the output layer the reachable state-space becomes an 'n-dimensional' one, 'n' being the number of different classes."

The reachable set obtained from NNV tool contains all possible states of a DNN (deep neural network)from bounded input sets" and a DNN is declared as Safe if, and only if, their reachable sets do not violate safety properties (i.e., have a non-empty intersection with any state satisfying the negation of the safety property)".

C. Verification of Autoencoder models: Although verification of Neural networks (e.g. feed-forward [11]–[14], Convolutional [15], [16], etc.) became quite popular with time and a major work is going on in several labs worldwide. But as per the authors knowledge there is no prior verification work in the domain of NN based autoencoders.

1 Method-1(Done by Nithin) : Safety and robustness verification of Autoencoder for regression using sigma method:

Autoencoder Architecture:

Autoencoder is a self-supervised Neural network which is used in many applications like classification, regression (timeseries forecasting) and anomaly detection methods. In this section, we are verifying autoencoder for regression tasks.

Like already mentioned in the previous section, **verification of the autoencoder for regression task is a novel problem and has not been addressed before in the prior literature.**

Here we are trying to novel approach to verify it using combination of NNV tool and 2 sigma method.

The autoencoder architecture is shown below. It is trained using Keras APIs and has 8 layers

```
29
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 29)                870
_________________________________________________________________
dense_1 (Dense)              (None, 16)                480
_________________________________________________________________
dense_2 (Dense)              (None, 8)                 136
_________________________________________________________________
dense_3 (Dense)              (None, 4)                 36
_________________________________________________________________
dense_4 (Dense)              (None, 2)                 10
_________________________________________________________________
dense_5 (Dense)              (None, 4)                 12
_________________________________________________________________
dense_6 (Dense)              (None, 8)                 40
_________________________________________________________________
dense_7 (Dense)              (None, 16)                144
_________________________________________________________________
dense_8 (Dense)              (None, 29)                493
=================================================================
Total params: 2,221
Trainable params: 2,221
Non-trainable params: 0
```

Fig 1 shows an autoencoder architecture used for verification

Autoencoder Verification Problem formulation:

We are trying to formally verify if a given autoencoder (Feedforward Neural network) meets some notion of specification defined over inputs and outputs.Here in this project, we consider input mean +/- 2 Sigma threshold as the input specification and output specification A +/- 1 as normalized values.

Given a pretrained feedforward/convolutional neural network *f: X -> Y,*

where typically assume X = R^n , Y = R^m.

A set of inputs , Xo subset of Y

Compute the set of outputs, *f(Xo)*

Finally check if ~P intersection f(Xo) == NULL ?

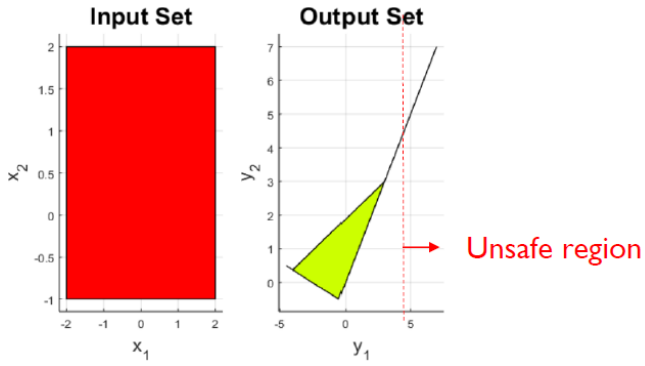


Fig 2 shows an reachable set and unsafe regions

Dataset and preprocessing

The dataset used to verify is credit card usage dataset which has 29 dimensions like time of transaction , amount of transaction etc.

The Autoencoder model is trained using credit card usage dataset which has 160000 records.epoch of 100 , batch size of 256.The trained model is saved in h5 format.The test data consist of 40000 samples.

NNV tool:

The Neural network verification (NNV) tool is used for verifying the safety and robustness of autoencoder. The NNV tool is written in matlab and uses the matlab MPT toolbox.

The NNV tool is available online at https://github.com/verivital/nnv.

