دانشکده مهندسی کامپیوتر

تولید خودکار داده آزمون در فازرهای قالب فایل

پایان‌نامه برای دریافت درجه کارشناسی ارشد در رشته مهندسی کامپیوتر
گرایش نرم‌افزار

دانشجو:

مرتضی ذاکری نصرآبادی

استاد راهنما:

دکتر سعید پارسا

شهریور ۹۷

تأییدیه‌ی هیأت داوران جلسه‌ی دفاع از پایان‌نامه

نام دانشکده: دانشکده مهندسی کامپیوتر

نام دانشجو: مرتضی ذاکری نصرآبادی

عنوان پایان‌نامه: تولید خودکار داده آزمون در فازرهای قالب فایل

تاریخ دفاع: شهریور ۹۷

رشته: مهندسی کامپیوتر

گرایش: نرم‌افزار

ردیف	سمت	نام و نام‌خانوادگی	مرتبه دانشگاهی	دانشگاه / مؤسسه	امضا
۱	استاد راهنمای اول	دکتر سعید پارسا	دانشیار	دانشگاه علم و صنعت ایران	
۲	استاد راهنمای دوم	–	–	–	
۳	استاد مشاور	–	–	–	
۴	استاد مدعو داخلی	دکتر محمد عبداللهی ازگمی	دانشیار	دانشگاه علم و صنعت ایران	
۵	استاد مدعو خارجی	دکتر مجتبی وحیدی‌اصل	استادیار	دانشگاه شهید بهشتی	

تأییدیه‌ی صحت و اصالت نتایج

باسمه تعالی

اینجانب مرتضی ذاکری نصرآبادی به شماره دانشجویی ۹۵۷۲۳۰۸۸ دانشجوی رشته مهندسی کامپیوتر مقطع تحصیلی کارشناسی ارشد تأیید می‌نمایم که کلیه‌ی نتایج این پایان‌نامه حاصل کار اینجانب و بدون هرگونه دخل و تصرف است و موارد نسخه‌برداری‌شده از آثار دیگران را با ذکر کامل مشخصات منبع ذکر کرده‌ام. درصورت اثبات خلاف مندرجات فوق، به تشخیص دانشگاه مطابق با ضوابط و مقررات حاکم (قانون حمایت از حقوق مؤلفان و مصنفان و قانون ترجمه و تکثیر کتب و نشریات و آثار صوتی، ضوابط و مقررات آموزشی، پژوهشی و انضباطی ...) با اینجانب رفتار خواهد شد و حق هرگونه اعتراض درخصوص احقاق حقوق مکتسب و تشخیص و تعیین تخلف و مجازات را از خویش سلب می‌نمایم. در ضمن، مسؤولیت هرگونه پاسخگویی به اشخاص اعم از حقیقی و حقوقی و مراجع ذی‌صلاح (اعم از اداری و قضایی) به عهده‌ی اینجانب خواهد بود و دانشگاه هیچ‌گونه مسؤولیتی در این خصوص نخواهد داشت.

نام و نام خانوادگی: مرتضی ذاکری نصرآبادی

تاریخ و امضا:

11/28/2018

m-zakeri@live.com

مجوز بهره‌برداری از پایان‌نامه

بهره‌برداری از این پایان‌نامه در چهارچوب مقررات کتابخانه و با توجه به محدودیتی که توسط استاد راهنما به شرح زیر تعیین می‌شود، بلامانع است:

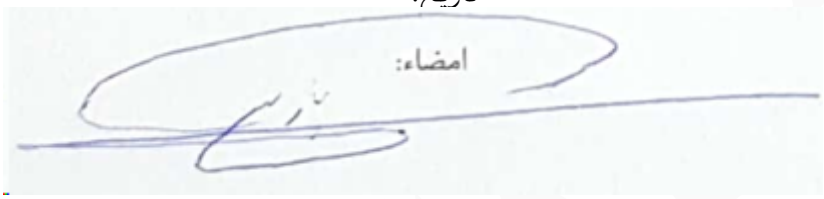
☐ بهره‌برداری از این پایان‌نامه برای همگان بلامانع است.

☑ بهره‌برداری از این پایان‌نامه با اخذ مجوز از استاد راهنما، بلامانع است.

☐ بهره‌برداری از این پایان‌نامه تا تاریخ ..... ممنوع است.

استاد راهنما: دکتر سعید پارسا

تاریخ:



تقــدیم

بـه دوستی که

می‌اندیشد،

می‌جوید،

می‌یابد

و

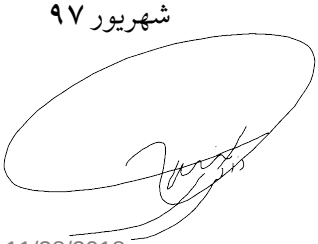
تغییر می‌دهد :)

قـدردانی

افرادی در دوره مهیج و کوتاه کارشناسی ارشد و در روند پژوهش و نگارش این پایان‌نامه به بنده کمک کرده‌اند که برخود لازم می‌دارم از زحمات ایشان سپاس‌گزاری نمایم. در ابتدا از استاد دانشمند و فرهیخته، آقای دکتر سعید پارسا که راهنمایی اینجانب را برعهده داشتند، صمیمانه قدردانی می‌کنم. بدون وجود حمایت‌ها و راهنمایی‌های ارزنده ایشان انجام این پایان‌نامه محقق نمی‌گردید. از داوران محترم پایان‌نامه آقایان دکتر محمد عبداللّهی ازگمی و دکتر مجتبی وحیدی‌اصل که با مطالعه و نقد سازنده خود، اسباب بهبود علمی و رفع نواقص این کار پژوهشی را فراهم نمودند، کمال تشکر را دارم. از همراهی‌ها و دلسوزی‌های همیشگی دو دوست و هم‌آزمایشگاهی بسیار عزیزم، آقای مهندس محسن امیریان و آقای مهندس سعید امیری، که بیش از ۶ سال با ایشان همکلاسی بودم، بی‌اندازه سپاس‌گزار هستم. از همکلاسی‌های بامعرفت و مهربانم در این دوره که همواره حامی و مشوّق بنده بوده‌اند، لحظه‌های تکرار نشدنی را کنار ایشان تجربه کرده‌ام و متأسفانه مجال نام‌بردن از آنها را در این متن کوتاه ندارم، بی‌نهایت ممنون هستم. آشنایی با این عزیزان را دستاوردی ارزشمند در زندگی خود می‌دانم و برای‌شان بهترین‌ها را آرزومندم. از دوستان گرامی، آقایان مهندس علی صابری و مهندس علی طاهری در آزمایشگاه رسانه دیجیتال دانشگاه صنعتی شریف، بابت انتقال مفاهیم اولیه یادگیری ژرف و آقای مهندس عرفان شرف‌زاده در آزمایشگاه سیستم‌های توزیع شده دانشکده مهندسی کامپیوتر، بابت پشتیبانی امور مربوط به پردازش ابری و ماشین‌های مجازی، تشکر می‌کنم. از آقای دکتر کارپتی برای وبلاگ و آموزش‌های مفید خود در زمینه یادگیری ژرف سپاس‌گزارم. از دیگر اساتید بزرگوارم در طول دوره کارشناسی ارشد از جمله آقایان دکتر محسن شریفی، دکتر بهروز مینایی و دکتر مهرداد آشتیانی (مدیر مرکز پردازش ابری دانشکده مهندسی کامپیوتر)، ممنون هستم. مهم‌تر از همه در پایان از خانواده عزیزم، به‌خصوص پدر و مادرم بابت حمایت‌های بی‌دریغ و همیشگی‌شان در همه مراحل زندگیم، مهربانانه سپاس‌گزاری کرده، دست ایشان را می‌بوسم.

مرتضی ذاکری نصرآبادی

شهریور ۹۷



11/28/2018
m-zakeri@live.com

تولید خودکار داده آزمون در فازرهای قالب فایل

چکیده

آزمون فازی (فازینگ) یک فن آزمون پویای نرم‌افزار است. در این فن با تولید و تزریق مکرر داده‌های آزمون بدشکل به نرم‌افزار تحت آزمون (SUT)، به‌دنبال یافتن خطاها و آسیب‌پذیری‌های احتمالی موجود در آن هستیم. برای نیل به این هدف، آزمون فازی نیازمند داده‌های آزمون متنوع است. مشکل اساسی، پیچیده بودن ساختار ورودی برنامه‌هایی است که فایل را به‌عنوان ورودی می‌پذیرند. بررسی‌ها نشان می‌دهد بسیاری از داده‌های آزمون تولیدی در این موارد، مسیرهای محدود و سطحی را می‌پیمایند؛ زیرا در همان مراحل اولیه به‌علت بدشکل، بودن توسط تجزیه‌گرِ SUT رد می‌شوند. استفاده از ساختار گرامری فایل‌ها برای تولید داده‌ها، منجربه افزایش پوشش کد می‌شود؛ اما، استخراج گرامر برای ساختار فایل، اغلب، دستی صورت می‌پذیرد که مستلزم صرف هزینه و زمان زیاد و مستعد خطای فراوان است. در این پایان‌نامه، روشی خودکار برای تولید داده آزمون به‌صورت ترکیبی ارائه می‌دهیم. در روش خود، از مدل‌های زبانی عصبی (NLMs) که با شبکه‌های عصبی مکرر (RNNs) ساخته می‌شوند، استفاده می‌کنیم. مدل‌های پیشنهادی با فنون یادگیری ژرف، قادر به یادگیری آماری ساختار فایل‌های پیچیده و سپس تولید داده‌های جدید متنی، به‌صورت مبتنی بر گرامر و داده‌های دودویی به‌صورت مبتنی بر جابه‌جایی، هستند. فاز (بدشکل‌سازی) داده‌های آزمون، نیز توسط دو الگوریتم فاز جدید، تحت عنوان الگوریتم‌های فاز عصبی، که از این مدل‌ها استفاده می‌کنند، صورت می‌پذیرد. از روش پیشنهادی خود، برای تولید داده و سپس آزمون فازی نرم‌افزار پیچیده MuPDF که فایل‌های قالبِ سندِ حمل‌پذیر (PDF) را به‌عنوان ورودی می‌پذیرد، استفاده نمودیم. برای آموزش مدل‌های مولد، یک پیکره بزرگ از فایل‌های PDF را گردآوری کردیم. آزمایش‌های ما نشان می‌دهد که داده‌های تولید شده با این روش، منجربه افزایش میزان پوشش کد اجرایی SUT و بهبود بیش از ۷ درصدی آن، در مقایسه با فازرهای قالب فایل مشهور، مثل AFL، می‌شود. آزمایش‌ها همچنین، بیان‌گر دقت یادگیری بهترِ NLMهای ساده‌تر در قیاس با مدل پیچیده‌تر کدگذار_کدگشا و نیز شکست این مدل، در میزان پوشش کد SUT، حین آزمون فازی، هستند.

واژگان کلیدی: آزمون فازی، داده آزمون، پوشش کد، یادگیری ژرف، شبکه عصبی مکرر.

فهرست مطالب

فهرست شکل‌ها

فهرست جدول‌ها

فهرست الگوریتم‌ها

فهرست کوته‌نوشت‌ها

A

ASCII ..... American Standard Code for Information Interchange

ASE ..... Automated Software Engineering

B

BPTT ..... Back Propagation Through Time

C

CAN ..... Creative Adversarial Networks

CFG ..... Control Flow Graph

CLI ..... Command Line Interface

CNN ..... Convolutional Neural Network

D

DAG ..... Directed Acyclic Graph

DFG ..... Data Flow Graph

G

GAN ..... Generative Adversarial Network

GRU ..... Gated Recurrent Units

GUI ..... Graphical User Interface

I

ISRT ..... Internet Security Threat Report

L

LM ..... Language Model

LSTM ..... Long-Short Term Memory

M

MOU ..... Multiple Objects Update

N

NLM ..... Neural Language Model

NLP ..... Natural Language Processing

P

PDF ..... Portable Document Format

R

RNN ..... Recurrent Neural Network

S

SDL ..... Security Development Lifecycle

SOU ..... Single Object Update

SUT ..... Software Under Test

فصل ۱

مقدمه

«من می‌گویم، امنیت، بالاترین اولویت ماست؛ زیرا برای همه چیزهای هیجان‌انگیزی که شما قادر به انجام دادن آن با کامپیوترها هستید — سازمان‌دهی زندگی‌تان، در ارتباط ماندن با دیگران، خلاق بودن — اگر ما مسائل امنیتی را حل نکنیم، مردم از همه این‌ها عقب خواهند ماند.»

✠ بیل گیتس

۱ـ۱ پیش‌زمینه

در حوزه مهندسی نرم‌افزار خودکار (ASE[1])، یکی از زمینه‌های مورد مطالعه و پژوهش، خودکارسازی فرایند آزمون نرم‌افزار، به‌عنوان یکی از مراحل مهم توسعه و ساخت یک سیستم نرم‌افزاری است. به‌طور کلی هدف از خودکارسازی، کاهش هزینه و زمان و افزایش دقت در اجرای یک فرایند است. آزمون فازی[۲] یکی از فنون آزمون خودکار نرم‌افزار است. آزمون فازی در یافتن خطا[۳]ها وآسیب‌پذیری[۴] ها در نرم‌افزارهای دنیای واقعی مانند مرورگرهای وب، ویرایش‌گرهای متن، پخش‌کننده‌های چندرسانه‌ای و غیره، بسیار مؤثر واقع شده

[1] Automated Software Engineering

[2] Fuzz Testing

[3] Fault

[4] Vulnerability

است [۱، ۲]. در این فن ورودی‌هایی بدشکل[1] توسط یک برنامه دیگر، یعنی با روش خودکار، تولید شده و به نرم‌افزار تحت آزمون (SUT[2]) تزریق می‌شود. SUT در عین حال، به امید یافتن خطا بر اثر پردازش ورودی تزریق شده، پایش[3] می‌شود. ورودی تولید شده که به برنامه داده می‌شود، نقش **داده آزمون**[4] را داشته و عامل اصلی نمایان‌سازی خطا(های) احتمالی موجود در برنامه با بردن آن به یک حالت خرابی[5] است. به‌همین علّت مهم‌ترین مرحله در فرایند آزمون فازی را می‌توان تولید خودکار داده‌های آزمون دانست، به‌نحوی که بیشترین خطاها، ایرادها و آسیب‌پذیری‌ها شناسایی گردند.

۱ ـ ۲ شرح مسئله

راه‌کارهای مطرح در فن آزمون فازی [۳—۶]، برای شناسایی خطاها و آسیب‌پذیری‌ها نیازمند تولید تعداد زیادی داده آزمون هستند. در نرم‌افزارهایی با ساختار ورودی ساده، تولید داده آزمون نیز ساده است. برای مثال می‌توان با روش تصادفی این کار را انجام داد. اما در نرم‌افزارهایی با ساختار ورودی پیچیده، مانند فایل با قالب مشخص تولید داده آزمون متنوع که بتواند مسیرهای اجرایی بیشتری را پوشش دهد، کار آسانی نیست. تعداد و عمق مسیرهای اجرایی در یک برنامه با ساختار ورودی پیچیده به مراتب بیشتر از یک برنامه با ساختار ورودی ساده است. بررسی‌ها نشان می‌دهد بسیاری از داده‌های آزمون تولید شده برای چنین نرم‌افزارهایی، مسیرهای یکسان و سطحی (کم عمق) را می‌پیمایند [۷] و در مجموع، آزمون‌های فازی معمول پوشش کد ضعیفی دارند [۸]. درصد بالایی از داده‌های آزمون ساخته شده به‌صورت تصادفی، از لحاظ ساختاری کاملاً نامعتبر هستند و در همان مراحل اولیه بررسی صحت فایل، به‌وسیله تجزیه‌گر[6] ورودی برنامه هدف، رد می‌شوند [۷، ۹]. در چنین شرایطی، قادر به نفوذ به عمق برنامه، کشف و آزمایش مسیرهای جدید نخواهیم بود. در واقع این نوع ورودی‌ها به نوعی تکراری و هدر رفته محسوب می‌گردند.

برای حل مسائل بالا، داده آزمون را با استفاده از قالب یا گرامر ورودی تولید می‌کنند، روش‌هایی مثل [۱۰]. قالب یا گرامر اما به صورت دستی و از روی مستندات تهیه می‌شود که با توجه به پیچیده بودن ساختار آن، عملی زمان‌بر، پرهزینه و مستعد خطا است [۱۱]. همچنین مستندات ساختار ورودی همواره دردسترس آزمون‌گر نیست. با این اوصاف روش مذکور تا به امروز، یکی از مؤثرترین روش‌های آزمون و یافتن خطا در برنامه‌هایی

[1] Malformed
[2] Software Under Test
[3] Monitor
[4] Test Data
[5] Failure
[6] Parser

مانند مرورگرهای وب بوده، که ساختار ورودی آن فایل‌هایی با قالب‌های متنوع و پیچیده هستند [۱۱، ۱۲]. به همین جهت، ارایه روشی برای خودکارسازی تولید داده آزمون بر مبنای قالب ورودی ارزشمند و حائز اهمیت است. پیش از ارایه یک روش جدید در ادامه ابتدا مسئله را دقیق‌تر تبیین کرده و راه‌حل‌های قبلی و نارسایی‌های هریک از آنها را مطالعه می‌کنیم.

۱ـ۲ـ۱ شهود اولیه

برای روشن شدن مسئله و شناسایی مشکلات موجود در تولید داده آزمون، مسئله را به زیر مسائل کوچک‌تر شکسته و از زوایای گوناگون تشریح می‌کنیم. ساختار پیچیده ورودی، ساختار پیچیده کد و تمایز داده و فراداده[1] سه زیر مسئله‌ای هستند که ما آنها را شناسایی کرده و در این بخش، مطرح می‌کنیم. در ادامه این پایان‌نامه تمرکز خود را بر روی حل این مسائل منعطف خواهیم کرد.

ساختار پیچیده ورودی

نخستین مورد حائز اهمیت ساختار ورودی برنامه است. در برنامه‌هایی با ورودی خط فرمان (CLI[2]) ساختارها به نسبت ساده هستند. اما برنامه‌هایی با ورودی فایل، ساختار ورودی بسیار پیچیده‌تری دارند. در واقع بسته به کاربرد، آنها یک یا چندین قالب فایل تعریف شده را پشتیبانی می‌کنند. همچنین برای یک قالب فایل شناخته‌شده ممکن است چندین نرم‌افزار وجود داشته باشد. یعنی در حالت کلی یک ارتباط چندبه‌چند بین قالب فایل ورودی و نرم‌افزار وجود دارد. هنگامی که یک نرم‌افزار برای آزمون انتخاب می‌شود هریک از قالب‌های فایلی که پشتیبانی می‌کند بخشی از کد نرم‌افزار را اجرا خواهند کرد.

فایل PDF[3] را می‌توان نمونه‌ای از یک ورودی پیچیده برای نرم‌افزارهای PDFخوان، مثل اغلب مرورگرهای وب، تلقی کرد. مجموعه اسناد توصیف کننده مشخصه‌ها[4]ی کامل قالب PDF بیش از ۱۳۰۰ صفحه است [۱۱]. جزئیات ساختار این قالب را در پیوست آ بیان کرده‌ایم. در ساختارهای پیچیده هر بایت و در مواردی هر بیت نقش ویژه‌ای ایفا می‌کند که تولید تصادفی آنها تنوعی در پوشش کد برنامه ایجاد نمی‌کند؛ زیرا اغلب در دام کدهای کنترل استثنا[5] می‌افتند. بنابراین داشتن یک درک حداقلی از ساختار در هنگام تولید داده جدید بسیار کمک کننده خواهد بود. چگونگی کسب این درک به‌صورت خودکار مسئله‌ای است که بایستی

[1] Metadata
[2] Command Line Interface
[3] Portable Document Format
[4] Specification
[5] Exception Handling

حل شود.

ساختار پیچیده کد

پیچیده بودن ساختار ورودی، منجربه پیچیده شدن کداجرایی و در نتیجه ممانعت از پوشش کد بالا در آزمون فازی خواهد شد. برای درک بهتر این مسئله برنامه ۱ ـ ۱ به زبان C را درنظر می‌گیریم. به‌خاطر سرعت بالای اجرا، بیشتر تجزیه‌گرهای قالب‌های پیچیده به این زبان نوشته می‌شوند. این برنامه یک فایل را از ورودی خوانده و براساس بایت‌های مشخصی در آدرس نسبی آن، مسیرهای معینی را اجرا می‌کند. چندین نکته قابل توجه در قطعه کد مذکور وجود دارد [۷] :

۱. **بایت‌های جادویی**[۱]: بایت دوم و بایت اول ابتدا برای اعتبارسنجی ورودی با مقادیر ثابتی مقایسه می‌شوند. اگر نتیجه این مقایسه صحیح نباشد؛ ورودی درجا رد می‌شود. در سطر ۱۳ این مثال ابتدا آدرسی نسبی ۱ با مقدار *0xEF* و سپس آدرس نسبی ۰ با مقدار *0xFD* مقایسه می‌گردد. بایت‌های جادویی در قالب‌های فایل بسیاری وجود دارند. از جمله قالب فایل jpeg که در ابزار djpeg[۲] با همین روش، اعتبارسنجی می‌شود.

۲. **شرط‌های تودرتو:** در اجرای برنامه، هر مسیر اجرایی مهم است. هرچند رسیدن به برخی مسیرها ممکن است دشوارتر باشد یا حتی امکان‌پذیر نباشد [۱۳] . در این مثال برای رسیدن به خط ۱۸ کد بایستی همه شرایط موجود در خط ۱۷ برقرار باشد که به‌نوبه خود نیاز هست تا شرط موجود در خط ۱۵ نیز برقرار شد و به همین ترتیب. لذا داده آزمون تولیدی باید تا حد زیادی معتبر باشد تا بتواند به عمق مدنظر دسترسی پیدا کند.

۳. **نشانگر[۳]ها:** برای رسیدن به کد خطادار در سطر ۱۹ بایستی شرط سطر ۱۸ ارضا شود. این مقایسه با یک توالی از نشانه‌ها انجام می‌شود که آدرس نسبی شروع آن لزوماً ثابت نیست؛ البته در این مثال ثابت نشان داده شده است. در قالب‌های فایل‌هایی مانند png، jpeg و gif این قبیل نشانگرها دیده می‌شود.

۴. **آدرس‌های نسبی متغیر:** برای رسیدن به مسیر اجرایی سطر ۱۸ یک مقایسه برمبنای آدرس‌های نسبی در سطر ۱۷ انجام می‌شود. آدرس نسبی به‌کار رفته در این مقایسه‌ها، از ورودی خوانده شده یا داخل برنامه محاسبه شده‌اند و بنابراین ممکن است که در هر بار اجرا متفاوت باشند. این امر برخلاف مورد بایت‌های جادویی است که آدرس نسبی ثابتی دارند.

[1] Magic Bytes

[2] https://linux.die.net/man/1/djpeg

[3] Markers

```
#include <stdio.h>
void main(int argc, char *argv[]){
  unsigned char buffer[1024]; //Fixed size buffer
  int fd, size, i, j;
  /* Some initialization here */
  if((fd = open(argv[1], O_RDONLY)) == -1)
    exit(0);
  fstat(fd, &s);
  size = s.st_size;
  if(size > 1024)
    return -1;
  read(fd, buffer, size);
  if(buffer[1] == 0xEF && buffer[0] == 0xFD) //Complex
 logic expression
    printf("Magic bytes matched!\n");
  else
    EXIT_ERORR("Invalid input file\n");
  if(buffer[i] == '%' && buffer[j] == '$' ){
    if(strcmp(&buffer[15], "MAZE", 4) == 0) //Nested
 condition
      /* Codes contain bug here */
    else{
      /* *** Render file here (lines of code) *** */
      close(fd);
      return 0;
      }
  else{
    EXIT_ERROR("Invalid bytes");
    close(fd);
    return 0;
    }
  close(fd);
}
```

برنامه ۱ـ۱: یک قطعه‌کد به عنوان نمونه‌ای از نرم‌افزار تحت آزمون در این پایان‌نامه، با ساختار تودرتو که چالش‌های پیچیدگی برنامه تحت آزمون و پوشش کد در آزمون فازی قالب فایل را نشان می‌دهد [۷](با تغییر).

تمایز داده و فراداده

برنامه مبتنی بر ورودی فایل، به‌طور معمول دو گام مجزا را برای پردازش یک فایل طی می‌کند: گام اول تجزیه[۱] فایل و گام دوم پرداخت[۲] آن. در مرحله تجزیه، فایل در حافظه بارگذاری، مقادیر فیلدهای آن خوانده شده و تبدیل به داده‌ساختارهای داخل حافظه اصلی (مثل بافر، ساختمان یا رکورد، آرایه و غیره) می‌شود. در این مرحله چنان‌چه اشکال[۳] نحوی در ساختار فایل باشد (فایل از مشخصه‌های قالب خود پیروی نکند)، باید توسط تجزیه‌گر تشخیص داده شود وگرنه منجربه اشکال فساد حافظه[۴] و خرابی برنامه می‌شود. در مرحله پرداخت، برنامه روی اطلاعات خوانده شده از فایل پردازش لازم را انجام می‌دهد و خروجی تولید می‌کند (مثلاً نمایش یک تصویر روی صفحه نمایش یا اجرای یک ویدئو و غیره) [۱۴]. خطاهای این مرحله معمولاً جدی‌تر بوده و تشخیص آن نیز مشکل‌تر است، زیرا در عمق بیشتری از کد اجرایی رخ می‌دهند. جایی که داده‌های آزمون کمتری به آن دست پیدا می‌کنند.

با توجه به توضیح بالا، می‌توان یک فایل را حاوی دو دسته از مقادیر دانست: اول، مقادیری که مشخص کننده ساختار آن فایل هستند؛ برای مثال نام فیلدها. این مقادیر را فراداده یا دادگان (داده برای داده) می‌نامند. دوم، مقادیری که مشخص کننده اطلاعات هر فیلد هستند یا همان داده‌های فایل. مسئله نهفته در اینجا آن است که رویکرد آشکارسازی خطا برای هر کدام از این قسمت‌ها متفاوت خواهد بود؛ چراکه طبیعت خطاهای هر قسمت با یکدیگر متفاوت بوده و همان‌طور که گفته‌شد در مراحل مختلفی هم روی می‌دهند. برای آشکار کردن خطاهای تجزیه‌گر، لازم است تا فایل‌هایی تولید کنیم که بخش فراداده آن بدشکل شده‌اند در حالی که برای آشکار کردن خطاهای بخش پرداخت، بایستی فایل‌هایی تولید کنیم که از لحاظ نحوی معتبر بوده و بخش داده آن بدشکل شده باشند. شرط لازم هر دو نوع بدشکل‌سازی داشتن سازوکاری برای تشخیص داده و فراداده از یکدیگر، در هنگام تولید داده‌های آزمون است.

همان‌طور که گفتیم، رسیدن به کدهای مرحله پرداخت یک فایل (منظور تزریق داده آزمونی است که منجربه اجرای آن شود) سخت‌تر است. در برنامه ۱ ـ ۱، فرض شده است که به عنوان مثال پرداخت فایل در خط ۲۱ انجام می‌شود؛ یعنی، بعد از گذشتن از تمامی شرایط و بررسی‌های انجام شده توسط تجزیه‌گر و انتقال فیلدهای داخلی فایل به حافظه اصلی (فیلدهایی مثل فیلد size در برنامه مذکور). هر داده آزمونی که یکی از شرایط قبل از خط ۲۱ را نداشته باشد، رد شده و آن اجرا از برنامه به اجرای خط ۲۱ منتهی نمی‌گردد. برای آن که درصد خوبی از داده‌های آزمون تولید شده به اجرای خط ۲۱ منجر شوند، بایستی یک فایل تقریباً معتبر و

[1] Parse
[2] Render
[3] Error
[4] Memory Corruption

پیروی کننده از قواعد قالب فایل مورد انتظار برنامه ۱ـ۱ را به عنوان داده آزمون تولید کرد.

آنچه از شهود داده شده در این قسمت نتیجه می‌شود آن است که از یک برنامه قابل اطمینان و غیر قابل نفوذ، انتظار می‌رود که تحت هیچ عنوان بر اثر پردازش یک ورودی دچار خطا نشود. تنها زمانی می‌توان این ادعا را داشت که مطمئن شویم برنامه ورودی‌های به اندازه کافی متنوع را پردازش کرده و در هیچکدام از آنها دچار خطا نشده است. ورودی‌ها بایستی قادر به اجرای بخش‌های زیادی از کد برنامه باشند. حالت ایده‌آل اجرای تمام کد یک برنامه پیچیده است.

۱ـ۲ـ۲ کارهای مرتبط

تعدادی کار در ارتباط با استخراج خودکار گرامر فایل انجام شده‌اند. Bastani و همکاران [۱۵] الگوریتمی برای تولید یک گرامر مستقل از متن روی یک مجموعه از ورودی‌های نمونه داده شده ارایه کرده‌اند، که در نهایت برای تولید داده‌های جدید مورد نیاز آزمون فازی استفاده می‌شود. این الگوریتم یک مجموعه از مراحل تعمیم‌پذیری را با معرفی ساختارهای تکراری و متناوب برای عبارت‌های منظم به‌کار می‌بندد و غیر پایانه‌ها را برای گرامر مستقل از متن در هم ادغام می‌نماید که به‌نوبه خود یک گرامر یکنواخت از زبان ورودی به‌دست می‌دهد؛ اما، این روش برای قالب‌هایی مثل PDF که ساختار مسطح (غیر تو در تو) ولی در عین حال محتوای مختلفی از انواع و جفت‌های کلید_مقدار دارند، مناسب نیست [۱۱].

AUTOGRAM [۱۶] نیز به‌صورت غیر_احتمالاتی یک گرامر مستقل از متن را یاد می‌گیرد. یک مجموعه ورودی داده‌شده و به‌صورت پویا مشخص می‌شود که چگونه ورودی‌ها در برنامه پردازش می‌شوند. در واقع برنامه تحت آزمون با آلودگی پویا مشاهده می‌شود که حافظه را با قطعات ورودی که از آنها می‌آیند، برچسب‌گذاری می‌کند. بخش‌هایی از ورودی‌ها که توسط برنامه پردازش می‌شود، نهادهای نحوی در گرامر می‌شوند.

در پژوهش‌های اخیر تمایل زیادی به استفاده از شبکه‌های عصبی برای تحلیل و تولید برنامه‌ها به‌وجود آمده‌است. در سال ۲۰۱۷، Godefroid [۱۱] و همکاران روش جدیدی را برای تولید داده آزمون جهت استفاده در آزمون فازی بر مبنای مدل کدگذار_کدگشا[1] [۱۷، ۱۸] ارایه کردند. در مقاله آنها، ساختار فایل PDF برای آزمون انتخاب شده است. ایده اصلی یادگیری یک مدل مولد روی مجموعه‌ای از ویژگی‌های اشیای داده‌ای PDF با داشتن مجموعه‌ای از نمونه‌های اولیه است. مدل کدگذار_کدگشا اجازه یادگیری متن با طول دلخواه را برای پیش‌بینی توالی بعدی کاراکترها، می‌دهد.

مدل استفاده شده توسط Godefroid و همکاران، مدل مبنایی وظایفی مانند ترجمه ماشینی یا تبدیل گفتار

[1] Encoder-Decoder Model

به نوشتار است که یادگیری ساختار فایل را نمی‌توان در این وظایف گنجاند؛ زیرا، این مدل برای نگاشت دو توالی با دامنه‌های مختلف به کار گرفته می‌شود و این در حالی است که یادگیری ساختار فایل چنین وظیفه‌ای نیست. یعنی می‌توان از مدل‌های ساده‌تری مانند مدل زبانی نیز برای یادگیری ساختار فایل استفاده کرد. روش پیشنهادی آنها، تنها ساختارهای متنی فایل را مورد یادگیری قرار می‌دهد. این در حالی است که ساختار فایل‌های پیچیده هم متنی و هم دودویی هستند. افزون بر این، الگوریتم پیشنهادی آنها برای تولید داده آزمون نیز مشکلاتی دارد. از جمله اینکه ممکن است هیچ‌گاه پایان نیابد. در فصل ۳، ضمن تشریح کامل این روش، مشکلات آن را نیز به‌صورت کامل‌تری بیان می‌کنیم. همچنین در فصل ۳، دو فازر قالب فایل دیگر تحت عنوان AFL [۱۹] و AFLافزوده [۲۰] که با روش‌هایی غیر از یادگیری گرامر سعی در بهبود پوشش کد SUT در فرایند آزمون فازی را دارند، نیز بررسی می‌کنیم و مشکلات آنها را بیان خواهیم کرد.

در هیچ‌کدام از کارهای قبلی، مسئله مطرح شده در ارتباط با تمایز میان داده و فراداده در هنگام آزمون فازی دیده نشده است. به عبارت دیگر، این دیدگاه به آزمون فازی قالب فایل، دیدگاهی نو است و ارزش آزمایش شدن دارد. امکان استفاده از الگوریتم ارایه شده در روش [۱۱]، برای تمایز میان داده و فراداده وجود دارد اما برای حل مابقی مشکلات، یک روش جدید را در فصل ۴ ، پیشنهاد خواهیم داد.

۱ ـ ۲ ـ ۳ فرضیه‌ها و اهداف

هدف اصلی در پایان‌نامه پیش‌رو، ارائه روشی کارا جهت یافتن خطاها و آسیب‌پذیری‌ها در نرم‌افزارهایی مثل PDFخوان‌ها بوده که ورودی آنها فایل با ساختار مشخص و معمولاً پیچیده است. در این راستا تولید خودکار فایل‌های ورودی با هدف افزایش پوشش کد[1] SUT از اهمیت ویژه‌ای برخوردار است. برای نیل بدین اهداف از فنون یادگیری ژرف[2] در یادگیری و درک خودکار ساختار فایل و سپس تولید فایل‌های جدید، استفاده خواهیم کرد.

چون ایده استفاده از یادگیری ماشینی در آزمون فازی جدید است، این حوزه هنوز برای پژوهشگران ناشناخته بوده و بنابراین یکی دیگر از اهداف این پایان‌نامه شناسایی، تعریف و تفکیک پارامترهای حاکم در حوزه مذکور است. به‌نظر می‌رسد که فنون یادگیری ماشینی راه‌گشای حل مسائل شرح داده شده در بخش ۱ ـ ۲ باشد. به‌همین جهت فراهم آوردن چارچوبی استاندارد برای شکل‌دهی به کارهای آتی، مفید و مثمر ثمر خواهد بود. به‌طور خلاصه ما چندین فرضیه در این پایان‌نامه درنظر داریم، که تدوین سازوکارهایی برای رد یا تأیید صحت آنها، اهدافِ ما خواهند بود:

[1] Code Coverage

[2] Deep Learning

- استفاده از فنون یادگیری ژرف بالأخص شبکه‌های عصبی مکرر ژرف، در یادگیری خودکار ساختار فایل، امکان‌پذیر و نتیجه‌بخش است.
- خودکارسازی کامل فرایند آزمون فازی مبتنی بر گرامر با ترکیب مدل یادگیری (مدل زبانی عصبی) و روش‌های فاز (بدـ شکل‌سازی) ورودی، به خوبی میسر می‌شود.
- روش‌های ترکیبی تولید داده آزمون، یعنی روش تولید مبتنی بر گرامر به همراه روش تولید مبتنی بر جابه‌جایی، منجر به افزایش پوشش کد SUT می‌گردند.
- امکان کشف خطاها و آسیب‌پذیری‌های احتمالی موجود در SUT از طریق آزمون فازی با داده‌های آزمون تولید شده از طریق مدل‌های یادگیری ژرف، وجود دارد.

۱ـ۳ روش پیشنهادی و نوآوری‌ها

در این پایان‌نامه یک روش برای یادگیری خودکار ساختار فایل و سپس تولید داده‌های آزمون بر اساس آن ارایه می‌شود. برای یادگیری از مدل زبانی (LM[1]) که یک مفهوم ابتدایی در پردازش زبان طبیعی (NLP[2])، است استفاده می‌کنیم. مدل زبانی را با استفاده از کلاس خاصی از شبکه عصبی ژرف[۳] موسوم به شبکه عصبی مکرر (RNN[4])، ایجاد می‌کنیم که در نتیجه به آن مدل زبانی عصبی (NLM[5]) هم گفته می‌شود. روش ارایه شده در اینجا، همچنین، بخش‌های غیرمتنی را نیز در آزمون فازی لحاظ می‌کند، به هیچ قالب فایل خاصی وابستگی نداشته و به سبب پیاده‌سازی با زبان پایتون قابلیت اجرا برروی هر ماشینی را دارد.

روش پیشنهادی در فصل ۴، در سه بخش کلی ارائه شده است. بخش اول به یادگیری ساختار فایل می‌پردازد (بخش ۴ـ۲) ، بخش دوم روشی برای تولید و بدشکل‌سازی همزمان داده‌های آزمون ارایه می‌دهد (بخش ۴ـ۳) و در نهایت بخش سوم یک فازر کاملاً پیمانه‌ای را معرفی می‌کند که از آن برای آزمون فازی قالب فایل استفاده خواهد شد (بخش ۴ـ۴) . در حالی که تمرکز اصلی بر روی نحوه تولید داده‌های آزمون است، اما برای انجام آزمون فازی به ابزارهای دیگری مانند تزریق کننده داده آزمون و نیز پایش SUT جهت ثبت خطاهای رخ داده شده نیاز است. نوآوری‌های اصلی روش پیشنهادی به طور خلاصه عبارتند از:

۱. یادگیری گرامر یا ساختار یک قالب فایل با استفاده از مدل‌های زبانی عصبی،

[1] Language Model
[2] Natural Language Processing
[3] Deep Neural Network
[4] Recurrent Neural Network
[5] Neural Language Model

۲ . تولید داده‌های آزمون متنی و دودویی همگام با بدشکل‌سازی آنها با استفاده از یک روش ترکیبی،

۳ . ایجاد یک فازر قالب فایل و یک مجموعه داده آزمون برای آزمون فازی نرم‌افزارهای PDF‌خوان،

۴ . و بررسی و شناسایی پارامترهای مؤثر در یادگیری ساختار فایل با استفاده از فنون یادگیری ژرف.

توضیح مبسوط‌تری از نوآوری‌های و دستاوردهای این پایان‌نامه در فصل ۶، ارایه شده است. در آن فصل همچنین مزایا و معایب فنون یادگیری ژرف در یادگیری ساختار فایل و نیز مزایا و معایب روش پیشنهادی بررسی و بیان شده‌اند.

۱ ـ ۴ اهمیت موضوع

مانند هر محصول دیگری، نرم‌افزار نیازمند آزمون[1] و راستی‌آزمایی است. ماهیت غیرقابل لمس و پیچیدگی ذاتی نرم‌افزار سبب می‌شود تا فرایند آزمون آن نیز متفاوت، پیچیده و پرهزینه باشد. اما این دشواری‌ها از اهمیت موضوع آزمون نمی‌کاهد. خطاهای نرم‌افزاری در مواردی سبب خسارت‌های مالی و جانی جبران ناپذیری شده‌اند. راکت آریان ۵[2] اروپا در سال ۱۹۹۶، تنها ۳۷ ثانیه پس از پرتاب منفجر شد. علت آن وقوع خطا در تبدیل نوع یک عدد ممیز شناور به عدد صحیح بود [۱۳]. وجود خطا در ماشین پرتودرمانی Therac-25[3]، سبب کشته شدن دست‌کم سه انسان بر اثر تشعشع بیش‌از حد پرتو، در سال‌های ۱۹۸۵ تا ۱۹۸۷ شد. مثال‌های دیگری از این قبیل در [۱۳، ۲۱] آمده است.

در مواردی وجود خطا منجربه آسیب‌پذیری می‌شود که امکان سوء استفاده و دسترسی‌های غیرمجاز را به مهاجمان[4] می‌دهد. باج‌افزار[5] WannaCrypt[6] که در نیمه اول سال ۲۰۱۷، بیش از ۲۳۰ هزار رایانه را در ۱۵۰ کشور جهان آلوده ساخت، از یک آسیب‌پذیری در هسته نسخه‌های قدیمی، سیستم عامل ویندوز شرکت مایکروسافت بهره‌برداری کرده بود. این باج‌افزار اطلاعات کاربر را رمزنگاری و برای رمزگشایی آن

[1] Test

[2] https://en.wikipedia.org/wiki/Ariane_5

[3] https://en.wikipedia.org/wiki/Therac-25

[4] Attacker

[5] Ransomware

[6] https://docs.microsoft.com/en-us/windows/security/threat-protection/wannacrypt-ransomware-worm-targets-out-of-date-systems-wdsi

درخواست پرداخت هزینه می‌کرد. شرکت سیمنتک[۱] در گزارش ISRT[2] خود در سال ۲۰۱۸[۳] [۲۲]، افزایش ۶۰۰ درصدی حملات در اینترنت چیزها[۴] (اینترنت اشیاء) و افزایش تهدیدات در دیگر حوزه‌های سایبری از جمله تلفن همراه، را اعلام کرده است. در هر حال، کشف خطا و آسیب‌پذیری احتمالی ناشی از آن، در نرم‌افزارهایی که به طور گسترده توسط همگان مورد استفاده قرار می‌گیرند، مثل سیستم‌عامل‌ها، مرورگرهای وب، PDFخوان‌ها و غیره، بسیار حائز اهمیت است؛ زیرا، در صورت برطرف نشدن آن خطر وقوع حملاتی مشابه حملات بالا دور از انتظار نخواهد بود.

هنگامی که نرم‌افزارها بزرگ می‌شوند، آزمون دستی پاسخ‌گو نیست و خودکارسازی آزمون اهمیت می‌یابد. آزمون فازی همان‌طور که در ابتدای فصل بیان شد، به عنوان یک فن مؤثر آزمون نرم‌افزار در شناسایی خطاهای حافظه و آسیب‌پذیری‌ها شناخته شده است. برای مثال چرخه حیات امن نرم‌افزار (SDL[5]) شرکت مایکروسافت [۶]، در مرحله درستی‌یابی[۷]، استفاده از آزمون فازی را به عنوان یک روش استاندارد، اجباری می‌کند [۲۳]. آزمون فازی جعبه سفید[۸] (رجوع کنید به بخش ۲ـ۱ـ۲)، حدود یک سوم کل آسیب‌پذیری‌های شناخته شده در سیستم عامل ویندوز ۷ این شرکت را کشف کرده است [۱۰]. شرکت گوگل در سال ۲۰۱۲، اطلاعاتی راجع‌به ابزار ClusterFuzz خود منتشر کرد که از آن برای آزمون فازی پروژه‌های Chromium [۹] (شامل Chromium OS و مرورگر وب Chromium) استفاده می‌کند [۱۲]. این شرکت همچنین به افرادی که موفق به کشف آسیب‌پذیری در پروژه‌های نام‌برده شوند، جوایزی اهدا می‌کند.

تولید داده آزمون را بایستی مهم‌ترین مرحله در آزمون فازی دانست؛ چراکه داده‌هایی که نرم‌افزار با آنها آزمون می‌شود عامل اصلی اجرا شدن کدهای قسمت‌های مختلف SUT است و در صورتی که خطایی در آنها وجود داشته باشد، تنها از این طریق است که خود را نشان می‌دهد. البته باید بدین مسئله توجه کرد که اجرای کد خطادار شرط لازم برای آشکارسازی خطا است ولی کافی نیست و روش تولید داده‌های آزمون، می‌بایست شرایط خاص بدشکل بودن را نیز محیا کند. تولید داده مبتنی بر گرامر، مؤثرترین روش آزمون برنامه‌هایی با ساختار ورودی پیچیده است [۲۴]. موارد بیان شده در این بخش، به‌خوبی اهمیت موضوع تولید خودکار داده آزمون در آزمون فازی و لزوم ارایه روش‌های جدید را توجیه کرده و انگیزه کافی را برای پژوهش در این

[1] https://www.symantec.com/
[2] Internet Security Threat Report
[3] https://www.symantec.com/security-center/threat-report
[4] Internet of Things
[5] Security Development Lifecycle
[6] https://www.microsoft.com/en-us/sdl
[7] Verification
[8] White Box
[9] https://www.chromium.org/

زمینه ایجاد می‌کنند.

۱ ـ ۵ ساختار پایان‌نامه

این پایان‌نامه در شش فصل و دو پیوست تنظیم شده است و ساختار ادامه آن به قرار زیر است. در فصل ۲ ادبیات موضوع شامل آزمون نرم‌افزار، آزمون فازی و یادگیری ژرف را مطرح می‌کنیم. در این فصل ابتدا معیارهای سنجش کیفیت آزمون و چگونگی محاسبه آنها را توضیح داده، سپس به معرفی آزمون فازی، فرایند کلی و روش‌های تولید داده آزمون در آن می‌پردازیم. در بخش پایانی مباحث یادگیری ژرف را با تمرکز بر مفاهیم مرتبط با یادگیری ساختار فایل، عنوان خواهیم کرد.

در فصل ۳ به پیشینه پژوهش و بیان کارهای مرتبط در تولید خودکار داده آزمون و نقد و بررسی آنها می‌پردازیم. به‌طور خلاصه برخی راه حل‌های دیگران برای مسائل مطرح شده در بخش ۱ ـ ۲ را معرفی و سپس مشکلات آنها را بیان می‌کنیم. روش پیشنهادی در راستای حل این مسائل و ارزیابی ما در مقایسه با نتایج ارائه شده قبلی در این فصل خواهد بود.

در فصل ۴ روش پیشنهادی خود را برای تولید داده آزمون مطرح می‌کنیم. روش پیشنهادی در این فصل، همان‌طور که بدان اشاره شد، یک روش تولید مبتنی بر مدل‌های زبانی است که ما جابه‌جایی‌های تصادفی را نیز به آن اضافه کرده و روشی ترکیبی خلق نموده‌ایم. در همین فصل، ما دو الگوریتم جدید را برای فاز داده‌های آزمون معرفی می‌کنیم.

در فصل ۵ معیارهای ارزیابی روش پیشنهادی، چیدمان آزمایش‌ها و نتایج حاصل از اجرای آنها را ذکر خواهیم کرد. مورد مطالعاتی ما در آزمایش‌های این فصل نرم‌افزار MuPDF[1] [۲۵] و قالب فایل PDF است که در ابتدای فصل آنها را مختصر معرفی خواهیم کرد.

در نهایت فصل ۶ را به بیان نتیجه‌گیری، یافته‌ها و نوآوری‌های پایان‌نامه، محدودیت‌های روش پیشنهادی و کارهای قابل انجام در آینده اختصاص داده‌ایم. همچنین در پیوست آ ساختار فایل PDF و در پیوست ب جزئیات پیاده‌سازی محصول نهایی پایان‌نامه را درج کرده‌ایم.

[1] https://mupdf.com/

فصل ۲

مفاهیم اولیه

«آزمون نرم‌افزار می‌تواند وجود خطاها را نشان دهد، اما هرگز نمی‌تواند نبود آنها را تضمین کند.»

✠ ادسخر دایکسترا

در بخش اول این فصل تعاریف پایه‌ای آزمون نرم‌افزار و معیارهای سنجش آزمون، روش اندازه‌گیری و گزارش آنها را بیان می‌کنیم. در بخش دوم، آزمون فازی، فنون مختلف به‌کار گرفته شده در این آزمون، آزمون فازی قالب فایل، روش‌های تولید خودکار داده آزمون در فازر[1] ها و معماری فازرها را توضیح می‌دهیم. بخش سوم و چهارم را به معرفی فنون یادگیری ژرف و استفاده از آنها در وظایف یادگیری مبتنی بر توالی[2] تخصیص داده‌ایم. در پایان نیز خلاصه‌ای از مطالب مطرح شده در فصل را می‌آوریم. مفاهیم اولیه بسیار گسترده‌تر از آنچه در این فصل مطرح گردیده هستند و به‌همین دلیل در بخش پایانی خواننده را به منابع مناسبی ارجاع داده‌ایم. همچنین مطالعه کامل‌تری در باب مطالب این فصل در گزارش سمینار کارشناسی ارشد اینجانب [۲۶] انجام شده است.

---

[1] Fuzzer

[2] Sequence

۲ ـ ۱ آزمون نرم‌افزار

مجموعه فنون کشف و آشکارسازی خرابی‌های نرم‌افزار در مراحل مختلف توسعه آن را **آزمون نرم‌افزار** گویند. منظور از خرابی بروز رفتار(های) ناخواسته و خلاف مشخصه‌ها در یک نرم‌افزار یا قسمتی از آن است. خرابی خود حاصل یک خطا (نقص) ایستا در نرم‌افزار است. حالت داخلی نادرست برنامه را که ناشی از یک خطا است، اشکال می‌گویند [۱۳]. واژه‌های خطا، اشکال و خرابی از حوزه اتکاپذیری[1] وارد آزمون نرم‌افزار شده‌اند [۲۱] و تعاریف یکسانی برای آنها وجود ندارد [۱۳، ۲۷].

آزمون نرم‌افزار منحصر به کد اجرایی برنامه نیست و شامل آزمون نیازمندی[2]ها، آزمون مشخصه‌ها و طراحی[3] نیز می‌گردد. در آزمون فازی اما مُنحصراً به کد اجرایی پرداخته می‌شود. لذا در ادامه این پایان‌نامه، منظور از آزمون، فقط آزمون مربوط به کداجرایی برنامه خواهد بود. واژگان حاکم بر حوزه آزمون نرم‌افزار بسیار گسترده می‌باشد. در اینجا روی برخی از مفاهیم پایه‌ای و مرتبط با آزمون فازی و تولید داده آزمون تمرکز می‌کنیم.

۲ ـ ۱ ـ ۱ معیارهای پوشش

متأسفانه آزمون تنها قادر است وجود خرابی را نشان دهد و نبود آن را تضمین نمی‌کند. به‌بیان صوری، مسئله یافتن تمامی خرابی‌ها در یک برنامه تصمیم‌ناپذیر[4] است [۱۳]. این موضوع سبب می‌شود به‌دنبال راه‌کارهایی برای اندازه‌گیری آزمون و تعیین میزان خوب بودن آن باشیم. در دهه‌های اخیر تمرکز بر روی معیارهای پوشش[5]، شناسایی، تفکیک و سنجش آنها بوده است. معیارهای پوشش بیشتر با هدف کمی‌سازی و اندازه‌گیری مقدار آزمون انجام شده مطرح شده‌اند؛ هرچند متقابلاً در تولید داده آزمون هم کاربرد دارند. چون با مسئله‌ای تصمیم‌ناپذیر مواجه هستیم، بایستی معیاری وجود داشته باشد که مشخص کند چه زمانی می‌توانیم آزمون را خاتمه دهیم و در این زمان آزمون تا چه‌حد خوب انجام شده است. آمان [6] و آفوت[7] [۱۳] معیارهای پوشش

[1] Dependability
[2] Requirement
[3] Design
[4] Undecidable
[5] Coverage Criteria
[6] P.Ammann (https://cs.gmu.edu/ pammann)
[7] J. Offutt (https://cs.gmu.edu/ offutt/)

را به‌صورت چهار معیار افراز فضای ورودی[۱]، پوشش گراف[۲]، پوشش منطق[۳] و پوشش ساختار نحوی[۴] مطرح کرده‌اند.

پوشش گراف در سطح کد اجرایی که به آن پوشش کد هم گفته می‌شود، شامل پوشش گراف جریان کنترل (CFG[5]) و گراف جریان داده (DFG[6]) برنامه می‌شود. پوشش منطق، مقداردهی و تعیین ارزش عبارات منطقی ظاهر شده در متن برنامه است. افراز فضای ورودی یعنی انتخاب از بین حالت‌های مختلف ترکیب ورودی‌ها و در نهایت پوشش ساختار نحوی، استفاده از قوانین گرامر برای اعتبارسنجی[۷] داده‌های ورودی یا تولید داده‌های جدید آزمون را شامل می‌شود.

معیارهای پوشش به دو روش مختلف قابل استفاده هستند. یک روش به این صورت است که مقادیر داده‌های آزمون منحصراً برای برآوردن یک معیار داده شده، تولید گردند. این روش در برخی موارد بسیار دشوار است. به‌ویژه اگر ابزاری برای تولید خودکار داده آزمون نداشته باشیم یا ساختار ورودی پیچیده باشد. روش دوم تولید داده‌های آزمون به‌طور کاملاً مجزا و سپس اندازه‌گیری میزان پوشش معیار مربوطه است. برای روش اول یک برنامه شناسنده[۸] برآورده شدن معیار و برای روش دوم یک برنامه مولد[۹] داده آزمون نیاز است. در صنعت استفاده از روش دوم مرسوم‌تر است [۱۳].

در این پایان‌نامه، معیار دوم از معیارهای پوشش (پوشش کد) با روش دوم را مبنا قرار می‌دهیم. به این صورت که یک روش تولید خودکار داده آزمون و اندازه‌گیری میزان پوشش کد ارایه خواهیم داد. این فن آزمون، به‌خوبی عملی بوده و آزمون فازی معمول نیز بر اساس این روش است. پوشش کد خود در سطوح مختلفی مطرح است؛ پوشش دستور[۱۰]، پوشش شاخه[۱۱] و پوشش مسیر[۱۲] سه سطح شناخته شده هستند. در پوشش دستور اجرای حداقل یک‌مرتبه هر دستور برنامه به عنوان نیازمندی تعریف و سپس اندازه‌گیری می‌شود. در پوشش شاخه اجرای حداقل یک‌مرتبه هر انشعاب برنامه به عنوان نیازمندی مطرح است و بلأخره در پوشش مسیر اجرای حداقل یک‌مرتبه هر مسیر اجرایی مد نظر است. بدیهی است که پوشش مسیر، پوشش شاخه و

[1] Input Space Partitioning
[2] Graph Coverage
[3] Logic Coverage
[4] Syntax-Based Coverage
[5] Control Flow Graph
[6] Data Flow Graph
[7] Validation
[8] Recognizer Program
[9] Generator Program
[10] Statement Coverage
[11] Branch Coverage
[12] Path Coverage

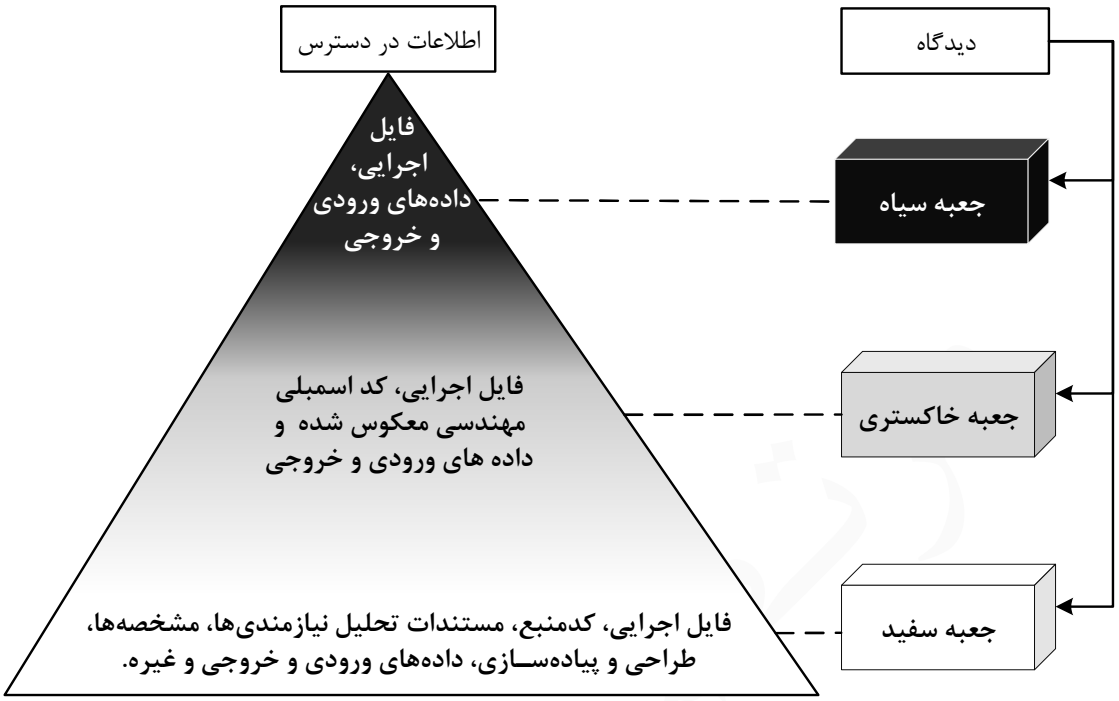
پوشش شاخه، پوشش دستور را شامل می‌شود. تعاریف و فهرست کاملی از سطوح پوشش کد و رابطه بین آنها در [۱۳] ذکر شده است.

شرکت مایکروسافت در محیط توسعه مجتمع ویژوال استادیو[۱]، معیارهای پوشش خط[۲]، پوشش بلوک پایه[۳] و پوشش خط جزئی[۴] را ارایه کرده است. پوشش خط همان پوشش دستور در کد سطح بالا (به‌جای کد اسمبلی/ماشین) است. پوشش بلوک پایه تعمیمی از پوشش دستور است که در آن یک توالی از دستورهای برنامه که شامل هیچ دستور پرشی در بین خود نیستند، یک دستور واحد در نظر گرفته می‌شوند. مزیت پوشش کد در سطح دستور و بلوک پایه[۵] سادگی اندازه‌گیری آن و عدم نیاز به وجود کد منبع[۶] برنامه است [۲۱]. پوشش خط جزئی منظور اجرا شدن برخی از قسمت‌های یک خط کد سطح بالا است. برای مثال در خطوطی با عبارات شرطی طولانی بر اثر ارزیابی مدار کوتاه[۷] ممکن است کلیه دستورات آن خط کد اجرا نگردد. در نتیجه پوشش خط جزئی معیاری متفاوت از پوشش خط است.

علاوه بر موارد ذکر شده معیارهای دیگری هم برای پوشش کد ارایه گردیده، از جمله پوشش دامنه[۸] که در **آزمایشگاه تحقیقاتی مهندسی معکوس[۹] دانشگاه علم و صنعت ایران**[۱۰] توسعه داده شده است [۲۸]. پوشش دامنه، نوعی از معیار افراز فضای حالت [۱۳] است که در آن با استناد به تئوری مجموعه‌ها، دامنه‌هایی برای هریک از ورودی‌های برنامه تعیین می‌شود. انتخاب ورودی از این دامنه‌ها در زمان آزمون، منجربه پوشش یک مسیر مشخص شده می‌گردد. از آنجایی که تنها یک بار اجرای یک مسیر، لزوماً خطای موجود در آن را آشکار نمی‌کند؛ در این روش می‌توان مسیر داده شده را به هر تعداد و هر بار با داده‌های آزمون متفاوت آزمون کرد. از این دیدگاه، پوشش دامنه یک روش تولید داده آزمون را ارایه می‌دهد.

در این پایان‌نامه، پوشش بلوک پایه و پوشش خط را برای گزارش میزان پوشش کد به کار می‌بریم. در بین این دو معیار، پوشش بلوک پایه در مقایسه با پوشش خط برنامه معیار مناسب‌تری است؛ زیرا، تعداد خطوط برنامه را می‌توان تغییر داد. برای مثال چندین دستور را در یک خط قرار داد یا اینکه یک دستور را در چند خط نوشت. تعداد بلوک پایه اما مستقل از نحوه آرایش و قرارگیری خطوط برنامه است و این معیار با تغییراتی مانند آنچه گفته شد، تغییر نمی‌کند؛ زیرا، در هر حالت تعداد بلوک‌های پایه یک برنامه و گراف جریان کنترل ثابت

[1] Visual Studio (visualstudio.microsoft.com)
[2] Line Coverage
[3] Basic Block Coverage
[4] Partial Line Coverage
[5] Basic Block
[6] Source Code
[7] Short-Circuit Evaluation
[8] Domain Coverage
[9] Reverse Engineering
[10] IUST Reverse Engineering Research Laboratory (http://parsa.iust.ac.ir/reverse-engineering-lab/)



شکل ۲ ـ ۱ : طرح‌واره‌ای از دیدگاه جعبه بر اساس اطلاعات در دسترس به‌صورت یک مثلث آزمون [۲۶].

است. البته این تأکید صرفاً برای مواردی است که ممکن است آرایش خطوط برنامه در آزمون‌های مختلف تغییر کند، بدون اینکه تغییری در خود کد ایجاد شده باشد.

۲ ـ ۱ ـ ۲ دیدگاه جعبه

نحوه اندازه‌گیری معیارهای بحث شده، تنظیم و سنجش آزمون، منوط به اطلاعات در دسترس از SUT است. آزمون نرم‌افزار را براساس اطلاعات در دسترس از SUT در زمان آزمون، به سه دسته/ حالت جعبه سیاه[۱]، جعبه سفید و جعبه خاکستری[۲] تقسیم‌بندی می‌کنند [۱۳، ۲۹]، که آن را دیدگاه جعبه به آزمون نرم‌افزار گوییم. شکل ۲ ـ ۱ این تقسیم‌بندی را به‌صورت یک طرح‌واره مثلثی نشان می‌دهد.

همان‌طور که در این طرح‌واره پیداست، در رأس مثلث، هیچ اطلاعی از ساختار داخلی SUT در اختیار آزمون‌گر نیست. در واقع با برنامه به مثابه یک جعبه سیاه که ورودی را دریافت و خروجی تولید می‌کند، رفتار می‌شود. این نوع آزمون معمولاً روی برنامه‌هایی که منتشر شده‌اند و توسط افرادی خارج از تیم توسعه و پشتیبانی محصول انجام می‌شود. هدف آن هم بیشتر کشف آسیب‌پذیری‌ها و ضعف‌های امنیتی است. در پایین

[1] Black Box

[2] Gray Box

مثلث، دیدگاه جعبه سفید وجود دارد. در اینجا تقریباً تمامی اطلاعات SUT در دسترس است. دیدگاه جعبه خاکستری مابین دو دیدگاه قبلی قرار دارد. در این دیدگاه از فنون مهندسی معکوس برای استخراج اطلاعات از SUT استفاده می‌شود. هریک از این دیدگاه‌ها مزایا و معایب خاص خود را دارند [۳۰]. آزمون فازی هم که در ادامه مطرح می‌شود، قابل تقسیم به‌این سه دسته است. آزمون فازی جعبه خاکستری، برای مثال، بسیار ثمربخش بوده است [۱۹، ۲۰]. روش پیشنهادی در این پایان‌نامه برای تولید داده آزمون، قابل استفاده در هر سه دسته آزمون فازی است. برای ارزیابی روش پیشنهادی در این پایان‌نامه، در فصل ۵، اما از آزمون جعبه سفید، استفاده کرده‌ایم تا بتوانیم معیارهای پوشش کد و بخش‌های اجرا شده را به‌دقت اندازه‌گیری نماییم. از طرفی SUT انتخابی ما متن باز و رایگان بوده و در این حالت کد منبع آن دردسترس است.

۲ ـ ۱ ـ ۳ اندازه‌گیری پوشش کد

هدف از مطرح کردن دیدگاه جعبه در بخش ۲ ـ ۱ ـ ۲، اشاره به امکانات لازم برای اندازه‌گیری معیارهای پوشش از جمله پوشش کد بود. در آزمون جعبه سفید که کد منبع برنامه وجود دارد، مشاهده پوشش کد برنامه با اضافه کردن دستوراتی به متن کد به آسانی امکان پذیر است. این عمل را ابزارگذاری[1] کد یا تجهیز برنامه به کد رهگیری می‌نامند [۱۴]. در محیط GNU/GCC (زبان‌های C و C++)، برای ابزارگذاری کافی است کد منبع برنامه را همراه با پرچم‌های `-fprofile-arcs` و `-ftest-coverage` کامپایل کنیم:

```
1  ~$ gcc -g -fprofile-arcs -ftest-coverage -o test test.c
2  ~$ ./test f1 f2
3  ~$ gcov test.c
4  ~$ more test.c.gcov
```

برنامه ۲ ـ ۱: ابزارگذاری کد در محیط GNU/GCC.

سطر اول برنامه ۲ ـ ۱ برنامه test را با پرچم‌های ذکر شده کامپایل می‌کند. سطر دوم برنامه را با دو آرگومان خط فرمان `f1` و `f2` اجرا می‌کند. سطر سوم پوشش کد اجرای اخیر برنامه را محاسبه و فایلی تحت عنوان `test.c.gcov` تولید می‌کند. در سطر چهارم این فایل برای مشاهده باز می‌شود. افزون بر این، پرچم‌هایی برای اخذ انواع سطوح پوشش کدی که بحث شد، از جمله پوشش شاخه، وجود دارد [۱۴].

محیط ویژوال استادیو ابزار vsinstr را برای ابزارگذاری کد برنامه و VSPerfMon را برای اندازه‌گیری پوشش کدِ زبان‌های پشتیبانی شده توسط این محیط، ارائه می‌دهد که البته در حضور کد منبع برنامه قابل استفاده

[1] Instrumenting

هستند. هنگامی که کد منبع در اختیار نباشد کار اندکی دشوار می‌شود. در این موارد، بایستی از ابزارهای مهندسی معکوس و ابزارگذاری کد باینری استفاده کرد. DynamoRIO [۱] [۳۱]، Pin [۲] [۳۲]، PaiMei [۳] [۱۴] و QEMU [۴] [۳۳] ابزارهایی برای این منظور فراهم کرده‌اند.

۲ـ۲ آزمون فازی

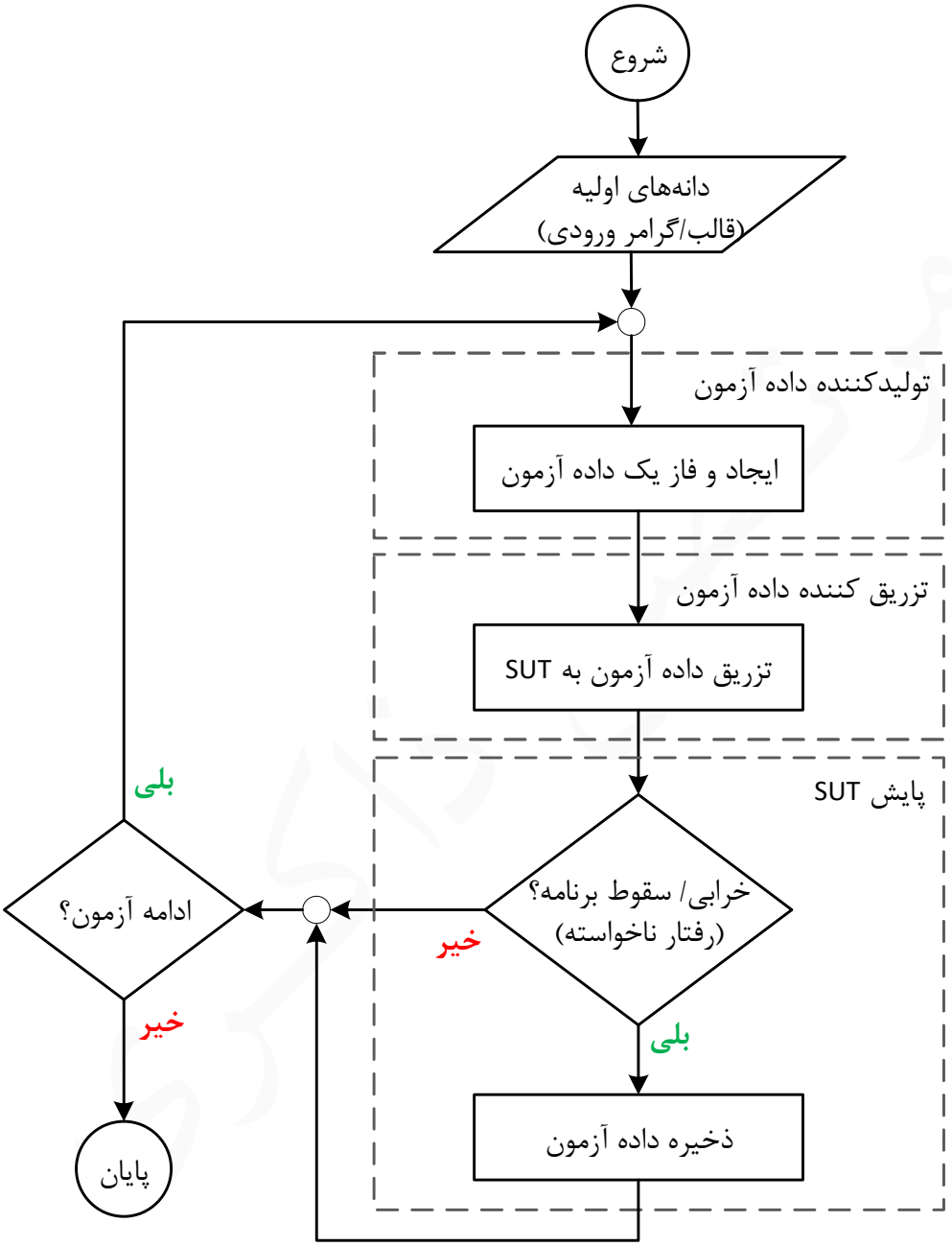
آزمون فازی [۳–۶] همان‌طورکه قبلاً هم گفتیم، فرایند ساده تولید و سپس تزریق یک ورودی ناخواسته (بدشکل شده یا نامتعارف) به SUT است. چنان‌چه برنامه بر اثر پردازش این ورودی ناخواسته دچار خرابی شود، حافظه برنامه مورد تحلیل قرار گرفته و خطای احتمالی موجود در کد آشکار می‌گردد. به دلیل اینکه SUT هنگام آزمون اجرا می‌شود، آزمون فازی، نوعی آزمون پویا است. چون SUT با تعداد ورودی‌های بسیار زیادی مورد آزمون قرار می‌گیرد، آزمون فازی را می‌توان نوعی‌آزمون فشار[۵] هم به‌شمار آورد. فرایند معمول آزمون فازی در شکل ۲ـ۲ نشان داده شده است.

۲ـ۲ـ۱ فازر

فازر ابزاری است که فرایند آزمون فازی را خودکار می‌کند. پیاده‌سازی هریک از پیمانه[۶]های شکل ۲ـ۲ و تجمیع آنها در کنار یکدیگر یک فازر را ایجاد می‌کند. تمرکز اصلی در یک فازر، نحوه تولید ورودی جدید به عنوان داده آزمون است به‌گونه‌ای که می‌توان آن را وجه تمایز اصلی فازرهای مختلف دانست. روش‌های تولید داده در فازرها قابل تفکیک به دو دسته کلی روش‌های مبتنی بر جابه‌جایی[۷] یا جهش و روش‌های مبتنی بر تولید[۸] هستند [۲۴].

در روش مبتنی بر جابه‌جایی، تعداد یک یا بیشتر داده ورودی معتبر[۹] به‌عنوان دانه اولیه[۱۰] برای تولید داده‌های آزمون بیشتر به‌کار می‌رود. دانه اولیه جهش (تغییر ناگهانی) می‌یابد تا داده آزمون دیگری تولید شود.

[1] http://www.dynamorio.org/
[2] https://software.intel.com/en-us/articles/pin-a-dynamic-binary-instrumentation-tool
[3] https://github.com/OpenRCE/paimei
[4] https://www.qemu.org/
[5] Stress Testing
[6] Module
[7] Mutation Based
[8] Generation Based
[9] Valid
[10] Initial Seed



شکل ۲ ـ ۲ : روندنمای فرایند آزمون فازی در حالت ساده که با اقتباس از مراجع مختلف ترسیم شده است. پیمانه‌های مورد نیاز برای خودکارسازی فرایند با مستطیل خطچین مشخص شده‌اند. شرایط ادامه آزمون می‌تواند بر عهده فرد آزمون‌گر قرار داده شود یا توسط خود فازر تعیین گردد.

جهش می‌تواند ساده باشد؛ شبیه معکوس کردن یک بیت، جایگزین کردن یک کاراکتر، یا پیچیده‌تر باشد؛ شبیه شناسایی و تکرار ساختارهای مشخص در داده، جایگزینی اعداد صحیح و حقیقی با مقادیر مرزی (بسیار بزرگ یا کوچک) و غیره. ساختن یک فازر مبتنی بر جابه‌جایی و تولید ورودی بد شکل و مخرب با آن، آسان است و نیاز به شناخت قبلی ساختار داده ورودی مورد جهش ندارد. عیب روش مبتنی بر جابه‌جایی اینجاست که این روش وابسته به تنوع ورودی‌های نمونه است. بدون وجود ورودی‌های مختلف با پیچیدگی کافی، این نوع فارز به پوشش کد بالایی دست نمی‌یابد [۱۲]. به عبارت دیگر دانه اولیه در فازر مبتنی بر جابه‌جایی بسیار حائز اهمیت است. AFL[1] [۱۹]، FileFuzz[2] [۲] و Radamsa[3] [۱۲] نمونه‌هایی از فـازرهای مبتنی بر جابه‌جایی هستند.

روش مبتنی بر تولید، داده‌های آزمون را به‌صورت کاملاً تصادفی یا از روی یک توصیف صوری مانند گرامر، قالب، یا مدل تولید می‌کند. در حالت دوم، از مشخصه‌های داده ورودی، برای ساخت یک مدل مولد[4] استفاده می‌شود. این روش بیشتر روی قالب‌های داده‌ای که مستنداتی از مشخه‌های آنها در دسترس است، به‌کار می‌رود که در مقایسه با فازرهای مبتنی بر جابه‌جایی، معمولاً به پوشش کد بالاتری دست می‌یابد. با این حال، همان‌طور که در فصل ۱ هم اشاره شد، زمان و هزینه زیادی باید صرف شود تا مشخصه‌های یک قالب داده، کامل فهمیده و مدل خوبی از آن تهیه شود؛ زیرا، این کار تمام خودکار نیست [۳۴] و از طرفی، مستندات ساختار ورودی همواره در دسترس آزمون‌گر، نیستند. SPIKEfile[5] [۲] و Peach[6] نمونه‌ای از فازرهای مبتنی بر تولید هستند. روش‌های ترکیبی نیز وجود دارد که از ویژگی‌های هر دو روش بیان شده در تولید مورد آزمون کمک می‌گیرد. یک مثال از فازرهای ترکیبی LangFuzz [۳۵] است.

۲ـ۲ـ۲ معماری فازرها

یک معماری مرجع برای فازرها وجود ندارد؛ ولی می‌توان از روی فرایند ترسیم شده برای آزمون فازی (شکل ۲ـ۲)، پیمانه‌های اصلی لازم برای یک فازر را پیشنهاد کرد. پیمانه اول تولید کننده داده آزمون‌است که داده‌های ورودی لازم برای آزمون را تولید می‌کند. دیدیم که روش‌های مختلفی برای تولید داده آزمون وجود دارند. پیمانه دوم تزریق کننده داده آزمون است که وظیفه آن تحویل داده‌های تولید شده توسط تولیدکننده به

[1] American Fuzzy Lop (http://lcamtuf.coredump.cx/afl/)
[2] http://www.fuzzing.org/
[3] https://gitlab.com/akihe/radamsa
[4] Generative Model
[5] http://fuzzing.org/
[6] https://www.peach.tech/

SUT است. هر برنامه واسط مختص به خود را دارد. واسط می‌تواند CLI، گرافیکی (GUI[1])، پروتکل شبکه یا فایل باشد. هدف این پایان‌نامه برنامه‌هایی هستند که ورودی آنها فایل است. معمولاً داده‌های ورودی به‌شکل فایل (مثل PDF) ساختار پیچیده‌تری نسبت به داده‌های CLI و پروتکل‌های شبکه دارند و تولید آنها سخت‌تر است. فازری که برای آزمون این برنامه‌ها توسعه داده می‌شود، فازر قالب فایل[2] هم نامیده می‌شود [۲].

آخرین پیمانه مورد نیاز ابزار یا محیط پایش است که SUT را پایش می‌کند تا در صورت بروز خرابی ضمن ذخیره حالت برنامه، محل وقوع خرابی را بتواند با تحلیل حافظه برنامه مشخص کند. البته ابزارهای مستقل زیادی برای این منظور وجود دارند که می‌توان از آنها بهره گرفت. Application Verifier[3] [۳۶] یک ابزار پایشِ قابل استفاده در سیستم عامل ویندوز است. ممکن است فازر، یک بازخورد از ابزار پایش برای تولید داده آزمون بعدی دریافت کند. این بازخورد عموماً حاوی اطلاعات مربوط به پوشش کد اجرای برنامه است. بر این اساس فازرها به دو دسته دارای حلقه بازخورد و بدون حلقه بازخورد تقسیم‌بندی می‌شوند [۲۴]. AFL [۱۹] یک فازر دارای حلقه بازخورد است. در این پایان‌نامه یک فازر ساده قالب فایل بدون حلقه بازخورد ارائه می‌دهیم. تمرکز برروی نحوه تولید داده آزمون به روش مبتنی بر تولید / گرامر است.

۲ـ۲ـ۳ آسیب‌پذیری

آزمون فازی می‌تواند برای اهداف مختلفی استفاده شود، اما مهمترین هدف آن تحلیل نرم‌افزار به منظور کشف آسیب‌پذیری‌ها است. یعنی می‌توان آن را نوعی آزمون نفوذ[4]، آزمون قدرت‌مندی[5] و آزمون امنیت[6] هم تعریف کرد. تعاریف گوناگونی برای اصطلاح آسیب‌پذیری نرم‌افزار وجود دارد. آسیب‌پذیری را می‌توان اجتماع سه عنصر دانست: وجود یک خطا در نرم‌افزار، دسترسی مهاجم به خطا و قابلیت مهاجم برای بهره‌برداری[7] از خطا [۲۴، ۳۷]. چنانچه پیش از این اشاره شد، خطا، نرم‌افزار را در یک حالت اشکال قرار می‌دهد که منجربه خرابی می‌شود. بعضی خطاها ممکن است تنها منجربه خرابی نرم‌افزار و بروز رفتار ناخواسته (مغایر با مشخصه‌ها) شوند. بنابراین صرف وجود یک خطا در نرم‌افزار آسیب‌پذیری محسوب نمی‌گردد [۲، ۱۴].

انواع مختلفی از خطاها وجود دارند که می‌توانند توسط فازرها کشف شوند؛ به‌ویژه آنهایی که از جنس نقض دسترسی به حافظه هستند. خطاهای فساد حافظه بدون شک رایج‌ترین و مؤثرترین روش بهره‌برداری

[1] Graphical User Interface
[2] File Format Fuzzer
[3] https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/application-verifier
[4] Penetration Testing
[5] Robustness Testing
[6] Security Testing
[7] Exploit

خرابکارانه از یک سیستم کامپیوتری محلی یا راه دور هستند. اگر حافظه بتواند به روشی خراب شود (یک آدرس بازگشت، یک اشاره‌گر پشته، یک اشاره‌گر تابع و غیره) اغلب اجرا می‌تواند به کد تهیه شده توسط مهاجم هدایت شود. بیشترین آسیب‌پذیری‌هایی که معمولاً توسط محققین حوزه امنیت کشف می‌شوند، مرتبط با میانگیر[۱] (یک ناحیه ثابت از حافظه اصلی در اختیار برنامه) هستند؛ از جمله سرریز میانگیر، سرریز پشته، سرریز عدد صحیح و غیره [۱]. در آزمون فازی به‌دنبال یافتن چنین خطاهایی از طریق تزریق ورودی بدشکل و خرابی حافظه SUT هستیم. مطالعه و طبقه‌بندی کاملی از انواع آسیب‌پذیری‌ها در [۲۶] انجام شده است.

۲ـ۳ یادگیری ژرف

یادگیری ژرف مجموعه‌ای از فنون یادگیری ماشینی است که در آن بردار ویژگی‌ها به‌صورت خودکار از داده‌های خام استخراج می‌گردد. ابزار اصلی و غالب در این فنون از یادگیری، شبکه‌های عصبی[۲] هستند. در حالت ساده شبکه‌های عصبی به مسئله یک تابع را نسبت داده و آن را مدل می‌کند. پارامترهای این تابع سپس در فرایند آموزش مدل، تخمین زده می‌شوند. در یادگیری ژرف تعداد این پارامترها صدها، هزارها و گاهی میلیون‌ها برابر افزایش می‌یابد که در نتیجه قدرت بازنمایی[۳] و حل مسئله مدل، افزایش خواهد یافت. شبکه عصبی در این حالت یک گراف با ژرفای بیش از یک یال، خواهد بود که به آن شبکه عصبی ژرف هم می‌گویند. شبکه عصبی ژرف بدین ترتیب قادر به استخراج و تمییز جزئی‌ترین ویژگی‌های مسئله و در نتیجه بهبود دقت حاصله، خواهد بود [۳۸].

وجود تعداد بسیاری پارامتر در مدل‌های یادگیری ژرف، سبب می‌شود که آموزش این مدل‌ها از لحاظ محاسباتی سنگین و زمان‌بر باشد. بسیاری از موانع تئوری و عملی آموزش این مدل‌ها به‌تازگی رفع شده‌اند. برای مسائل مختلف، مدل‌های متفاوتی با شبکه‌های عصبی ژرف طراحی و ساخته می‌شوند. شبکه عصبی روبه‌جلو[۴] ساده‌ترین نوع است. شبکه عصبی پیچشی (CNN[5]) مناسب وظایف بینایی ماشین و شبکه عصبی مکرر (RNN) مناسب وظایف پردازش زبان طبیعی (NLP) (وظایف مبتنی بر توالی) ایجاد شده‌اند. وجه مشترک همه شبکه‌های نام‌برده، بلوک سازنده آن یعنی عصب[۶] است [۳۹].

[1] Buffer
[2] Neural Network
[3] Representation
[4] Feed-forward Neural Network
[5] Convolutional Neural Network
[6] Neuron

۲ـ۳ـ۱ عصب

بلوک محاسباتی پایه در بستر شبکه‌های عصبی، عصب یا واحد[1] است. یک عصب تعدادی عدد حقیقی را به‌عنوان ورودی گرفته، محاسباتی روی آنها انجام داده و یک خروجی تولید می‌کند. محاسبه بدین صورت است که عصب جمع وزن‌دار ورودی‌هایی که به آن وارد می‌شوند را به همراه یک مقدار اضافی به عنوان بایاس[2] حساب می‌کند. سپس برای جلوگیری از محدود شدن فضای توابع قابل تخمین به فضایی خطی، یک تابع غیرخطی، موسوم به تابع انگیزش[3] (تابع فعّالیت)، روی حاصل جمع اعمال می‌شود. توابع انگیزش متداول عبارتند از sigmoid و tanh. شکل ۲ـ۳ ساختمان داخلی یک عصب تنها با ورودی‌های x_1، x_2 و x_3، متقابلاً وزن‌های w_1، w_2 و w_3 و مقدار بایاس b را نشان می‌دهد. محاسبات این عصب مطابق رابطه‌های ۲ـ۱ و ۲ـ۲ است.

$$z = b + \sum_i w_i x_i \qquad (۲ـ۱)$$

$$y = \sigma(z) \qquad (۲ـ۲)$$

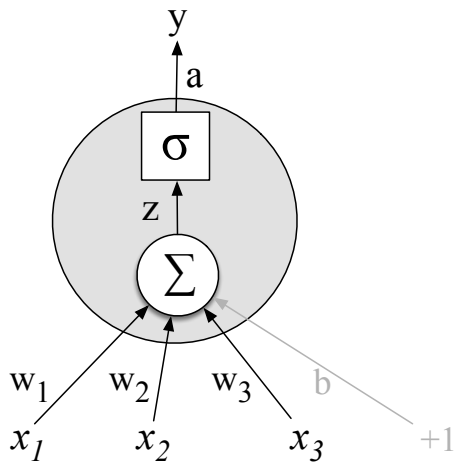
رابطه ۲ـ۱ را می‌توان به صورت ضرب داخلی بردارهای W و x هم نشان داد که نمایش مرسوم‌تری است [۳۹]:

$$z = b + W.x \qquad (۲ـ۳)$$

۲ـ۳ـ۲ شبکه عصبی روبه‌جلو

شبکه روبه‌جلو را می‌توان با یک گراف جهت‌دار بدون دور (DAG[4]) نشان داد. هر گره یک عصب را نشان می‌دهد. گره‌هایی که مسیری به یکدیگر ندارند، یک لایه را تشکیل می‌دهند. ورودی هر لایه، خروجی عصب‌های لایه قبلی است. هر لایه بازنمایی یک مفهوم[5] در قالب یک تابع است و مسئله ترکیبی از این مفاهیم

[1] Unit
[2] Bias
[3] Activation Function
[4] Directed Acyclic Graph
[5] Concept



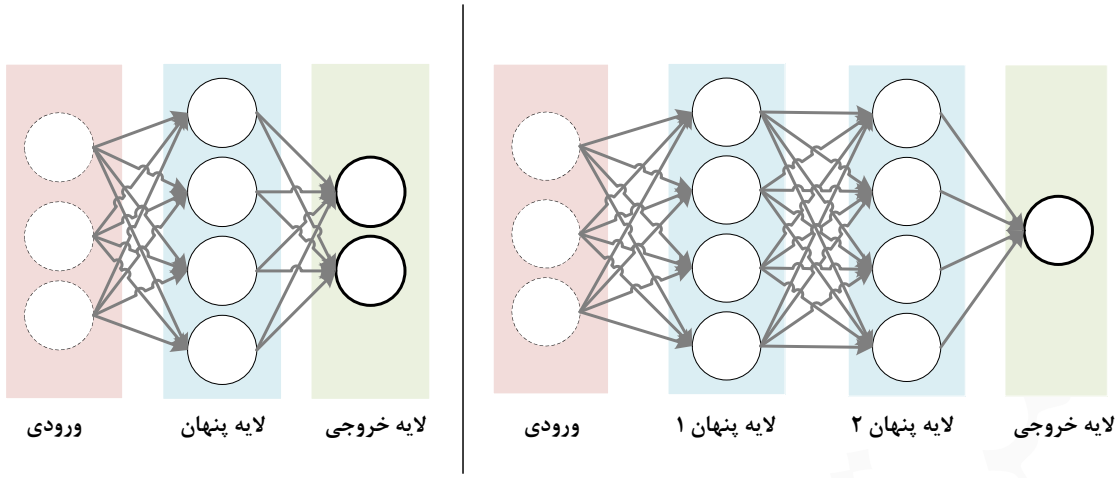
شکل ۲ـ۳: ساختمان داخلی یک عصب با ورودی‌های x_1، x_2 و x_3، به‌ترتیب با وزن‌های w_1، w_2 و w_3، مقدار بایاس b و خروجی y [۳۹].

است. یک شبکه با دو لایه تابع $f(x) = W_2\sigma(W_1 x)$ را پیاده می‌کند که W_1 و W_2 ماتریس پارامترهای تابع و σ تابع انگیزش است. $f(x) = W_3\sigma(W_2\sigma(W_1 x))$ معرف یک شبکه سه‌لایه خواهد بود. شکل ۲ـ۴ دو گراف محاسباتی با ژرفای ۲ و ۳ لایه (به ترتیب از چپ به راست) را نشان می‌دهد. لایه‌های میانی را لایه‌های پنهان[1] هم می‌گویند. توجه شود که ورودی اول شبکه یک لایه محاسباتی نیست و بنابراین در شمارش لایه‌ها، شمرده نمی‌شود.

۲ـ۳ـ۳ آموزش شبکه روبه‌جلو

هدف از آموزش شبکه، تعیین مقادیر مناسب برای ضرایب و بایاس‌ها و سپس استفاده از شبکه برای انجام وظیفه[2] مورد نظر است. داده‌هایی که در فرایند آموزش شبکه استفاده می‌گردد مجموعه آموزش[3] نامیده می‌شوند. داده‌هایی که شبکه با آن مورد سنجش قرار می‌گیرد، مجموعه آزمون[4] نامیده می‌شوند و مجموعه سوم از داده‌ها که در طول فرایند آموزش شبکه برای انتخاب بهترین راهکارهای یادگیری به‌کار گرفته می‌شوند، به نام مجموعه ارزیابی[5] شناخته می‌شوند [۳۸].

[1] Hidden Layer
[2] Task
[3] Training Set
[4] Test Set
[5] Validation Set



شکل ۲ ـ ۴: گراف محاسباتی برای دو شبکه عصبی روبه‌جلوی فرضی. چپ: یک شبکه عصبی دو لایه (یک لایه پنهان متشکل از چهار عصب و یک لایه خروجی متشکل از دو عصب) به‌همراه سه ورودی. راست: یک شبکه عصبی سه‌لایه (دو لایه پنهان هر کدام متشکل از چهار عصب و یک لایه خروجی متشکل از یک عصب) و به‌همراه سه ورودی.

شبکه عصبی در یک راهبرد بانظارت[۱] آموزش می‌بیند. در یادگیری بانظارت[۲] هر نمونه در مجموعه داده، علاوه بر بردار ویژگی‌ها با یک برچسب[۳] همراه است که نقش ناظر را ایفا می‌کند. طبقه‌بندی[۴] نمونه‌ای از وظایف یادگیری بانظارت است. در این وظیفه برای هر ورودی، کلاس خروجی آن در زمان آموزش مشخص است.

در فرایند آموزش ابتدا پارامترهای شبکه مقداردهی اولیه می‌شوند (معمولاً به‌صورت تصادفی). سپس به ازای هر ورودی از مجموعه آموزش، عصب‌های هر لایه خروجی خود را تولید می‌کنند تا زمانی که خروجی نهایی شبکه تولید شود. این مرحله را گذرِجلو[۵] می‌گویند. از آنجایی که خروجی درست برای ورودی مربوطه در زمان آموزش دردسترس است (یادگیری بانظارت)؛ اختلاف مقدار محاسبه شده توسط شبکه ($\hat{y}$) و مقدار واقعی (y)، توسط تابعی موسوم به تابع خطا[۶] یا تابع هزینه[۷] یا تابع هدف محاسبه می‌شود [۴۰]. توابع مختلفی برای این منظور می‌توان تعریف کرد. تابع خطای مطلق میانگین[۸] (رابطه ۲ ـ ۴)، تابع خطای میانگین

[1] Supervised
[2] Supervised Learning
[3] Label
[4] Classification
[5] Forward Pass
[6] Error Function
[7] Cost Function
[8] Mean Absolute Error

مربعات[1] (رابطه ۲ـ۵) و تابع خطای آنتروپی متقاطع[2] (رابطه ۲ـ۶) توابعی هستند که بسته به وظیفه مورد نظر استفاده می‌شوند [۴۱].

$$L_{MAE}(\hat{y}, y; W, b) = \frac{1}{n}\sum_{i=1}^{n} |\hat{y}_i - y_i| \qquad (۲ـ۴)$$

$$L_{MSE}(\hat{y}, y; W, b) = \frac{1}{n}\sum_{i=1}^{n} (\hat{y}_i - y_i)^2 \qquad (۲ـ۵)$$

$$L_{CE}(\hat{y}, y; W, b) = -\sum_{i=1}^{n} y_i \log \hat{y}_i \qquad (۲ـ۶)$$

در روابط ۲ـ۴ تا ۲ـ۶، n تعداد نمونه‌ها یا تعداد مشاهده‌ها است. در این پایان‌نامه از تابع خطای آنتروپی متقاطع استفاده می‌کنیم که برای وظیفه‌های طبقه‌بندی با بیش از دو کلاس مناسب است [۴۰]. وقتی میزان خطا در گذرجلو مشخص شد، هدف تغییر میزان پارامترهای شبکه به‌نحوی است که خطا کمینه شود (رابطه ۲ـ۷):

$$f^* = arg\min_{W,b} L(\hat{y}, y; W, b) \qquad (۲ـ۷)$$

که f^* تابع یادگیری شده و مقادیر W و b پارامترهای شبکه هستند. سایر مقادیر را که در طول این فرایند یادگیری نمی شوند، ابَر پارامتر[3] می‌نامند. از جمله ابر پارامترها می‌توان به تعداد لایه‌ها و تعداد عصب‌ها در هر لایه اشاره کرد، که از آنها برای تعیین اندازه شبکه استفاده می‌گردد.