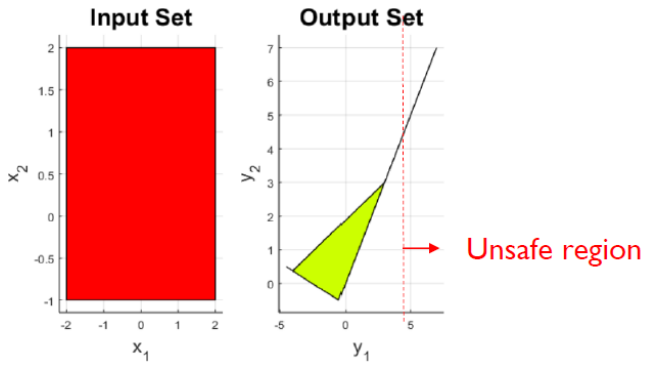


Fig 3 shows an unsafe region

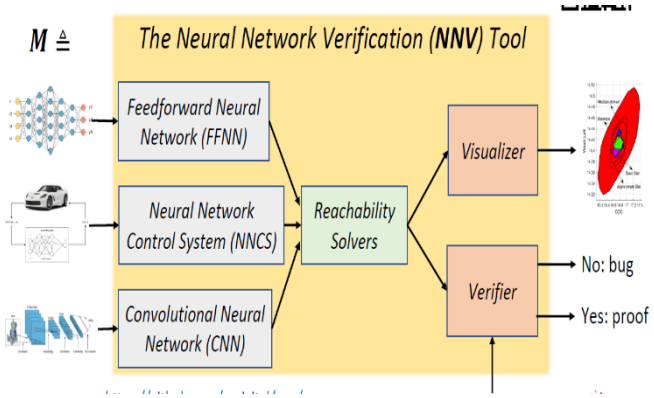


Fig 4 shows an NNV tool block diagram used to verify autoencoder

The Keras h5 autoencoder model needs to be imported into the NNV MATLAB tool using ImportKerasNetworks() API and needs to be converted into .mat file. Then it must be converted into CNN based autoencoder model.

Input test data:

We need to define a normalized input set of upper and lower bounds. The credit card usage dataset has 84807 records as test set. The upper and lower bounds are selected as mean+/- 2 sigma as shown in figure 5. This input set is converted into Starset.

A Starset can be a complex representation of a bounded convex polyhedron and is tuple of <c,V,P>. where where c belongs to R^n, V = {v1,v2,…,vm} is a set of basis vectors. P(Alpha) = [alpha1,alpha2,…alpham]' is a predicate variable.

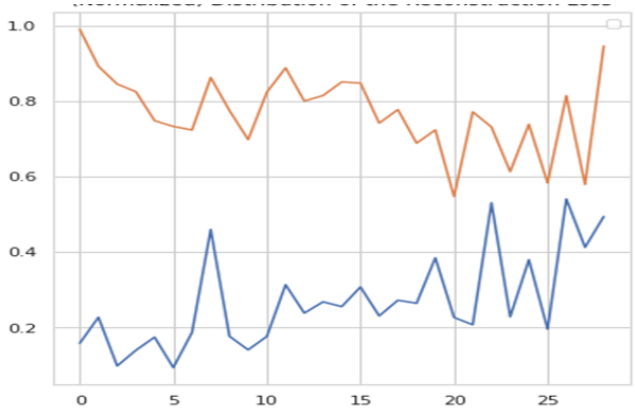


Fig 5 shows input test data upper bound(mean+2 Sigma) and (mean-2Sigma) lower bound.

The input starset is then converted into ImageStar set suitable for feeding into a CNN. ImageStar sets are an extension of starsets used to represent infinite sets of multi-channel images, But here it is used to represent credit card usage time-series data.

Reachability calculation for input test data using NNV tool :

The imageStar output from nnv tool calculates the normalized output reachability set as shown in figure 6.

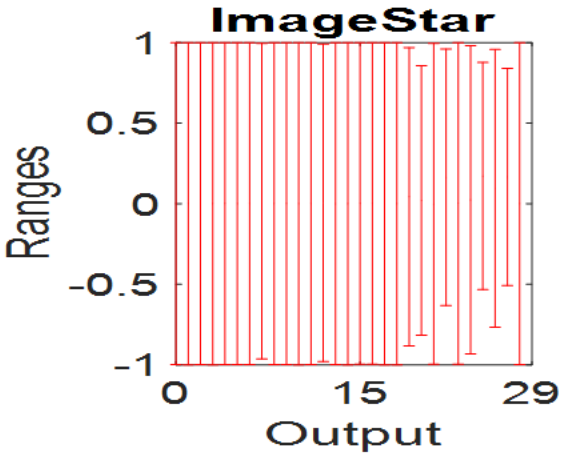


Fig 6 shows an normalized output reachability set for the normalized input test data (ImageStar) generated by NNV tool shown in fig 5.

Output bounds calculation from Keras Autoencoder model:

The output from the Keras autoencoder is analysed and lower and upper bound is as shown in the figure 7.

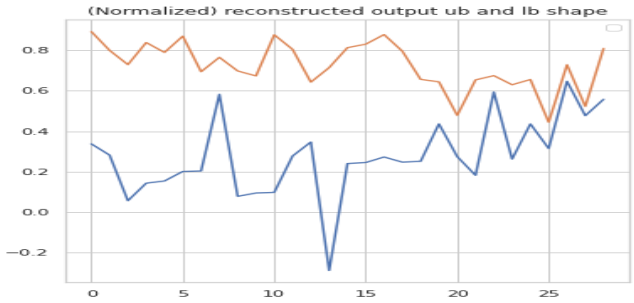


Fig 7 shows an normalized output set generated from the normalized input set by using keras autoencoder model for normalized input shown in fig 4

Method 2(Done by Neelanjana):

Please refer the submission done by Neelanjana in Brightspace to compare her methodology and experiments.