برای کمینه‌سازی خطا، مشتقات جزئی تابع خطا نسبت به هریک از پارامترها در لایه خروجی محاسبه می‌شود (گرادیان) و سپس این روند از طریق روال پس‌انتشار[4] تا ابتدای شبکه پیش می‌رود و مقدار هر پارامتر درسوی کاهش گرادیان با ضریبی موسوم به نرخ یادگیری[5] بروزرسانی می‌شود. نرخ یادگیری عدد کوچکی (در مقیاس صدم و هزارم) در نظر گرفته می‌شود که برای همگرایی آهسته و پیوسته شبکه و جلوگیری از

[1] Mean Squared Error
[2] Cross Entropy Error
[3] Hyper-parameter
[4] Backpropagation
[5] Learning Rate

واگرا شدن آن لازم است. نرخ یادگیری یکی دیگر از ابَر پارامترهای مورد نیاز آموزش شبکه‌های ژرف است. الگوریتم پس‌انتشار در واقع یک کاربرد بازگشتی از قاعده مشتق زنجیری در حساب دیفرانسیل است [۴۰]. در اینجا به بیان جزئیات الگوریتم‌های بهینه‌سازی نمی‌پردازیم؛ چارچوب‌های یادگیری ژرف انواع مختلفی از این الگوریتم‌ها را پیاده‌سازی و در اختیار قرار داده‌اند. توضیحات کاملی از این الگوریتم‌ها در [۳۸] آمده است.

شرایط توقف آموزش

فرایند آموزش شبکه می‌تواند تا بی‌نهایت ادامه داشته باشد. بنابراین بایستی یک قانون برای توقف آن تعریف شود. گزینه‌های زیادی وجود دارد که تحت چه شرایطی آموزش را خاتمه دهیم. همچنین ترکیب این شرایط نیز امکان‌پذیر است. برخی از این شرایط عبارتند از [۴۱]:

۱. هنگامی که تعداد مشخصی دوره[1] طی شود (هر دوره برابر تعداد تکرارهایی است که برای مرور کل مجموعه آموزش لازم است).

۲. هنگامی که خطای مجموعه ارزیابی (برای تعداد مشخصی دوره متوالی) کاهش پیدا نکند.

۳. هنگامی که تغییرات میزان خطا (برای تعداد مشخصی دوره متوالی) کمتر از یک حد آستانه شود.

۴. هنگامی که یک زمان مشخص از فرایند آموزش سپری شود.

منظم‌سازی

مشکل رایجی که هنگام آموزش شبکه عصبی باید جلوگیری شود اثر بیش‌برازش[2] است. بیش‌برازش یعنی با کاهش خطا روی مجموعه آموزش، خطای مجموعه‌های ارزیابی و آزمون به‌طور ناگهانی افزایش یابد [۳۸]. یک دلیل می‌تواند به‌اندازه کافی بزرگ نبودن اندازه مجموعه آموزش باشد. دلیل دیگر می‌تواند پیچیدگی بسیار بالای مدل باشد. پیچیدگی مدل با تعداد پارامترهای آن مشخص می‌شود. در پیچیدگی بالا ممکن است مدل، اختلال[3]های موجود در ورودی‌ها را نیز لحاظ کند و متناسب یا برازنده آن شود. در نتیجه هنگام ارزیابی مدل روی داده‌های مجموعه آزمون، خطا بالا می‌رود [۴۱].

راهکارهای مختلفی برای مقابله با مسئله بیش‌برازش مدل پیشنهاد شده است. از این راهکارها تحت عنوان فنون منظم‌سازی[4] یا تنظیم مدل یاد می‌شود. اغلبِ روش‌های شناخته شده، پارامترهای دارای مقادیر بالا را

[1] Epoch
[2] Overfitting
[3] Noise
[4] Regularization

که اثرات نوسانی شدیدی دارند، با تابعی جریمه می‌کنند [۴۰]. روش دیگر برای منظم‌سازی Dropout [۴۲] است که بسیار مؤثر و ساده بوده و در مدل‌های یادگیری ژرف معمولاً از آن استفاده می‌شود. در طول آموزش Dropout یک عصب را فقط با احتمال p (یک ابر پارامتر) فعال نگه می‌دارد و در غیر این صورت آن را صفر می‌کند. در نتیجه پیچیدگی مدل تنظیم می‌شود.

۲ـ۳ـ۴ شبکه عصبی مکرر

شبکه‌های روبه‌جلو در حل مسائلی که ترتیب ورودی در آنها مهم است، مثل ترجمه ماشینی و سایر وظیفه‌های NLP، نارسایی دارند. برای مثال در ترجمه ماشینی هر واژه به واژه‌های قبلی خود وابسته است. در این وظیفه‌ها ورودی به‌صورت یک توالی $x =< x^{(1)}, x^{(2)}, x^{(3)}, ..., x^{(n)} >$ است. غالب وظایف حوزه پردازش زبان طبیعی بدین صورت هستند.

شبکه‌های عصبی مکرر کلاسی از شبکه‌های عصبی هستند که به‌صورت یک گراف جهت‌دار دارای دور بیان می‌شوند. به‌عبارت دیگر ورودی هریک از لایه(های) پنهان یا لایه خروجی افزون‌بر خروجی لایه قبل، شامل ورودی‌ای از گام زمانی[1] قبل به‌صورت بازخورد نیز می‌شود. شکل ۲ـ۵ یک RNN را نشان می‌دهد که در آن لایه پنهان از گام‌های زمانی قبلی بازخورد گرفته است. در هر گام زمانی t (از $t = 1$ تا $t = T$) یک بردار $x^{(t)}$ از توالی ورودی پردازش می‌شود. معادلات گذرجلو شبکه در t عبارتند از [۳۸]:

$$z^{(t)} = Ux^{(t)} + Wh^{(t-1)} + b \qquad (۸ـ۲)$$

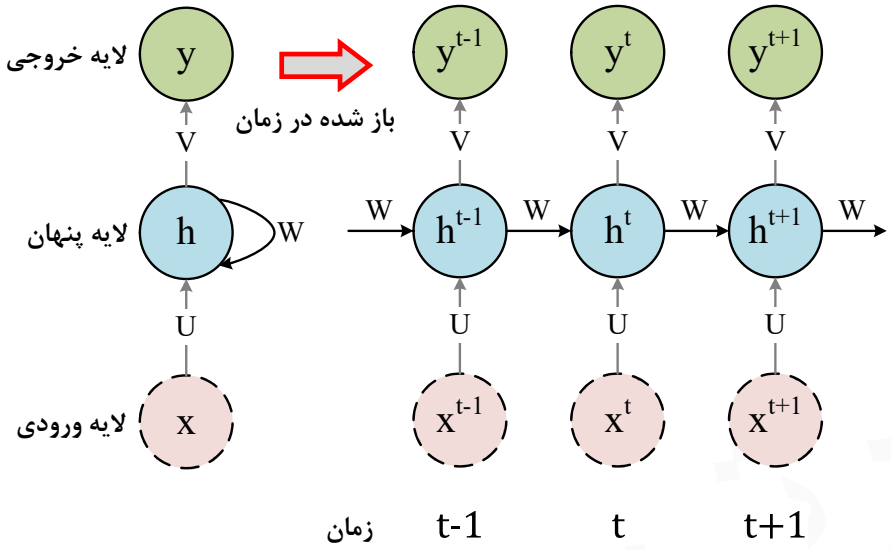
$$h^{(t)} = \sigma(z^{(t)}) \qquad (۹ـ۲)$$

$$y^{(t)} = Vh^{(t)} + c \qquad (۱۰ـ۲)$$

$$\hat{y}^{(t)} = softmax(y^{(t)}) \qquad (۱۱ـ۲)$$

در روابط ۲ـ۸ تا ۲ـ۱۱، b و c بایاس و ماتریس‌های U، V و W به‌ترتیب وزن یال‌های لایه ورودی

[1] Time Step



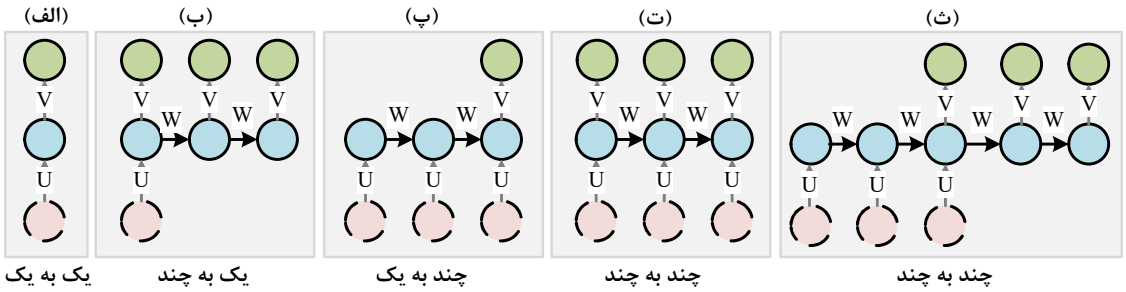
شکل ۲ـ۵: گراف محاسباتی مربوط به یک شبکه عصبی مکرر با یک لایه پنهان که یک توالی ورودی از مقادیر $x =< x^{(1)}, x^{(2)}, x^{(3)}, ..., x^{(n)} >$ را به یک توالی خروجی از مقادیر $y =< y^{(1)}, y^{(2)}, y^{(3)}, ..., y^{(n)} >$ نگاشت می‌کند. فرض شده است که خروجی y احتمال‌های نرمال نشده است، بنابراین خروجی نهایی شبکه یعنی $\hat{y}$ از اعمال تابع بیشینه هموار روی y حاصل می‌شود. چپ: شبکه عصبی مکرر به‌صورت یال بازگشتی (گراف جهت‌دار دارای دور). راست: همان شبکه به‌صورت باز شده در زمان، به نحوی که هر گره با برچسب مرحله زمانی مشخص شده است [۳۸].

به پنهان، پنهان به خروجی و پنهان به پنهان، پارامترهای شبکه هستند. در لایه خروجی به‌جای تابع انگیزش، تابع بیشینه هموار[1] اعمال می‌شود. این تابع خروجی شبکه را به‌شکل یک توزیع احتمالی معتبر تبدیل می‌نماید و معمولاً در لایه خروجی مدل‌هایی که برای طبقه‌بندی استفاده می‌شوند، قرار می‌گیرد. تابع بیشینه هموار یک بردار kتایی از اعداد حقیقی را به عنوان ورودی دریافت نموده و یک بردار kتایی از مقادیر حقیقی در بازه $[۰, ۱]$ را به عنوان خروجی می‌دهد به‌طوری که جمع مؤلفه‌های آن برابر یک خواهد بود (رابطه ۲ـ۱۲):

$$softmax(x_i) = \frac{e^{x_i}}{\sum_{j=.}^{k} e^{x_j}} \quad for\ i=1\ to\ k \qquad (۱۲ـ۲)$$

در شکل ۲ـ۵، RNN با یک لایه پنهان نشان داده شده است. اما می‌توان RNNژرف با چندین لایه پنهان نیز داشت. همچنین طول توالی‌های ورودی و خروجی می‌تواند بسته به مسئله مورد نظر متفاوت باشد.

[1] Softmax



شکل ۲ـ۶: طرح‌واره‌ای از انواع حالت‌های مختلف شبکه عصبی مکرر براساس طول توالی ورودی و طول توالی خروجی. (الف): شبکه عصبی استاندارد، (ب): شبکه یک به چند، (پ): شبکه چند به یک، (ت) و (ث): شبکه‌های چند به چند [۴۰].

کارپتی[۱] [۴۰] RNNها را از منظر طول توالی ورودی و طول توالی خروجی به چند دسته تقسیم‌بندی کرده است. شکل ۲ـ۶ این دسته‌بندی را نشان می‌دهد.

آموزش شبکه عصبی مکرر

الگوریتم پس‌انتشار برای آموزش RNN هم قابل استفاده است. در واقع RNN همان شبکه عصبی روبه‌جلو است که یک بازخورد از وزن‌های گام زمانی قبل دارد. در نتیجه خطا هنوز به‌صورت روی‌به‌عقب از گام زمانی t منتشر می‌شود. بسته به طول توالی ورودی و میزان منابع محاسباتی این انتشار تا گام زمانی $t = ۱$ ادامه می‌یابد یا تحت محدودیت تعداد مشخصی گام[۲] متوقف می‌شود. نسخه‌ای از الگوریتم پس‌انتشار که می‌تواند به‌کل طول توالی ورودی اعمال شود، پس‌انتشار در زمان (BPTT[3]) نام دارد. محدودیت منابع محاسباتی ایجاب می‌نماید که طول توالی ورودی مقدار مشخصی قرار داده شود یا پس‌انتشار در تعداد محدودی گام‌زمانی متوقف شود [۳۸].

حافظه کوتاه‌مدت بلند

RNN در حالت استاندارد، برای توالی‌های طولانی که وابستگی بین ورودی‌هایی با فاصله زیاد وجود دارد، قادر به حفظ این وابستگی‌ها نیست؛ به‌بیان دیگر گرادیان در دوره‌های زمانی بلند مدت تمایل به ناپدید شدن[۴] یا انفجار[۵] (زیاد شدن بی‌اندازه) دارد. در حالت ناپدید شدن گرادیان، یادگیری مدل متوقف می‌شود. مدل

[1] A. Karpathy (http://karpathy.github.io/)
[2] Step
[3] Back Propagation Through Time
[4] Vanishing
[5] Explosion

حافظه کوتاه‌مدت بلند (LSTM[1]) برای مقابله با مسئله بالا و افزایش قدرت به‌خاطرسپاری RNN ایجاد شده است. هسته این مدل سلول حافظه C است که در آن بر خلاف شبکه مکرر استاندارد که خروجی هر واحد تنها با اعمال یک تابع انگیزش ایجاد می‌شود، خروجی با محاسبات پیچیده‌تری تعیین می‌شود که به‌ویژه هدف آنها منظم‌سازی و حفظ قدرت گرادیان است. ایده اصلی در LSTM تخصیص مسیری مجزا علاوه بر یال بازخوردی حالت پنهان، برای جریان حافظه است.

جزئیات عملکرد LSTM در [۳۸] توضیح داده شده است. امروزه به‌طور پیشفرض از LSTM در پیاده‌سازی RNN استفاده می‌شود [۴۳]. البته راه‌کارهای جایگزین دیگری مشابه LSTM، نیز مطرح شده‌اند مانند GRU[2] [۱۸]، اما عملکرد آنها بهتر از LSTM نیست. مطالعه جامعی روی RNN و انواع آن برای یادگیری توالی‌ها در [۴۴] انجام شده است. کلیه مدل‌های یادگیری ژرف معرفی شده در این پایان‌نامه از LSTM در هسته خود استفاده می‌کنند.

۲ـ۴ مدل زبانی

در فصل ۱ به استفاده از LM در روش پیشنهادی خود، اشاره کردیم. LM یک مفهوم پایه در NLP است که امکان پیش‌بینی نشانه بعدی در یک توالی را فراهم می‌کند. به‌بیان دقیق‌تر LM عبارت است از یک توزیع احتمالی روی یک توالی از نشانه‌ها (اغلب واژه‌ها) که احتمال وقوع یک توالی داده شده را مشخص می‌کند. در نتیجه می‌توان بین چندین توالی داده شده برای مثال چند جمله، آن را که محتمل‌تر است، انتخاب کرد. LM برای توالی $x =< x^{(۱)}, x^{(۲)}, x^{(۳)}, ..., x^{(n)} >$ به‌صورت زیر تعریف می‌شود [۴۵]:

$$p(x) = \prod_{t=۱}^{n} p(x^{(t)}|x^{<t}) \qquad (۱۳ـ۲)$$

در رابطه ۲ـ۱۳ هر جمله منفرد $p(x^{(t)}|x^{<t})$ احتمال شرطی نشانه $x^{(t)}$ به شرط وقوع (ظاهر شدن) t نشانه قبلی $x^{<t}$ در توالی است که به آن زمینه[3] یا تاریخچه[4] نیز می‌گویند. در عمل محاسبه این احتمال به‌صورت رابطه ۲ـ۱۳ تقریباً غیر ممکن است؛ زیرا، نیازمند داشتن همه توالی‌های ممکن هستیم. مدل‌های سنتی n-gram برای غلبه بر چالش‌های محاسباتی، با استفاده از فرض مارکوف رابطه ۲ـ۱۳ را به درنظر گرفتن تنها $n-۱$ نشانه قبلی محدود می‌کنند. اگرچه در بسیاری از مسائل این مدل‌ها به خوبی پاسخ‌گو هستند؛

[1] Long-Short Term Memory
[2] Gated Recurrent Units
[3] Context
[4] History

اما، برای توالی‌های طولانی (بیشتر از ۴ یا ۵ نشانه) و نیز مشاهده نشده مناسب نیستند. در حالتی که نشانه فعلی پیش از این مشاهده نشده باشد، احتمال صفر به آن نسبت داده می‌شود که سبب صفر شدن احتمال پایانی می‌گردد. برای حل این مشکل، مدل‌های n-gram نیازمند اعمال فنون هموارسازی[1] هستند [۳۹]. این فنون، راه‌حل‌هایی برای از بین بردن احتمال‌های صفر پیشنهاد می‌دهند. در هر صورت، مدل‌های n-gram با محدودیت‌های جدی روبه‌رو هستند.

۲ـ۴ـ۱ مدل زبانی عصبی

می‌توان از RNN برای ایجاد مدل زبانی استفاده کرد. این مدل‌ها را مدل زبانی عصبی (NLM) می‌گویند. استفاده از RNN هر دو مشکل اشاره شده در بالا را برطرف می‌کند. یعنی هم امکان پیش‌بینی توالی‌های طولانی‌تر فراهم می‌شود و هم وقوع احتمالات صفر از بین می‌رود [۴۵]. شکل ۲ـ۷ یک معماری از RNN برای ساخت مدل زبانی در NLP را نشان می‌دهد. چون هدف مدل زبانی پیش‌بینی نشانه بعدی است، توالی x را در ورودی شبکه قرار داده و خروجی متناظر با آن را همان توالی x که یک واحد روبه‌جلو شیفت داده شده است، می‌گذارند. این روش آموزش شبکه Teacher Forcing می‌گویند [۳۸].

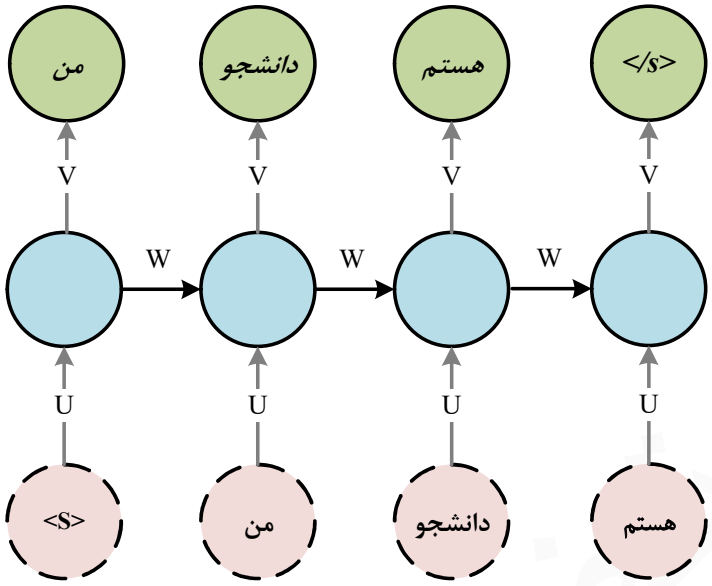
مدل‌های زبانی را مدل مولد نیز می‌نامند؛ زیرا یک توزیع احتمالی روی توالی‌های یک زبان می‌دهند که با نمونه‌برداری از آن می‌توان توالی‌های جدید تولید کرد. در NLP نشانه‌های هر توالی واژه‌ها هستند، اما می‌توان این مدل را در سطح کاراکتر نیز آموزش داد. ایده اصلی در این پایان‌نامه نیز همین است. هر فایل زبان و قواعد مخصوص به خود را دارد که در مشخصه‌های قالب فایل ذکر شده است. با ایجاد یک مدل زبانی برای این زبان در سطح کاراکتر، امکان پیش‌بینی نشانه بعدی از روی یک توالی از نشانه‌های داده شده فراهم می‌شود. با نمونه برداری از این مدل زبانی می‌توان داده‌های آزمون جدیدی ایجاد کرد.

۲ـ۴ـ۲ ارزیابی مدل زبانی

چگونه می‌توان میزان خوب بودن یک مدل زبانی را تعیین کرد؟ یک روش برای ارزیابی مدل زبانی، قرار دادن آن در یک وظیفه مشخص و اندازه‌گیری میزان دقت نتایج حاصله است. به این روش ارزیابی بیرونی[2] می‌گویند [۳۹]. برای مثال در یک وظیفه تصحیح خودکار غلط‌های املایی، تعداد واژه‌های نادرستی که به‌درستی با واژه‌های صحیح جایگزین شده‌اند را شمارش می‌کنیم و بر تعداد کل واژه‌های غلط تقسیم می‌نماییم. ارزیابی

[1] Smoothing

[2] Extrinsic Evaluation



شکل ۲ ـ ۷: طرح‌واره‌ای از معماری یک مدل زبانی ایجاد شده با استفاده از RNN. $< s >$ نشانه شروع توالی و $< /s >$ نشانه خاتمه توالی در ابتدا و انتهای هر توالی موجود در مجموعه آموزش قرار داده می‌شود. بدین ترتیب، تولید توالی جدید پس از پیش‌بینی نشانه پایان، متوقف می‌گردد [۴۵].

بیرونی، زمان‌بر، پرهزینه و مختص به وظیفه اجرا شده خواهد بود. روش دیگر استفاده از ارزیابی درونی[1] است [۳۹]. معیار سرگشتگی[2] با هدف ارزیابی درونی مدل زبانی، به‌صورت زیر تعریف می‌شود [۴۶]:

$$PP_{LM}(x) = \sqrt[n]{\prod_{i=1}^{n}(\frac{1}{p(x^{(i)}| < x^{(1)},...,x^{(i-1)} >)}} = 2^{-\frac{1}{n}\sum_{i=1}^{n}\log_2 p(x^{(i)}|<x^{(1)},...,x^{(i-1)}>)} \quad (۱۴ ـ ۲)$$

در رابطه ۲ ـ ۱۴، x توالی مورد ارزیابی است. همان‌طور که مشاهده می‌شود سرگشتگی در ارتباط مستقیم با خطای آنتروپی متقاطع (رابطه ۲ ـ ۶) است که میزان اختلاف بین مدل و نمونه‌های مجموعه آزمون را نشان می‌دهد. بنابراین هرچه‌قدر میزان سرگشتگی پایین‌تر باشد، مدل زبانی بهتر است. پایه توان و پایه لـگاریتم در رابطه ۲ ـ ۱۴ می‌تواند مقادیری به‌غیر از ۲ انتخاب شود. اما در هر حالت هر دو پایه بایستی یکسان باشند تا تساوی اول (بالا) و دوم (پایین) رابطه برقرار گردد. معمولاً از پایه لگاریتم طبیعی در این رابطه استفاده

[1] Intrinsic Evaluation
[2] Perplexity

می‌شود [۴۷].

برای درک بهتر نحوه ارزیابی توسط این معیار، توالی $x =< x^{(1)}, x^{(2)}, x^{(3)}, ..., x^{(n)} >$ را در نظر می‌گیریم که متشکل از V نشانه متفاوت است. V مجموعه واژگان[1] زبان است. در غیاب مدل زبانی (بدترین حالت)، تنها می‌توانیم ادعا کنیم که هر نشانه دست‌کم با احتمال $\frac{1}{V}$ در توالی رخ می‌دهد (ضریب انشعاب[2]). بدیهی است که این احتمال برای همه نشانه‌ها یکسان است. برای توالی x پس از جایگزینی احتمال آن در رابطه ۲ـ۱۴ داریم:

$$PP_{LM}(x) = \sqrt[n]{\prod_{i=1}^{n}(\frac{1}{\frac{1}{V}})} = \sqrt[n]{V^n} = V$$

بنابراین در بدترین حالت میزان سرگشتگی برابر با اندازه مجموعه واژگان زبان است. حال چنان‌چه از مدل زبانی استفاده کنیم، احتمال نسبت داده شده به وقوع هر نشانه، نسبت به حالت عدم استفاده از مدل زبانی، افزایش و در نتیجه سرگشتگی کاهش خواهد یافت. در ارزیابی مدل‌های این پایان‌نامه از معیار سرگشتگی استفاده می‌کنیم.

۲ـ۵ خلاصه

در این فصل مفاهیم اولیه سه حوزه کلی مطرح در این پایان‌نامه یعنی آزمون نرم‌افزار، آزمون فازی و یادگیری ژرف را بیان کردیم. آزمون نرم‌افزار به‌عنوان یک مسئله تصمیم‌ناپذیر تعریف گردید و معیارهای پوشش برای ارزیابی آن معرفی شدند. آزمون فازی به عنوان یک فن مطرح در آزمون نرم‌افزار، معرفی و از سه دیدگاه مختلف مورد طبقه‌بندی و بحث قرار گرفت: نخست، از دیدگاه اطلاعات در دسترس از SUT به سه دسته آزمون جعبه سیاه، جعبه سفید و جعبه خاکستری تقسیم گردید. دوم، از دیدگاه روش‌های تولید داده آزمون به سه دسته مبتنی بر جابه‌جایی، مبتنی بر تولید و نیز ترکیبی تقسیم‌بندی شد. سوم، با توجه به دریافت بازخورد از نتیجه اجرا به دو دسته دارای بازخورد و بدون بازخورد تفکیک گشت. ابزار خودکارسازی آزمون فازی تحت عنوان کلی فازر معرفی و معماری آن تشریح شد. فازرها با توجه به طبقه‌بندی‌های فوق در بحث آزمون فازی و نیز با توجه به نوع ورودی SUT متفاوت هستند. فازرهای قالب فایل مربوط به برنامه‌هایی با ورودی فایل هستند. برای اطلاعات بیشتر در زمینه آزمون نرم‌افزار خواننده را به [۱۳، ۲۷] ارجاع می‌دهیم. در زمینه آزمون فازی نیز خواننده را به [۲، ۲۴، ۲۹] رجوع می‌دهیم.

در بخش پایانی این فصل، یادگیری ژرف به عنوان زیرشاخه‌ای از یادگیری ماشینی با استفاده از شبکه‌های

[1] Vocabulary

[2] Branching Factor

عصبی مصنوعی، تفهیم شد. شبکه عصبی مورد استفاده در یادگیری ژرف را شبکه عصبی ژرف گویند. RNN نوع خاصی از شبکه‌های عصبی ژرف است که برای پردازش وظیفه‌های مبتنی بر توالی مثل مدل زبانی استفاده می‌شود. ساختار یک قالب فایل زبان مختص به خود را دارد و هدف از مطرح کردن اصول اولیه یادگیری ژرف استفاده از مدل‌های این فن، در یادگیری ساختار فایل، جهت تولید داده آزمون جدید است که در فصل ۴ به آن خواهیم پرداخت. الگوریتم‌های آموزش شبکه‌های عصبی ژرف ریاضیات گسترده‌ای دارد که به‌طور خلاصه به موارد مهم آن اشاره شد. جهت مطالعه مبسوط این الگوریتم‌ها خواننده را به [۳۸] ارجاع می‌دهیم. همچنین توضیح کاملی از روش‌های یادگیری توالی با استفاده از RNNها در [۴۴] آمده است. در فصل بعد برخی از تازه‌ترین کارهای مرتبط با مفاهیم مطرح شده در این فصل توضیح داده می‌شوند.

فصل ۳

کارهای مرتبط

«یک شب تاریک و طوفانی بود. پاییز ۱۹۹۴، در آپارتمان خود در مادیسون نشسته بودم. آن شب من از طریق خط تلفن به سیستم‌های یونیکس دانشگاه متصل شدم. با بارش سنگین باران اختلال زیادی روی خط اتصالی بود که در اجرای فرمان‌های ارسالی من دخالت می‌کرد. رقابتی بین سرعت نوشتن یک فرمان قبل از آنکه اختلال آن را خراب کند وجود داشت. چیزی که مرا شگفت زده می‌کرد این حقیقت بود که اختلال سبب ایجاد خرابی و سقوط برنامه‌ها می‌شد و شگفت‌آورتر برنامه‌هایی بود که سقوط می‌کرد: ابزارهای رایج یونیکس!»

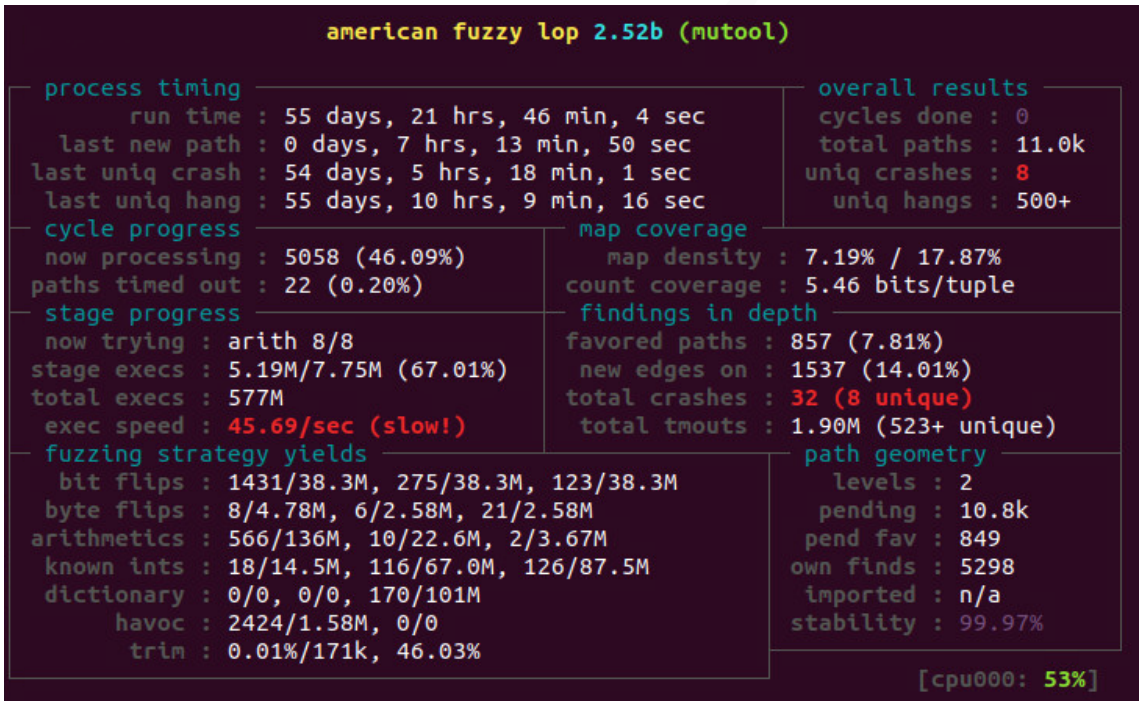
✠ بارتون میلر، مبدع آزمون فازی

کارهای بسیاری برای بهبود آزمون فازی اولیه که داده‌های آزمون را به‌صورت تصادفی تولید می‌کرد [۳]، انجام شده است. فازرهای مبتنی بر تولید در برنامه‌های با ساختار ورودی پیچیده پوشش کد بیشتری نسبت به فازرهای مبتنی بر جابه‌جایی فراهم می‌کنند [۳۴]، اما کاملاً خودکار نیستند [۱۱]. در مقابل فازرهای مبتنی بر جابه‌جایی سعی کرده‌اند تا با استفاده از الگوریتم های تکاملی مثل ژنتیک داده آزمون‌های بهتری را برای جابه‌جایی انتخاب کنند. AFL [۱۹] نمونه‌ای موفق از فازرهای مبتنی بر جابه‌جایی است. در سال ۲۰۱۷، فنون یادگیری ماشینی به هر دودسته از فازرهای بالا اعمال شده‌اند. در فازرهای مبتنی بر تولید، برای یادگیری خودکار گرامر فایل [۱۱] و در فازرهای مبتنی بر جابه‌جایی برای پیش‌بینی بهترین مکان جابه‌جایی [۲۰]. هریک از این کارها نواقص و محدودیت‌هایی دارند. در این فصل به معرفی، نقد و بررسی این کارها می‌پردازیم.

۳ـ۱ فازر AFL

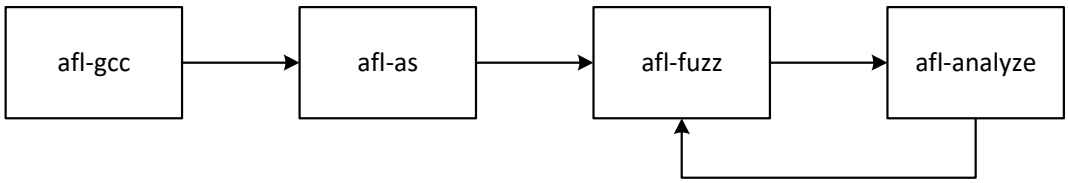
AFL [۱۹] یک فازر قالب فایلِ جعبه خاکستری، با تولید داده آزمون مبتنی بر جابه‌جایی، دارای بازخورد، متن‌باز و رایگان است که توسط Michal Zalewski[1] توسعه داده شده است. این فازر روی سیستم عامل‌های خانواده یونیکس قابل اجرا است. راه‌اندازی آن ساده بوده و واسط کاربری خوبی برای دنبال کردن جزئیات آماری فرایند آزمون فازی و خطاهای کشف شده دارد. در شکل ۳ـ۱ یک نمونه از اجرای این فازر را در عمل نشان داده‌ایم.

به‌طور پیش‌فرض AFL برای سنجش پوشش کد، به کد منبع SUT نیاز دارد. برنامه‌های نوشته شده به زبان‌های C، C++ و Objective-C در آزمون فازی جعبه سفید با AFL قابل استفاده هستند. همچنین نسخه‌هایی از AFL برای برنامه‌های نوشته شده به زبان‌های Go و Python توسط دیگران انتشار یافته است. AFL در آزمون فازی جعبه سیاه، برای ابزارگذاری و اخذ اطلاعات زمان اجرا، از ابزار QEMU [۳۳] استفاده می‌کند. این فازر به‌حدی موفق بوده که پژوهش‌های زیادی برای بهبود جنبه‌های مختلف آن انجام شده است. اما چنان‌چه خواهیم دید روی قالب فایل‌هایی با ساختار پیچیده نمی‌تواند به پوشش خوبی دست پیدا کند [۲۰].



شکل ۳ـ۱: اجرای AFL بر روی ابزار mutool از نرم‌افزار MuPDF [۲۵]. این تصویر جزئیات اجرا بعد از گذشت ۵۵ روز از آغاز فرایند آزمون فازی را نشان می‌دهد. آزمون تا زمانی که کاربر $ctrl+z$ را فشار ندهد، اجرا می‌شود.

[1] http://lcamtuf.coredump.cx/



شکل ۳ـ۲ : مؤلفه‌های فازر AFL و ارتباط آنها با یکدیگر [۱۹].

۳ـ۱ـ۱ معماری AFL

شکل ۳ـ۲ مؤلفه‌ها[1]ی اصلی فازر AFL و ترتیب استفاده از آنها در هنگام آزمون فازی را نشان می‌دهد. در این معماری چهار مؤلفه مشاهده می‌شود:

- **afl-gcc.** یک جایگزین برای gcc یا clang استاندارد است که برای کامپایل کد منبع SUT استفاده می‌شود و خروجی آن به afl-as ارسال می‌شود. در رویکرد جعبه سفید استفاده مرحله اول کامپایل SUT با afl-gcc است.
- **afl-as.** کد کامپایل شده با afl-gcc را با تزریق کدهای اسمبلی ابزارگذاری می‌کند. ابزارگذاری به نحوی است که پوشش انشعاب یا پرش را ضبط می‌کند. خروجی afl-as فایل دودویی قابل اجرای SUT است که سپس توسط afl-fuzz استفاده می‌شود.
- **afl-fuzz.** همانطور که انتظار می‌رود، هسته اصلی فازر است که عملیات فاز ورودی را انجام می‌دهد. این ابزار فایل دودویی و داده آزمون ورودی را دریافت و با توجه به اطلاعات دریافتی از afl-analyze فرایند آزمون را ادامه می‌دهد. این مؤلفه برای آزمون فازی جعبه سیاه، مستقیماً به‌کار می‌رود. afl-fuzz همچنین مسئول چاپ واسط کاربری روی ترمینال است.
- **afl-analyze.** اثر ورودی اجرا شده بر پوشش کد را بررسی می‌کند و اطلاعات مربوط به پوشش کد را به عنوان بازخوردی به afl-fuzz می‌دهد تا برای جابه‌جایی‌های بهتر در ادامه استفاده کند.

حلقه بازخوردی که در شکل ۳ـ۲ وجود دارد نشان دهنده تکاملی بودن فرایند آزمون فازی در AFL است. AFL در هسته خود از الگوریتم ژنتیک استفاده می‌کند. یک فایل ورودی جهش یافته، مفید و مورد توجه است اگر قسمت‌های جدیدی از کد دودویی را اجرا کرده باشد یا تعداد اجرای کدهای قبلی مشاهده شده را افزایش داده باشد. این ویژگی با عنوان Input Gain شناخته می‌شود [۲۰]. AFL، سپس این داده آزمون را به انتهای

[1] Component

صف داده‌های آزمون، اضافه می‌کند. به این ترتیب حاصل فرایند آزمون فازی در AFL، علاوه بر اجرای SUT و اندازه‌گیری پوشش کد، یک مجموعه داده آزمون جهش یافته با ویژگی‌های متمایز است.

برای مقایسه نحوه عملکرد AFL و یک فازر تصادفی، جزئیات الگوریتم‌های این دو فازر را مرور می‌کنیم. الگوریتم ۳ـ۱، فرایند فاز ورودی در یک فازر تصادفی و الگوریتم ۳ـ۲ همین فرایند را در AFL نشان می‌دهد. تابع *Mutate* یک بایت از ورودی را به‌صورت درجا، با استفاده از فنونی مثل وارون‌کردن بیت، وارون‌کردن بایت، چرخش بیت یا عملیات ریاضی و منطقی، جابه‌جا می‌کند. تابع *Execute* برنامه را با ورودی جابه‌جا شده اجرا و خرابی‌های احتمالی را گزارش می‌دهد. خطوط سایه زده شده در دو الگوریتم (خطوط ۱۰، ۱۴ و ۱۵) قسمت‌های متفاوت را نشان می‌دهد [۲۰].

الگوریتم ۳ـ۱ Basic-Random Fuzzing [۲۰]

Input: *Seeds*, Target program *P*
Output: *MaliciousInputs*

```
1  for Seed ∈ Seeds do
2      for iterations ← 0 to limit do
3          input ← Seed
4          length ← len(Seed)
5          mutations ← RandInt(length)
6          for mut ← 0 to mutations do
7              byte ← RandInt(length)
8              mutate(input, byte)
9          end
10         result ← Execute(P, input)
11         if result is crash then
12             Append input to MaliciousInputs
13         end
14
15
16     end
17 end
```

الگوریتم ۳ـ۲ AFL Fuzzing [۲۰]

Input: *Seeds*, Target program *P*
Output: *MaliciousInputs*

```
1  for Seed ∈ Seeds do
2      for iterations ← 0 to limit do
3          input ← Seed
4          length ← len(Seed)
5          mutations ← RandInt(length)
6          for mut ← 0 to mutations do
7              byte ← RandInt(length)
8              Mutate(input, byte)
9          end
10         result, cov ← Execute(P, input)
11         if result is crash then
12             Append input to MaliciousInputs
13         end
14         if HasInputGain(cov) then
15             Append input to Seeds
16         end
17     end
18 end
```

۳ـ۱ـ۲ مشکلات AFL

آزمون فازی از لحاظ محاسباتی سنگین است. یعنی حتی یک Input Gain کوچک نیاز نیازمند هزارها تا میلیون‌ها جابه‌جایی است [۲۰]. از طرفی همه جابه‌جایی‌های یکسان نیستند. AFL با بهره‌گیری از بازخورد داده‌های آزمون بهتری را انتخاب می‌کند، اما همچنان آنها را به صورت تصادفی جهش می‌دهد یا جابه‌جا می‌کند. در نتیجه تعداد زیادی داده آزمون تکراری تولید می‌شود که لزوما تأثیری بر بهبود آزمون ندارند. از طرفی در ساختارهای پیچیده، تغییر برخی قسمت‌ها سبب می‌شود تا داده آزمون ورودی نامعتبر شود و سریعاً توسط تجزیه‌گر مردود اعلام گردد. در نتیجه تعداد زیادی داده آزمون هدر رفته خواهیم داشت. AFLافزوده[۱] [۲۰] مسئله انتخاب تصادفی مکان‌های جابه‌جایی را مورد مطالعه و راه‌کارهایی برای هوشمندسازی آن ارائه داده

[1] Augmented

است (بخش ۳ـ۲).

محدودیت دیگر AFL قابلیت حمل[1] پایین آن است. AFL به‌خاطر استفاده از فراخوانی‌های سیستمی سیستم عامل Linux تنها در توزیع‌های آن قابل استفاده است. نسخه‌ای از AFL، تحت عنوان WinAFL[2] برای سیستم عامل ویندوز توسعه داده شده است که البته سرعت پایین و سربار بالایی دارد. WinAFL برای افزایش سرعت پیشنهاد می‌کند که تنها یک تابع از برنامه که هدف اصلی آزمون فازی است، ابزارگذاری شود.

به‌دلیل مشکلاتی که اشاره شد، اجرای AFL و هرگونه فازر مشابه آن، روی نرم‌افزارهایی مثل PDFخوان‌ها که ساختار ورودی آنها پیچیده است به پوشش کد به‌مراتب کمتری دست می‌یابد که حتی با افزایش زمان آزمون نیز نتایج بهبود چندانی نمی‌کند. در این پایان‌نامه یک روش ترکیبی برای تولید داده آزمون ارائه می‌دهیم که پوشش کد بهتری نسبت به فازرهای مبتنی بر جابه‌جایی محض برای ساختارهای پیچیده فراهم کرده و در عین حال ویژگی‌های مثبت جابه‌جایی در فایل‌های دودویی را نیز به‌کار می‌بنند.

۳ـ۲ فازر AFLافزوده

AFLافزوده [۲۰] تلاش می‌کند با استفاده از فنون یادگیری ماشینی مکان‌های مناسبی را برای جابه‌جایی بایت‌ها پیدا کند. چارچوب ارائه شده در این کار شامل فازر AFL و یک مدل است که مکان‌های مفید برای جابه‌جایی را مشخص می‌کند. در طول اجرا فازر ابتدا مدل را برای اخذ آدرس مکان‌های جهش پرس‌وجو می‌کند و جابه‌جایی‌های فازر را روی مکان‌های بازگردانیده شده متمرکز می‌کند.

برای آموزش مدل فایل ورودی از مجموعه دانه اولیه، فایل جهش یافته و میزان پوشش کد هر دو فایل پس از اجرا توسط SUT مورد نیاز است. در فرایند آموزش اگر پوشش کد فایل جهش یافته و فایل والد یکسان باشد (یعنی میزان Input Gain صفر باشد)، مکان‌های جابه‌جا شده مکان‌های امیدبخشی نخواهند بود و چنان‌چه پوشش کد فایل جابه‌جا شده افزایش داشته باشد (یعنی میزان Input Gain بزرگ‌تر از صفر باشد)، آنگاه جابه‌جایی‌ها امیدبخش محسوب می‌شوند. با تعریف یک تابع خطا که بسته به مقدار Input Gain امتیازهایی به هریک از دو حالت بالا نسبت می‌دهد، مدل در حالت بانظارت آموزش می‌بیند. ورودی مکان‌های جهش در فایل و خروجی میزان امتیاز کسب شده بر اثر هریک از این مکان‌ها است. صورت‌بندی ریاضی مطالب عنوان شده، به‌طور مفصل در [۲۰] ذکر شده است.

یک ویژگی مثبت این مقاله آموزش و استفاده از چندین مدل متنوع یادگیری ژرف و نیز آزمون چندین برنامه هدف در انجام آزمایش‌ها است که سبب می‌شود تأثیر مدل‌های مختلف را بتوان با یکدیگر مقایسه کرد. برای

[1] Portability

[2] https://github.com/ivanfratric/winafl

ایجاد مجموعه آموزش، فازر AFL با تعداد ۱۸۰ دانه اولیه برای هر SUT به مدت ۲۴ ساعت اجرا گردیده و اطلاعات لازم ضبط شده‌اند.

الگوریتم ۳ـ۳، فرایند فاز ورودی و انجام آزمون فازی در AFLافزوده را نشان می‌دهد. خطوط سایه زده شده (خطوط ۲، ۱۱، ۱۲ و ۱۳) قسمت‌های اضافه شده به الگوریتم اصلی AFL هستند. چون جابه‌حایی‌های AFL در حالت عادی به‌صورت تصادفی است (خطوط ۷ و ۸ در الگوریتم ۳ـ۲ و خطوط ۸ و ۹ در الگوریتم ۳ـ۳)، تغییر انجام شده در AFLافزوده بدین صورت است که ابتدا مکان‌های مناسب جابه‌جایی برای یک ورودی را توسط مدل پیش‌بینی می‌کند، سپس اجازه می‌دهد تا AFL ورودی را جابه‌جا کند. حال چنان‌چه مجموع شباهت جابه‌جایی‌های AFL با جابه‌جایی‌های گزارش شده که مطلوب مدل است، کمتر از یک حد آستانه α باشد (خط ۱۲ الگوریتم ۳ـ۳)؛ این داده آزمون تولیدی برای اجرا ارسال نمی‌گردد. محاسبه شباهت دو رشته بیتی نیز با استفاده از ترکیب عطفی آنها (خط ۱۲ الگوریتم ۳ـ۳) به‌راحتی امکان‌پذیر است. چنان‌چه دو رشته مشابه نباشند مجموع بیت‌های ترکیب عطفی آنها کمتر از مجموع بیت‌های هریک از آنها خواهد بود و بالعکس.

۳ـ۲ـ۱ مشکلات AFLافزوده

الگوریتم AFLافزوده روی چندین قالب فایل و چندین برنامه محتلف، مورد آزمایش قرار گرفته است؛ از جمله قالب‌های فایل PNG، XML و PDF. کمترین بهبود گزارش شده بازهم مربوط به قالب فایل PDF و نرم‌افزار MuPDF است. علت آن هم ساختار پیچیده و هم حجم بالای فایل‌های PDF است. میزان درصد پوشش کد گزارش شده برای نرم‌افزار MuPDF توسط AFL اصلی ۱۱٫۶۳ و توسط AFLافزوده در بهترین مدل ۱۱٫۸۰ بوده که افزایش ناچیز ۰٫۱۷ درصدی را نشان می‌دهد. در بخش نتایج این مقاله همچنین هیچ ارزیابی روی دقت مدل‌های مختلف آموزش داده شده انجام نشده و به یک نتیجه‌گیری کلی بسنده شده است.

یک تأکید مهم که در مستندات AFL به آن اشاره شده است کوچک نگاه داشتن فایل‌های مجموعه دانه اولیه است (توصیه شده است که از فایل‌هایی با حجم زیر ۱ کیلوبایت استفاده شود)؛ زیرا، در صورتی که از فایل‌های با حجم بالایی استفاده شود سرعت فازر به‌شدت کاهش می‌یابد. در واقع جابه‌جایی‌های بسیاری بایستی روی یک دانه اعمال گردد تا یک داده آزمون جدید ایجاد شود. این در حالی است که برخی ساختارها مثل PDF ذاتاً حجم بالایی دارند. مثلاً یک فایل PDF تنها حاوی عبارت Hello Wrold حجمی در حدود ۱ کیلوبایت دارد. برای این ساختارها، تولید یک فایل جدید از ابتدا، مثلاً از روی یک مدل، سرعت تولید داده آزمون را افزایش می‌دهد. استفاده از یک روش ترکیبی یعنی ثابت نگه داشتن بخش‌هایی از فایل و تولید بخش‌های دارای اهمیت در ساختار یک فایل سرعت تولید داده را حتی بیشتر از تولید یک فایل از ابتدا هم

الگوریتم ۳ـ۳ Augmented-AFL Fuzzing [۲۰]

Input: $Seeds$, Target program P
Output: $MaliciousInputs$

```
for Seed ∈ Seeds do
    bytemask ← QueryModel(Seed)
    for iterations ← 0 to limit do
        input ← Seed
        length ← len(Seed)
        mutations ← RandInt(length)
        for mut ← 0 to mutations do
            byte ← RandInt(length)
            Mutate(input, byte)
        end
        diff ← input ⊕ Seed
        if ∑ diff ∧ bytemask < α then
            Continue
        end
        result, codecov ← Execute(P, input)
        if result is crash then
            Append input to MaliciousInputs
        end
        if HasInputGain(codecov) then
            Append input to Seeds
        end
    end
end
```

افزایش می‌دهد. روش ارائه شده در این پایان‌نامه یک روش ترکیبی است.

۳ـ۳ یادگیری و فاز

مقاله یادگیری و فاز[1] [۱۱] روشی برای تولید داده آزمون جهت استفاده در آزمون فازی بر مبنای RNN و مدل کدگذار_کدگشا[2] [۱۷، ۱۸] ارائه کرده است. در این مقاله ساختار فایل PDF و مرورگر وب Edge[3] شرکت مایکروسافت، برای آزمون انتخاب شده‌اند. ایده اصلی یادگیری یک مدل مولد زبان روی مجموعه‌ای از ویژگی‌های اشیای PDF با داشتن مجموعه‌ای از نمونه‌های اولیه است. ساختار قالب فایل PDF و اشیای داده‌ای آن، در پیوست آ توضیح داده شده است.

مدل کدگذار_کدگشا [۱۷، ۱۸] از دو RNN تشکیل شده است. یک شبکه کدگذار که یک توالی ورودی با بُعد متغیر را به یک نمایش بعد ثابت تبدیل می‌کند (بردار زمینه) و یک شبکه کدگشا که یک توالی ورودی بُعد ثابت را می‌گیرد و به یک توالی ورودی با بعد متغیر تولید می‌کند. شبکه کدگشا توالی‌های خروجی را با استفاده از کاراکتر خروجی پیش‌بینی شده که در مرحله‌زمانی t به عنوان کاراکتر ورودی برای مرحله‌زمانی $t+1$ تولید شده است، تولید می‌کند. یک بازنمایی از معماری این مدل در شکل ۳ـ۳ نشان داده شده است. این مدل نخستین بار در وظیفه ترجمه ماشینی استفاده شده است [۱۸]. با توصیف بیان شده، این معماری امکان یادگیری توزیع شرطی $p(< y^{(1)}, y^{(2)}, y^{(3)}, ..., y^{(n)} > | < x^{(1)}, x^{(2)}, x^{(3)}, ..., x^{(n)} >)$ را فراهم می‌کند.

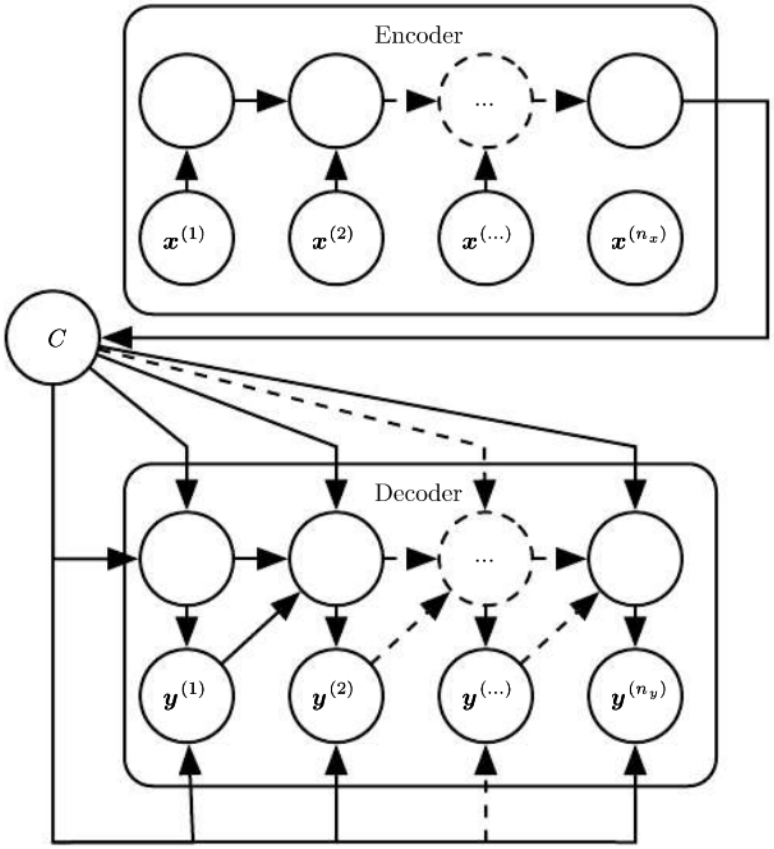
۳ـ۳ـ۱ آموزش مدل کدگذار_کدگشا

مدل کدگذار_کدگشا در روش یادگیری و فاز، با استفاده از تکه اشیای PDF که هر یک به‌صورت توالی‌ای از کاراکترها در نظر گرفته شده است، آموزش می‌بینند. برای آموزش، ابتدا همه فایل‌های حاوی اشیای PDF به‌شکل یک توالی از کاراکترها به یکدیگر الحاق می‌شوند که منجربه ایجاد یک توالی بزرگ از کاراکترها به‌صورت $\hat{s} =< s^{(1)}, s^{(2)}, s^{(3)}, ..., s^{(n)} >$ خواهد شد. سپس توالی $\hat{s}$ به چند زیر توالی آموزشی با طول ثابت d شکسته می‌شود به نحوی که iامین نمونه آموزشی عبارت است از: $t_i = \hat{s}[i * d : (i+1) * d]$ که $s[l : u]$ یک زیرتوالی از s بین اندیس‌های l و u را نشان می‌دهد. در ادامه، توالی خروجی برای هر توالی آموزشی t_i عبارت است از توالی ورودی که یک واحد به سمت راست شیفت داده شده است؛ یعنی:

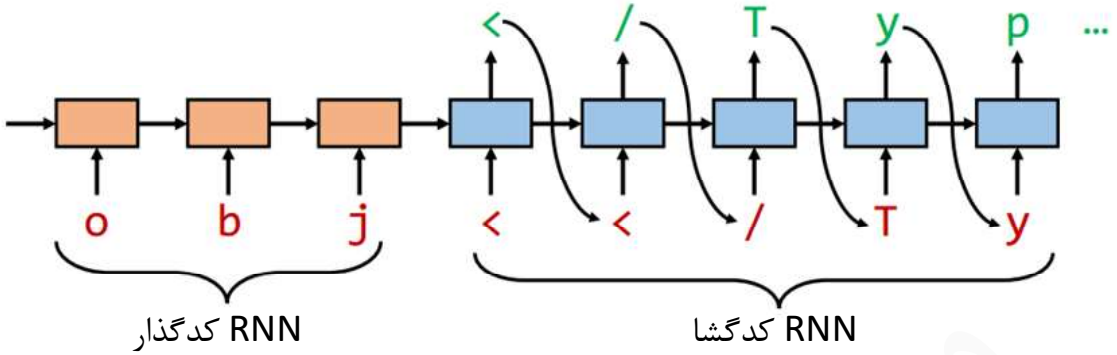
[1] Learn&Fuzz

[2] Encoder-Decoder

[3] مرورگر جدید شرکت مایکروسافت که همراه با سیستم عامل ویندوز ۱۰ معرفی شد.



شکل ۳ـ۳: مثالی از معماری مدل کدگذارـ کدگشا، متشکل از دو RNN و بردار زمینه C. این مدل برای یادگیری تولید توالی خروجی $< y^{(1)}, y^{(2)}, y^{(3)}, ..., y^{(n_y)} >$ از روی بردار زمینه، که نماینده توالی ورودی $< x^{(1)}, x^{(2)}, x^{(3)}, ..., x^{(n_x)} >$ است، به‌کار می‌رود [۳۸].



شکل ۳ـ۴: مدل RNN کدگذارـ کدگشا برای تولید اشیای داده‌ای PDF [۱۱].

$o_{t_i} = \hat{s}[i*d+1:(i+1)*d+1]$. مدل سپس به‌صورت انتهابه‌انتها[1] آموزش می‌بیند تا یک مدل مولد روی مجموعه همه توالی‌های ایجاد شده، یادگیری شود. شکل ۳ـ۴ مدل ارائه شده در این مقاله را نشان می‌دهد.

از آنجایی که مدل بحث شده در حالت بدون نظارت آموزش می‌بیند، برچسب‌های تولید شده به‌طور صریح برای مشخص کردن این که مدل یادگرفته شده چه‌قدر خوب عمل می‌کند، آزموده نشدند. به‌جای آن چندین مدل آموزش داده می‌شود که در تعداد دوره‌ها (بخش ۲ـ۳ـ۳) متفاوت هستند. مدل در ۱۰، ۲۰، ۳۰، ۴۰ و ۵۰ دوره آموزش داده شده است. در تنظیمات شبکه ارائه شده در این مقاله، یادگیری روی هر دوره حدود ۱۲ دقیقه طول می‌کشد و بنابراین مدل با ۵۰ عدد حدودا ۱۰ ساعت زمان نیاز دارد تا کامل شود. شبکه به‌صورت RNN با ۲ لایه مخفی که هر لایه ۱۲۸ سلول LSTM دارد، است.

۳ـ۳ـ۲ تولید داده‌های جدید

از مدل یادگیری شده برای تولید اشیای جدید PDF استفاده شده است. راهبردهای مختلفی برای تولید اشیا، بسته به راهبرد نمونه‌برداری که برای نمونه‌برداری از توزیع یادگیری شده به‌کار می‌رود، وجود دارند. در این مقاله، با یک پیشوند[2] از توالی "obj" (که آغاز یک شی را مشخص می‌کند) شروع و سپس مدل را پرس‌وجو کرده تا یک توالی از کاراکترهای خروجی تولید گردد، تا زمانی که مدل "endobj" را که مشخص کننده انتهای یک شی است، تولید نماید. سپس این مقاله، سه راهبرد مختلف را برای تولید اشیای جدید از روی مدل، استفاده کرده است [۱۱]:

[1] End to End
[2] Prefix

- **NoSample.** در این راهبرد تولید، کاراکتر پیش‌بینی شده با بالاترین احتمال به‌عنوان کاراکتر بعدی انتخاب می‌شود که در واقع انتخابی حریصانه[۱] است. این راهبرد برای تولید اشیای PDFای که خوش‌شکل[۲] و سازگار[۳] هستند، کاملاً نتیجه‌بخش است ولی تعداد اشیائی را که می‌توان تولید کرد، محدود می‌کند. با دادن یک پیشوند مثل "obj" بهترین توالی از کاراکترهای بعدی به‌صورت یکتا مشخص می‌شود و بنابراین این راهبرد همان شی را نتیجه می‌دهد. این محدودیت مغایر با اهداف آزمون فازی بوده و مانع مفید بودن استفاده از آن در آزمون فازی می‌شود.

- **Sample.** در این راهبرد تولید، به‌جای انتخاب محتمل‌ترین کاراکتر پیش‌بینی شده، از توزیع یادگرفته شده، برای نمونه‌برداری کاراکتر بعدی در توالی‌ای که پیشوند آن داده شده است، استفاده می‌شود. راهبرد نمونه‌برداری قادر به تولید مجموعه گوناگونی از اشیا با ترکیب الگوهای مختلفی که از اشیای موجود یادگرفته شده است، خواهد بود. به دلیل نمونه‌برداری اشیای تولید شده، تضمینی نیست که آنها به میزان راهبرد قبلی، خوش‌شکل باشند، که این ویژگی از منظر آزمون فازی مفید است.

- **SampleSpace.** این راهبرد ترکیب دو راهبرد قبلی است. SampleSpace توزیع احتمالی برای تولید کاراکتر بعدی را تنها زمانی که پیشوند توالی کنونی، با فضای خالی خاتمه می‌یابد، نمونه‌برداری می‌کند. برای میان توکن[۴]ها (یعنی پیشوندهایی که با فضای خالی خاتمه نمی‌یابند)، بهترین کاراکتر را انتخاب می‌کند (راهبرد NoSample) است که در مقایسه با راهبرد Sample، ورودی‌های خوش‌شکل‌تری تولید می‌شود. چون نمونه‌برداری در پایان کاراکترهای فضای خالی صورت می‌گیرد؛ نمونه‌های جدید نیز در این راهبرد، ساخته خواهند شد.

۳ـ۳ـ۳ اعمال آزمون فازی

برای فاز داده‌های آزمون تولید شده، مقاله الگوریتم جدیدی تحت عنوان SampleFuzz (الگوریتم ۳ـ۴)، ارائه کرده است. SampleFuzz توزیع یادگیری شده $D(x,\theta)$، احتمال فازینگ یک کاراکتر t_fuzz و احتمال آستانه p_t که تصمیم می‌گیرد آیا کاراکتر پیش‌بینی شده اصلاح شود یا نه، را به عنوان ورودی می‌پذیرد. سپس برای توالی خروجی seq، الگوریتم $D(x,\theta)$ را نمونه‌برداری می‌کند تا کاراکتر بعدی یعنی c و احتمال آن $p(c)$ را در یک مرحله زمانی مشخص t به‌دست آورد. اگر احتمال $p(c)$ از آستانه فراهم شده توسط کاربر (p_t)

[1] Greedy
[2] Well-formed
[3] Consistent
[4] Token

بالاتر باشد؛ یعنی، اگر مدل مطمئن باشد که کاراکتر بعدی به احتمال زیاد c است، الگوریتم تصمیم می‌گیرد تا یک کاراکتر دیگر c' که احتمال کمینه $p(c')$ را دارد، جایگزین c کند. البته این اصلاح (فاز) تنها در صورتی که عدد تصادفی تولید شده p_fuzz از ورودی t_fuzz مقدار بیشتری داشته باشد، انجام می‌شود، که بدین ترتیب اجازه می‌دهد تا کاربر بتواند بر روی احتمال (درصد) کاراکترهای فاز شده کنترل داشته باشد.

الگوریتم ۳ـ۴ SampleFuzz [۱۱]

Input: $D(x,\theta)$, t_fuzz, p_t
Output: seq

```
1  seq ← "obj"
2  while ~ seq.EndsWith("endobj") do
3      c, p(c) ← sample(D(seq,θ))     /* Sample c from the learnt distribution
       */
4      p_fuzz ← Random(0,1)     /* Random variable to decide whether to fuzz
       */
5      if p_fuzz > t_fuzz ∧ p(c) > p_t then
6          c ← argmin_c'{p(c') ~ D(seq,θ)}        /* Replace c by c' (with lowest
           likelihood) */
7      end
8      seq ← seq + c
9      if len(seq) > MaxLen then
10         seq ← "obj"                                  /* Reset the sequence */
11     end
12 end
13 Return seq
```

الگوریتم SampleFuzz اطمینان می‌یابد که طول شی با عدد $MaxLen$ محدود شود؛ اما، شرط حلقه $while$ الگوریتم تضمین نمی‌کند که همواره خاتمه یابد. با این حال، نویسندگان گزارش کرده‌اند که در آزمایش‌های انجام شده در عمل، الگوریتم همواره پایان یافته‌است. چیدمان آزمایش‌ها و نیز تحلیل جامع نتایج در [۱۱] آمده است که نشان از برتری الگوریتم SampleFuzz در مقایسه با روش تصادفی و افزایش نسبی پوشش کد در سطح دستورات برنامه است.

۳ـ۳ـ۴ مشکلات روش یادگیری و فاز

مقاله یادگیری و فاز برای نخستین بار از یادگیری ماشینی در تولید داده آزمون فازی استفاده کرده است. در عین حال روش ارائه شده در این مقاله دارای مشکلات و کاستی‌هایی است که عبارتند از:

۱. همواره با یک پیشوند ثابت شروع به تولید داده جدید می‌کند که این سبب می‌شود تنوع کمتری حاصل شود. در نتیجه ناچار به پیچیده کردن فرایند نمونه برداری از مدل می‌شود. همچنین این پیشوند کوتاه است و حاوی اطلاعات بعدی یک شی داده‌ای نیست. می‌توان مدل‌هایی با قابلیت پذیرفتن پیشوندهای متفاوت ایجاد کرد.

۲. مدل کدگذارـ کدگشا معمولاً برای نگاشت توالی‌های با طول و دامنه‌های متفاوت به یکدیگر، استفاده می‌شود. بارزترین مثال وظیفه ترجمه ماشینی است که هدف آن نگاشت جمله‌های یک زبان به زبان دیگر است [۱۷]. در این وظیفه، طول جمله زبان مقصد لزوماً برابر طول معادل همان جمله در زبان مبدأ نیست. به‌همین خاطر نیازمند معماری پیچیده کدگذارـ کدگشا هستیم. وظیفه یادگیری خودکار ساختار فایل بیشتر به ایجاد یک مدل زبانی روی الفبای زبان فایل ورودی شباهت دارد که در نتیجه استفاده از یک مدل زبانی عصبی در این وظیفه، منطقی‌تر و احتمالاً نتیجه‌بخش‌تر به نظر می‌رسد. چنان‌چه در روش یادگیری و فاز هم می‌بینیم که ساختار و وظایف شبکه کدگذار و کدگشا به خوبی تفکیک و تفهیم نگردیده است و صرفاً به پیچیده‌تر شدن مدل انجامیده است.

۳. فقط داده‌های متنی فایل تولید و فاز شده‌اند در حالی که یک فایل باینری در حالت کلی حاوی ترکیبی از داده‌های متنی و غیرمتنی است. می‌توان یک روش ترکیبی برای این منظور ارائه داد.

۴. الگوریتم فاز ارائه شده، همان‌طور که دیدیم، ممکن است هیچ‌گاه پایان نیابد یا اجرای الگوریتم برای تولید یک نمونه خیلی طولانی شود که خوب به‌نظر نمی‌رسد. می‌توان با تعریف شرایطی از این اتفاق جلوگیری کرد. خاتمه‌یافتن در هرصورت، یک ویژگی است که در تعریف اصطلاح الگوریتم وجود دارد و از این بابت این مسئله بسیار مهم تلقی می‌شود.

۵. معماری مدل پیشنهادی به نحوی است که توالی‌های تولید شده برای آموزش در برگیرنده همه اطلاعات و وابستگی‌های ساختار فایل نیستند. در هنگام تولید توالی‌های اموزشی هر بار به اندازه d واحد از روی مجموعه داده پرش می‌کنیم. اگر d کوچک باشد، توالی‌های کوچکی خواهیم داشت که لزوماً شامل یک وابستگی طولانی نیستند و اگر d بزرگ انتخاب شود، تعداد وابستگی‌های کمتری تشخیص داده می‌شود. به این نکته بایستی توجه کرد که وابستگی‌های فایل‌های با قالب مختلف، یکسان نیستند. می‌توان راهکارهای بهتری در نحوه تولید ورودی و خروجی شبکه یافت.

۶. در هر بار اجرا فقط یک نمونه را تولید می‌کند و بنابراین نمونه‌های تولید شده در چندین اجرای متوالی کاملاً مستقل از یکدیگر هستند. ممکن است چند نمونه متوالی به‌هم وابستگی داشته باشند در این صورت بایستی بتوان به‌نحوی این پارامتر را در تولید نمونه‌ها گنجاند.

۷. و بلأخره در مقاله یادگیری و فاز تنها از یک مدل یادگیری ژرف برای تولید داده آزمون استفاده شده است که می‌توان با انتخاب مدل‌های مختلف یا تغییر در ابرپارمترهای در مورد تأثیر نوع مدل نیز بحث کرد و آزمایش‌های جدید را تجربه نمود.

۳ـ۴ دیگر کارهای مرتبط

تعداد دیگری کار در زمینه تولید داده آزمون در فازرها و به‌ویژه در زمینه یادگیری ساختار فایل انجام شده است. در فصل اول، به‌طور خلاصه به برخی از آنها اشاره کردیم [۱۵، ۱۶]. بیشتر این روش‌ها بسیار پیچیده بوده، درک و پیاده‌سازی آنها مشکل است و سربار زیادی را به عملیات آزمون فازی یا عملیات پیش‌پردازش، عملیات پیش از انجام آزمون فازی، تحمیل می‌کنند. در ضمن اغلب آنها در نهایت محدود به ساختارهای ورودی خاصی می‌شوند. تمرکز ما بر روی روش‌هایی است که کاملاً عملیاتی بوده و قابل استفاده بر روی نرم‌افزارهای دنیای واقعی هستند.

در آزمایشگاه تحقیقاتی مهندسی معکوس دانشگاه علم و صنعت ایران، تعدادی زیادی پروژه، کار پژوهشی و پایان‌نامه در ارتباط با آزمون فازی و تولید داده آزمون، نگارش شده است [۱]. یعقوبی [۴۸] یک روش خودکار تولید داده آزمون با استفاده از الگوریتم ژنتیک برای آزمون فازی جعبه سیاه مرورگرهای وب، ارائه داده است. هدف الگوریتم در این فرایند تکاملی، تشخیص مجموعه‌ای از برچسب‌های HTML است که حداکثر درهم ریختگی حافظه و کاهش سرعت اجرایی مرورگر را موجب شوند. در واقع دسته برچسب‌های HTML به‌عنوان صفحات ورودی هر مرورگر، طوری ساخته می‌شوند که در هر مرحله نسبت به دسته برچسب‌های قبل، میزان مصرف حافظه اصلی را بالا ببرند.

امکان بهره‌گیری از این روش برای آزمون فازی و شناسایی خطاها و آسیب‌پذیری‌های دیگر نرم‌افزارهایی که کد منبع آنها در دسترس نیست نیز وجود دارد. اما تمرکز این روش روی تولید برچسب‌های HTML بوده و سرعت همگرایی آن کند است. علاوه بر این، تنها هدف الگوریتم ژنتیک در اینجا، افزایش حافظه تعریف

[۱] فهرست کاملی از کارهای مرتبط انجام شده، از طریق وب‌سایت رسمی آزمایشگاه به نشانی http://parsa.iust.ac.ir قابل دسترسی هستند. همچنین فهرستی سازمان‌دهی شده و کامل در ارتباط با منابع مختلف آزمون فازی در نشانی https://github.com/secfigo/Awesome-Fuzzing وجود دارد.

شده است در حالی که می‌دانیم صرف افزایش حافظه منجربه کشف خطا نمی‌شود. در سیستمی با حافظه پایین ممکن است سیستم عامل برنامه را پیش از سقوط متوقف سازد یا برنامه فقط بر اثر کمبود حافظه نتواند یک ورودی به اندازه کافی بزرگ را پردازش کند. در ساختارهای پیچیده که تجزیه‌گر سخت‌گیری دارند، تولید داده به این روش بسیار زمان‌بر است؛ زیرا، در عمل خیلی از داده‌های آزمون تولیدی مورد پردازش قرار نمی‌گیرند و حافظه‌ای به آنها تخصیص داده نمی‌شود.

امینی [۴۹] یک روش تکاملی با هدف حل مشکل بزرگی فضای ورودی، برای تولید داده آزمون ارائه داده است. در روش‌هایی مشابه این روش مسئله تولید داده آزمون به صورت یک مسئله جست‌وجو تعریف می‌گردد و سپس از الگوریتم‌های اکتشافی برای یافتن یک حل بهینه استفاده می‌شود. در راهکار امینی، زمانی که یک داده آزمون یک الگوی آسیب‌پذیر[1] را شناسایی می‌کند، داده آزمون بعدی به‌گونه‌ای تولید می‌شود که منجربه افشای آن آسیب‌پذیری گردد. در نتیجه احتمال کشف آسیب‌پذیری‌های شناسایی شده بهبود خواهد داشت. الگوی آسیب‌پذیر در سطح کد منبع برنامه قابل شناسایی بوده و بنابراین این روش در فازرهای جعبه سفید قابل استفاده است. البته می‌توان آن را به فازرهای جعبه خاکستری هم تعمیم داد. روش مذکور روی برنامه‌های دنیای واقعی ارزیابی نشده است. همچنین یافته‌های آزمایش‌های آن، با هیچ روش هوشمند تولید داده آزمونی، مقایسه نگردیده است.

۳ـ۵ خلاصه

جدول ۳ـ۱، یک جمع‌بندی از مزایا و معایب سه کار اصلی معرفی شده در این فصل [۱۱، ۱۹، ۲۰] را در کنار یکدیگر نشان می‌دهد. همچنین فازر FileFuzz [۲]، که یک فازر قالب فایل تصادفی و بسیار اولیه است به‌عنوان یک فازر مبنایی برای مقایسه، در جدول آورده شده است. البته تا به امروز، فازرهای بسیار زیادی در زمینه آزمون فازی قالب فایل توسعه داده شده‌اند که مجال بررسی همه آنها در این پایان‌نامه نبوده و در اینجا صرفاً آن دسته از کارهایی را بیان کردیم که قصد داریم در روش پیشنهادی خود مشکلات موجود در آنها را تا حد امکان برطرف کنیم. برخی از این فازرها مانند [۱۰] به صورت داخل سازمانی و یا تجاری ارایه شده‌اند و کد منبع آنها در دسترس نیست. عدم دسترسی به کد منبع یک فازر و در نتیجه عدم امکان تغییر در پیمانه‌های مختلف آن را می‌توان انگیزه‌ای برای توسعه فازرهای جدید دانست.

به‌طور خلاصه، در این فصل ابتدا فازر قالب فایل AFL [۱۹] را به‌عنوان یک فازر با روش تولید داده آزمون مبتنی بر جابه‌جایی مورد مطالعه قرار دادیم و دیدیم که به علت جابه‌جایی تصادفی بایت‌های فایل، تعداد زیادی داده آزمون بیهوده برای ساختارهای پیچیده ایجاد می‌کند. سپس AFLافزوده [۲۰] را توضیح

[1] Vulnerable Pattern

جدول ۳ـ۱: مقایسه کارهای مرتبط

فازر	مشخصه‌ها	مزایا	معایب
FileFuzz [۲]	• مبتنی بر جابه‌جایی • جعبه سیاه	• سادگی پیاده‌سازی • عدم نیاز به اطلاع داشتن از ساختار ورودی و ساختار SUT	• تعیین مکان جابه‌جایی‌ها به‌صورت کاملاً تصادفی • پوشش کد بسیار پایین و غیر قابل قبول، برای بیشتر نرم‌افزارها پس از آزمون
AFL [۱۹]	• مبتنی بر جابه‌جایی • مبتنی بر بازخورد • جعبه خاکستری	• بیشینه کردن پوشش کد با بهره‌گیری از الگوریتم ژنتیک • قابل استفاده به صورت جعبه‌سیاه، جعبه‌خاکستری و جعبه‌سفید	• تعیین مکان جابه‌جایی‌ها به‌صورت تصادفی • تولید تعداد زیادی داده آزمون هدر رفته • سرعت پایین فاز در فایل‌های با حجم بیشتر از ۱ کیلوبایت
AFLٔافزوده [۲۰]	• مبتنی بر جابه‌جایی • مبتنی بر بازخورد • جعبه خاکستری	• تعیین مکان (آدرس نسبی) جابه‌جایی‌ها از طریق یک مدل ارزش‌دهی به بایت‌ها	• پوشش کد پایین برای نرم‌افزارهای با ساختار ورودی پیچیده
Learn&Fuzz [۱۱]	• مبتنی بر تولید • جعبه سفید	• یادگیری گرامر فایل و تولید داده‌های آزمون جدید با استفاده از مدل کدگذار_کدگشا • تعیین مکان فاز داده آزمون توسط مدل یادگیری شده • تعیین مقدار فاز شده داده آزمون	• حذف کامل داده‌های دودویی • شروع تولید داده آزمون با یک پیشوند ثابت • عدم تضمین پایان یافتن الگوریتم فاز • در دسترس نبودن فازر، مجموعه داده و دیگر پارامترهای مهم مربوط به آموزش مدل

دادیم از که فنون یادگیری ماشینی برای تعیین محل مکان‌های جابه‌جایی استفاده می‌کند. ایده اصلی در این روش، آن است که در قالب‌های فایل، معمولاً فقط قسمت‌های کوچکی از فایل برای تجزیه‌گر مهم هستند و در نتیجه بیشتر جابه‌جایی آن‌ها منتهی به اجرای مسیرهای جدید SUT می‌گردد. این روش نیز روی قالب پیچیده‌ای مانند PDF بهبود ناچیزی داشته است. با این حال روی قالب‌های کاملاً دودویی مثل قالب فایل PNG در زمان برابر به پوشش کد بهتری دست یافته است.

در ادامه فصل، روش یادگیری و فاز [۱۱] را توضیح دادیم که از مدل کدگذار_کدگشا برای تولید و فاز داده آزمون استفاده می‌کند. ایده اصلی این روش یادگیری خودکار گرامر برای یک قالب فایل متنی و سپس تولید داده با روش مبتنی بر تولید است. این روش با یک روش تصادفی که در ابتدای فصل نیز مطرح شد، مقایسه شده و نتایج بهبود داشته است. روش یادگیری و فاز روی ابزار متن بسته مرورگر Edge شرکت مایکروسافت و توسط محققین همین شرکت مورد آزمایش قرار گرفته و هیچ‌گونه کدمنبع و مجموعه داده‌ای از آن منتشر نشده است. در مقابل اما فازر AFL متن‌باز و رایگان است. AFLافزوده نیز به‌تازگی توسط توسعه‌دهندگان آن منتشر شده است.

در پایان این فصل مروری اجمالی بر دو کار پیشین انجام شده در زمینه آزمون فازی و تولید خودکار داده آزمون در آزمایشگاه مهندسی معکوس دانشگاه علم و صنعت ایران، که این پایان‌نامه در آنجا نگارش یافته است، داشتیم. هر دو کار [۴۸، ۴۹] از الگوریتم ژنتیک برای تولید داده‌های آزمون استفاده کرده بودند که مزایا و معایب هریک نیز بیان شد. در این پایان‌نامه ما از الگوریتم‌های یادگیری ژرف برای تولید خودکار داده آزمون استفاده خواهیم کرد. با توجه به مزایا و معایبی که در کلیه روش‌های مطرح در این فصل شناسایی و معرفی کردیم، در فصل بعد یک روش ترکیبی که بتواند از ویژگی‌های خوب جابه‌جایی تصادفی فایل‌های دودویی در کنار تولید فایل‌های متنی براساس مدل، استفاده کند، ارائه خواهیم کرد. هدف بهبود برخی کاستی‌های شناسایی شده در کارهای مطرح در این فصل و حل مسائل اصلی اشاره شده در فصل اول است.

فصل ۴

روش پیشنهادی

«اولین قانون هر فناوری مورد استفاده در یک کسب‌وکار این است که اِعمال خودکارسازی به یک عملیات کارآمد، کارآمدی را افزایش می‌دهد. دومین قانون این است که اِعمال خودکارسازی به یک عملیات ناکارآمد، ناکارآمدی را افزایش می‌دهد.»

✠ بیل گیتس

هدف، در روش پیشنهادی همان‌طور که پیش از این هم گفته شد، ایجاد یک مدل برای یادگیری خودکار گرامر قالب فایل و سپس استفاده از آن در تولید داده‌های آزمون جدید است. معماری مدل یادگیرنده ساختار فایل، نحوه آموزش، مجموعه داده لازم برای آموزش مدل، نحوه تولید داده‌های آزمون و در نهایت چگونگی انجام آزمون فازی، مباحثی هستند که در این فصل به آنها خواهیم پرداخت. در ابتدا به بیان کلیات روش پیشنهادی پرداخته و سپس هریک از مراحل را با جزئیات تشریح می‌کنیم.

۴ـ۱ کلیات

شکل ۴ـ۱ یک روندنمای ساده از کلیات مراحل انجام روش پیشنهادی ما را نشان می‌دهد. در ساختار برخی قالب‌های فایل پیچیده مانند PDF، بسیاری از قسمت‌ها به صورت متنی هستند. اما ممکن است قسمت‌های دودویی هم وجود داشته باشد. در روش پیشنهادی خود ما ابتدا کلیه قسمت‌های متنی فایل‌های موجود در

دانه‌های اولیه را استخراج، سپس این قسمت‌ها را به یکدیگر الحاق می‌کنیم. حاصل این کار ایجاد یک پیکره[1] بزرگ است که در بطن خود مشخصه‌های قالب یک فایل را دارد. در ادامه این مجموعه را مطابق با ضوابط فنون یادگیری ماشینی به سه مجموعه آموزش، ارزیابی و آزمون تقسیم‌بندی با نسبت‌های مشخص می‌کنیم. سپس یک مدل مولد را ایجاد و آن را روی داده‌های مجموعه آموزش، با تعریف یک روش یادگیری، آموزش می‌دهیم. در نهایت از این مدل برای ایجاد داده‌های آزمون جدید و اعمال آزمون فازی استفاده خواهیم کرد. اما چون مدل صرفاً قادر به تولید داده‌های متنی است در هنگام ایجاد یک داده آزمون، با فنونی که در ادامه خواهیم گفت، قسمت‌های دودویی را نیز به داده آزمون تزریق می‌نماییم.

۴ ـ ۱ ـ ۱ تمایز داده‌های متنی و دودویی

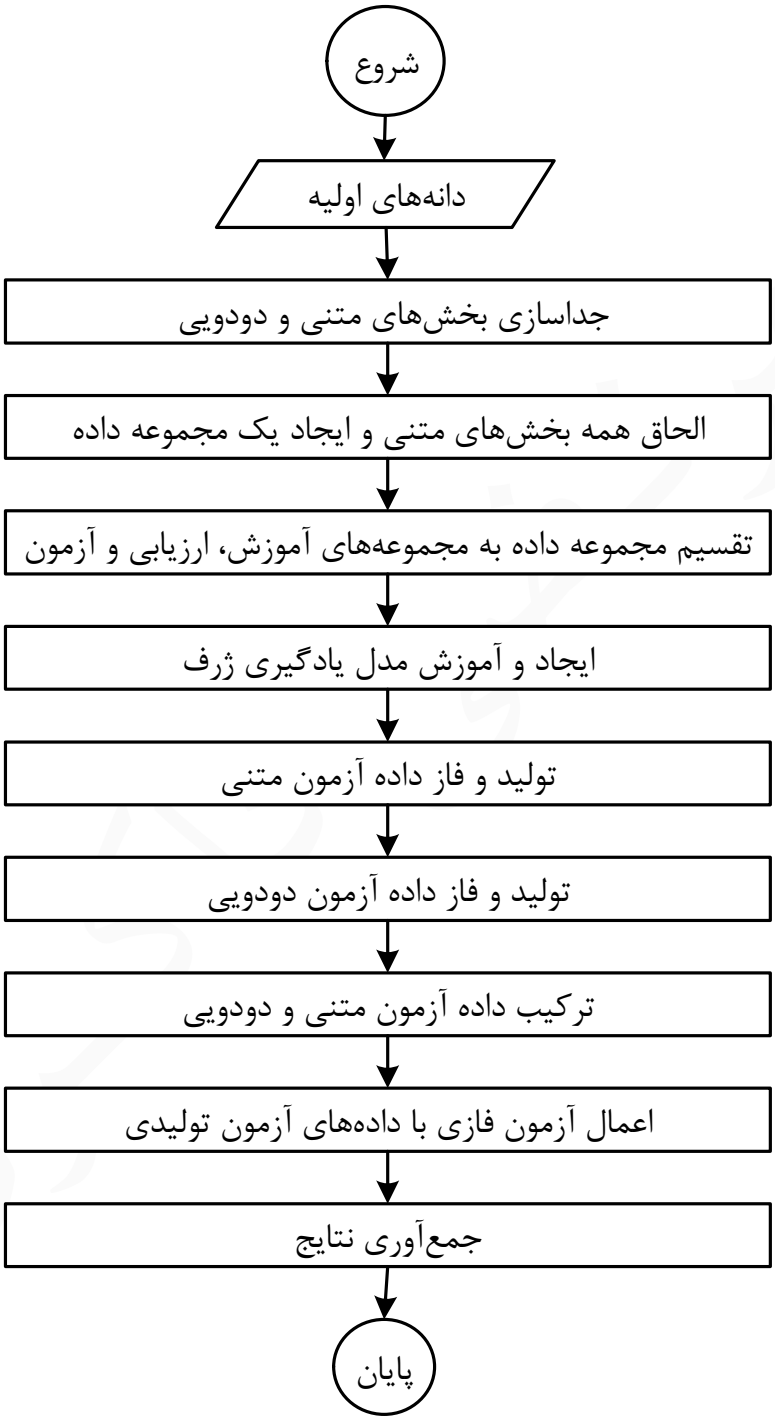
گفتیم مدل مولد را روی بخش متنی قالب فایل ایجاد می‌کنیم. اما مفهوم دقیق متنی و دودویی و معیار جداسازی آنها چیست و دلیل تأکید ما بر این موضوع از کجا نشئت می‌گیرد؟ منظور از داده‌های متنی در اینجا مجموعه کدهای ASCII[2] است. هنگامی که درسطح بیت یک فایل را بررسی می‌کنیم تمایزی میان واژه‌های متنی و دودویی نیست؛ زیرا همه چیز با مفهوم صفر و یک بیان می‌شود که دودویی است. اما اگر فایل را به‌صورت یک توالی از بایت‌ها در نظر بگیریم، با توجه به اینکه هر بایت ۲۵۵ حالت ممکن خواهد داشت، می‌توانیم آن فایل را یک زبان متشکل از تکرار حداکثر ۲۵۵ نشانه بدانیم. کدهای ASCII حتی این تعداد را به ۱۲۸ نشانه که کاراکترها و نشانه‌های متداول زبان انگلیسی هستند، محدود می‌کنند. مشکل بخش‌های دودویی آن است که اغلب در سطح بیت تفسیر می‌شوند و یادگیری وابستگی‌های آنها با شبکه‌های عصبی، که داده‌های محدودی را در زمان آموزش می‌بینند، امکان‌پذیر نیست. از طرفی هنگامی که نسبت کوچکی از داده‌ها به‌صورت دودویی در میان قسمت‌های متنی ظاهر می‌شوند می‌توان آنها را با روش‌های جابه‌جایی تصادفی فاز کرد.

ما در روش پیشنهادی خود داده‌های دودویی را از قالب فایل حذف کرده اما به‌جای آن یک توکن خاص موسوم به توکن دودویی[3] را جایگزین می‌کنیم. بدین ترتیب در فرایند آموزش مدل توزیع آماری نحوه ظاهر شدن قسمت‌های دودویی نیز فراگیری می‌شود. حال در فرایند تولید داده‌های آزمون جدید مدل مکان‌های احتمالی ظاهر شدن قسمت دودویی را با پیش‌بینی وقوع توکن مربوطه، تعیین می‌کند. سپس توکن را با یک قسمت دودویی که به‌صورت تصادفی (یا هر روش دیگر) فاز شده است جایگزین می‌کنیم. یعنی وارون عملی که پیش از فرایند آموزش انجام دادیم را پس از فرایند تولید، انجام می‌دهیم. با کلیت گفته شده در اینجا یک روش ترکیبی تولید خودکار داده آزمون خواهیم داشت.

[1] Corpus

[2] American Standard Code for Information Interchange

[3] Binary Token



شکل ۴ـ۱ : روندنمای روش پیشنهادی در حالت کلی.

۴ ـ ۱ ـ ۲ تمایز داده و فراداده

در فصل ۱ ، تمایز داده و فراداده به عنوان یک مسئله در تولید داده آزمون مطرح گردید. مدل مطرح در روش پیشنهادی این امکان را فراهم می‌کند تا بتوانیم به‌صورت احتمالی داده و فراداده را از یکدیگر تشخیص دهیم. از آنجایی که فراداده‌ها در ساختار یک فایل از پیش تعریف شده و تقریباً ثابت هستند، تکرار آنها به مراتب بیشتر از تکرار داده‌ها است که محتویات یک فایل را تشکیل می‌دهند. مدل بدین ترتیب در توزیع آماری یادگیری شده احتمال بیشتری را به فراداده‌ها نسبت می‌دهد. با تعیین یک آستانه مشخص برای هریک از انواع داده و فراداده در هنگام تولید داده آزمون قادر به تصمیم‌گیری در مورد نوع آنها هستیم. ما نتیجه این تصمیم را در نحوه فاز داده آزمون جدید به‌کار می‌بریم، بدین صورت که دو الگوریتم فاز با اهداف فاز فراداده برای مرحله تجزیه و فاز داده برای مرحله پرداخت فایل ارائه خواهیم داد.

۴ ـ ۱ ـ ۳ مثال انگیزشی

تمرکز اصلی ما در روش پیشنهادی، بر روی پیمانه تولید داده آزمون در یک فازر قالب فایل است. شکل ۴ ـ ۲ ، مراحل تولید داده آزمون از طریق روش پیشنهادی را روی قالب فایل PDF، نشان می‌دهد. سه مرحله اصلی در این شکل وجود دارد که به ترتیب با شماره‌های یک تا سه، شماره‌گذاری شده‌اند. مطابق این مراحل، ابتدا پیکره‌ای از فایل‌های از پیش موجود و معتبر جمع آوری گردیده و به عنوان مجموعه داده انتخاب می‌شوند. در مرحله اول، یعنی پیش‌پردازش، هریک از فایل‌های ورودی پیمایش شده، بخش‌های متنی آن جدا و به یکدیگر الحاق می‌شوند. همچنین بخش‌های دودویی داخل فایل، نیز به‌صورت مجزا، پس از جدا شدن، نگهداری می‌شوند. در فایل PDF، بخش‌هایی دودویی بین کلیدواژه‌های stream و endstream می‌آیند که در پیش‌پردازش آنها را با توکن دودویی stream جایگزین می‌کنیم. خروجی این مرحله دو مجموعه است. یک مجموعه داده متنی و یک مجموعه داده دودویی. در مجموعه داده متنی، مکان قسمت‌های دودویی با توکن دودویی مشخص شده است.