Conclusion:

The ranges of the keras produced output bounds shown in Fig 7 are within the ranges of NNV generated normalized output bounds as shown in Fig 6.

This verifies the input specification that any random test input within +/- 2 sigma bounds as shown in fig 5 are within the bounds produced by the NNV tool output as show in Fig 6. So this verifies the Autoencoder for the regression task for credit card usage dataset.

REFERENCES

[1] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, “Extract-ing and composing robust features with denoising autoencoders,” inProceedings of the 25th international conference on Machine learning,2008, pp. 1096–1103.

[2] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, P.-A. Manzagol, andL. Bottou, “Stacked denoising autoencoders: Learning useful represen-tations in a deep network with a local denoising criterion.”Journal ofmachine learning research, vol. 11, no. 12, 2010.

[3] Z. Chen, C. K. Yeo, B. S. Lee, and C. T. Lau, “Autoencoder-basednetwork anomaly detection,” in2018 Wireless Telecommunications Sym-posium (WTS), 2018, pp. 1–5.

[4] M. Chen, X. Shi, Y. Zhang, D. Wu, and M. Guizani, “Deep featureslearning for medical image analysis with convolutional autoencoderneural network,”IEEE Transactions on Big Data, pp. 1–1, 2017.

[5] C. Zhou and R. C. Paffenroth, “Anomaly detection with robust deepautoencoders,” inProceedings

of the 23rd ACM SIGKDD internationalconference on knowledge discovery and data mining, 2017, pp. 665–674.

[6] M. Sakurada and T. Yairi, “Anomaly detection using autoencoders withnonlinear dimensionality reduction,” inProceedings of the MLSDA 20142nd Workshop on Machine Learning for Sensory Data Analysis, 2014,pp. 4–11.

[7] W. Sun, S. Shao, R. Zhao, R. Yan, X. Zhang, and X. Chen, “A sparseauto-encoder-based deep neural network approach for induction motorfaults classification,”Measurement, vol. 89, pp. 171–178, 2016.

[8] N. Munir, J. Park, H.-J. Kim, S.-J. Song, and S.-S. Kang, “Performanceenhancement of convolutional neural network for ultrasonic flaw classi-fication by adopting autoencoder,”NDT & E International, vol. 111, p.102218, 2020.

[9] C. Zhang, W. Gao, J. Song, and J. Jiang, “An imbalanced dataclassification algorithm of improved autoencoder neural network,” in2016 Eighth International Conference on Advanced Computational Intelligence (ICACI), 2016, pp. 95–99.

[10] H.-D. Tran, X. Yang, D. M. Lopez, P. Musau, L. V. Nguyen, W. Xiang,S. Bak, and T. T. Johnson, “Nnv: The neural network verification toolfor deep neural networks and learning-enabled cyber-physical systems,”inInternational Conference on Computer Aided Verification. Springer,2020, pp. 3–17.

[11] H.-D. Tran, D. M. Lopez, P. Musau, X. Yang, L. V. Nguyen, W. Xiang,and T. T. Johnson, “Star-based reachability analysis of deep neuralnetworks,” inInternational Symposium on Formal Methods. Springer,2019, pp. 670–686.

[12] S. Dutta, S. Jha, S. Sankaranarayanan, and A. Tiwari, “Learning andverification of feedback control systems using feedforward neural net-works,”IFAC-PapersOnLine, vol. 51, no. 16, pp. 151–156, 2018.

[13] R. Ehlers, “Formal verification of piece-wise linear feed-forward neuralnetworks,” inInternational Symposium on Automated Technology forVerification and Analysis. Springer, 2017, pp. 269–286.

[14] W. Xiang, H.-D. Tran, and T. T. Johnson, “Specification-guidedsafety verification for feedforward neural networks,”arXiv preprintarXiv:1812.06161, 2018.

[15] K. D. Dvijotham, R. Stanforth, S. Gowal, C. Qin, S. De, and P. Kohli,“Efficient neural network verification with exactness characterization,”inUncertainty in Artificial Intelligence. PMLR, 2020, pp. 497–507.

[16] H.-D. Tran, S. Bak, W. Xiang, and T. T. Johnson, “Verification ofdeep convolutional neural networks using imagestars,” inInternationalConference on Computer Aided Verification. Springer, 2020, pp. 18–42.

[17] B. Zong, Q. Song, M. R. Min, W. Cheng, C. Lumezanu, D. Cho, andH. Chen, “Deep autoencoding gaussian mixture model for unsupervisedanomaly detection,” inInternational Conference on Learning Represen-tations, 2018.

[18] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, andM. Vechev, “Ai2: Safety and robustness certification of neural networkswith abstract interpretation,” in2018 IEEE Symposium on Security andPrivacy (SP). IEEE, 2018, pp. 3–18