در مرحله دوم، یک مدل زبانی عصبی برروی مجموعه داده حاصل از جداسازی بخش‌های متنی پیکره آموزش داده می‌شود. مانند هر وظیفه یادگیری ماشینی در این مرحله ابتدا مجموعه داده ورودی به سه مجموعه آموزش، آزمون و ارزیابی تقسیم می‌گردد. خروجی این مرحله یک مدل مولد است که قادر به تولید داده‌های جدید با پیروی از توزیع حاکم بر داده‌های مجموعه آموزش بوده و کاراکتر به کاراکتر داده‌های جدید را تولید می‌کند.

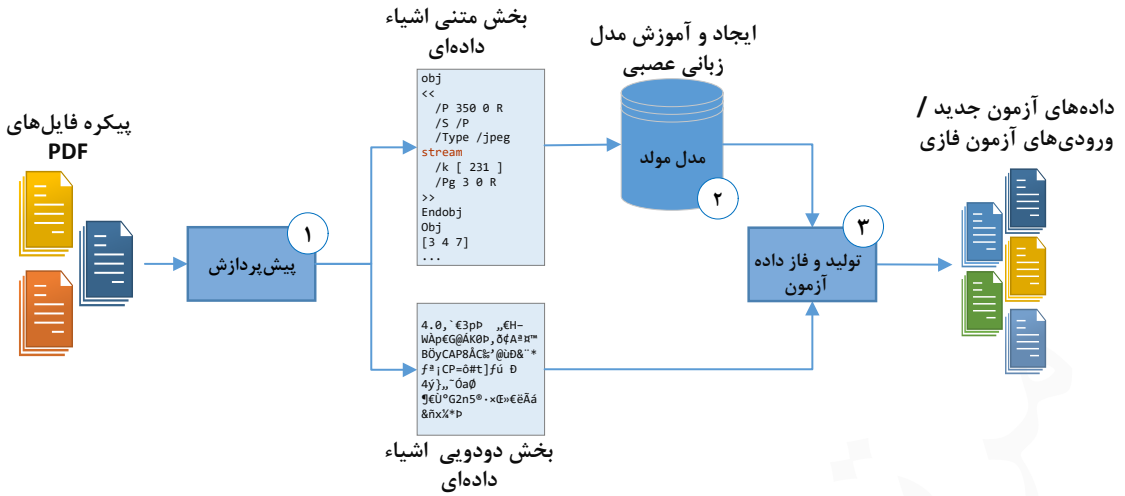
مرحله سوم با روش‌هایی اقدام به تولید داده‌های آزمون جدید، در این‌جا فایل‌های PDF، می‌کند. در این مرحله بخش‌های متنی فایل جدید از طریق مدل مولد، یعنی با روش مبتنی بر تولید، ایجاد می‌گردد. عمل

بدشکل کردن داده‌های متنی نیز همزمان با تولید آنها صورت می‌گیرد. بدین صورت که برخی از کاراکترها بعد از تولید، بر اساس شرایطی که در ادامه فصل توضیح داده می‌شوند، با کاراکترهای دیگری جایگزین می‌شوند. در همین مرحله، بخش‌هایی دودویی نیز در صورت پیش‌بینی توسط مدل مولد، با یک روش مبتنی بر جابه‌جایی، تولید و در مکان مناسب خود قرار می‌گیرند. منظور از پیش‌بینی در اینجا، تولید توکن دودویی stream در توسط مدل مولد است. در تولید بخش دودویی یک داده آزمون از بخش‌های دودویی پیکره اولیه که در مرحله یک جداسازی شده است به‌عنوان دانه اولیه استفاده می‌کنیم. در واقع آنها را با یکی از عملگرهای جابه‌جایی که در فصل ۲ به آن اشاره شد به‌سادگی جابه‌جا کرده و یک داده آزمون دودویی جدید و فاز شده می‌سازیم.

مرحله سوم در مجموع تعدادی داده آزمون را با روشی ترکیبی، تولید می‌کند. این داده‌های آزمون سپس می‌توانند به عنوان داده‌های آزمون در فرایند آزمون فازی استفاده شوند. همچنین می‌توان از آن به عنوان دانه اولیه برای فازرهای قالب فایل مبتنی بر جابه‌جایی نیز استفاده کرد. برای ارزیابی روش پیشنهادی خود که در فصل بعد به آن خواهیم پرداخت، در انتهای این فصل یک فازر قالب فایل ساده را مطرح می‌کنیم. این فازر قادر است تا پس از تولید داده‌های آزمون به روش شرح داده شده آنها را به عنوان ورودی به SUT تزریق کرده و خطاهای احتمالی را در صورت وجود گزارش کند. مرحله پیش‌پردازش بسیار ساده بوده و نیاز به توضیح خاصی ندارد. این مرحله ممکن است برای قالب‌های فایل مختلف، کاملاً متفاوت باشد. مثلاً برای قالب‌های فایل متنی مثل HTMLو XML پیش‌پردازش تنها الحاق کردن کلیه فایل‌های موجود در پیکره است. مراحل دوم و سوم، یعنی ایجاد و آموزش مدل مولد و تولید و فاز داده آزمون، به ترتیب در بخش ۴ـ۲ و بخش ۴ـ۳ توضیح داده شده‌اند. همچنین معماری فازر قالب فایل پیشنهادی در بخش ۴ـ۴ بیان گردیده است.

۴ـ۲ مدل

یک فایل را می‌توان نمونه تولید شده توسط زبانی که قالب آن فایل را بیان می‌کند دانست. با این فرض می‌توان یک مدل زبانی برای هر قالب فایلی ایجاد کرد. مدل زبانی عصبی عملکرد بهتری نسبت به مدهای زبانی سنتی دارد. با توجه به ذات مبتنی بر توالی بایت‌های یک فایل، وابستگی بایت فعلی به یک توالی از بایت‌های قبلی و نیز تجزیه ترتیبی (چپ به راست و بالا به پایین) آن توسط تجزیه‌گر SUT در بسیاری از موارد، در روش پیشنهادی خود، از RNN برای ایجاد مدل زبانی و یادگیری ساختار فایل استفاده می‌کنیم. در فصل ۲ دیدیم که RNN برای یادگیری وظایف مبتنی بر توالی ابداع شده است. به‌طور دقیق‌تر در هنگام آموزش از یک مدل RNN چند به یک، با معماری شکل ۲ـ۶ (پ) استفاده خواهیم کرد و در هنگام تولید داده جدید با فراخوانی همین مدل به‌صورت متوالی، مدلی از نوع شکل ۲ـ۶ (ت) خواهیم داشت.



شکل ۴ـ۲ : مراحل یادگیری ساختار فایل، تولید و فاز داده آزمون برای فایل PDF.

هدف RNN در اینجا ایجاد یک مدل زبانی عصبی روی ساختار فایل با دیدن یک پیکره از نمونه‌های آموزشی است. مطابق آنچه گفتیم، مدل زبانی مدلی مولد است، یعنی پس از آموزش، می‌توان از آن برای تولید داده‌های آزمون جدید استفاده کرد، که در ادامه نحوه این کار را توضیح خواهیم داد. در این پایان‌نامه دو مدل با معماری مختلف بر مبنای RNN می‌سازیم. در هسته عصب‌های RNN هر دو مدل از سلول LSTM استفاده می‌کنیم که قابلیت یادگیری توالی‌های با طول زیاد را دارد. مدل اول LSTM یک‌سویه[۱] نام دارد که معماری آن مشابه شکل ۲ـ۵ است. مدل دوم LSTM دوسویه[۲] است.

LSTM دوسویه توالی ورودی را به دو صورت روبه‌جلو[۳] و روبه‌عقب[۴] می‌بینند. در واقع LSTM دوسویه از دو LSTM یک‌سویه تشکیل شده است. یکی از آنها توالی را از چپ به راست پردازش می‌کند و دیگری از راست به چپ. در نتیجه هر گذرجلو LSTM دو خروجی خواهد داشت. یک تابع ادغام[۵] برای ادغام خروجی هریک از LSTMهای روبه‌جلو و روبه‌عقب به‌کار می‌رود. تابع ادغام می‌تواند هر تابعی که قادر به ترکیب دو بردار با طول‌های یکسان است، درنظر گرفته شود؛ از جمله: جمع، ضرب، الحاق و غیره. ما از تابع ادغام جمع برای مدل پیشنهادی خود استفاده می‌کنیم که درایه‌های نظیر به نظیر هر یک از بردارهای خروجی را با یکدیگر جمع می‌زند.

علاوه‌بر نوع مدل LSTM، برای LSTM یک‌سویه مدل‌هایی با تعداد عصب‌های متفاوت و نیز تعداد

[1] Unidirectional
[2] Bidirectional
[3] Forward
[4] Backward
[5] Merge Function

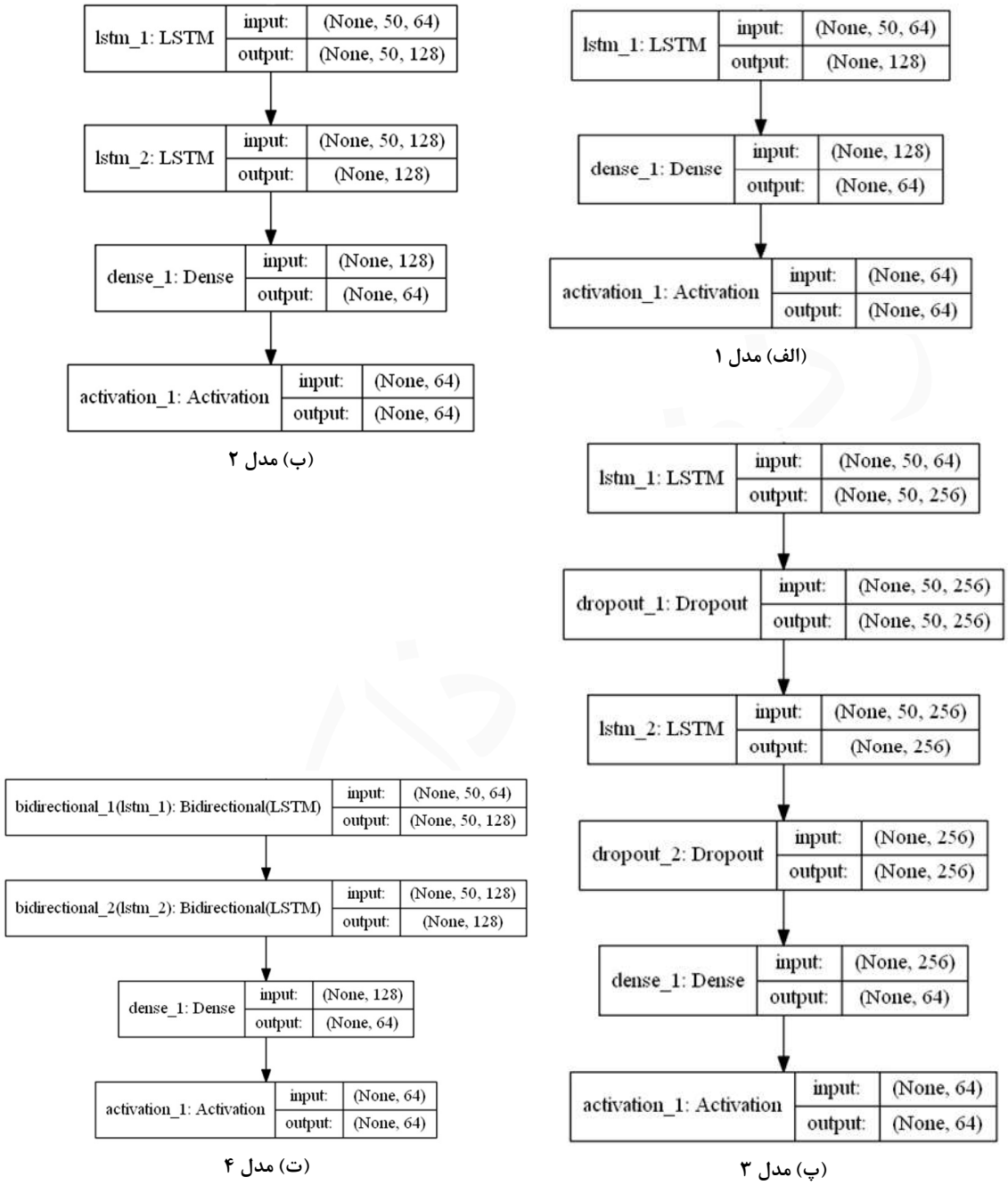
جدول ۴ـ۱ : مدل‌های یادگیری ژرف طراحی شده، پارامترها و ابرپارامترهای آنها.

شماره (نام) مدل	نوع (معماری) مدل	تعداد لایه پنهان	تعداد عصب در هر لایه	تعداد پارامترها
۱	Unidirectional LSTM (Many to One)	۱	۱۲۸	۱۲۷۵۸۴
۲	Unidirectional LSTM (Many to One)	۲	۱۲۸	۲۳۸۶۵۶
۳	Unidirectional LSTM (Many to One)	۲	۲۵۶	۸۷۰۴۶۴
۴	Bidirectional LSTM (Many to One)	۲	۱۲۸	۴۶۹۰۵۶

لایه‌های پنهان متفاوت را ساخته‌ایم. هدف از این کار مشاهده تأثیر مدل‌های با پیچیدگی مختلف در یادگیری ساختار فایل و تولید داده‌های آزمون و آزمایش این فرضیه است که آیا مدل‌های پیچیده‌تر، مدل‌های ساده‌تر را شکست می‌دهند؟ یعنی موفق به افزایش پوشش کد و یا شناسایی خطاهای بیشتری می‌شوند یا خیر. جدول ۴ـ۱ مدل‌های طراحی شده برای استفاده در آزمایش‌های روش پیشنهادی و مشخصه‌های هریک را نشان می‌دهد. یکی از مهم‌ترین مشخصه‌ها تعداد پارامترهای هر مدل است. گراف محاسباتی هریک از مدل‌های جدول ۴ـ۱ در شکل‌های ۴ـ۳الف تا ۴ـ۳ت آمده است. اندازه ماتریس‌های ورودی و خروجی در گراف مرتبط با تعداد لایه‌های پنهان و نیز تعداد عصب‌ها در هر لایه است که در ادامه بیشتر توضیح داده خواهد شد. همچنین در ادامه پایان‌نامه، هریک از مدل‌های معرفی شده، با شماره ذکر شده برای آنها در جدول ۴ـ۱ شناخته می‌شوند.

۴ـ۲ـ۱ آموزش مدل

فرایند آموزش همه مدل‌های جدول ۴ـ۱ یکسان است. در واقع برای آموزش هر مدل تنها نیاز داریم که ورودی و خروجی شبکه عصبی ژرف را مشخص نماییم. پس از استخراج مجموعه داده اولیه که حاوی یک پیکره متوالی از کاراکترهای در بردارنده فایل است، آن را به مجموعه‌های آموزش، آزمون و ارزیابی تقسیم می‌کنیم. چون خروجی مدل مولد در هربار اجرا یک توالی خواهد بود، طول توالی‌های مجموعه داده معیار مناسبی برای تقسیم‌بندی آن به مجموعه‌های سه‌گانه یاد شده است به نحوی که هر مجموعه توزیع یکسانی از فراوانی طول‌های مختلف باشد. از مجموعه‌های آموزش و ارزیابی در هنگام آموزش مدل و از مجموعه آزمون در هنگام تولید داده‌های جدید از روی مدل، استفاده خواهیم کرد. تفکیک مجموعه داده به سه مجموعه



(الف) مدل ۱

(ب) مدل ۲

(پ) مدل ۳

(ت) مدل ۴

شکل ۴ـ۳: گراف محاسباتی مدل‌های عصبی ژرف جدول ۴ـ۱ . اعداد داخل مستطیل‌ها اندازه ماتریس‌های ورودی و خروجی هر لایه را نشان می‌دهند. گراف‌ها با استفاده از توابع کتابخانه Keras [۵۰] تولید شده‌اند.

آموزش، ارزیابی و آزمون در وظایف یادگیری ماشینی یک امر بدیهی است، اما استفاده از آن برای آزمون فازی و تولید داده‌های آزمون جدید تا قبل از این، انجام نشده و چنان‌چه در ادامه خواهیم دید، ایده جدیدی است. در واقع ما از داده‌های مجموعه آزمون برای تولید داده‌های آزمون جدید استفاده خواهیم کرد.

هر مجموعه بدین ترتیب حاوی تعدادی فایل است و هر فایل یک توالی از کاراکترهای ASCII است. شروع و پایان هریک از این توالی‌ها پیش از فرایند آموزش، با توکن‌های شروع و پایان که نشانه‌های مجزایی (بیرون از مجموعه داده) هستند برچسب‌گذاری می‌شود. البته بسیاری از قالب‌های فایل چنین توکن‌هایی را به‌عنوان بخشی از ساختار خود دارند. برای مثال فایل‌های HTML با برچسب $< html >$ شروع و با برچسب $< /html >$ خاتمه می‌یابند. فایل‌های XML و PHP نیز چنین هستند. در این حالت نیازی به افزودن توکن‌های شروع و پایان جدید نیست و می‌توان از همین توکن‌ها استفاده کرد. دلیل افزودن توکن‌های شروع و پایان در این مرحله، به نحوه ساخت مدل‌های زبانی عصبی باز می‌گردد. در واقع این مدل‌ها با دیدن توکن آغازین پیش‌بینی را شروع و با تولید توکن پایانی، پیش‌بینی را خاتمه می‌دهند.

به‌منظور آموزش هر مدل چند به یک و ایجاد یک مدل زبانی عصبی، ابتدا با الحاق محتوای فایل‌های مجموعه آموزش، یک توالی بزرگ از کاراکترها به‌شکل $S =< s^{(1)}, s^{(2)}, s^{(3)}, ..., s^{(n)} >$ ایجاد می‌کنیم. سپس این توالی را به تعدادی توالی ورودی کوچکتر با طول ثابت d می‌شکنیم طوری که iامین توالی ورودی برابر است با:

$$x_i = S[i * j : (i * j) + d] \qquad (۱ـ۴)$$

در رابطه ۴ـ۱ ، $s[l, u]$ برشی از توالی s بین شاخص‌های l و u بوده و j میزان پرش به جلو در انتخاب توالی ورودی بعدی از توالی اصلی را نشان می‌دهد. ابرپارامتر d در اینجا، در واقع همان بردار زمینه در مدل زبانی است. انتخاب d به پارامترهایی مانند فاصله وابستگی‌های بین بایت‌های مختلف در یک فایل و طول فایل‌های مجموعه داده، بستگی دارد. وقتی از LSTM به عنوان بلوک‌های سازنده RNN در ساخت NLM استفاده شود، می‌توان وابستگی‌های طولانی‌تر را به خوبی یاد گرفت. در عمل و برای وظایف مدل زبانی، این مقدار را بین ۴۰ تا ۱۰۰ نشانه در نظر می‌گیرند.

گفتیم شبکه عصبی در حالت بانظارت آموزش می‌بیند؛ یعنی، برای هر ورودی یک برچسب خروجی نیاز است. در اینجا کاراکتر بعدی، که هدف پیش‌بینی آن است، را به عنوان برچسب خروجی برای هر توالی ورودی نسبت می‌دهیم. بنابراین برچسب خروجی برای iامین توالی ورودی برابر خواهد بود با:

$$Y_{x_i} = S[(i * j) + d + ۱] \qquad (۲ـ۴)$$

پس از تولید همه نمونه‌های آموزشی و برچسب‌های متناظر آنها، مدل به صورت انتهابه‌انتها آموزش داده می‌شود؛ مشابه روش یادگیری و فاز [۱۱] . یعنی، داده‌های خام مجموعه آموزش، بدون استخراج هیچ‌گونه ویژگی خاصی به شبکه داده می‌شوند و در زمان استفاده نیز، شبکه داده‌های نهایی را بدون نیاز به هیچگونه پساپردازشی تولید می‌کند. در این حالت مدل قادر پیش‌بینی مقدار احتمال شرطی $p(x^{(i+d+1)}| < x^{(i)},...,x^{(i+d)} >)$ خواهد بود. روند تولید نمونه‌های آموزشی در الگوریتم ۴ـ۱ نشان داده شده است. همچنین شکل ۴ـ۴، یک فایل HTML و سه‌توالی آموزشی اول تولید شده برای آن توسط الگوریتم ۴ـ۱ را به همراه موقعیت قرارگیری آنها در شبکه نشان می‌دهد. توالی ورودی کلی در این مثال، همه کاراکترهای فایل HTML و پارامترهای d و j به‌ترتیب برابر ۳ و ۱ قرار داده شده‌اند. مقادیر در نظر گرفته شده کاملاً فرضی هستند.

شبکه عصبی با مقادیر عددی کار می‌کند. بنابراین بایستی یک نمایش عددی برای هر کاراکتر در مجموعه آموزش در نظر گرفت. یک روش مرسوم در مدل‌های مبتنی بر کاراکتر استفاده از بردارسازی One-hot یا ۱ از k است. در این روش یک بردار به طول اندازه مجموعه واژگان برای هر کاراکتر درنظر می‌گیرند که تنها یکی از شاخص‌های آن برابر ۱ است. بدین ترتیب برای نمایش V کاراکتر، V بردار متفاوت خواهیم داشت. در اینجا نیز از همین روش استفاده می‌کنیم.

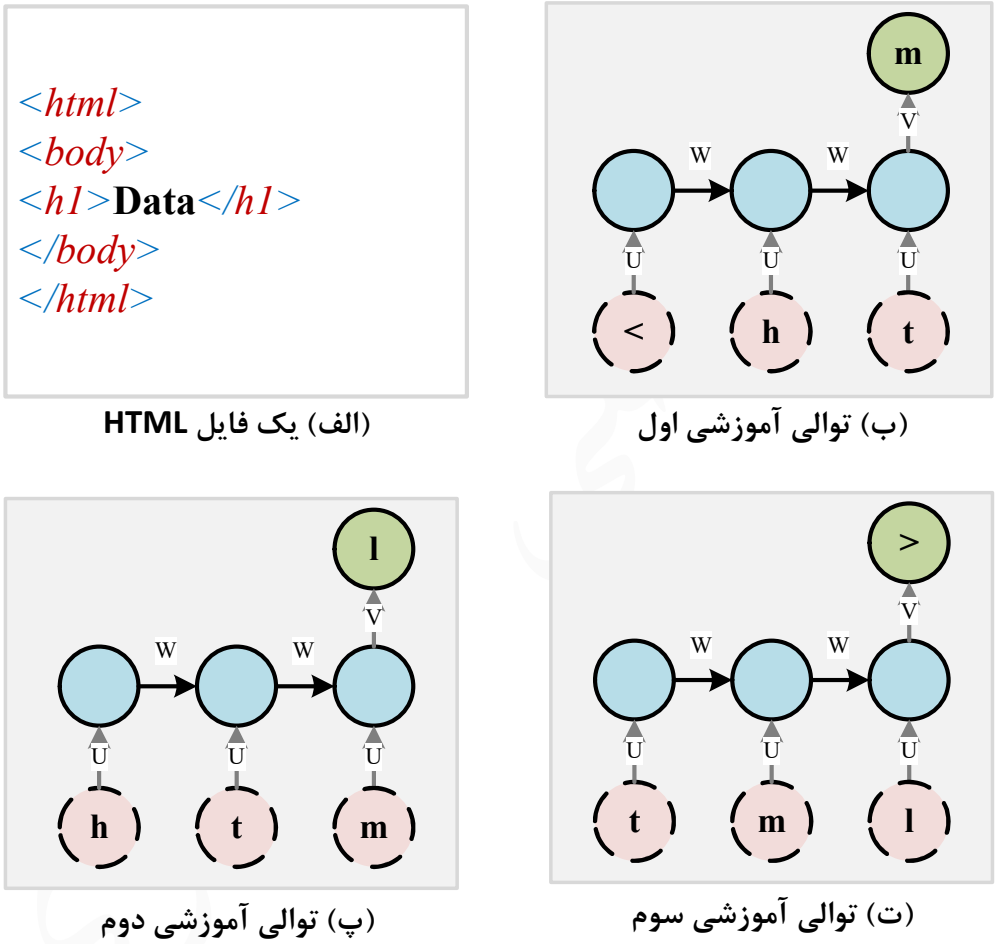
خروجی شبکه نیز که به‌شکل یک کاراکتر تعریف شده است در واقع یک بردار One-hot است. در فرایند آموزش تنها یکی از شاخص‌های این بردار یک است. اما در مرحله آزمون بردار تولیدی حاوی مقادیر حقیقی مختلفی در بازه $(-\infty,+\infty)$ برای هر شاخص است. با اعمال تابع بیشینه هموار (رجوع شود به رابطه ۲ـ۱۲)، در خروجی این لایه، می‌توان خروجی را به یک توزیع احتمالی معتبر تبدیل کرد.

الگوریتم ۴ـ۱ ProduceTrainingSamples

Input: Total sequence S, Input sequences lenght d, Jump step j

Output: Input sequences x, Output labels Y

```
x ← List()                                   /* Make an empty list */
Y ← List()                                   /* Make an empty list */
for i ← 1 to Len(S) − d do
    x[i] ← S[i ∗ j : (i ∗ j) + d]
    Y[i] ← S[(i ∗ j) + d + 1]
end
Return x, Y
```



(الف) یک فایل HTML

(ب) توالی آموزشی اول

(پ) توالی آموزشی دوم

(ت) توالی آموزشی سوم

شکل ۴ـ۴: مثالی از نحوه آموزش مدل‌های چند به یک روی یک فایل HTML ساده. تنها سه‌توالی آموزشی اول در این شکل نشان داده شده است. اما الگوریتم ۴ـ۱ کل فایل را به توالی‌های آموزشی تقسیم می‌کند.

۴ـ۲ـ۲ تولید داده‌های جدید

پس از اتمام فرایند آموزش، مدل برای تولید داده‌های جدید مورد پرس‌وجو قرار می‌گیرد. در این مرحله ابتدا یک پیشوند ثابت از کاراکترهای شروع کننده یک توالی از **فایل‌های مجموعه آزمون** به طول d، که توکن شروع در ابتدای آن است، به شبکه خورانیده می‌شود[۱]. شبکه توزیع احتمالی کاراکتر $d+۱$ام را به‌عنوان خروجی تولید می‌کند که شامل یک بردار One-hot از مقادیر احتمال‌های تمام کاراکترهای دیده شده هنگام آموزش است. سپس یک کاراکتر از این توزیع احتمالی، انتخاب شده و به پیشوند ورودی الحاق می‌گردد. در این حالت طول پیشوند $d+۱$ است. در مرحله بعد سمت چپ‌ترین کاراکتر پیشوند حذف شده و d کاراکتر باقی مانده دومرتبه به شبکه داده می‌شود تا توزیع احتمالی کاراکتر $d+۲$ حاصل شود. این روند تا زمان تولید توکن پایانی که خاتمه یافتن یک توالی را پیش‌بینی می‌کند، ادامه خواهد داشت. در [۱۱] سه راهبرد تولید داده جدید از توزیع احتمالی پیش‌بینی شده توسط مدل معرفی شده که در بخش ۳ـ۳ـ۲ آنها را دیدیم.

در اینجا ما از راهبرد نمونه‌برداری که روشی مرسوم برای تولید داده از یک توزیع آماری است استفاده می‌کنیم. یعنی هر بار از توزیع احتمالی پیش‌بینی شده به عنوان یک توزیع توزیع چندجمله‌ای[۲] نمونه‌برداری می‌نماییم. البته راهبردهای جدیدی را نیز معرفی خواهیم کرد. در زبان پایتون و با استفاده از بسته numpy می‌توان به‌صورت زیر از یک توزیع چندجمله‌ای نمونه برداری کرد:

```
1 import numpy as np
2 sample = np.random.multinomial(n, prob_vals, size=None)
```

برنامه ۴ـ۱: نمونه‌برداری از توزیع چندجمله‌ای در پایتون.

پارامتر تنوع

در [۱۱] انتخاب حریصانه به‌دلیل وجود پیشوند ثابت راهبرد ناکارمدی معرفی شده است. در روش پیشنهادی ما، مدل پیشوندهای متغیری را می‌پذیرد. بنابراین می‌توانیم از راهبرد حریصانه هم استفاده نماییم. با این حال تعداد داده‌های تولیدی در این راهبرد محدود به تعداد نمونه‌های مجموعه آزمون است که البته تعداد کمی نخواهند بود.

[۱] دقت شود که ویژگی فایل‌های مجموعه آزمون آن است که قبلاً در فرایند آموزش توسط مدل مشاهده نشده‌اند و این امر تأثیر مستقیمی در ارزیابی خوب بودن مدل و تنوع داده‌های تولید شده توسط مدل دارد. بدون وجود مجموعه آزمون نمی‌توانیم معیارهایی مانند سرگشتگی را به‌درستی حساب کنیم. به‌همین سبب جداسازی مجموعه آزمون را تأکید می‌کنیم.

[2] Multinomial Distribution

در مقابل راهبرد حریصانه، نمونه‌برداری باعث ایجاد اشیای متنوع می‌شود که لزوماً خوش‌شکل نیستند. به‌همین دلیل پارامتری به‌نام تنوع[1] را با هدف کنترل داشتن برروی تنوع داده‌های تولیدی مطرح می‌کنیم. تنوع D به‌صورت یک عدد حقیقی در بازه $(0, +\infty)$ تعریف می‌گردد. در هنگام آزمون مدل، مقادیر حقیقی پیش‌بینی شده توسط مدل، ابتدا بر D تقسیم می‌شوند و سپس تابع بیشینه همواره به آن اعمال می‌شود تا بردار خروجی حاصل گردد. در نتیجه هرچه‌قدر مقدار D کوچک‌تر (نزدیک به صفر) انتخاب شود، نمونه‌برداری به راهبرد حریصانه نزدیک شده، داده‌های خوش‌شکل‌تری تولید می‌گردد اما از تنوع آن کاسته می‌شود. بالعکس هرچه قدر مقدار D بزرگتر از یک انتخاب شود، اختلاف بین مقادیر پیش‌بینی شده کم، تنوع داده‌های تولید شده زیاد و متقابلاً احتمال بدشکل بودن آنها افزایش می‌یابد.

n ـ فهرست بهتر

یک راهبرد دیگر تولید داده جدید، استفاده از n ـ فهرست بهتر است. در این راهبرد در هر بار پیش‌بینی مدل پس از اعمال تابع بیشینه هموار، تعداد n احتمال بالای بردار خروجی انتخاب و نگهداری می‌شود. سپس با هریک از این احتمال‌ها پیش‌بینی مرحله بعد انجام می‌شود. برای انتخاب n ـ فهرست بهتر بین دو پیش‌بینی مقادیر احتمال‌های هر دو فهرست در یکدیگر ضرب و n حاصل‌ضرب بالاتر انتخاب می‌شود. پس از B مرحله، n پیشـوند Bتـایی خواهیم داشت که پیشوند با بالاترین احتمال پیشوند خروجی خواهد بود. این راهبرد البته زمان و حافظه مصرفی بالایی نیاز دارد. لذا مقادیر n و B بایستی کوچک انتخاب شوند. در ترجمه ماشینی معمولاً از این راهبرد نیز استفاده می‌شود. ما در این پایان‌نامه و در آزمایش‌های خود، راهبرد بالا را ارزیابی نکردیم، اما می‌توان از آن در عمل استفاده کرد.

۴ـ۳ الگوریتم‌های فاز عصبی

این بخش را می‌توان هسته روش پیشنهادی و نوآوری‌های ارایه شده در این پایان‌نامه دانست. در اینجا توضیح می‌دهیم که داده‌های آزمون چگونه با روشی خودکار و ترکیبی تولید و فاز می‌شوند. هنگامی که داده‌های آزمون را با استفاده از مدل‌های بخش ۴ـ۲ و راهبردهای معرفی شده در بخش ۴ـ۲ـ۲ تولید می کنیم یک تنوع ذاتی در داده‌ها وجود خواهد داشت و در هر صورت داده‌های آزمون جدید محسوب می‌شوند. اما اهداف یادگیری قالب فایل و آزمون فازی قالب فایل در تضاد بایکدیگر هستند. هدف الگوریتم یادگیری ایجاد داده‌های خوش‌شکل است که بتواند از تجزیه‌گر اولیه عبور و مسیرهای عمیق را اجرا کند و هدف آزمون فازی بدشکل

[1] Diversity

کردن ورودی است به‌نحوی که تجزیه‌گر و دیگر قسمت‌های کد اجرا شده را به حالت خرابی ببرد. در این بخش، ما الگوریتم‌های فاز عصبی را با هدف ایجاد یک مصالحه بین دو هدف قبلی و تولید نهایی داده آزمون برای یک فازر قالب فایل، معرفی می‌کنیم.

فاز کردن داده آزمون همزمان با تولید آن از روی مدل صورت می‌گیرد و مرحله مجزایی نیست. الگوریتم‌های ۴ـ۲ و ۴ـ۳ نحوه تولید داده آزمون که در بخش ۴ـ۲ـ۲ بحث شد و فاز همزمان آن را نشان می‌دهند. هر دو الگوریتم به‌عنوان ورودی مدل یادگیری شده M، پیشوند شروع توالی P (حاوی توکن شروع)، تنوع نمونه‌برداری D، نرخ فاز FR، توکن پایان ET و توکن دودویی BT را پذیرفته و داده آزمون TD را به‌عنوان خروجی باز می‌گردانند.

حلقه اصلی هر یک از الگوریتم‌ها، تا زمانی که ET تولید نشده باشد ادامه می‌یابد. چنان‌چه طول توالی تولید شده از یک حد آستانه $MaxLen$ بیشتر شد ولی ET کماکان تولید نشده باشد، آنگاه ET به‌صورت خودکار به انتهای TD اضافه می‌شود تا الگوریتم از حلقه تکرار بیرون آید. بدین ترتیب مطمئن می‌شویم که الگوریتم همواره خاتمه خواهد یافت. سپس در صورتی که مدل حضور یک داده دودویی در TD را پیش‌بینی کرده باشد (یعنی BT در توالی TD پیدا شود)، ابتدا یک بخش دودویی جایگزین BT شده و سپس این بخش با روش‌های جابه‌جایی تصادفی یا هر روش دلخواه دیگر به‌صورت مجزا فاز می‌شود و در نهایت داده آزمون نهایی توسط الگوریتم بازگردانیده می‌شود. ما در عمل برای خط ۱۹ هر دو الگوریتم، عملگر معکوس کردن یک بایت را در نظر گرفتیم. بدین ترتیب که درصد ثابتی از بایت‌های دودویی را در هر بخش دودویی انتخاب شده، به‌صورت تصادفی انتخاب نموده و مقادیر بیت‌های هر بایت انتخابی را معکوس می‌کنیم.

۴ـ۳ـ۱ فاز عصبی داده

تفاوت اصلی در الگوریتم‌های فاز عصبی داده و فاز عصبی فراداده در هنگام اعمال عملیات فاز است. در بخش ۴ـ۱ـ۲ گفتیم مدل به داده‌ها احتمال کمتری نسبت می‌دهد. بنابراین در الگوریتم ۴ـ۲، هرگاه احتمال کاراکتر پیش‌بینی شده از یک حد آستانه α کمتر باشد، متوجه می‌شویم که کاراکتر جزو بخش داده فایل بوده که در این حالت آن را با کاراکتری که کمترین احتمال را دارد جایگزین می کنیم. با این امید که کم‌احتمال‌ترین کاراکتر، ناخواسته‌ترین کاراکتر نیز بوده و ممکن است منجربه خرابی شود.

خط ۷ الگوریتم ۴ـ۲ بیان می‌دارد که برای فاز بایستی شرایط دیگری علاوه بر شرط فوق برقرار باشد. از جمله عدد تصادفی p_fuzz که لازم است از FR یعنی نرخ فاز کمتر شود. هرچه‌قدر مقدار FR بزرگتر انتخاب شود شرط $p\_fuzz < FR$ برای اعداد بیشتری تحقق می‌یابد. بدین ترتیب آزمون‌گر می‌تواند روی درصد کاراکترهای فاز شده کنترل داشته باشد. همچنین می‌خواهیم که مجموعه کاراکترهای توالی‌های ET و

BT فاز نشود؛ زیرا از آنها در قسمت‌های بعد استفاده می‌کنیم. لذا دو شرط دیگر به شرط‌های خط ۷ الگوریتم ۴ـ۲ اضافه می‌شوند. بنابراین فاز کاراکتر پیش‌بینی شده تنها در حالتی که ترکیب عطفی هر ۴ شرط درست باشد، صورت می‌گیرد.

تغییر یک بایت یا یک کاراکتر از بخش‌های داده یک فایل، معمولاً بدشکل بودن خاصی را ارائه نمی‌دهد. به طور شهودی تصور کنید که به جای عدد ۱۲۵ عدد ۹۲۵ قرار داده شود یا هر البته هر کاراکتر دیگری در موقعیت رقم ۱ . در بد شکل کردن داده هدف ما جایگزین کردن یک مقدار مرزی به جای داده موجود است. مثلاً برای مورد گفته شده جایگذاری عددی به فرم ۹...۹۹۹ خوب به نظر می‌رسد. در واقع در فاز عصبی داده، به دنبال روشی برای انتشار فاز به کاراکترهای بعدی نیز هستیم. وقتی یک کاراکتر فاز شد، دو راهکار برای تولید کاراکترهای بعدی پیش‌رو داریم. راهکار اول تولید کاراکترهای بعدی، مستقل از کاراکتر فاز شده است؛ یعنی اینکه مدل در تولید کاراکتر tام هیچ آگاهی از فاز شدن کاراکتر مرحله $t-1$ام خود نداشته باشد. برای این منظور کافی است تا کاراکتر اولیه تولید شده توسط مدل، یعنی کاراکتر قبل از فاز و جایگزین شدن، را به پیشوند شروع توالی P اضافه کنیم. بدین ترتیب مدل مستقل از اینکه کاراکتر فاز شده است یا نه به تولید داده‌های مبتنی بر گرامر (داده‌های خوش‌شکل) ادامه می‌دهد. راهکار دوم تولید کاراکترهای بعدی، وابسته به کاراکتر فاز شده است؛ یعنی اینکه اثر کاراکتر فاز شده به کاراکترهای تولید شده در مراحل بعد سرایت کند. برای این منظور کاراکتر فاز شده را به پیشوند شروع توالی P اضافه کرده و مدل را به سمت بد شکل کردن کاراکترهای بعدی متمایل می‌کنیم. این راهکار در الگوریتم فاز عصبی داده به دلیل شرح داده شده استفاده گردیده است. خط ۱۰ الگوریتم ۴ـ۲، کاراکتر فاز شده یعنی کاراکتر c را به پیشوند P اضافه می‌کند. در همین خط برای ثابت ماندن طول پیشوند که در واقع ورودی مدل است، کاراکتر اول یعنی قدیمی‌ترین کاراکتر را از پیشوند حذف می‌کنیم. در کل این خط عمل **انتشار فاز به جلو** را پیاده‌سازی می‌کند.

۴ـ۳ـ۲ فاز عصبی فراداده

در الگوریتم فاز عصبی فراداده (الگوریتم ۴ـ۳)، می‌خواهیم کاراکترهای فراداده را تغییر دهیم. برای این منظور بررسی می‌کنیم که احتمال کاراکتر پیش‌بینی شده توسط مدل یعنی $p(c)$ بیشتر از حد آستانه β باشد، در این صورت کاراکتر با کمترین احتمال را جایگزین می‌کنیم. در اینجا اما به‌دنبال تغییر تنها یکی از کاراکترها در هر توکن فراداده هستیم. لذا در خط ۱۱ الگوریتم ۴ـ۳ تغییری در پیشوند بعدی مدل ایجاد نمی‌کنیم تا مدل به همان روند تولید داده خوش‌شکل خود ادامه دهد بدون اینکه متوجه فاز شدن کاراکتر پیش‌بینی کرده گام زمانی قبلی خود باشد. در واقع در فاز عصبی فرا داده، انتشار فاز به جلو را انجام نمی‌دهیم. ایده ما برای این کار این است که در اغلب مواقع یک تغییر کوچک، مثلاً تغییر یک بایت در قسمتی که قالب فایل را بیان

می‌کند سبب گمراه شدن تجزیه‌گر می‌شود. این عمل درست در خلاف مورد داده است که یک بایت معمولاً تغییری ایجاد نمی‌کند و این مقادیر مرزی (بیش از حد کوچک، بیش از حد بزرگ، خالی یا صفر) هستند که مسبب خرابی می‌شوند. مقادیر داده معمولاً به‌صورت یک توالی از بایت‌ها ادامه می‌یابند.

در الگوریتم ۴ـ۳ پارامتر نرخ فاز FR همچنان وجود دارد، اما دو شرطی که بررسی کننده وجود توکن‌های BT و ET هستند را حذف کردیم تا علاوه بر آسان‌تر کردن شرط فاز بتوانیم آنها را نیز به‌عنوان بخشی از قالب فایل، فاز کنیم؛ زیرا، تعداد خطاهایی که در تجزیه‌گر رخ می‌دهند، معمولاً بیشتر هستند. در نهایت خطوط سایه زده شده در الگوریتم ۴ـ۳ (خطوط۷، ۸ و ۱۱) محل‌های تفاوت این الگوریتم با الگوریتم فاز عصبی داده (الگوریتم ۴ـ۲) را نشان می‌دهد.

حد آستانه $MexLen$ که حداکثر طول داده آزمون تولیدی را در الگوریتم‌های ۴ـ۲ و ۴ـ۳ مشخص می‌کند نیز به‌صورت یک عدد صحیح تصادفی در بازه $[a, b)$ مشخص می‌شود. مقادیر این بازه می‌تواند براساس میانگین طول فایل‌های مجموعه آزمون، انتخاب شود. بدین صورت که کران‌های آن برابر اختلاف واریانس طول از میانگین باشند. مقادیر ابرپارامترهای موجود در هر دو الگوریتم بایستی توسط آزمون‌گر مطابق قالب فایل مورد آزمون و نکته‌های بیان شده، تنظیم گردد. در فصل ۵ که ارزیابی روش پیشنهادی را مطرح می‌کنیم، مقادیر در نظر گرفته شده را برای قالب فایل PDF بیان می‌کنیم.

۴ـ۴ پیاده‌سازی

جزئیات پیاده‌سازی روش پیشنهادی در پیوست ب آمده است. برای پیاده‌سازی مدل‌های یادگیری ژرف از کتابخانه سطح بالای یادگیری ژرف Keras [۵۰] که به زبان پایتون نوشته شده است استفاده کردیم. این کتابخانه مجموعه‌ای مفید از توابع و APIها برای ساخت انواع مختلف شبکه‌های عصبی را در اختیار قرار می‌دهد؛ اما، برای اجرا نیاز به یک چارچوب سطح پایین دارد که ما Tensorflow [۴۷] را بدین منظور انتخاب کردیم. Tensorflow همچنین یک ابزار به نام Tensorboard دارد که از آن می‌توان برای مصورسازی گراف‌های محاسباتی و نیز مشاهده نمودارهای خطاو دقت در فرایندهای آموزش و آزمون استفاده کرد. برای آموزش مدل‌ها ما از تابع خطای CE (رابطه ۲ـ۶ در بخش ۲ـ۳ـ۳) و بهینه‌ساز Adam [۵۱] با نرخ‌های یادگیری ۱×۱۰^{-۳} و ۱×۱۰^{-۴} استفاده کردیم.

هدف این پایان‌نامه همان‌طور که پیش از این گفته شد، ارائه یک روش تولید خودکار داده آزمون است که در بخش‌های گذشته این فصل بحث شد. اما تولید داده آزمون به‌تنهایی کافی نیست و برای ارزیابی لازم است تا یک فازر قالب فایل با همه پیمانه‌های شکل ۲ـ۲ در اختیار داشته باشیم. برای این منظور نیاز به دو پیمانه تزریق کننده مورد آزمون و پایش SUT نیز داریم. اغلب نرم‌افزارهایی که یک فایل را به‌عنوان ورودی

الگوریتم ۴ـ۲ DataNeuralFuzz

Input: Learnt model M, Sequence prefix P, Diversity D, Fuzzing rate FR, End token ET, Binary token BT

Output: Test data TD

$TD \leftarrow P$
$MaxLen \leftarrow$ `RandInt` (a, b)
while *not EndsWith (TD, ET)* **do**
  $predicts \leftarrow$ `Predict` $(M(P))$
  $c, p(c) \leftarrow$ `Sample` $(predicts, D)$ `/* Sample c from the learnt model */`
  $p\_fuzz \leftarrow$ `Random` $(0, 1)$ `/* Decide whether to fuzz */`
  if $p\_fuzz < FR \wedge p(c) < \alpha \wedge c \notin Chars(BT) \wedge c \notin Chars(ET)$ **then**
    $c \leftarrow argmin_{c'}\{p(c') \in predicts\}$ `/* Fuzz c by c' where c' is the lowest likelihood */`
  end
  $TD \leftarrow TD + c$
  $P \leftarrow P[1:] + c$ `/* Propagate fuzz to prefix and next generated data */`
  if $Len(TD) > MaxLen$ **then**
    $TD \leftarrow TD + ET$
    Break
  end
end
if $BT \in TD$ **then**
  $TD \leftarrow$ `AddBinaryPart` (TD)
  $TD \leftarrow$ `MutateBinaryPart` (TD)
end
Return TD

الگوریتم ۴ـ۳ MetadataNeuralFuzz

Input: Learnt model M, Sequence prefix P, Diversity D, Fuzzing rate FR, End token ET, Binary token BT

Output: Test data TD

```
1  TD ← P
2  MaxLen ← RandInt (a, b)
3  while not EndsWith (TD, ET) do
4      predicts ← Predict (M(P))
5      c, p(c) ← Sample (predicts, D)        /* Sample c from the learnt model */
6      p_fuzz ← Random (0, 1)                /* Decide whether to fuzz */
7      if p_fuzz < FR ∧ p(c) > β then
8          c' ← argmin_c"{p(c") ∈ predicts}  /* Fuzz c by c' where c' is the
               lowest likelihood */
9      end
10     TD ← TD + c'
11     P ← P[1 :] + c                        /* Don't propagate fuzz to prefix */
12     if Len (TD) > MaxLen then
13         TD ← TD + ET
14         Break
15     end
16 end
17 if BT ∈ TD then
18     TD ← AddBinaryPart (TD)
19     TD ← MutateBinaryPart (TD)
20 end
21 Return TD
```

می‌پذیرند یک واسط خط فرمان نیز دارند که می‌توان از این طریق یک فایل را به آنها داد. در غیر اینصورت نیز می‌توان یک برنامه کوچک برای تزریق ورودی نوشت. به‌طور کلی در آزمون فازی قالب فایل تزریق ورودی ساده است. برای پایش نیز ابزارهای مستقلی وجود دارد که می‌توان از آنها استفاده کرد. ما آزمون فازی را تحت سیستم عامل ویندوز انجام دادیم[۱] و ابزار Application Verifier [۳۶] که در بخش ۲ـ۲ـ۲ معرفی شد را برای پایش انتخاب کردیم که مایکروسافت استفاده از آن را در SDL توصیه کرده است.

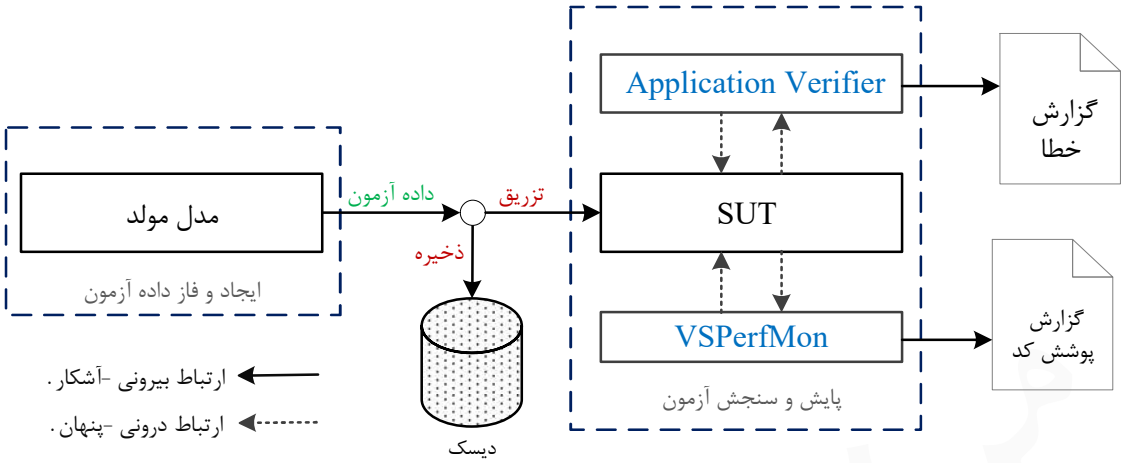
Application Verifier فایل اجرایی SUT را می‌گیرد و در هر بار اجرای SUT یک فایل حاوی وقایع را با یک شماره ترتیبی ثبت می‌کند. در صورتی که برنامه دچار خطای زمانِ اجرا (شامل یکی از انواع خطاهای فساد حافظه، دسترسی غیر مجاز و غیره) شود، نوع خطا در این فایل ثبت و ضبط می‌گردد. سپس می‌توان برنامه را به‌همراه داده آزمون مسبب خطا (که از روی شماره ترتیبی فایل وقایع قابل تشخیص است) در یک محیط اشکال‌زدا مانند WinDbg مجدداً اجرا و ضمن مشاهده خطا، محل دقیق آن را آشکار ساخت.

از سازوکاری که در بالا توضیح داده شد برای آزمون فازی نرم‌افزار MuPDF که یک نرم‌افزار پر استفاده است و قالب پیچیده PDF را به‌عنوان ورودی می‌پذیرد، در آزمایش‌ها استفاده کرده‌ایم. ما همچنین پوشش کد این نرم‌افزار را با استفاده ابزار VSPerfMon اندازه‌گیری کردیم تا عملکرد تولید کننده داده آزمون را بسنجیم. شکل ۴ـ۵ یک طرح‌واره کلی از معماری (مؤلفه‌ها و ارتباط بین آنها) فازر پیشنهادی ما در این پایان‌نامه را نشان می‌دهد. با تعویض پیمانه پایش، این فازر در سیستم‌های عامل دیگر نیز قابل اجرا خواهد بود. این فازر را می‌توان در گروه فازرهای جعبه خاکستری، مبتنی بر روش ترکیبی تولید داده آزمون و بدون حلقه بازخورد به‌شمار آورد. VSPerfMon برای اندازه‌گیری پوشش کد در حالت جعبه‌سفید استفاده می‌شود که در صورت عدم وجود کد منبع SUT بایستی آن را با یکی از ابزارهای سنجش پوشش کد باینری در بخش ۲ـ۱ـ۳ جایگزین کرد.

۴ـ۵ خلاصه

مدل‌های زبانی عصبی در سطح کاراکتر قابلیت یادگیری یک توزیع آماری روی یک توالی از کاراکترها را دارند. در این فصل ما ساختار یک فایل را به صورت یک توالی از کاراکترها (بایت‌ها) مدل کرده و با ارائه چندین مدل زبانی عصبی و چندین راهبرد تولید داده‌های جدید از این مدل‌ها، یک روش تولید خودکار داده آزمون ارائه دادیم که قابل استفاده در فازرهای قالب فایل مختلف است. ما همچنین دو الگوریتم با عنوان‌های فاز داده عصبی و فاز فراداده عصبی ارائه کردیم. هر کدام از این الگوریتم‌ها بدشکل‌سازی داده آزمون ورودی

[۱] هرچند که روش پیشنهادی کاملاً مستقل از سیستم عامل است. برنامه تولید خودکار داده آزمون به زبان پایتون نوشته شده و روی سکوهای مختلف قابل اجرا است.



شکل ۴ـ۵: معماری فازر قالب فایل پیشنهادی. فازر به‌صورت کاملاً پیمانه‌ای پیاده‌سازی شده است. پیکان‌ها ارتباطات بین مؤلفه‌ها را نشان می‌دهند. ارتباط بیرونی، سازوکار ارتباطی است که توسط ما پیاده‌سازی شده یا قابل تغییر است و ارتباط درونی سازوکار ارتباطی است که توسط پیمانه‌های شخص ثالث استفاده می‌شود.

را با هدف خاصی انجام می‌دهند: اولی با هدف جابه‌جایی داده‌های یک فایل به منظور ایجاد خطا در هنگام پرداخت آنها و دومی با هدف جابه‌جایی فراداده‌های یک فایل برای اینجا خطا در تجزیه‌گر اولیه ساختار فایل تحت فاز.

در بخش پایانی فصل نیز یک فازر قالب فایل را طراحی و پیاده‌سازی کردیم که از آن می‌توان برای آزمون فازی برنامه‌هایی با ورودی فایل استفاده کرد. خروجی این فازر علاوه بر گزارش خطاهای یافت شده حین فرایند آزمون، گزارش میزان پوشش کد و ذخیره کلیه داده‌های آزمون تولید شده نیز است. معماری فازر پیشنهادی کاملاً پیمانه‌ای بوده و با تعویض پیمانه پایش، امکان استفاده از آن در سیستم‌های عاملی به‌جز ویندوز فراهم می‌شود. فازر معرفی شده یک فازر جعبه خاکستری و مجهز به یک روش تولید داده آزمون ترکیبی است. در فصل ۵، به ارزیابی روش پیشنهادی و بررسی یافته‌های حاصل از آن می‌پردازیم.

فصل ۵

ارزیابی روش پیشنهادی

«من شکست نخورده‌ام. من فقط ۱۰,۰۰۰ راه پیدا کردم که کار نمی‌کند.»

✠ توماس ادیسون

ما روش پیشنهادی خود را روی قالب فایل PDF به‌عنوان یک قالب فایل با ساختار پیچیده و ترکیبی و نرم‌افزار متن‌باز و رایگان MuPDF [۲۵] مورد ارزیابی قرار دادیم. در این فصل به تشریح چیدمان آزمایش‌ها، معیارهای ارزیابی، یافته‌های حاصل از آزمایش‌ها و نتیجه‌گیری از آنها می‌پردازیم. هدف علاوه بر اثبات کارایی و خوب بودن روش پیشنهادی، شناسایی و تفکیک پارامترهای مهم در هنگام استفاده از فنون یادگیری ماشینی در آزمون فازی و تولید خودکار داده آزمون است. استفاده از یادگیری ماشینی در آزمون فازی حوزه پژوهشی بدیع و ناشناخته‌ای است. بنابراین در این فصل معیارهایی را که در هر روش پیشنهادی بایستی مورد ارزیابی قرار بگیرند، نیز مشخص خواهیم نمود. سپس روش پیشنهادی خود را با آنها مورد ارزیابی قرار می‌دهیم.

۵ـ۱ مورد مطالعاتی

نرم‌افزار MuPDF [۲۵] مجموعه غنی از کتابخانه‌ها، APIها و ابزارهای کاربردی برای کار با فایل‌های PDF است. این نرم‌افزار مشتمل بر سه قسمت API، ابزار mutool و ابزار PDF-خوان است. ابزار mutool برای ایجاد و تغییر فایل‌های PDF استفاده می‌شود. ابزار PDF-خوان نیز برای نمایش فایل‌های PDF به‌کار می‌رود. بسته نرم‌افزاری کامل، به زبان C نوشته شده و در سیستم عامل‌های ویندوز، لینوکس و Mac و همچنین اندروید قابل استفاده است. به‌دلیل قابلیت حمل بالا و کاربرد زیادی که دارد، پیدا کردن خطا در آن حائز اهمیت است. در

آزمایش‌ها ما ابزار PDFخوان نسخه ویندوز را مورد آزمون قرار دادیم. منظور از عبارت MuPDF در ادامه این فصل همان ابزار PDFخوان است.

MuPDF البته در نسخه‌های جدید قالب‌های فایل دیگری را نیز پشتیبانی می‌کند؛ از جمله قالب‌های فایل XPS و EPub. اما قالب فایل PDF بدون شک رایج‌ترین قالب اسناد و انتشارات الکترونیکی است که بیشتر مردم از آن استفاده می‌کنند. از این‌رو تضمین کیفیت و امنیت PDFخوان‌ها به‌ویژه در تلفن‌های همراه اهمیت دو چندانی دارد. افزون بر آنچه گفته شد قالب فایل PDF و نرم‌افزار MuPDF تمامی مسائلی را که در بخش ۱ـ۲ بدان اشاره کردیم، یعنی ساختار پیچیده ورودی، ساختار پیچیده کد و غیره، دارند و از طرفی در فصل ۳ دیدیم که فازرهای مبتنی بر جابه‌جایی مانند AFL و AFLافزوده نتواسته بودند به پوشش کد خوبی حین آزمون فازی این نرم‌افزار دست‌یابند. بنابراین انتخاب آنها به‌عنوان مورد مطالعاتی به‌دلایل بالا و نیز در جهت فراهم شدن امکان مقایسه با کارهای پیشین، قابل توجیه است. آزمایش‌ها برای هر قالب فایل و هر نرم‌افزار دیگری به‌نحوی که در ادامه این فصل توضیح داده می‌شود قابل اجرا است.

ساختار قالب فایل PDF در پیوست آ به‌طور کامل بررسی شده است. بخش عمده این ساختار را اشیای متنی تشکیل می‌دهند. از مدل‌های معرفی شده در بخش ۴ـ۲ برای یادگیری ساختار اشیای متنی استفاده می‌کنیم؛ زیرا یادگیری فایل PDF به‌صورت یکجا با توجه به اینکه قسمت‌هایی از آن برای آدرس‌دهی استفاده می‌شوند امکان‌پذیر نیست. اما MuPDF یا هر PDFخوان دیگری فایل‌های کامل PDF را به عنوان ورودی می‌پذیرد به همین لازم است تا اشیای تولید شده را به یک فایل تبدیل کنیم. برای این کار اشیای جدید را با استفاده از سازوکار توضیح داده شده در پیوست آ در مورد بروزرسانی یک فایل PDF، به یک فایل معتبر از پیش موجود، که آن را میزبان[1] می‌نامیم، اضافه می‌کنیم. در نتیجه یک فایل PDF جدید در اختیار خواهیم داشت که ساختار آن معتبر بوده و برخی اشیای آن تغییر داده شده و بازنویسی شده‌اند.

۵ـ۲ معیارهای ارزیابی

در تولید خودکار داده آزمون به روش یادگیری ژرف، تعداد زیادی پارامتر وجود دارد که می‌توان تأثیر آنها را در فرایند آزمون فازی سنجید. اما مهم‌ترین هدف در فرایند آزمون فازی همان‌گونه که پیش از این هم ذکر شد یافتن خطا در SUT است. هدف مهم بعدی افزایش میزان پوشش کد برای نیل هر چه بیشتر به هدف اول است. هدف دیگر که مختص به این روش است یادگیری هر چه بهتر ساختار فایل ورودی است. براساس اهداف اشاره شده، معیارهای ارزیابی زیر را در این فصل لحاظ کرده‌ایم:

- **خطا و دقت مدل‌های یادگیری ژرف.** شامل خطا و دقت هر مدل روی مجموعه‌های آموزش و ارزیابی،

[1] Host

میزان و نحوه تغییر آنها با گذشت زمان. این معیارها توسط کد نوشته شده به ابزار Tensorboard گزارش شده و در حین فرایند آموزش یا در پایان آن قابل مشاهده هستند.

- **سرگشتگی.** برای ارزیابی میزان خوب بودن مدل زبانی در پیش‌بینی نشانه بعدی توالی داده شده. معیار سرگشتگی از رابطه ۲ـ۱۴ و با جایگذاری خطای مدل روی مجموعه ارزیابی، برای آن مدل محاسبه می‌گردد.

- **پوشش کد.** در هر اجرای SUT، تعداد خطوط برنامه و نیز تعداد بلوک‌های اولیه‌ای که اجرا شده‌اند، شمارش می‌شود. با داشتن کل خطوط و بلوک‌های اولیه درصد پوشش کد نیز قابل محاسبه است. برای یک مجموعه آزمون، یعنی چندین بار اجرای متوالی SUT پوشش کد عبارت خواهد بود از اجتماع پوشش کد تک‌تک هریک از اجراها. بنابراین منظور از اصطلاح کلی پوشش کد در ادامه پوشش بلوک‌های پایه است.

- **خطا و آسیب‌پذیری.** SUT تحت ابزار پایش Application Verifier اجرا می‌شود که خطاهای احتمالی را ثبت و شماره‌گذاری می‌کند. سپس می‌توان با تحلیل این خطاها در یک محیط اشکال زدا، ضمن تعیین مکان و علت خطا، امکان قابل بهره‌برداری بودن آنها را بررسی کرد.

آزمایش‌ها بایستی به‌گونه‌ای طراحی شوند که هریک از آنها تأثیر یک پارامتر خاص را بر روی معیارهای بالا، به‌طور واضح نمایان سازد؛ اما، پارامـــترها کدامند؟ ما در این بخش، چندین پارامتر و مقادیر قابل آزمایش برای هریک را شناسایی و معرفی می‌کنیم. فهرستی از این پارامترها و مقادیری که می‌توانند بپذیرند در جدول ۵ـ۱ ذکر شده است. همان‌طور که مشاهده می‌شود برخی از پارامترها مقادیری پیوسته و برخی دیگر مقادیری گسسته می‌پذیرند. بر این باور هستیم که هر روش تولید خودکار داده آزمون با استفاده از یادگیری ماشینی روی چنین پارامترهایی بنا می‌شود، لذا شناسایی و تفکیک این پارامترها پیش از هرگونه طرح آزمایشی لازم و ارزشمند است.

نوع مدل در سطر اول جدول ۵ـ۱ برای هر مدل، خود دارای اَبَرپارمترهایی مانند تعداد عصب، لایه، نرخ یادگیری و غیره است که در فصل قبل به آنها اشاره شد. بنابراین جدول ۵ـ۱ یک دید جامع از همه پارامترهایی که تا اینجا معرفی شدند، در اختیار می‌گذارد. در بخش بعد چیدمان آزمایش‌ها خود را براساس این پارامترها شرح خواهیم داد.

جدول ۵ـ۱: پارامترهای مؤثر در تولید خودکار داده آزمون با روش‌های یادگیری ماشینی و مقادیر قابل آزمایش برای آنها.

ردیف	پارامتر	مقادیر قابل آزمایش
۱	نوع مدل	• انواع مدل‌های یادگیری ژرف
۲	راهبرد	• حریصانه • نمونه برداری • ترکیبی (حریصانه + نمونه‌برداری) • چند فهرست بهتر
۳	تنوع (D)	• $(۰,\infty)$
۴	روش فاز	• بدون فاز • تصادفی • مبتنی بر مدل • ترکیبی
۵	الگوریتم فاز	• فاز داده • فاز فرادادە
۶	تعداد دوره آموزش	• $\{۱, ۲, ۳, ...\}$
۷	نوع فایل	• متنی • متنی + دودویی
۸	استفاده از میزبان	• استفاده • عدم استفاده
۹	تعداد میزبان	• $\{۱, ۲, ۳, ...\}$
۱۰	میزان تغییر میزبان	• ثابت • متغیر

جدول ۵ـ۲: مشخصه‌ها و زمان آموزش مدل‌های جدول ۴ـ۱ در فرایند آموزش.

پارامتر / شماره مدل	۱	۲	۳	۴
طول توالی‌های ورودی (d)	۵۰	۵۰	۵۰	۵۰
گام پرش (j)	۳	۳	۱	۱
تعداد دوره	۵۰	۵۰	۵۰	۵۰
زمان تقریبی یک دوره (ساعت: دقیقه)	۱:۰۰′	۱:۴۵′	۵:۴۵′	۹:۳۰′
حجم تقریبی مدل (مگابایت)	۱٫۲۴	۲٫۷۶	۹٫۹۹	۵٫۴۱

۵ـ۳ چیدمان آزمایش‌ها

برای آموزش مدل‌های یادگیری معرفی شده در بخش ۴ـ۲ (جدول ۴ـ۱) و تولید داده‌های آزمون از یک سیستم فیزیکی با پردازنده گرافیکی NVidia GTX 1080، پردازنده مرکزی Intel Core i7 و ۲۰ گیگابایت حافظه اصلی به همراه سیستم عامل Ubuntu 16.04 x64 استفاده کردیم. آزمون‌های فازی را نیز روی ماشین مجازی با پردازنده Intel Core i7 و ۸ گیگابایت حافظه اصلی به همراه سیستم عامل Windows 10 x64 انجام دادیم. ما همچنین از نسخه نهایی MuPDF در زمان انجام آزمایش‌ها یعنی نسخه MuPDF 1.11(2017-04-11) Final برای آزمون فازی استفاده کردیم. باتوجه به نهایی بودن نسخه انتظار می‌رود خطاهای نسخه تا حد زیادی، در مقایسه با نسخه‌های آلفا، بتا و RC برطرف شده باشند و لذا شناسایی خطا کار سخت‌تری خواهد بود. مشخصه‌های کامل مدل‌ها در فرایند آموزش به‌همراه تعداد دوره و زمان آموزش هر دوره، در جدول ۵ـ۲ آمده است.

پیچیدگی مدل‌های جدول ۵ـ۲ با افزایش شماره، افزایش می‌یابد. یعنی مدل ۲ از مدل ۱ پیچیده‌تر درنظر گرفته شده است. همان‌طور که در جدول هم مشخص است مدل‌های پیچیده‌تر زمان آموزش طولانی‌تری دارند. برای مدل‌های پیچیده‌تر منطقی است که نمونه‌های آموزشی بیشتری داشته باشیم. در نتیجه برای مدل‌های ۳ و ۴ گام پرش را ۱ و برای مدل‌های ۱ و ۲ گام پرش را ۳ در نظر گرفتیم. همچنین برای مدل شماره ۳ از روش منظم‌سازی Dropout [۴۲] $p = ۰٫۳$ استفاده کردیم. در نهایت هر مدل را دست‌کم برای ۵۰ دوره روی داده‌های مجموعه آموزش، آموزش دادیم.

۵ـ۳ـ۱ مجموعه داده

آموزش موفق شبکه‌های عصبی ژرف مستلزم داشتن مجموعه داده به‌اندازه کافی بزرگ و مناسب است. در یادگیری ماشینی هرچه‌قدر تعداد داده‌ها بیشتر باشد، یادگیری بهتر انجام خواهد شد. هنوز در بسیاری از زمینه‌ها

مجموعه داده کافی برای آموزش وجود ندارد و این یکی از محدودیت‌های استفاده از شبکه‌های عصبی ژرف است. برای یادگیری آماری ساختار اشیای فایل PDF پیکره بزرگی از فایل‌های PDF جمع‌آوری کردیم؛ زیرا چنین پیکره‌ای از قبل وجود نداشت. بخشی از این پیکره شامل مجموعه داده‌های آزمون PDF خوان Mozilla[1] است که در مرورگر وب Firfox و دیگر پروژه‌های این شرکت استفاده می‌شود و به‌صورت رایگان دردسترس است. بخش دیگری از آن PDFهای استفاده شده در فازرهای دیگر مثل AFL است و بلأخره بخشی بزرگی نیز از طریق وب جمع‌آوری شد به‌نحوی که تنوع خوبی از لحاظ اندازه و محتوا داشته باشند. پیکره نهایی در حال حاضر شامل ۶۱۰۰ فایل PDF مختلف و طبقه‌بندی شده است که ما آن را تحت نام IUST PDF Corpus[2] منتشر کرده‌ایم و از آن می‌توان در موارد دیگر نیز استفاده کرد.

از فایل‌های داخل این پیکره که اندازه آنها بین ۱ تا ۹۰۰ کیلوبایت و تا حداکثر ۷۹۳۵ کیلوبایت متغیر است، در حدود ۵۰۴۱۵۳ تعداد شیء داده‌ای PDF با طول‌های گوناگون استخراج شد که به عنوان مجموعه داده جهت یادگیری ساختار اشیای داده‌ای PDF در نظر گرفته شده‌اند. این اشیاء سپس همگی در یک فایل الحاق شدند که حجم فایل نهایی در حدود ۷۰ مگابایت است. از این تعداد در حدود ۱۳۷۱۵۷ شیء حاوی جریان‌های دودویی بودند که محتوای همه این جریان‌ها پس از شناسایی با توکن دودویی stream جایگزین شدند؛ زیرا، همان‌طور که قبل از این هم اشاره کردیم نیازی به شرکت دادن آنها در فرایند آموزش نیست و بعداً مجدداً آنها را اضافه می‌کنیم.

۵ـ۳ـ۲ پیش‌پردازش داده‌ها

تعداد اشیای استخراج شده بسیار زیاد هستند. قبل از انجام عملیات یادگیری مدل بایستی تعدادی از این اشیا را به نحوی از مجموعه داده حذف کرد. برای این منظور عملیات پیش‌پردازشی طی مراحل مختلف روی مجموعه داده انجام دادیم. مهمترین ویژگی در دسترس در تنظیم ابرپارامترهای مدل یادگیری ژرف در اینجا طول اشیای داده‌ای PDF است که هم در فرایند آموزش و هم در فرایند تولید نقش تأثیرگذاری دارد. لذا داشتن توزیعی همگن از اشیای PDF برحسب طول آنها فرضیه‌ای است که در اینجا در نظر گرفته شده است و در عملیات پیش‌پردازش نیز معیار طول اشیای داده‌ای بوده است. ویژگی طول اشیاء از این منظر مهم است که شبکه LSTM بر روی توالی‌های بسیار طولانی عملکرد ضعیفی دارد و بهتر است چنین توالی‌هایی را در نظر نگیریم. مراحل پیش‌پردازش داده‌ها به‌ترتیب در زیر ذکر شده است.

۱. هر شیء داده‌ای یک موجودیت مستقل درنظر گرفته شده و ابتدا مجموعه داده برحسب طول اشیای

[1] https://github.com/mozilla/pdf.js/tree/master/test/pdfs

[2] https://github.com/m-zakeri/iust_deep_fuzz/tree/master/dataset

داخل آن مرتب گردید. سپس کلیه اشیاء داده‌ای به‌شکل

```
obj
/
endobj
```

و

```
obj
null
endobj
```

از مجموعه داده اولیه حذف شدند؛ زیرا، حاوی بار اطلاعاتی مفیدی نیستند.

۲. پس از اعمال مرحله ۱ تعداد اشیای مجموعه داده به ۴۹۴۹۷۹ کاهش یافت. اکنون به‌طور تقریبی هر صدک از مجموعه داده شامل حدود ۴۹۵۰ شیء داده‌ای PDF است. برای حذف هرچه بیشتر داده‌های پرت صدک‌های اول و آخر را نیز از مجموعه داده حذف کردیم. به‌این ترتیب در پایان این مرحله تعداد ۴۸۵۰۸۰ شیء داده‌ای PDF باقی ماند.

۳. مرحله اصلی پیش‌پردازش خارج کردن داده‌های outlier و extreme value با استفاده از معیار iqr توسط ابزار WEKA[1] [۵۲] است. ورودی این مرحله مجموعه داده مرحله ۲ است و خروجی آن حاوی تعداد ۴۷۷۱۰۴ شی داده‌ای PDF است. برای تعیین دو مقدار نامبرده یعنی outlier و extreme value به ترتیب از ضرایب ۳ برای ضریب outlier و ۶ برای ضریب extreme value استفاده شد؛ که مقادیر پیش‌فرض WEKA برای پاک‌ساز[2] iqr هستند.

۴. در این مرحله هدف تقسیم مجموعه داده به سه بخش مجموعه آموزش ، مجموعه ارزیابی و مجموعه آزمون است. از آنجایی که یادگیری مدل ما در حالت بدون نظارت انجام می‌شود[3]، از مجموعه آزمون فقط برای انتخاب پیشوند جهت تولید اشیای جدید استفاده خواهد شد. به‌عبارت دیگر مدل روی مجموعه آموزش، آموزش داده می‌شود و در هنگام تولید اشیای جدید PDF مقادیر اولیه از مجموعه آزمون برداشته می‌شود. برای این منظور ۲۵ درصد از کل تعداد اشیای PDF را برای مجموعه آزمون و ۷۵ درصد را برای مجموعه آموزش/ ارزیابی در نظر گرفتیم. برای ایجاد این تفکیک نیز از فیلتر

[1] https://www.cs.waikato.ac.nz/ml/weka/

[2] Filter

[3] برچسب‌ها از خود مجموعه آموزش برداشته می‌شوند.

جدول ۵ـ۳: نحوه و درصدهای تقسیم مجموعه داده به مجموعه‌های آموزش، ارزیابی و آزمون.

مجموعه	تعداد شیء	درصد
آموزش	۲۶۸۳۷۱	۵۶/۲۵
ارزیابی	۸۹۴۵۷	۱۸/۷۵
آزمون	۱۱۹۲۷۶	۲۵
جمع	۴۷۷۱۰۴	۱۰۰

StratifiedRemoveFolds موجود در ابزار WEKA با دانه اولیه ۱۰۱ (کوچک‌ترین عدد اول سه رقمی) و مقدار $fold = ۳$ (یعنی بعد از درهم‌آمیزی[1] و تقسیم کل مجموعه داده به ۴ قسمت با نسبت‌هایی برابر با مجموعه داده اولیه، قسمت سوم آن برای مجموعه آزمون کنار گذاشته شود) استفاده شد. در نهایت تعداد ۳۵۷۸۲۸ شی در مجموعه‌های آموزش/ ارزیابی و تعداد ۱۱۹۲۷۶ شی داده‌ای هم در مجموعه آزمون قرار گرفتند.

۵. در تکمیل مرحله ۴، نیاز است تا مجموعه آموزش/ ارزیابی جدا شده، حاوی ۳۵۷۸۲۸ شیء خود به دو مجموعه مجزای آموزش و ارزیابی تقسیم شود. برای این منظور این مجموعه آموزش/ ارزیابی به ۴ قسمت تقسیم گردید و قسمت چهارم به عنوان مجموعه ارزیابی در نظر گرفته شد. سه قسمت اول هم مجموعه آموزش را تشکیل می‌دهند. درصد تعداد اشیاء هریک از مجموعه‌های آموزش، ارزیابی و آزمون در جدول ۵ـ۳ آمده است.

۶. آخرین مرحله پیش‌پردازش نرمال‌سازی[2] متن است. در این مرحله برخی کاراکترهای بسیار کم تکرار (فراوانی نسبی نزدیک صفر) حذف یا با کاراکترهای مناسبی جایگزین شدند. در واقع طی این مرحله اندازه مجموعه واژگان کاراکترها از ۹۶ به ۶۴ تقلیل یافت. علت انجام این مرحله، آن است که کاراکترهای با تعداد کم داده پرت محسوب می‌شوند و به‌دلیل اینکه در مجموعه آموزش تعداد تکرار آنها پایین است عملاً احتمال بسیار ناچیزی به آنها تخصیص داده می‌شود که در واقع ضرورتی به این کار نیست. در عین حال با انجام این کار، حجم مکعب داده‌ها کمتر شده و فرایند آموزش سریع‌تر انجام می‌شود.

[1] Shuffle

[2] Normalization

۵ـ۳ـ۳ انتخاب فایل‌های میزبان

در [۱۱] انتخاب فایل‌های میزبان به‌صورت انتخاب سه فایل با کمترین حجم از پیکره انجام گرفته است. در واقع این انتخاب بدون هدف خاص و کاملاً تصادفی صورت گرفته است. پیرو روش آنها ما نیز برای آزمایش‌های خود سه فایل را به‌عنوان فایل‌های میزبان انتخاب کردیم، با این تفاوت که برای انتخاب فایل‌های میزبان ابتدا میزان پوشش کد را به‌صورت مجزا برای همه فایل‌های IUST PDF Corpus به‌دست آورده و سپس سه فایل با بیشترین میزان پوشش کد، کمترین میزان پوشش کد و نیز میزان پوشش کد میانگین، به‌ترتیب با نام‌های host1_max، host2_min و host3_avg به‌عنوان فایل‌های میزبان‌انتخاب شدند. هدف از این کار بررسی تأثیر روش پیشنهادی بر بهبود میزان پوشش کد در فایل‌های با اختلاف پوشش کد زیاد است و سپس انجام آزمون فازی براساس بهترین میزبان است. ما همچنین برای هر آزمایش پوشش کد مجموع هر سه میزبان را با ادغام پوشش کدهای هریک از آنها تحت عنوان host123 گزارش کردیم. این کار کمک می‌کند تا بتوانیم نتایج دو آزمایش کاملاً جدا را در حالت کلی با یدکدیگر مقایسه کنیم.

۵ـ۳ـ۴ پوشش کد مبنا

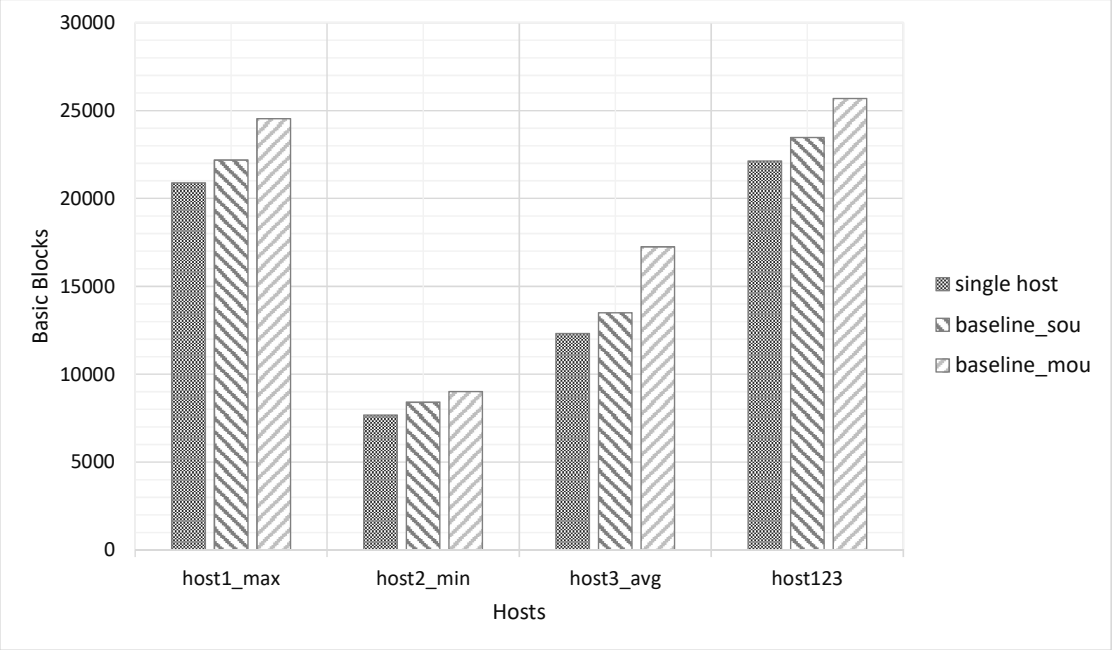
برای آنکه بتوانیم درک معنادار و مقایسه‌پذیری از پوشش کدهای به‌دست آمده در هر آزمایش داشته باشیم. ابتدا پوشش کد مبنا را مطرح، اندازه‌گیری و گزارش می‌کنیم. پوشش کد مبنا، پوشش کد در غیاب استفاده از روش پیشنهادی برای تولید خودکار داده آزمون است. برای این منظور، ابتدا از مجموعه آزمون، تعداد ۱۰۰۰ شیء داده‌ای PDF استخراج و در دو حالت به فایل‌های میزبان تزریق می‌نماییم:

- **بروزرسانی یک شیء** (SOU[1]): برای هر فایل میزبان در هر مرحله یک شیء را انتخاب و آخرین شیء میزبان را با آن بازنویسی می‌کنیم، یعنی در واقع تغییر تنها یک شیء از فایل میزبان در هر مرحله بدون توجه به اندازه و تعداد اشیای فایل میزبان.

- **بروزرسانی چندین شیء** (MOU[2]): تغییر درصد ثابتی از اشیای فایل میزبان در هر مرحله. در این روش ما برای host1_max تعداد $\frac{۱}{۵}$، برای host2_min تعداد $\frac{۱}{۳}$ و برای host3_avg تعداد $\frac{۱}{۴}$ اشیای هر فایل را به صورت تصادفی انتخاب و با اشیای انتخابی از مجموعه آزمون بازنویسی کردیم. یعنی در این روش هر میزبان را به نسبت بیشتری تغییر می‌دهیم، به‌امید اینکه این تغییرات سبب افزایش پوشش کد شود. می‌توان اشیاء را باروش‌هایی غیر از روش تصادفی نیز برای بازنویسی انتخاب کرد؛ به‌عنوان مثال

[1] Single Object Update

[2] Multiple Objects Update



شکل ۵ـ۱: پوشش کد برای هریک از میزبان‌ها و پوشش کدهای مبنا.

انتخاب براساس ترتیب نزولی یا صعودی ID هر شیء یا براساس بزرگی آدرس آنها. در این پایان‌نامه ما فقط روش تصادفی را آزمایش کردیم.

به این ترتیب سه مجموعه ۱۰۰۰ فایلی از حالت اول و سه مجموعه ۱۰۰۰ فایلی از حالت دوم حاصل می‌گردد که آنها را به ترتیب baseline_sou[1] و baseline_mou[2] می‌نامیم. پوشش کد هر مجموعه و نیز پوشش کد اجتماع هر سه میزبان به عنوان یک مقدار پایه برای آزمایش‌های بعدی اندازه‌گیری و تعیین شدند. نمودار شکل ۵ـ۱ مقادیر پوشش کد مبنا را در مقایسه با پوشش کد هریک از میزبان‌ها به‌صورت تنها نشان می‌دهد. لازم به ذکر است که هر یک از میزبان‌ها پیش از بروزرسانی‌های افزایشی برای تولید داده جدید بررسی شدند و معتبر بودن ساختار آنها تأیید گردید. لذا، صرف انجام این کار به منزله آزمون فازی نخواهد بود؛ زیرا، هنوز داده‌ها را بدشکل (فاز) نکرده‌ایم. هرچند ممکن است در این فرایند نیز خطایی قابل شناسایی باشد.

مشاهدات

- میزان پوشش کد هریک از baselineها از پوشش کد فایل میزبان به تنهایی بیشتر است. یعنی تغییر میزبان‌ها منجر به اجرای بلوک‌های پایه جدید و احتمالاً مسیرهای اجرایی جدیدی، شده است.

[1] Baseline for Single Object Update

[2] Baseline for Multiple Object Update

- میزان پوشش baselineها متناسب با میزبان‌ها است. یعنی به‌عنوان مثال host1_max بیشترین پوشش کد را در هر سه حالت تنها، baseline_sou و baseline_mou داشته است. این بدین معنی است که انتخاب فایل میزبان بسیار حائز اهمیت است و بخش قابل توجهی از پوشش کد در هر حالت مربوط به فایل میزبان است.

- میزان پوشش کد baseline_mou در تمامی میزبان‌ها از baseline_sou بیشتر است. این بدین معنی است که با افزایش میزان تغییر فایل شاهد افزایش میزان پوشش کد بوده‌ایم.

- بیشترین پوشش کد مربوط به اجتماع پوشش کدهای baseline_mou در host123 است که نشان می‌دهد هر کدام از میزبان‌ها مجموعه دستورات متفاوتی را اجرا کرده‌اند.

- درنهایت اعداد و ارقامِ پوشش کد در محدوده ۲۵هزار بلوک پایه بوده که نشان می‌دهد MuPDF نرم‌افزاری پیچیده است.

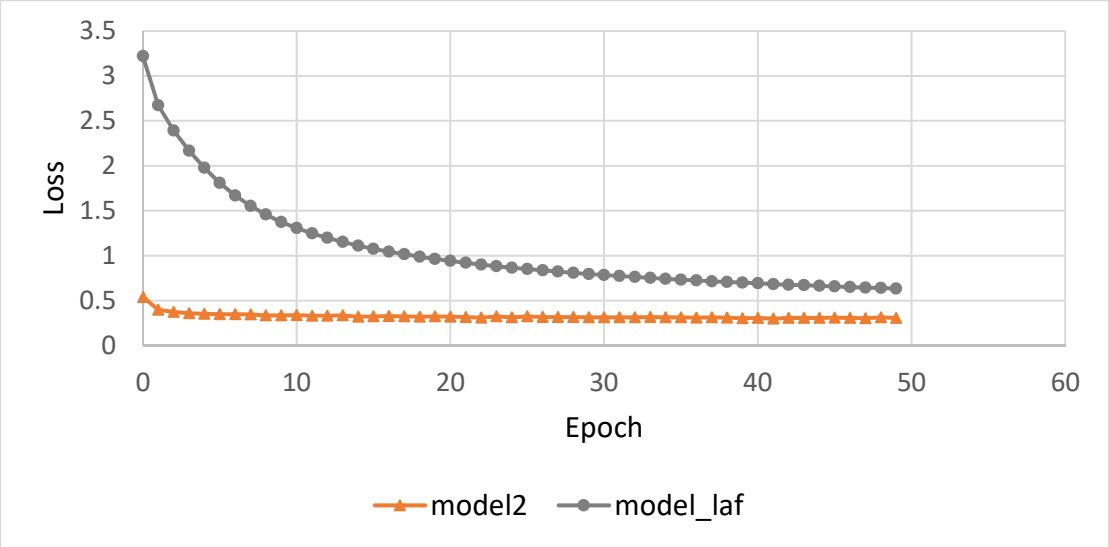
۵ـ۴ آزمایش‌ها، یافته‌ها و مقایسه نتایج

در این بخش آزمایش‌های صورت گرفته و نتایج حاصل از آن را بیان می‌کنیم. همچنین نتایج را با سایر کارهای مرتبط مقایسه خواهیم کرد. چون SUT مورد استفاده در یادگیری و فاز [۱۱] برای آزمون دردسترس نبود، ما روش یادگیری و فاز را نیز پیاده‌سازی و آن را روی ابزار MuPDF آزمایش کردیم. به‌این ترتیب توانستیم مقایسه معناداری بین روش پیشنهادی خود با این روش، به‌عنوان مرتبط‌ترین کار انجام شده داشته باشیم. نتایج نشان می‌دهد که روش پیشنهادی ما در معیار پوشش کد و همچنین در معیارهای خطا، دقت و سرگشتگی بهبود داشته است و توانسته روش یادگیری و فاز، که در آزمایش‌ها با نام laf آن را نشان می‌دهیم، را شکست دهد. روش پیشنهادی ما همچنین نسبت به AFL و AFLافزوده بهبودهایی داشته است که خواهیم دید.

چون مجموعه داده در مدل‌های یادگیری ژرف بسیار مهم است و روی نتایج تأثیر می‌گذارد در هنگام آموزش مدل laf مجموعه داده را مطابق مقیاس ارائه شده در [۱۱] لحاظ کردیم. بدین زمان آموزش کمتر برای هر دوره در حدود ۱۵ دقیقه به طول انجامید. سایر پارامترها مانند طول توالی‌های آموزشی d را که در مقاله به آن اشاره نشده بود نیز مشابه مقادیر آن در روش پیشنهادی قرار دادیم.

جدول ۵ـ۴: سرگشتگی، دقت و خطای مدل‌های مختلف روی مجموعه‌های آموزش و ارزیابی.

معیار / مدل	۱	۲	۳	۴	laf
سرگشتگی	۱/۴۴۰	۱/۳۹۱	۱/۳۳۵	۱/۳۵۰	۱/۸۶۰
بیشینه دقت آموزش	۰/۸۸۶	۰/۹۰۲	۰/۸۹۳	۰/۹۰۹	۰/۸۲۰
بیشینه دقت ارزیابی	۰/۸۸۴	۰/۸۹۵	۰/۹۰۴	۰/۹۰۵	۰/۸۰۰
کمینه خطای آموزش	۰/۳۵۳	۰/۲۹۸	۰/۳۲۴	۰/۲۷۶	۰/۶۲۳
کمینه خطای ارزیابی	۰/۳۶۵	۰/۳۳۰	۰/۲۸۹	۰/۳۰۰	۰/۷۲۴



شکل ۵ـ۲: نمودار تغییرات خطای مدل‌های ۲ و laf در دوره‌های ۱ تا ۵۰.

۵ـ۴ـ۱ سرگشتگی، خطا و دقت مدل‌ها

جدول ۵ـ۴ سرگشتی، خطا و دقت مدل‌های چهارگانه پیشنهادی و مدل یادگیری و فاز را بعد از گذشت ۵۰ دوره، روی مجموعه‌های آموزش و ارزیابی نشان می‌دهد. سرگشتی مطابق رابطه ۲ـ۱۴ و خطا نیز از رابطه ۲ـ۶ برای مدل‌ها محاسبه شده است که در فصل ۲ آنها را بیان کردیم. دقت نیز معیاری است که توسط Keras برای هر مدل حساب می‌شود. همچنین شکل ۵ـ۲ روند تغییرات خطا در فرایند آموزش را برای مدل ۲ و مدل laf نشان می‌دهد. مدل ۲ در این نمودار به‌این دلیل انتخاب گردیده که بیشترین میزان شباهت را از لحاظ معماری و مقادیر اَبرپارامترها با مدل laf دارد.

مشاهدات

- سرگشتگی و خطای همه مدل‌های پیشنهادی از مدل یادگیری و فاز کمتر و دقت آنها از یادگیری و فاز بیشتر است. یعنی مدل زبانی عصبی با معماری توضیح داده شده در بخش ۴ـ۲ ، در یادگیری ساختار فایل از مدل کدگذارـ کدگشا [۱۱] بهتر عمل کرده است.

- بیشترین دقت در بین تمامی مدل‌ها مربوط به مدل ۴ است. مدل ۴ LSTM دوسویه است. شبکه LSTM دوسویه هنگام آموزش علاوه‌بر کاراکترهای قبلی، کاراکترهای بعدی را نیز در نظر می‌گیرد. بنابراین توانسته است به دقت بیشتر دست یابد. کمترین میزان سرگشتگی نیز متعلق به مدل ۳ است. این مدل همان‌طور که در جدول نیز مشخص است دارای کمترین خطای ارزیابی است. البته اختلاف بین مقادیر مختلف در همه مدل‌های پیشنهادی کم است که نشان می‌دهد همه مدل‌ها قادر به یادگیری و درک ساختار فایل بوده‌اند. سرگشتگی بیشینه، حالت بدون استفاده از مدل یادگیری، برای تنظیمات ما برابر ۶۴ (اندازه بردار واژگان) خواهد بود.

- در شکل ۵ـ۲ ، منحنی خطای مدل ۲ در همه دوره‌ها، از منحنی خطای مدل laf پایین‌تر است. البته دوره‌ها زمان متفاوتی دارند بنابراین مقایسه نظیربه‌نظیر آنها ممکن است جالب به‌نظر نیاید. با این‌حال در بازه زمانی مساوی از شروع فرایند آموزش نیز این وضعیت فوق برقرار است. برای مثال پایان دوره ۱ برای مدل ۲ مصادف با پایان دوره ۱۲ برای مدل laf خواهد بود که باز هم خطای مدل laf بیشتر است. ما فرایند آموزش را برای هر دو مدل تا دوره ۱۰۰ ادامه دادیم و تغییر خلافی مشاهده نشد. چنان‌چه در بخش ۵ـ۴ـ۴ خواهیم دید پوشش کد مدل ۲ نیز در دوره‌های مختلف از مدل laf بالاتر است.

۵ـ۴ـ۲ پوشش کد مدل‌های مولد

در آزمایش این بخش، مدل‌های مولد پیشنهادی خود را برای اولین مرحله تولید داده آزمون محک می‌زنیم. برای تولید داده‌های آزمون جدید از روش نمونه‌برداری و پارامتر تنوع D در همه آزمایش‌ها استفاده کردیم. همچنین هربار با یک پیشوند تصادفی از مجموعه آزمون تولید داده از مدل را شروع می‌کنیم. می‌خواهیم ببینیم برای میزبان‌های مختلف کدام مدل مولد و نیز کدام تنوع تولید داده، به پوشش کد بالاتری دست می‌یابد. هدف مقایسه کارکرد مدل‌ها و انتخاب مناسب‌ترین مدل برای استفاده در آزمون فازی است. برای این منظور هریک از مدل‌های جدول ۴ـ۱ را به تعداد ۵۰ دوره آموزش دادیم. در پایان هر دوره یک نمونه از مدل آموزش دیده را ذخیره کردیم. سپس مدل با کمترین خطا را از بین نمونه‌های ذخیره شده، برای تولید داده برگزیدیم.

در مرحله بعد برای هر مدل در هنگام تولید داده با هریک از تنوع‌های تولید ۰/۵، ۱ و ۱/۵، تعداد ۱۰۰۰ فایل PDF با هر میزبان تولید کرده‌ایم. یعنی در مجموع با هر مدل و هر میزبان ۳۰۰۰ فایل را تولید می‌کنیم. چون ۴ مدل و ۳ میزبان داریم، درکل ۳۶۰۰۰ فایل تولید می‌شود و چون دو روش بروزرسانی شیء SOU و MOU هم نیاز است در نهایت ۷۲۰۰۰ فایل تولید می‌کنیم. دقت شود که در این مرحله هنوز عملیات آزمون فازی یعنی فاز ورودی، مشاهده رفتار برنامه در اشکال‌زدا و ذخیره خطاهای احتمالی، اعمال نشده است و تنها تأثیر مدل‌ها، تنوع تولید داده از آنها و میزبان‌ها، بر روی میزان پوشش کد مورد نظر است.

برای مقایسه با baseline_sou تعداد ۱۰۰۰ شیء داده‌ای از هر مدل تولید کرده و برای مقایسه با baseline_mou تعداد ۳۰۰۰ شیء داده‌ای با هر مدل تولید می‌کنیم. تولید ۱۰۰۰ فایل PDF جدید با استفاده از مدل‌ها برای حالت SOU (با اندازه بافر ۱۰۰[1]) به‌طور میانگین ۶۰ دقیقه و برای حالت MOU به‌طور میانگین ۱۹۰ دقیقه به‌طول انجامید. در پایان پوشش کد نرم‌افزار MuPDF روی هریک از ۷۲ مجموعه ۱۰۰۰ فایلی اندازه‌گیری شد. ما همچنین ادغام پوشش‌های کد را برای هر تنوع در قالب host123 اندازه گرفتیم. اندازه‌گیری پوشش کد برای هر مجموعه نیز در حدود ۶۰ دقیقه زمان برد. کلیه نتایج به تفکیک میزبان‌ها در شکل‌های ۵ـ۳آ تا ۵ـ۳ت ذکر شده است.

مشاهدات

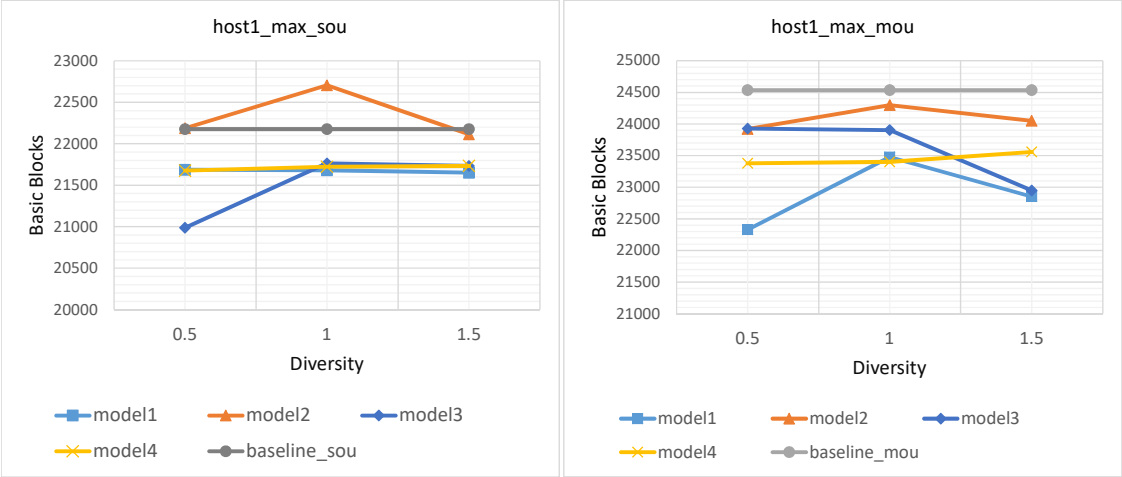
- پوشش کد داده‌های تولیدی از پوشش کد مبنا در اکثر موارد کمتر است؛ زیرا، اشیای تولید شده به خوش‌شکلی اشیای واقعی نیستند. البته دربرخی موارد شاهد افزایش پوشش کد هستیم. از جمله برای host2_min در حالت MOU.

- تولید داده با تنوع ۱ در بیشتر مدل‌ها و روی اکثریت میزبان‌ها پوشش کد بهتری داشته است. یعنی پارامتر تنوع واقعاً سبب بدشکل شدن و بالا رفتن گوناگونی داده‌های تولید شده و در نتیجه پارامـتری مؤثر بوده است.

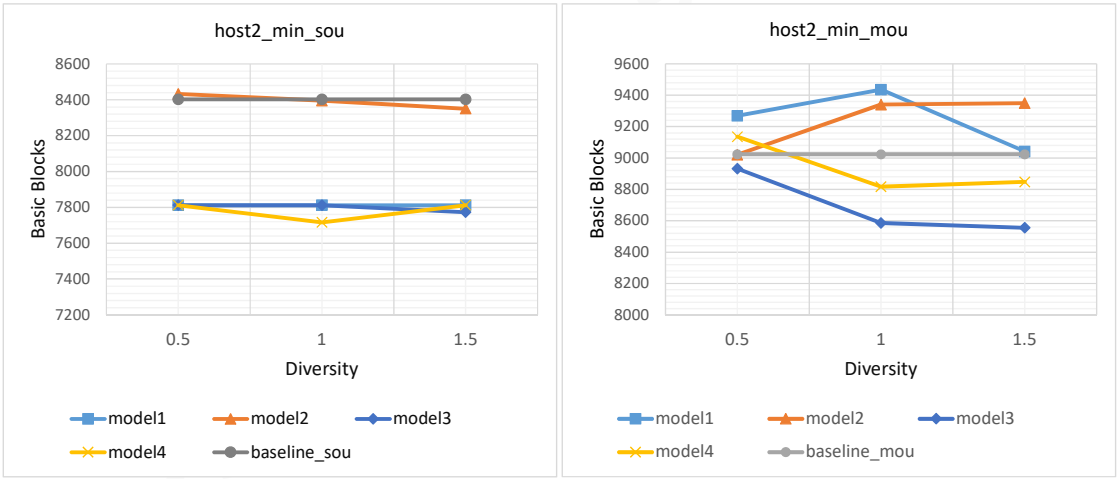
- افزایش تنوع در مدل LSTM دوسویه در حالت کلی سبب افزایش پوشش کد شده است؛ اما در دیگر مدل‌ها خیر.

- تقریباً در همه نمودارها داده‌های تولید شده توسط مدل ۲، پوشش کد بیشینه را نتیجه داده است. یعنی مدل‌های ژرف ساده بهتر از مدل‌های ژرف پیچیده همچون LSTM دوسویه (مدل ۴)، عمل کرده‌اند. به‌همین دلیل مدل ۲ با تنوع تولید داده ۱ پیروز نهایی این سری از آزمایش‌های ما هستند.

[1] یعنی هر بار یک لیست از ۱۰۰ شیء توسط مدل مولد بازگردانیده می‌شود.

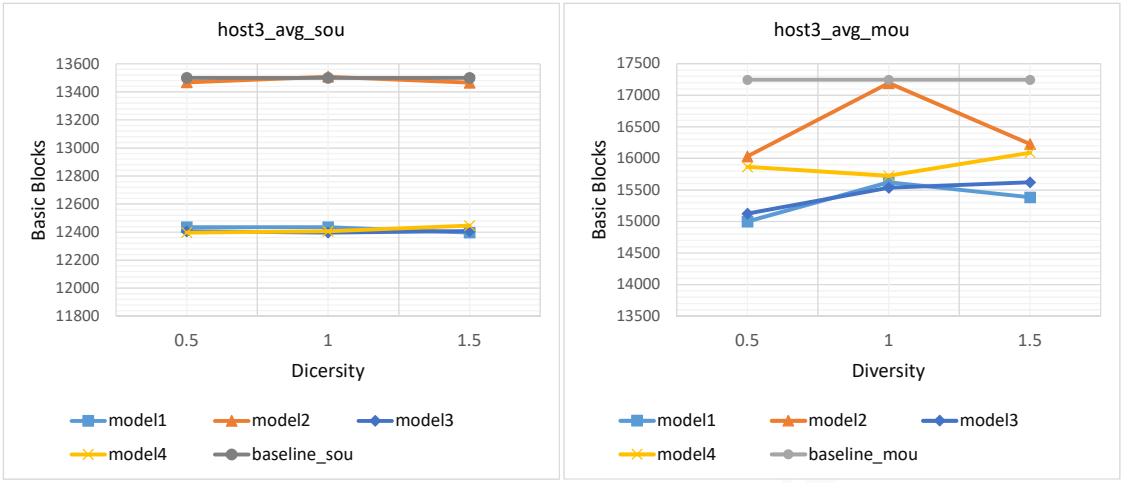


(آ) نمودار تغییرات پوشش کد در تنوع‌های نمونه‌برداری ۰٫۵ تا ۱٫۵ برای host1_max.

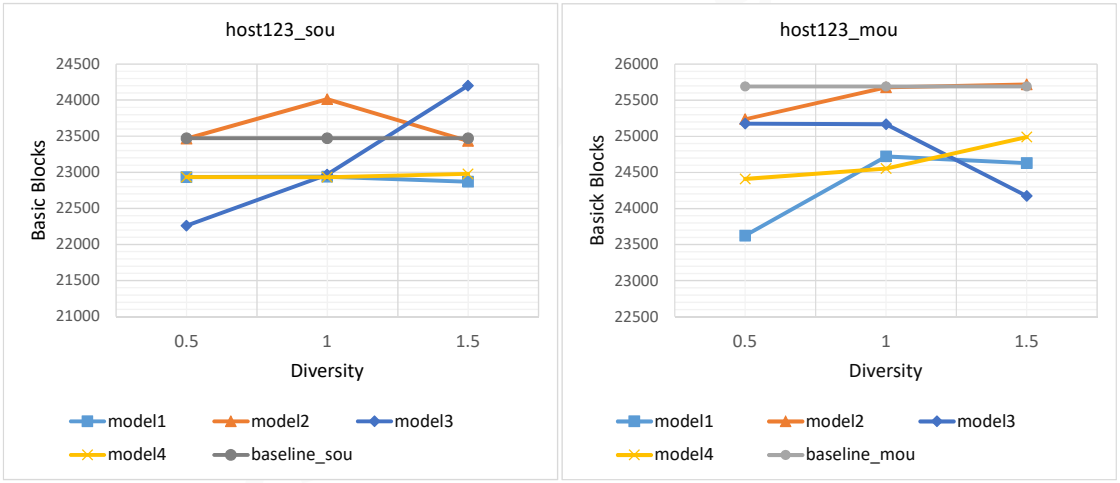


(ب) تغییرات پوشش کد در تنوع‌های نمونه‌برداری ۰٫۵ تا ۱٫۵ برای host2_min.

شکل ۵ـ۳: نمودار تغییرات پوشش کد مدل‌های مختلف برحسب تنوع.



(پ) تغییرات پوشش کد در تنوع‌های نمونه‌برداری ۰/۵ تا ۱/۵ برای host3_avg.



(ت) تغییرات پوشش کد در تنوع‌های نمونه‌برداری ۰/۵ تا ۱/۵ برای host123.

شکل ۵ـ۳: (ادامه.) نمودار تغییرات پوشش کد مدل‌های مختلف برحسب تنوع.

۵ـ۴ـ۳ مقایسه با مدل کدگذارـ کدگشا

برای مقایسه با [۱۱] ابتدا مدل کدگذارـ گشای توضیح داده شده را پیاده‌سازی و آن ۵۰ دوره آموزش دادیم. سپس ۱۰۰۰ فایل PDF را با این مدل تولید و پوشش کد مجموع آنها را اندازه‌گیری کردیم. چون راهبرد نمونه‌برداری برای مدل مذکور نیز بهترین روش گزارش شده است، ما نیز از نمونه‌برداری برای تولید داده‌ها با این مدل استفاده کرده‌ایم. برای مدل‌های چهارگانه خود نیز بالاترین پوشش کد کسب شده در میان همه تنوع‌های آزمایش شده در بخش ۵ـ۴ـ۲ را به‌عنوان نماینده پوشش کد برای هر مدل انتخاب کردیم. در نهایت دو حالت SOU و MOU را به‌صورت مجزا مورد ارزیابی و مقایسه قرار دادیم. نتایج در شکل ۵ـ۴ نشان داده شده است. شکل ۵ـ۴آ حالت SOU و شکل ۵ـ۴ب حالت MOU را نشان می‌دهد.

مشاهدات

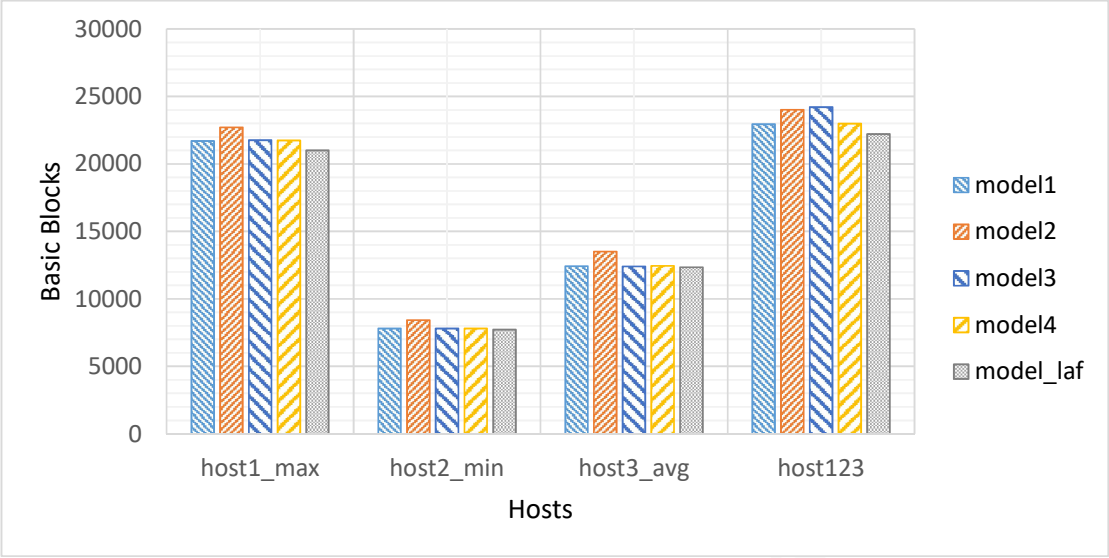
- در حالت SOU و برای host1_max همه مدل‌های پیشنهادی پوشش کد بالاتری داشته‌اند. در همین حالت دو میزبان دیگر اختلاف پوشش کد ناچیز بوده است. اما در حالت اجتماع پوشش کدها یعنی host123 مدل‌های پیشنهادی بهتر ظاهر شده‌اند یعنی هر مدل برای هر میزبان مجموعه دستورات متفاوتی را اجرا کرده است.

- در حالت MOU مدل‌های پیشنهادی ما با اختلاف بهتر هستند. این نشان می‌دهد که تغییر بیشتر فایل‌های میزبان به افزایش پوشش کد، در تعداد داده آزمون مساوی، می‌انجامد. بنابراین حالت MOU را برای انجام آزمون فازی توصیه می‌کنیم.

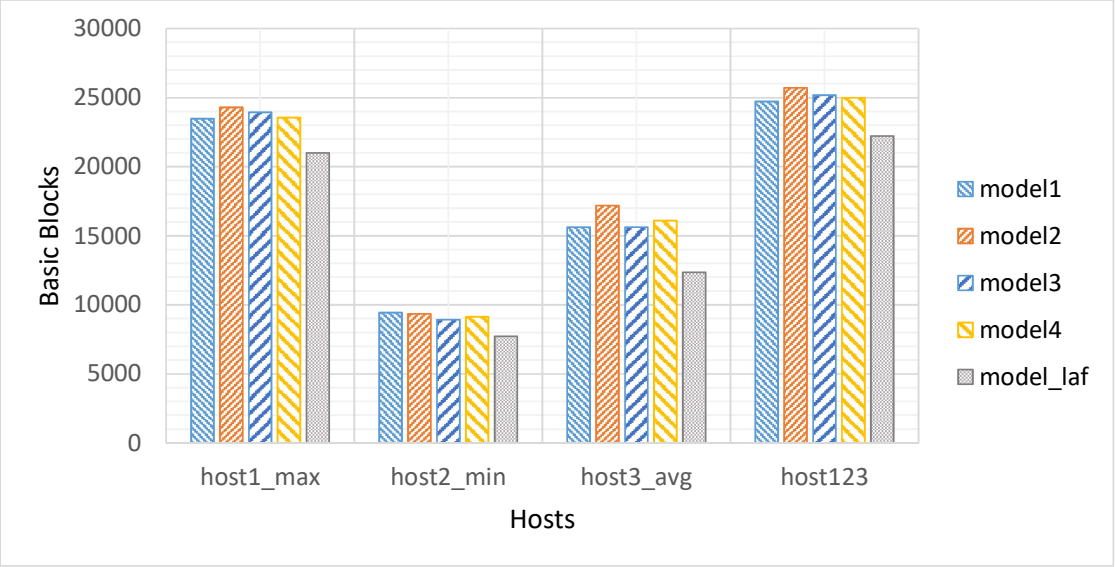
- در هر دو حالت اختلاف بین پوشش کد مدل‌ها برای host2_min کمتر و برای host1_max بیشتر به‌چشم می‌خورد. این امر نشان می‌دهد که انتخاب فایل میزبان در مواردی که نیاز به آن هست، مانند فایل PDF که ساختار پیچیده‌ای دارد و یادگیری کامل آن به‌آسانی محقق نمی‌شود، بسیار حائز اهمیت است و تأثیر چشم‌گیری روی ارتقاء پوشش کد دارد. این انتخاب نبایست تصادفی انجام شود، چون دیدیم افزایش پوشش کد رابطه مستقیمی با پوشش کد فایل میزبان دارد. بنابراین فایل میزبان با پوشش کد بیشتر را برای انجام آزمون فازی توضیه می‌کنیم.

۵ـ۴ـ۴ مقایسه در دوره‌های مختلف

یک پارامتر قابل ارزیابی که در جدول ۵ـ۱ مطرح کردیم و تأثیر آن روی پوشش کد در [۱۱] نیز بررسی شده است تعداد دوره‌های آموزش است. به نظر می‌رسد که پوشش کد ارتباط مستقیمی با تعداد دوره‌های آموزش

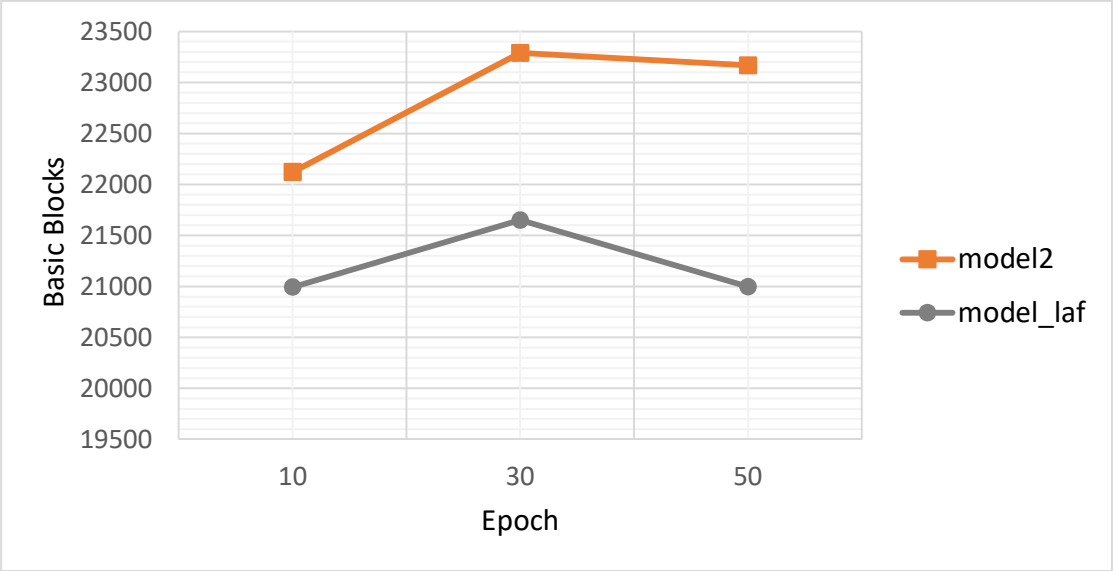


(آ) حالت SOU.



(ب) حالت MOU.

شکل ۵ـ۴: نمودار پوشش کد مدل‌های مختلف در مقایسه با مدل laf برحسب فایل‌های میزبان.



شکل ۵ـ۵: نمودار تغییرات پوشش کد برحسب دوره برای مدل‌های ۲ و laf.

مدل ژرف داشته باشد. برای این منظور پوشش کد ۱۰۰۰ فایل تولید شده با استفاده از مدل ۲ را در سه دوره ۱۰، ۳۰ و ۵۰ اندازه‌گیری کردیم. سپس همین کار را برای مدل laf نیز انجام دادیم. برای آن که نتایج قابل اطمینان باشند و اثر پارامترهای تصادفی از بین برود، هر آزمایش را سه مرتبه با سه مجموعه داده مجزا تکرار و میانگین پوشش کدها اندازه گرفتیم. نتایج در شکل ۵ـ۵ نشان داده شده‌اند.

مشاهدات

- برای هر دو مدل پوشش کد از دوره ۱۰ به ۳۰ افزایش و سپس در دوره ۵۰ کاهش یافته است. یعنی رابطه مستقیم معناداری بین پوشش کد و تعداد دوره‌های آموزش مدل وجود ندارد.

- با بررسی مقدار خطای مدل در هر دوره متوجه شدیم تا زمانی که نرخ کاهش خطای مدل در فرایند آموزش زیاد است، پوشش کد نیز افزایش می‌یابد؛ چراکه ساختار فایل بهتر یادگیری شده و در نتیجه مدل قادر به تولید داده‌های خوش‌شکل‌تری خواهد بود. هنگامی که نرخ کاهش خطا از یک حد آستانه کمتر می‌شود، افزایش پوشش کد نیز ثابت شده یا اندکی کاهش می‌یابد.

- در همه دوره‌های شکل ۵ـ۵ پوشش کد مدل پیشنهادی از مدل laf بیش‌تر بوده است که نشان می‌دهد این مدل ساختار فایل را بهتر یادگرفته است.

۵ـ۴ـ۵ آزمون فازی عصبی

در پنجمین و آخرین آزمایش نرم‌افزار MuPDF را با استفاده از فازر پیشنهادی در بخش ۴ـ۵، مورد آزمون فازی قرار دادیم. برای این منظور الگوریتم‌های فاز عصبی داده و فاز عصبی فراداده (الگوریتم‌های ۴ـ۲ و ۴ـ۳) را که در بخش ۴ـ۳ معرفی کرده بودیم، پیاده‌سازی و با استفاده از هریک از آنها تعداد ۱۰۰۰۰ فایل PDF را ایجاد کردیم. تنظیمات پارامترهای ورودی و ثوابت موجود در الگوریتم‌های پیشنهادی به قرار جدول ۵ـ۵ است. ستون آخر جدول ۵ـ۵، همچنین مقادیر مجاز برای هریک از ورودی‌ها و ثوابت داده شده را نشان می‌دهد. حداکثر طول یک شیء داده‌ای PDF را در بازه متغیر ۴۵۰ تا ۵۵۰ انتخاب کرده‌ایم؛ زیرا، به طور میانگین طول اشیای استخراج شده مجموعه‌های آموزش و آزمون، در این بازه قرار دارد. در آزمایش‌های آتی خود در نظر داریم تا آزمون فازی را با مقادیر متنوع انتخابی از مجموعه‌های داده شده انجام دهیم و بدین ترتیب قادر خواهیم بود تا اثر هریک از این پارامترها را به طور دقیق‌تری بررسی کنیم.

ما همچنین نسخه‌هایی از الگوریتم‌های ۳ـ۱ [۲] و ۳ـ۴ [۱۱] را پیاده‌سازی و آزمون فازی را با تولید داده آزمون از طریق این الگوریتم‌ها نیز انجام دادیم. برای حالت تصادفی (الگوریتم ۳ـ۱) از ابزار FileFuzz که یک فازر تصادفی تحت سیستم عامل ویندوز است و در بخش ۲ـ۲ـ۱ معرفی شد، استفاده کردیم. متأسفانه در روش یادگیری و فاز [۱۱]، تقریباً اکثر ابرپارامترهای مورد نیاز در هنگام آزمایش‌ها نامشخص هستند و نویسندگان اشاره‌ای به مقادیر آنها نکرده‌اند. برای این هریک از این ابرپارامترها نظیر d، ما همان مقدار استفاده شده در الگوریتم‌های روش پیشنهادی را استفاده کردیم، تا بدین ترتیب شرایط یکسانی بر آزمایش‌ها حاصل باشد. در همه آزمایش‌ها، host1_max به‌عنوان میزبان[1] استفاده شد. نتایج حاصل از پوشش کد روش‌های مختلف در جدول ۵ـ۶ آمده است. همچنین جدول ۵ـ۷ میزان بهبود در پوشش کد روش پیشنهادی در مقایسه با کارهای مرتبط را نشان می‌دهد.

مشاهدات

- الگوریتم فاز عصبی فراداده به پوشش کد کمتری دست پیدا کرده است، زیرا همان‌طور که انتظار می‌رود دستکاری بخش کوچکی در از قالب فایل ممکن است آن را کاملاً نامعتبر کند و در مرحله اولیه تجزیه توسط تجزیه‌گر رد شود. بنابراین الگوریتم در رسیدن به هدف خود یعنی فاز فراداده موفق بوده است.
- هر دو الگوریتم فاز عصبی داده و فاز عصبی فراداده پوشش کد بهتری نسبت به SampleFuzz داشته‌اند که نشان از عملکرد بهتر مدل‌های پیشنهادی در یادگیری ساختار فایل و عملکرد بهتر الگوریتم‌های

[1] در تولید داده آزمون به روش تصادفی و روش مبتنی برجابه‌جایی، به‌جای میزبان، به یک یا تعدادی دانه اولیه نیاز داریم که در اینجا از همان host1_max استفاده گردید.

جدول ۵ـ۵: مقادیر مجاز و مقادیر داده شده به ثوابت و ورودی‌های الگوریتم‌های فازی عصبی داده و فاز عصبی فرا داده در هنگام آزمون فازی قالب فایل PDF

پارامتر ورودی / ثابت	مقادیر استفاده شده	مقادیر مجاز
Learnt model M	۲	$۱,۲,۳,۴,laf$
Sequence prefix P	انتخاب از مجموعه آزمون	ثابت رشته‌ای
Diversity D	۱	$(۰,+\infty)$
Fuzzing rate FR	۰/۱۰	$(۰,۱]$
End token ET	endobj	ثابت رشته‌ای
Binary token BT	stream	ثابت رشته‌ای
(a,b)	(۴۵۰,۵۵۰)	$(len(P),+\infty)$
(DataNeuralFuzz) α	۰/۵۰	$(۰,۱)$
(MetadataNeuralFuzz) β	۰/۹۰	$(۰,۱)$

جدول ۵ـ۶: نتایج پوشش کد حاصل از آزمون فازی نرم‌افزار MuPDF با الگوریتم‌های تولید داده آزمون مختلف.

الگوریتم تولید داده آزمون / معیار	پوشش بلوک پایه	درصد	پوشش خط‌کد	درصد
DataNeuralFuzz	۲۳۷۱۹	۱۹/۳۶	۱۸۶۷۳	۲۰/۸۱
MetadataNeuralFuzz	۲۲۵۸۳	۱۸/۴۳	۱۷۸۹۴	۱۹/۹۵
SampleFuzz [۱۱]	۲۰۹۵۷	۱۷/۱۰	۱۶۷۹۳	۱۸/۷۲
RandomFuzz (FileFuzz) [۲]	۷۵۶۳	۶/۱۷	۵۰۰۲	۵/۵۸

جدول ۵ـ۷: میزان بهبود پوشش کد ابزار MuPDF توسط الگوریتم‌های روش پیشنهادی در مقایسه با کارهای مرتبط. اعداد داخل جدول به‌صورت درصد هستند. مقادیر هر خانه برابر اختلاف پوشش کد حاصله از الگوریتم‌های ستون و سطر مربوط به آن مقدار است.

	روش پیشنهادی	
روش‌های موجود	DataNeuralFuzz	MetaDataNeuralFuzz
SampleFuzz [۱۱]	+۲/۲۶	+۱/۸۳
AFL [۲۰]	+۷/۷۳	+۶/۸۰
AugmentedAFL [۲۰]	+۷/۵۶	+۶/۶۳
RandomFuzz (FileFuzz) [۲]	+۱۳/۱۹	+۱۲/۲۶

پیشنهادی در فاز کردن این فایل‌ها دارد. پوشش کد حاصل شده در این آزمایش‌ها همچنین از پوشش کد گزارش شده در [۲۰] که حاصل از آزمون فازی MuPDF با AFL و AFL افزوده بیشتر است. اعداد مربوط به آنها قبلاً در بخش ۳ـ۲ـ۱ ذکر شدند. جدول ۵ـ۷ در این بخش نیز اختلاف این مقادیر را با مقادیر حاصل از پوشش کد روش پیشنهادی نشان داده است.

- به‌وضوح می‌توان برتری روش‌های هوشمند تولید داده آزمون را نسبت به روش‌های تصادفی مشاهده کرد. همانطور که در ابتدای این پایان‌نامه گفتیم روش‌های تصادفی در ساختارهای پیچیده پوشش کد بسیار کمی را نتیجه می‌دهند. پوشش کد الگوریتم فاز عصبی داده بیش از ۳ برابر الگوریتم فاز تصادفی است.

- باوجود استفاده از هوش‌مصنوعی و الگوریتم‌های هوشمند در تولید داده آزمون همچنان شاهد پوشش کد پایینی (کمتر از ۲۰ درصد) در پایان عملیات آزمون هستیم. به‌نظر می‌رسد رسیدن به پوشش کد بالا در ساختارهای پیچیده، با آزمون فازی جعبه سیاه کار دشواری باشد. از طرفی فنون آزمون جعبه سفید مانند روش‌های اجرای نمادین نیز روی این ساختارها به‌علت پیچیدگی بالا و وجود محدودیت‌های فراوان سخت، زمان‌بر و تا حد زیادی غیرممکن است و کماکان آزمون فازی مؤثرتر بوده است.

خطاها و آسیب‌پذیری‌های شناسایی شده

در بررسی گزارش‌های تولید شده توسط Application Verifier پس از هر آزمون، هیچ‌گونه خطایی مشاهده نکردیم. باتوجه به‌اینکه ما نسخه نهایی نرم‌افزار MuPDF را مورد آزمایش قرار دادیم، تصور می‌شود که بیشتر خطاهای آن در نسخه‌های آزمایشی برطرف شده باشد و در نتیجه پیدا کردن خطای جدید سخت خواهد بود. از طرفی MuPDF نرم‌افزاری تحت توسعه فعال و جامعه توسعه‌دهنده و کاربری بزرگی است که سبب می‌شود تا از کیفیت خوبی برخوردار باشد. با این حال الگوریتم فاز عصبی داده چندین مورد استفاده از توابع ناامن را شناسایی کرد که Application Verifier آنها را درقالب هشدارهای امنیتی اعلام کرده است.

پیش از آزمایش‌های اصلی ما فازر و Application Verifier روی نرم‌افزارهای کوچکی که خطای آنها مشخص بود اجرا کردیم. به‌نظر می‌رسد Application Verifier روی ویندوز ۱۰ نسخه ۶۴ بیتی قادر به تشخیص خطاهای حافظه برنامه‌های ۳۲ بیتی نیست؛ زیرا موفق به شناسایی این خطاها نشدیم. برای برنامه‌های ۶۴ بیتی اما این مشکل وجود نداشت. لذا ما هر دو نسخه ۳۲ و ۶۴ بیتی MuPDF را مورد آزمون قرار دادیم.

فازر، برنامه PDFخوان را با داده آزمون ورودی باز و بعد از ۱۰ ثانیه آن را بسته و برای تزریق داده آزمون بعدی اقدام می‌کند. همزمان پوشش کد و گزارش وضعیت حافظه نیز ثبت و ذخیره می‌شوند. بنابراین زمان

آزمون برای هر مجموعه ۱۰۰۰۰ فایلی حدود ۲۸ ساعت به‌طول انجامید[۱]. برای همه نرم‌افزارهای دیگر آزمون به‌این شیوه قابل انجام خواهد بود. گفتیم که آزمون فازی نوعی آزمون فشار است. در نظر داریم تا آزمون فازی با این روش را با استفاده از ۱۰۰ هزار فایل تولید شده و حتی بیشتر انجام دهیم. در این صورت امکان سقوط برنامه MuPDF افزایش می‌یابد.

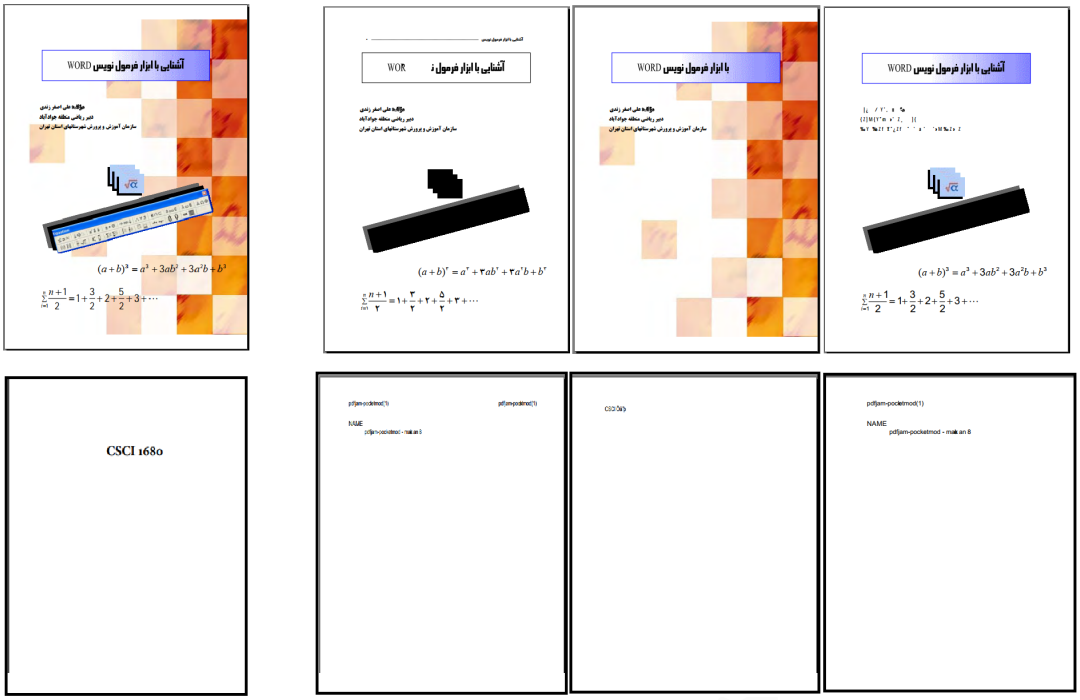
تنوع داده‌های تولید شده

مدل‌های ما در روش پیشنهادی قادر به تولید داده‌های متنوع و به‌صورت کنترل شده بدشکل هستند. الگوریتم فاز عصبی داده به‌خوبی در تغییر محتویات فایل‌های PDF عمل کرده و می‌تواند فایل‌هایی با داده‌های جدید بدون تغییر در ساختار ایجاد کند. شکل ۵ـ۶ نمونه‌ای از داده‌های تولید شده توسط الگوریتم فاز عصبی داده را نشان می‌دهد. همان‌طورکه در این شکل پیداست فایل‌های PDF در عین معتبر بودن حاوی داده‌های جدیدی هستند که با مقادیر مرزی در فرایند تولید داده آزمون جایگذاری شده‌اند. به‌دلیل معتبر بودن تعداد بیشتری از فایل‌های تولید شده توسط الگوریتم فاز عصبی داده پوشش کد این الگوریتم از پوشش کد الگوریتم فاز عصبی فراداده بیشتر است. اما هر دو الگوریتم مورد نیاز هستند؛ زیرا، هرکدام مرحله مجزایی از مراحل پردازش فایل در کد برنامه را هدف آزمون قرار می‌دهند.

۵ـ۵ خلاصه

در این فصل، ابتدا پارامترهای قابل بررسی و مؤثر در روش‌های تولید داده آزمون مبتنی بر یادگیری ژرف را معرفی و سپس روش پیشنهادی خود را که در فصل ۴ مطرح کرده بودیم، روی نرم‌افزار MuPDF و برای مهمترین این پارامترها مورد آزمایش و ارزیابی قرار دادیم. نتایج در حالت کلی حاکی از بهبود میزان پوشش کد در آزمون فازی MuPDF هنگام استفاده از مدل‌های و الگوریتم‌های تولید داده آزمون پیشنهاد شده است. به‌خصوص نسبت به تعدادی از روش‌های قبلی از جمله [۱۱] که مشابه‌ترین کار مرتبط بود، آمار و ارقام بهتری هم در دقت مدل‌ها و هم در پوشش کد مشاهده می‌شود. علاوه‌بر این نتیجه‌گیری کلی آزمایش‌های مختلف ما چندین واقعیت تجربی دیگر را مشخص کرد که مهمترین آنها عبارتند از:

[۱] Application Verifier و ابزارگذاری کد سربارهایی را به زمان هربار اجرای برنامه اضافه می‌کنند. همچنین فاصله زمانی کوتاهی بین تزریق داده‌های آزمون متوالی درنظر گرفته شد؛ زیرا، در مواردی سقوط برنامه ناشی از خطای دیگر بخش‌های سیستم است. با لحاظ این زمان‌ها از سقوط‌های این چنینی با اطمینان زیادی جلوگیری می‌نماییم.



شکل ۵ـ۶: نمونه‌ای از داده‌های آزمون متنوع تولید شده توسط الگوریتم فاز عصبی داده در فرایند آزمون فازی قالب فایل PDF. در هر ردیف، سمت چپ‌ترین فایل، فایل میزبان را نشان می‌دهد و ۳ فایل سمت راست فایل‌های حاصل از مدل مولد هستند. تغییر در محتوا به وضوح قابل مشاهده است.

- LSTM دوسویه به‌عنوان مدل زبانی می‌تواند به دقت بیشتر و خطای کمتری روی مجموعه داده دست پیدا کند. با این حال مدل‌های ژرف ساده مانند LSTM یک‌سویه بدون لایه‌های Dropout توانستند مدل‌های پیچیده‌تری مثل LSTM دوسویه را در آزمون فازی شکست دهند. نتیجه‌گیری مشابهی در [۲۰] بیان شده است.

- فایل‌های PDFای که پوشش کد بیشتری دارند، هنگام تغییر نحوه بروزرسانی افزایشی نیز، اختلاف پوشش کد بیشتری نسبت به دیگر فایل‌ها فراهم می‌کنند.

- تنوع تولید داده ۱ برای بیشتر مدل‌ها منجربه پوشش کد بهتری می‌شود.

- افزایش دوره‌های آموزش لزوماً به افزایش پوشش کد نمی‌انجامد اما تا زمانی که خطای مدل‌ها را به‌نحو خوبی کاهش دهد، پوشش کد را افزایش می‌دهد.

- روش‌های ترکیبی تولید داده آزمون مانند الگوریتم‌های فاز عصبی داده و فاز عصبی فرادادە، که بخش‌های دودویی را نیز در فرایند آزمون فازی شرکت می‌دهند منجربه پوشش کد بیشتری می‌شوند.

- آزمون فازی با روش‌های تولید داده هوشمند، آزمون فازی تصادفی را همواره شکست می‌دهد ولی هنوز هم روی ساختارهای خیلی پیچیده به پوشش کد آن‌چنان بالایی منتهی نمی‌گردد، این امر ارزشمند بودن کار بر روی روش‌های تولید داده آزمون در آینده را نشان می‌دهد. البته دلیل پوشش کد پایین برای نرم‌افزار MuPDF بیشتر آن است که این نرم‌افزار قالب‌های فایل دیگری غیر از PDF را پشتیبانی می‌کند. لذا اجرای آن با فایل‌های PDF تنها کدهای مربوط به تجزیه و پرداخت PDF را سبب می‌شود. بنابراین بایستی قالب‌های دیگر را نیز در فرایند آزمون شرکت داد که بدون شک منجر به افزایش پوشش کد می‌شود.

اثر تعدادی از پارامترهای دیگر مطرح شده در جدول ۵ـ۱ مانند راهبردهای تولید داده از مدل، در این آزمایش‌ها بررسی نشدند که جای دارد آزمایش‌هایی برای آنها نیز طراحی کرد. با این حال باتوجه به محدودیت‌های که برای هریک از دیگر راهبردها مثل راهبرد حریصانه برشمرده شد، راهبرد نمونه‌برداری مناسب‌ترین راهبرد به‌نظر می‌رسد.

فصل ۶

نتیجه‌گیری و کارهای آتی

«ما ممکن است امیدوار باشیم که ماشین‌ها در نهایت در همه زمینه‌های هوشمند با انسان رقابت کنند، اما بهترین زمینه برای شروع کدام است؟!»

✠ آلِن تورینگ

۶ـ۱ نتیجه‌گیری

دکتر اندرو انجی[1] هوشِ مصنوعی را یک الکتریسیته جدید می‌نامد که می‌تواند تحول بزرگ بعدی را در صنعت رقم بزند. یادگیری ژرف و نگاه متفاوت آن به حل مسئله تا همین الان این تحول بزرگ را در وظیفه‌هایی مانند پردازش تصویر، پرازش صوت، پردازش متن و ترجمه ماشینی رقم زده است. سال ۲۰۱۷ آغاز استفاده از یادگیری ماشینی در آزمون فازی و تولید داده آزمون بود [۱۰، ۲۰]. پژوهشگران مایکروسافت برای نخستین بار از این فنون در آزمون فازی استفاده کردند. در این پایان‌نامه ما از مدل‌های یادگیری ژرف برای یادگیری ساختار فایل‌های پیچیده و سپس تولید داده آزمون جدید به‌منظور استفاده در فرایند آزمون فازی استفاده کردیم. به‌طور خاص هر فایل را می‌توان نمونه‌ای مشتق شده از زبان یا گرامر ساختار آن دانست. براین اساس ما با استفاده از شبکه‌های عصبی مکرر اقدام به ایجاد یک مدل زبانی عصبی برای هر ساختار فایلی می‌کنیم که یک توزیع احتمالی از چگونگی وقوع نشانه‌ها در یک فایل را با آموزش روی یک مجموعه داده تخمین می‌زند. سپس این مدل زبانی را برای تولید فایل‌های جدید به‌کار می‌بندیم. یک ویژگی مفید امکان تشخیص و تمایز بین داده و

[1] Andrew Ng (http://www.andrewng.org/)

فراداده در یک فایل با استفاده از چنین مدل‌هایی است در نتیجه می‌توان جابه‌جایی آزمون فازی را با استناد به تفکیک داده و فراداده به‌نحو هوشمندتری انجام داد. دو الگوریتم پیشنهادی در این راستا نشان دادند چنین روشی قادر به اجرای بخش‌های بیشتری از کد یک SUT است و پوشش کد بیشتری را رقم می‌زند که در نتیجه امکان شناسایی خطا نیز افزایش خواهد یافت.

در فصل ۱ ، مسائلی را در باب سخت بودن تولید داده آزمون برای رسیدن به پوشش کد بالا در آزمون فازی قالب‌های فایل با ساختار پیچیده مثل PDF مطرح کردیم و دیدیم که چنان‌چه بتوان با استفاده از گرامر ورودی داده‌های آزمون را تولید کرد، تعداد داده‌هایی که در مراحل اولیه توسط کدهای مدیریت استثنای تجزیه‌گر رد می‌شوند کاهش یافته و قادر به نفوذ به مسیرهای عمیق‌تر برنامه خواهیم بود. با این ایده و براساس مطالعات اولیه و کارهای قبلی که در فصل‌های ۲ و ۳ به آنها اشاره کردیم، در فصل ۴ یک روش یادگیری ساختار فایل را پیشنهاد دادیم که تولید داده آزمون مبتنی بر گرامر را خودکار می‌کند. در فصل ۵ هم ابتدا پارامترهایی را که در تولید خودکار داده آزمون از روی مدل‌های یادگیری ژرف، نقش دارند شناسایی و سپس آزمایش‌هایی را برای بررسی و مقایسه تأثیر هریک از این پارامترهای طرح‌ریزی، پیاده‌سازی و اجرا نمودیم.

یک نتیجه‌گیری مهم که آزمایش‌های ما نشان می‌دهند این است که افزایش ظرفیت محاسباتی و پیچیدگی شبکه‌های عصبی ژرف که در نتیجه قدرت حل مسئله یک شبکه را بالا می‌برد لزوماً منجر به نتایج بهتری در آزمون فازی نخواهد شد و مشاهده کردیم که مدل‌های ساده مدل‌های پیچیده‌تر را در پوشش کد SUT شکست می‌دهند. این مدل‌ها طبیعتاً به‌زمان کمتری برای آموزش نیاز داشته و تولید داده با استفاده از آنها نیز سریع‌تر خواهد بود. بنابراین از همه نظر مقرون به صرفه هستند.

روش پیشنهادی در این پایان‌نامه، در عین مزایایی که برای آن برشمرده شد محدودیت‌ها و نقاط ضعفی دارد. بزرگترین محدودیت را می‌توان عدم اطلاع مدل مولد از وضعیت SUT دانست. به‌عبارت بهتر مدل مولد در غیاب SUT و تنها روی مجموعه داده‌های یک قالب فایل آموزش دیده و سپس اقدام به تولید داده‌های آزمون می‌کند. فازر پیشنهادی نیز فاقد یک حلقه بازخورد برای دریافت اطلاعات زمان اجرای SUT است. این در حالی است که همواره بازخوردهای اجرای SUT می‌تواند حاوی اطلاعات خوبی برای پیش‌برد ادامه فرایند آزمون در اختیار فازر قرار دهد. این محدودیت می‌تواند به‌عنوان کار آتی مورد بررسی قرار گیرد. مشکل بعدی که البته مختص به روش پیشنهادی ما نبوده و هر روش یادگیری ماشینی در این زمینه با آن مواجه است نیاز به تعدادی زیادی فایل به‌عنوان مجموعه داده است. برای قالب‌های مشهور فایل مانند PDF که مورد مطالعاتی این پایان‌نامه بود، این مشکل پُررنگ نیست اما اگر برای یک قالب خاص امکان تهیه مجموعه داده بزرگی نباشد، استفاده از این روش تقریباً غیر ممکن خواهد بود.

در هر حال تلاش در راستای موضوع این پایان‌نامه صرف نظر از نتایج تجربی آن به دلیل پیوند دو شاخه به‌ظاهر کمتر مرتبط در علم کامپیوتر یعنی یادگیری ژرف و آزمون فازی ارزشمند به‌نظر می‌رسد. به‌ویژه که

طراحی و آموزش شبکه‌های عصبی ژرف کاری مهیّج بوده و این مسیر پژوهشی نیز در ابتدای راه خود است. در این بین دو حوزه مذکور که این پایان‌نامه بر آنها بیان شده، نیز هرکدام در حالت کلی دارای مزایا و معایبی هستند که ممکن است برخی از آنها را قبلاً هم ذکر کرده باشیم، با این حال در پایان این بخش نگاهی کوتاه به مزایا، معایب و آینده هریک از این دو حوزه خواهیم داشت؛ زیرا می‌توانند در تعیین روند پژوهش‌های آتی مؤثر واقع شوند.

۶ـ۱ـ۱ مزایا و معایب یادگیری ژرف

یادگیری ژرف و به تبع آن شبکه‌های عصبی ژرف در انجام وظایف ساده برای انسان، سخت برای ماشین بسیار موفق ظاهر شده‌اند. با توجه به افزایش قدرت محاسبات انجام حجم وسیعی از محاسبات در مسائل پیچیده، ارزان‌تر از نوشتن یک الگوریتم خاص می‌باشد به‌نحوی که در آینده شاهد افزایش ظرفیت‌های سخت‌افزاری برای توسه چنین مدل‌هایی در مقیاس‌های بسیار بزرگ خواهیم بود. دو ویژگی بسیار مهم این شبکه‌ها عبارتند از تعمیم‌پذیر[1]ی و تطبیق‌پذیر[2]ی [۳۸]. تعمیم‌پذیری بدین معنی است که در صورت آموزش صحیح، شبکه برای ورودی‌های جدید نیز درست کار خواهد کرد. تطبیق‌پذیری یعنی در صورتی که داده‌ها تغییر کند، شبکه هم توانایی تغییر خواهد داشت. یعنی یک مدل می‌تواند برای حل خانواده‌ای از مسائل مشابه طراحی و ساخته شود و هر بار با داده‌های مختلفی آموزش ببیند.

شبکه‌های عصبی سراسر فایده و نوش‌داروی حوزه محاسبات جدید نیستند و در عین حال معایبی دارند. از جمله اینکه آموزش آنها سخت و بسیار مستعد خطا است. در واقع دقت نتایج بستگی زیادی به مجموعه آموزش دارد؛ به نحوی که یک مجموعه آموزش ضعیف (کوچک یا نادرست) عملاً شبکه غیرقابل استفاده‌ای را نتیجه می‌دهد. دیگر آنکه قوانین مشخصی برای طراحی یک شبکه جهت کاربردی خاص وجود ندارد. به عبارت بهتر تعیین ابرپارامترها مسئله تصمیم‌ناپذیر است؛ یعنی یک نمی‌توان به صورت الگوریتمی بهترین مجموعه ابرپارامتر را برای یک شبکه در یک وظیفه خاص تعیین کرد. این کار معمولاً با سعی و خطا انجام می‌شود و بلأخره اینکه نمی‌توان به فیزیک یا قانون حاکم بر مسئله حل‌شده توسط شبکه پی برد و تنها با مشتی اعداد سروکار خواهیم داشت که روی یک مجموعه یک هدف خواسته شده را بهینه کرده‌اند.

[1] Generalization

[2] Adaptive

۶ـ۱ـ۲ مزایا و معایب آزمون فازی

آزمون فازی چندین سودمندی دارد. در درجه نخست سادگی و راحتی خودکارسازی فرایند شرح داده شده است. به همین دلیل، فازرهای بسیاری توسعه داده شده است. برای استفاده از فازرهای موجود تنها انتخاب SUT و فراهم ساختن تعدادی داده آزمون اولیه یا قالب ورودی لازم است. فازر می‌تواند به‌صورت یک پردازه پس‌زمینه[1] و بدون دخالت اضافی کاربر، در یک حلقه بی‌نهایت، به مدت طولانی اجرا شود. برای ساخت یک فازر جدید نیز کافی است پیمانه‌های شکل ۲ـ۲ ، توسعه داده شده و در کنار هم قرار گیرند.

آزمون فازی همچنین کاستی‌های آزمون معمولی که به‌صورت دستی صورت می‌پذیرد را جبران می‌کند. آزمون‌های نوشته شده به صورت دستی تا حدودی تمایل به پیش‌دانسته‌های ذهنی فرد آزمون‌گر در مورد کد دارند. آزمون فازی از انحراف یاد شده مستثنی است و می‌تواند ورودی‌های بدشکل را به نحوی تولید کند که پیش از آن هیچگاه، به ذهن فرد یا افراد آزمون‌گر نرسیده است. با گذشت سه دهه از ابداع آزمون فازی این روش جایگاه ویژه‌ای در صنعت و نیز در پژوهش یافته است و به همین دلیل کار بر روی آن ارزشمند و اثر بخش است.

در حالی که آزمون فازی در پیدا کردن خطاهای فساد حافظه، بسیار خوب عمل می‌کند، خطاهای پیچیده‌تر مانند خطاهای منطقی به ندرت توسط این آزمون قابل شناسایی هستند؛ چرا که آشکارسازی آنها نیازمند به اتمام رسیدن اجرای برنامه بدون خطای حافظه و داشتن سروش[2] آزمون است. افزون بر این تحریک برخی خطاهای حافظه برای فازرها بسیار سخت خواهد بود. مسئله انفجار مسیر که ناشی از وجود حلقه‌های تکرار و شرط‌های تودرتو است مانع از اجرای نمادین برنامه‌های پیچیده می‌شود. در نتیجه به‌کارگیری اجرای نمادین در آزمون فازی جعبه سفید در بسیاری موارد ممکن نیست و این مسئله به پوشش کد پایین و خوب آزمون نشدن برنامه می‌انجامد.

آزمون فازی ممکن است برای اهداف سوء مورد استفاده قرار گیرد. یعنی شناسایی آسیب‌پذیری‌ها توسط فرد مهاجم و بهره‌برداری از آنها جهت حمله به یک سیستم نرم‌افزاری. در نتیجه روش‌هایی برای مقابله با امکان آزمون‌پذیری یک نرم‌افزار مطرح شده‌اند که می‌توان آنها را در طبقه روش‌های ضد مهندسی معکوس دانست. از جمله این روش‌ها می‌توان به حوزه‌ای جدید که اخیراً تحت عنوان ضد فازینگ[3] مطرح شده است، اشاره کرد که در آن اقداماتی برای کندسازی یا به‌کلی مانع شدن کشف خطاها توسط آزمون فازی انجام می‌گیرد. کاهش کارآمدی و پوشش خطاها دو رویکرد از میان رویکردهای موجود در این زمینه هستند. دیگر فنون مقابله با مهندسی معکوس نظیر مبهم‌سازی کد را نیز می‌توان در ضد فازینگ به‌کار گرفت. چنان‌چه این رویکردها

[1] Background Process

[2] Oracle

[3] Anti-fuzzing

به بلوغ خوبی برسند و توسعه‌دهندگان آنها را در برنامه‌های خود لحاظ نمایند، در آینده بایستی به دنبال فنون جایگزین برای روش آزمون فازی جعبه خاکستری و به‌طور کلی آزمون فازی باشیم. البته استفاده از این فن آزمون در بین توسعه دهندگان و پیش از انتشار نسخه نهایی نرم‌افزار همچنان در صنعت ادامه خواهد یافت.

۶ـ۲ نوآوری‌ها

مجموعه نـوآوری‌ها، دستاوردها و محصولات کار پژوهشی ما در قالب این پایان‌نامه عبارت است از:

۱. خودکارسازی فرایند یادگیری ساختار فایل ورودی و تولید داده‌های آزمون برای آزمون فازی قالب فایل با به‌کارگیری روش‌های جدید یادگیری ژرف.

۲. بهبود پوشش کد SUT با به‌کار بست روشی ترکیبی یعنی تولید فراداده و داده‌های متنی، مبتنی بر گرامر و سپس تزریق داده‌های دودویی مبتنی بر جابه‌جایی تصادفی در مکان‌های تعیین شده.

۳. شناسایی و استخراج پارامترهای مؤثر در تولید داده آزمون به‌روش یادگیری ژرف و ارزیابی تأثیر آنها با ارایه مجموعه‌ای از آزمایش‌های کنترل شده.

۴. معرفی یک مجموعه دانه اولیه و یک مجموعه داده آزمون برای آزمون فازی قالب فایل PDF تحت یک پیکره از فایل‌های PDF به‌همراه پوشش کدهای آنها.

۵. طراحی و پیاده‌سازی یک فازر قالب فایل ساده با معماری کاملاً پیمانه‌ای و قابل حمل، مجهز به پیمانه تولید خودکار داده‌های آزمون با استفاده از مدل‌های مولد.

به‌طور خلاصه در این پایان‌نامه یک مجموعه داده، چهار مدل مولد، دو الگوریتم فاز و یک فازر قالب فایل پیشنهاد و معرفی گردید که همه آنها در قالب یک بسته نرم‌افزاری تحت عنوان IUST Deep Fuzz منتشر شده‌اند. از این محصول می‌توان در عمل برای آزمون فازی و شناسایی خطاهای نرم‌افزارهایی با ورودی فایل استفاده کرد. تنظیمات کنونی بر روی قالب فایل PDF تعیین شده است اما به‌راحتی قابل تغییر است. نرم‌افزارهای با ورودی فایل علاوه‌بر PDFخوان‌ها، شامل خانواده وسیع کامپایلرها، مرورگرهای وب، محیط‌های توسعه مجتمع و غیره می‌شوند، که همه آنها در طبقه برنامه‌های کاربردی و بسیار مهم قرار می‌گیرند. تقریباً در همه موارد مذکور، داده کافی برای ایجاد مدل مولد وجود دارد. به‌عنوان مثال در آزمون مرورگرها، ایجاد یک مدل مولد برای تولید فایل‌های ترکیبی HTML، CSS و JavaScript داده آزمون بهتری نسبت به تولید تنها یکی از این فایل‌ها [۴۸] فراهم می‌کند و از طرفی برای هر سه قالب فایل نام‌برده مجموعه داده به فراوانی یافت می‌شود.

۶ـ۳ ملاحظات اعتبارسنجی

در این پایان‌نامه بر بهبود معیار پوشش کد در آزمون فازی، به‌طور مکرر تأکید ورزیدیم. منظور از پوشش کد در اینجا پوشش دستور (یا در حالت کلی‌تر پوشش بلوک پایه است). در مواردی به پوشش مسیر نیز اشاره کردیم و به‌عنوان مثال در مورد پوشش مسیرهای اجرایی عمیق برنامه صحبت به میان آوردیم. ذکر این نکته ضروری است که معیارهای پوشش دستور و پوشش مسیر متفاوت هستند و همچنان‌که در فصل ۲ دیدیم، پوشش مسیر معیار سخت‌گیرانه‌تری نسبت به پوشش دستور است؛ یعنی ممکن است تمامی دستورات برنامه در یک یا چند اجرا، حداقل یک‌بار اجرا شوند ولی لزوماً تمامی مسیرهای اجرایی پوشش داده نشوند.

یک برنامه با یک دستور *if* ساده را در نظر بگیرید. اگر برنامه در حالتی اجرا شود که حاصل ارزیابی عبارت شرطی درست شود، همه دستورات (همچنین همه بلوک‌های پایه) برنامه اجرا می‌شوند. در این حالت پوشش بلوک پایه معادل ۱۰۰ درصد خواهد بود. در حالی که این برنامه در بَدَوی‌ترین شکل خود، دو مسیر اجرایی دارد: یک مسیر که از بدنه دستور شرطی *if* عبور می‌کند و مسیر دیگر که وارد بدنه دستور *if* نمی‌شود. برای سناریوی اجرای ذکر شده، پوشش مسیر ۵۰ درصد است. بنابراین تفاوت معناداری میان پوشش دستور و پوشش مسیر در این مثال وجود دارد. ممکن است خواننده با این ابهام روبه‌رو شود که در چنین حالتی، نتایج گزارش شده موردی بوده و قابل تعمیم نیست.

چنان‌چه یک دستور (بلوک پایه) جدید اجرا شود، می‌توان گفت یک مسیر جدید اجرا شده است. بنابراین از این منظر، تلاش برای افزایش پوشش بلوک پایه منجر به افزایش پوشش مسیر نیز می‌گردد. اما دو مجموعه برابر از پوشش‌های دستور، لزوماً پوشش مسیر برابری به دست نمی‌دهند. اندازه‌گیری پوشش مسیر دشوارتر بوده و سربار بیشتری به آزمون تحمیل می‌کند. از لحاظ تئوری نیز، برخی از مسیرهای ایستای برنامه، غیر قابل دسترسی هستند و با وجود حلقه‌های تکرار در برنامه، تعداد مسیرهای اجرایی در مواردی بی‌نهایت می‌شود. به سبب این‌گونه مسائل، ابزارهای معرفی شده در فصل ۲، هیچ‌کدام پوشش مسیر اجرایی را اندازه‌گیری نمی‌کنند. در بیشتر کارهای مربوط به آزمون فازی منظور از پوشش کد، همان پوشش در سطح دستورات برنامه است. بنابراین آمار و ارقام ارایه شده در این پایان‌نامه نیز مشابه کارهای پیشین برحسب همان میزان پوشش دستورات و در حالت کلی‌تر پوشش بلوک پایه بنا شده است.

در آزمایش‌هایی که پوشش دستور دو مجموعه داده آزمون، دقیقاً برابر باشد، برای مقایسه لازم است تا پوشش مسیر اندازه‌گیری گردد. در آزمایش‌های انجام شده توسط ما، همواره پوشش بلوک پایه مجموعه‌های آزمون استفاده شده پس از گرفتن اجتماع، متفاوت بوده است که نشان از متفاوت بودن پوشش مسیرها نیز دارد. لذا نتایج گزارش شده، صحت داشته و دلالت بر بهبود میزان پوشش کد SUT در حالت کلی دارد. در هر صورت، ما در نظر داریم تا آزمایش‌های خود را در مقیاس بسیار بزرگ‌تری و روی قالب‌های فایل مختلف،

تکرار کرده و اندازه‌گیری درست پوشش مسیر را نیز برای آزمایش‌های جدید انجام دهیم.

۶ـ۴ کارهای آتی

پایان‌نامه پیشِ‌رو، مباحث تولید داده آزمون، آزمون فازی و یادگیری ژرف (و به طور خاص‌تر مدل‌های زبانی عصبی) را به یکدیگر پیوند زده است. هر سه موضوع یاد شده از موضوعات مهم، کاربردی و داغ در پژوهش‌های علوم و مهندسی کامپیوتر هستند. پیشنهادهای زیادی برای کارهای آتی مرتبط با موضوع این پایان‌نامه مطرح است که در ذیل به چندین مورد از آنها اشاره می‌کنیم.

۱. **استفاده از دیگر مدل‌ها و معماری‌های شبکه‌های عصبی ژرف در تولید داده آزمون.** به‌عنوان مثال می‌توان از مدل‌های GAN[1] [۵۳] برای تولید داده آزمون استفاده کرد. به‌طور خلاصه GAN از دو شبکه عصبی تشکیل شده است. یک شبکه مولد که داده‌های جدید را تولید می‌کند و یک شبکه که آنها ارزیابی می‌کند. هدف شبکه مولد این است که داده‌هایی تولید که از دید شبکه ارزیاب خطای کمتری داشته باشند. از GAN در تولید محیط‌های جدید در بازی‌های رایانه‌ای استفاده شده است اما کاربرد آنها محدود به‌این مورد نیست.

۲. **استفاده از یادگیری ژرف در دیگر فازرها.** به‌عنوان نمونه یادگیری ساختار پروتکل‌های شبکه که می‌تواند برای تولید داده در فازرهای شبکه و پروتکل‌ها استفاده شود؛ این مسئله به‌ویژه در آزمون پروتکل‌هایی با ساختار ناشناخته مثل بات‌نت‌ها قابل توجه است. در این مورد یادگیری ممکن است به اهداف مهندسی معکوس کمک نماید.

۳. **تولید داده آزمون براساس اهداف مختلف.** هدف آزمون فازی و فازر نبایست لزوماً افزایش پوشش کد یا به‌عبارتی Input Gain تعیین شود. داده‌های آزمونی که توابع ناامن را فراخوانی می‌کنند برای مثال حائز اهمیت هستند [۴۹]. ایجاد مدل‌های یادگیری که امتیاز را به فراخوانی مجموعه‌ای از توابع ناامن نسبت دهند، به‌این امید که آزمون برنامه با ورودی‌هایی که این مسیرها را اجرا می‌کند، منجربه وقوع خطای حافظه می‌شود، می‌تواند جالب و قابل توجه باشد.

۴. **افزودن حلقه بازخورد به فازر پیشنهادی در این پایان‌نامه.** قبلاً نیز اشاره شد که یک محدودیت روش پیشنهادی عدم استفاده از حقله بازخورد است. می‌توان این حلقه را در قالب یک فازر جدید اضافه

[1] Generative Adversarial Network

کرد یا برای مثال از روش پیشنهادی برای تولید دانه‌های اولیه برای فازرهایی مثل AFL استفاده کرد که در این صورت بایستی یک مرحله کمینه‌سازی دانه[1] نیز برای افزایش کارایی AFL ارایه شود.

۵. **افزودن تعامل کاربر به آزمون فازی جعبه سیاه.** تعامل کاربر با برنامه منجربه اجرای کدهای مربوط به کارهای کاربر می‌شود. ما در برخی آزمایش‌های خود مشاهده کردیم که این امر تأثیر زیادی بر میزان پوشش کد دارد. البته این مورد با آزمون فازی واسط کاربر متفاوت است؛ زیرا، لزوماً همه تعامل‌ها مربوط به واسط کاربر نیستند و بعضی از آنها کدهای مربوط به مرحله پرداخت فایل را اجرا می‌کنند. تعامل کاربر نیازمند دخالت مستقیم کاربر و انجام عملیاتی از طریق صفحه‌کلید یا دیگر ابزارهای ورودی است که در نتیجه خودکار نیست. بنابراین می‌توان فازری طراحی کرد که برروی یک داده آزمون ورودی تولید شده یک دنباله از عملیات کاربر را به‌طور خودکار و به‌صورت ترتیبی یا تصادفی انجام دهد. ما این مورد را در کار بعدی خود لحاظ می‌کنیم.

۶. **استفاده از یادگیری ژرف در دیگر مراحل آزمون نرم‌افزار.** آزمون نرم‌افزار تنها مختص به مرحله تولید داده آزمون و مکان‌یابی خطا نیست. می‌توان در دیگر مراحل نیز از فنون یادگیری ژرف استفاده کرد. یک زمینه پژوهشی نو، ترمیم خودکار برنامه‌ها پس از مشخص شدن مکان خطا است. برای این منظور استفاده از مدل CAN[2] [۵۴] ایده جالبی به‌نظر می‌رسد. CAN نوع خاصی از GAN است که تلاش می‌کند عوامل خلاقیت و تولید داده جدید را به آن اضافه کند. در ترمیم خودکار برنامه نیز نیازمند سازوکاری برای تنوع‌بخشی به قسمت‌های خطادار باهدف ترمیم آن قسمت‌ها، هستیم.

■■■■■

[1] Seed Minimization

[2] Creative Adversarial Networks

مراجع

[1] A. Takanen, J. DeMott, and C. Miller. *Fuzzing for software security testing and quality assurance*. Norwood, MA, USA: Artech House, Inc., 1 ed. , 2008.

[2] M. Sutton, A. Greene, and P. Amini. *Fuzzing: brute force vulnerability discovery*. Addison-Wesley Professional, 2007.

[3] B. P. Miller, L. Fredriksen, and B. So, "An empirical study of the reliability of Unix utilities," *Commun. ACM*, vol.33, pp.32–44, Dec. 1990.

[4] B. P. Miller, D. Koski, C. Pheow, L. V. Maganty, R. Murthy, A. Natarajan, and J. Steidl, "Fuzz revisited: a re-examination of the reliability of Unix utilities and services," tech. rep., University of Wisconsin-Madison, 1995.

[5] J. E. Forrester and B. P. Miller, "An empirical study of the robustness of Windows NT applications using random testing," in *Proceedings of the 4th Conference on USENIX Windows Systems Symposium - Volume 4*, WSS'00, (Berkeley, CA, USA), pp.6–6, USENIX Association, 2000.

[6] B. P. Miller, G. Cooksey, and F. Moore, "An empirical study of the robustness of MacOS applications using random testing," in *Proceedings of the 1st International Workshop on Random Testing*, RT '06, (New York, NY, USA), pp.46–54, ACM, 2006.

[7] S. Rawat, V. Jain, A. Kumar, L. Cojocar, C. Giuffrida, and H. Bos, "Vuzzer: application-aware evolutionary fuzzing," in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, Feb. 2017.

[8] U. Kargén and N. Shahmehri, "Turning programs against each other: high coverage fuzz-testing using binary-code mutation and dynamic slicing," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2015, (New York, NY, USA), pp.782–792, ACM, 2015.

[9] S. Veggalam, S. Rawat, I. Haller, and H. Bos, "Ifuzzer: an evolutionary interpreter fuzzer using genetic programming," in *Computer Security – ESORICS 2016* (I. Askoxylakis, S. Ioannidis, S. Katsikas, and C. Meadows, eds.), (Cham), pp.581–601, Springer International Publishing, 2016.

[10] P. Godefroid, M. Y. Levin, and D. Molnar, "Sage: whitebox fuzzing for security testing," *Queue*, vol.10, Jan. 2012.

[11] P. Godefroid, H. Peleg, and R. Singh, "Learn&fuzz: machine learning for input fuzzing," in *Proceedings of the 32Nd IEEE/ACM International Conference on Automated Software Engineering*, ASE 2017, (Piscataway, NJ, USA), pp.50–59, IEEE Press, 2017.

[12] A. Kettunen, *Test harness for web browser fuzz testing*. M.Sc. Thesis, University of Oulu, 2014.

[13] P. Ammann and J. Offutt. *Introduction to software testing*. Cambridge University Press, 2016.

[14] N. Rathaus and G. Evron. *Open source fuzzing tools*. Syngress Publishing, 2007.

[15] O. Bastani, R. Sharma, A. Aiken, and P. Liang, "Synthesizing program input grammars," *SIGPLAN Not.*, vol.52, pp.95–110, June 2017.

[16] M. Höschele and A. Zeller, "Mining input grammars from dynamic taints," in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, ASE 2016, (New York, NY, USA), pp.720–725, ACM, 2016.

[17] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in Neural Information Processing Systems 27* (Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, eds.), pp.3104–3112, Curran Associates, Inc., 2014.

[18] K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *CoRR*, vol.abs/1406.1078, 2014.

[19] M. Zalewsky, "American fuzzy lop," [Online]. Available: http://lcamtuf.coredump.cx/afl/, [Accessed: 2017-10-11].

[20] M. Rajpal, W. Blum, and R. Singh, "Not all bytes are equal: neural byte sieve for fuzzing," *CoRR*, vol.abs/1711.04596, 2017.

[21] E. Dubrova. *Fault-tolerant design*. Springer Publishing Company, Incorporated, 2013.

[22] Symantec, "ISTR internet security threat report," tech. rep., 2018. https://www.symantec.com/content/dam/symantec/docs/reports/istr-23-2018-en.pdf.

[23] Microsoft Corporation, "Simplified implementation of the SDL," tech. rep., 2010.

[24] C. Chen, B. Cui, J. Ma, R. Wu, J. Guo, and W. Liu, "A systematic review of fuzzing techniques," *Computers and Security*, vol.75, pp.118–137, 2018.

[25] Artifex Software, Inc., "MuPDF," [Online]. Available: https://mupdf.com/, [Accessed: 2018-07-25].

[۲۶] م. ذاکری نصرابادی، *بررسی روش‌های تولید خودکار داده‌های آزمون برای استفاده در فازرهای مبتنی بر قالب فایل*. سمینار کارشناسی ارشد، دانشکده کامپیوتر، دانشگاه علم و صنعت ایران، آبان ۱۳۹۶.

[27] Paul C. Jorgensen. *Software testing a craftsman's approach*, vol.47. CRC Press Taylor & Francis Group, fourth edi ed. , 2014.

[28] E. Nikravan and S. Parsa, "A reasoning-based approach to dynamic domain reduction in test data generation," *International Journal on Software Tools for Technology Transfer*, May 2018.

[29] R. Mcnally, K. Yiu, and D. Grove, "Fuzzing: the state of the art," *DSTO Defence Science and Technology Organisation*, p.55, 2012.

[30] M. E. Khan and F. Khan, "A comparative study of white box, black box and grey box testing techniques," *International Journal of Advanced Computer Science and Applications*, vol.3, no.6, pp.12–15, 2012.

[31] D. L. Bruening, *Efficient, transparent, and comprehensive runtime code manipulation*. Ph.D. thesis, Cambridge, MA, USA, 2004. AAI0807735.

[32] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood, "Pin: building customized program analysis tools with dynamic instrumentation," *SIGPLAN Not.*, vol.40, pp.190–200, June 2005.

[33] QEMU Team, "QEMU," [Online]. Available: https://www.qemu.org/, [Accessed: 2018-07-27].

[34] C. Miller and Z. Peterson, "Analysis of mutation and generation-based fuzzing," *White Paper, Independent Security Evaluators*, pp.1–7, 2007.

[35] C. Holler, K. Herzig, and A. Zeller, "Fuzzing with code fragments," in *Proceedings of the 21st USENIX Conference on Security Symposium*, Security'12, (Berkeley, CA, USA), pp.38–38, USENIX Association, 2012.

[36] Microsoft, "Application verifier (appverif.exe)," [Online]. Available: https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/application-verifier, [Accessed: 2018-07-18].

[37] Network Working Group, "Internet security glossary," https://tools.ietf.org/html/rfc2828, [Accessed: 2017-10-11].

[38] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[39] D. Jurafsky and J. H. Martin. *Speech and language processing (3rd ed. draft)*. 2017. https://web.stanford.edu/~jurafsky/slp3/.

[40] A. Karpathy, *Connecting images and natural language*. Ph.D. Thesis, Stanford University, 2016.

[41] B. Sautermeister, *Deep learning approaches to predict future frames in videos*. M.Sc. Thesis, Technical University of Munich, 2016.

[42] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol.15, pp.1929–1958, 2014.

[43] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "LSTM: a search space odyssey," *CoRR*, vol.abs/1503.04069, 2015.

[44] Z. C. Lipton, "A critical review of recurrent neural networks for sequence learning," *CoRR*, vol.abs/1506.00019, 2015.

[45] M. T. Luong, *Neural machine translation*. Ph.D Thesis, Stanford university, 2016.

[46] T. Mikolov, *Statistical language models based on neural networks*. Ph.D. Thesis, Brno University of Technology, 2012.

[47] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. J. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Józefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. G. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. A. Tucker, V. Vanhoucke, V. Vasudevan, F. B. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: large-scale machine learning on heterogeneous distributed systems," *CoRR*, vol.abs/1603.04467, 2016.

[۴۸] س.م. یعقوبی، *طراحی و پیاده‌سازی فازر با هدف تعیین آسیب‌پذیری‌های مرورگر وب*. پایان‌نامه کارشناسی ارشد، دانشکده مهندسی کامپیوتر، دانشگاه علم و صنعت ایران، آبان ۱۳۹۲.

[۴۹] س. امینی، *طراحی و پیاده‌سازی یک روش تولید داده آزمون به‌منظور کشف آسیب‌پذیری‌های نرم‌افزار*. پایان‌نامه کارشناسی ارشد، دانشکده مهندسی کامپیوتر، دانشگاه علم و صنعت ایران، آبان ۱۳۹۵.

[50] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[51] D. P. Kingma and J. Ba, "Adam: a method for stochastic optimization," *CoRR*, vol.abs/1412.6980, 2014.

[52] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining moftware: an update," *SIGKDD Explor. Newsl.*, vol.11, pp.10–18, Nov. 2009.

[53] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems 27* (Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, eds.), pp.2672–2680, Curran Associates, Inc., 2014.

[54] A. M. Elgammal, B. Liu, M. Elhoseiny, and M. Mazzone, "CAN: creative adversarial networks, generating "art" by learning about styles and deviating from style norms," *CoRR*, vol.abs/1706.07068, 2017.

پیوست آ

ساختار فایل PDF

در این پیوست به‌طور خلاصه ساختار کلی یک فایل PDF را به‌عنوان یک ساختار فایل پیچیده بیان می‌کنیم. در فصل ۱ اشاره کردیم که ویژگی‌های کامل قالب PDF بسیار زیاد است. بیشتر این ویژگی‌ها، در حدود ۷۰ درصد، در ارتباط با توضیح اشیای داده[1] و ارتباط آنها بین بخش‌های مختلف یک فایل PDF هست. تمرکز اصلی در این پیوست نیز بر روی ساختار اشیای داده خواهد بود. ما ساختار کلی PDF را در دو بخش فیزیکی و منطقی بررسی خواهیم کرد.

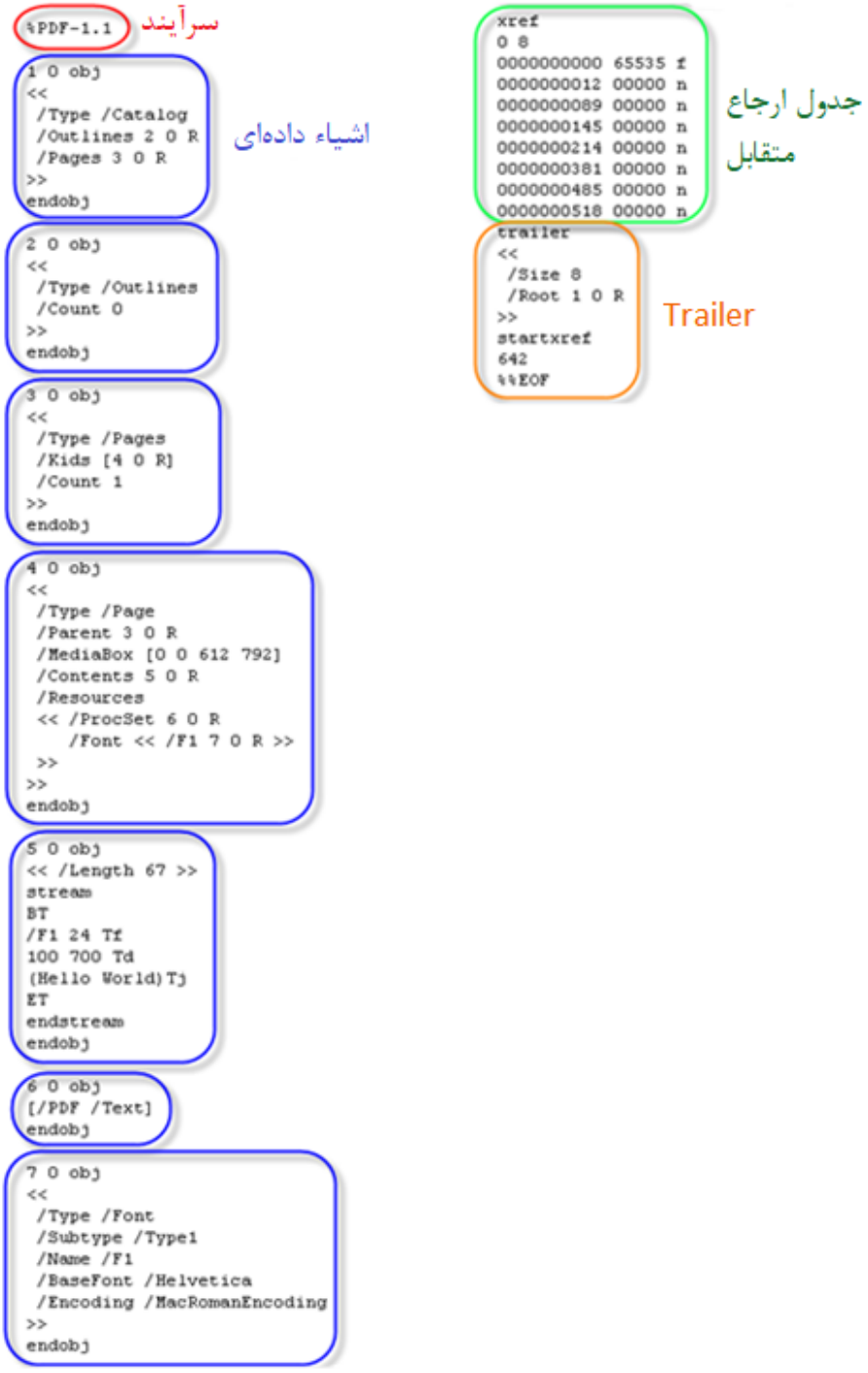
فایل‌های PDF در یک قالب متنی کدگذاری می‌شوند که ممکن است شامل جریان دودویی[2] مانند تصویر و غیره باشند. یک فایل PDF ترکیبی از دست‌کم یک بدنه[3] است. یک بدنه از سه بخش تشکیل شده است: شیء[4] (obj)، جدول ارجاع متقابل[5] (xref) و Trailer. در ابتدای هر فایل یک سرآیند قرار می‌گیرد که با عبارت PDF% آغاز شده و در ادامه آن یک عدد که نگارش قالب PDF را مشخص می‌کند، آمده است. شکل آ-۱ یک فایل PDF شامل متن Hello Wrold را که در یک ویراستار متنی باز شده است، نشان می‌دهد. در ادامه این ساختار را بررسی می‌کنیم.

آ-۱ ساختار فیزیکی

جزئیات ساختار فیزیکی بدنه فایل PDF به شرح زیر است:

- **اشیاء.** داده و فراداده در پرونده‌های PDFدر یک واحد اولیه که شیء نامیده می‌شود سازماندهی می‌شوند. اشیا همگی قالب مشابهی دارند که در شکل آ-۱ مشخص است و همچنین یک ساختار بیرونی مشترک هم دارند. اولین خط یک شیء شناسه آن است که برای ارجاع‌های غیر مستقیم استفاده

[1] Data Object
[2] Binary Stream
[3] Body
[4] Object
[5] Cross-reference Table



شکل آـ۱ : یک فایل ساده PDF باز شده در یک ویراستار متنی شامل عبارت Hello Wrold. چپ: سرآیند فایل و اشیای داده‌ای؛ اشیاء با مستطیل آبی (شماره‌های ۱ تا ۷) مشخص شده‌اند. راست: ادامه محتویات همان فایل، شامل جدول ارجاع متقابل و Trailer.

می‌شود. در ادامه عدد تولیدی آن است که اگر شی با یک نسخه جدیدتر بازنویسی شود، افزایش می‌یابد. سپس رشته "obj" که شروع یک شیء را مشخص می‌کند آمده و پس از آن محتویات شیء قرار داده می‌شود. رشته "endobj" نیز پایان یافتن محدوده یک شیء را مشخص می‌کند.

- **جدول ارجاع متقابل.** این جدول در بدنه PDF شامل آدرس نسبی اشیای مورد ارجاع قرار گرفته داخل یک فایل به صورت بایت می‌باشد. در شکل آ ـ ۱ این جدول شامل ۷ شیء با شناسه یک تا ۷ و یک مکان نگهدارنده برای شناسه صفر است که به هیچ شیئی اشاره نمی‌کند.

- **Trailer.** این قسمت از بدنه فایل PDF شامل یک فرهنگ لغت[1] از اطلاعات بدنه، مثل تعداد کل اشیای داخل فایل (/size 8)، شناسه و شماره ترتیبی شیء ریشه (/root 1 0 R) و غیره است. فرهنگ‌لغت بین نمادهای >> و << قرار می‌گیرد. ادامه Trailer شامل startxref است که آدرس نسبی شروع جدول ارجاع متقابل در فایل را مشخص می‌کند. این فن اجازه می‌دهد تا بدنه از پایان با خواندن startxref پویش شود، سپس به جدول ارجاع متقابل بازگشته و آن را نیز پویش می‌کند. بدین ترتیب تنها اشایی پویش می‌شوند که نیاز هستند. در شکل آ ـ ۱ آدرس شروع جدول ارجاع متقابل که پس از startxref در Trailer آمده است برابر با ۶۴۲ است. در صورتی که آدرس‌های بایت‌های این فایل بررسی شود، مشاهده خواهد شد که بایت ۶۴۲ام شروع جدول ارجاع متقابل (کاراکتر 'x') است. در پایانِ Trailer نیز عبارت %%EOF قرار می‌گیرد که پایانِ فایل را مشخص می‌کند.

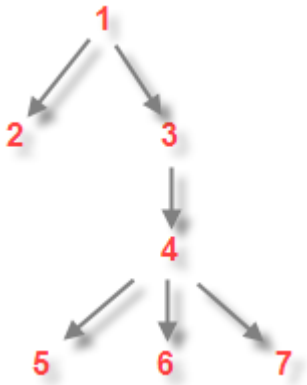
ساختار جدول ارجاع متقابل و Trailer به‌نسبت ساده و در فایل‌های گوناگون مشابه است، اما اشیای PDF انواع محتلفی دارند. برای مثال شیء شماره ۱ در شکل آ ـ ۱، حاوی یک ساختار فرهنگ لغت است لذا بین نمادهای >> و << واقع‌شده و حاوی کلیدهایی می‌شود که با کاراکتر / شروع شده و در ادامه مقادیر آنها آمده است. مثلا مقدار 2 0 R یک ارجاع به شیئی در همین فایل با شناسه ۲ و شماره ترتیبی ۰ است. از آنجایی که یک فایل ممکن است خیلی بزرگ باشد؛ آدرس نسبی شیئی که به آن ارجاع داده شده است از طریق جدول ارجاع متقابل که یک جدول دسترسی تصادفی است، قابل دستیابی خواهد بود.

اشیای PDF تنها محدود به فرهنگ لغت نمی‌شوند گرچه به‌نظر می‌آید که بیشتر آنها دست کم شامل یک فرهنگ لغت هستند. شیء شماره ۵ در شکل آ ـ ۱ برای نمونه، حاوی یک جریان دودویی است که بین واژه‌های کلیدی stream و endstream قرار گرفته است. تصاویر داخل PDF نمونه‌ای از جریان‌های دودویی هستند که به این صورت می‌توانند ظاهر شوند. شکل آ ـ ۲، یک شیء حاوی تصویر را نشان می‌دهد که در یک ویراستار متنی باز شده است. شیء شماره ۶ در شکل آ ـ ۱ یک آرایه از کلیدها را شامل می‌شود. مقادیر آرایه می‌توانند انواع مختلفی داشته باشند که در هر صورت بین نمادهای] و [قرار می‌گیرند و با کاراکتر فاصله از یکدیگر تمییز داده می‌شوند. به‌دلیل تنوع ذکر شده در ساختار اشیای PDF، قواعد تعریف و ترکیب این اشیاء بیشترین بخش از توضیحات مشخصه‌های قالب فایل PDF را تشکیل می‌دهند.

[1] Dictionary

```
1397 0 obj
<</Filter/FlateDecode/Length 160>>
stream
xœ32U0P0RÐ5T02T06WH1ä*T0´EOT
SUB(CAN[(CANéEMSTXåEDr.—"'—¾§¯BIQi*—~8PDC1—¾BELP
—¾S€³,!—¾K´!—A,—¾›¾³³ESCESC`c£ï
ÔÍ¥ïœŸSš›WFFÔbgÇåé¢ ¶ðô\ûKÿÿêÿ¿tÊÏ(Ks#„
$¸ÿM›þ²*z}fŽÓáNULFACKSOH.WO…@.NULÕH1£
endstream
endobj
```

شکل آ ـ ۲: یک شی PDF حاوی یک تصویر. محتوای دودویی تصویر بین stream و endstream قرار گرفته است.



شکل آ ـ ۳: ساختار منطقی فایل PDF نشان داده شده در شکل آ ـ ۱. گره‌های درخت شناسه اشیاء در فایل PDF و یال‌های آن معرف نحوه فراخوانی هر شیء توسط شیء دیگر هستند.

آ ـ ۲ ساختار منطقی

آنچه در بخش آ ـ ۱ صحبت شد مربوط ساختار فیزیکی یک فایل PDF و نحوه قرار گرفتن اجزای فایل در کنار یکدیگر بود (مرتبط با گام پویش در هنگام اجرا). ساختار منطقی فایل PDF، یعنی قواعد حاکم بر نحوه تفسیر و پردازش آنچه در یک فایل قرار دارد و ارتباط بین اجزا، یک ساختار سلسله مراتبی است (مرتبط با گام پرداخت در هنگام اجرا). شناسه شیء ریشه در Trailer مشخص می‌شود. به‌همین ترتیب شیء یا اشیای بعدی مورد نیاز در شیء ریشه معلوم می‌گردد. در فایل PDF شکل آ ـ ۱ شیء شماره ۱ ریشه است. اشیای شماره ۲ و ۳ داخل بدنه شیء ۱ مورد دسترسی قرار می‌گیرند و در نهایت ترتیب نحوه دسترسی‌ها به‌صورت یک درخت قابل نمایش است که ساختار منطقی فایل را نشان می‌دهد. شکل آ ـ ۳ ساختار منطقی فایل نشان داده شده در شکل آ ـ ۱ را نشان می‌دهد.

ساختار فیزیکی یک فایل PDF را می‌توان بدون تغییر ساختار منطقی آن، به شکل دیگری تبدیل کرد. برای مثال در شکل آ ـ ۱ اشیاء به‌ترتیب صعودی شناسه خود در فایل ظاهر شده بودند. می‌شود ترتیب قرارگرفتن

```
xref
0 8
0000000000 65535 f
0000000565 00000 n
0000000509 00000 n
0000000440 00000 n
0000000273 00000 n
0000000169 00000 n
0000000136 00000 n
0000000012 00000 n
```

شکل آـ۴: جدول ارجاع متقابل بروزرسانی شده در حالتی که اشیاء به ترتیب نزولی شناسه خود در فایل PDF ظاهر شده‌اند.

اشیاء را به‌صورت نزولی در آورد. در این صورت جدول ارجاع متقابل بایستی بروزرسانی شود؛ زیرا، هر شی در در مکان متفاوتی نسبت به قبل قرار گرفته است. اما این پدیده تأثیری بر ساختار منطقی ندارد. ساختار منطقی کماکان همان ساختار شکل آـ۳ و نتیجه اجرا نیز با فایل قبلی یکسان خواهد بود. جدول ارجاع متقابل برای حالتی که اشیاء به صورت نزولی در فایل ظاهر شده باشند، در شکل آـ۴ نشان داده شده است.

به‌طور کلی اشیاء PDF می‌توانند در مکان‌های تصادفی داخل یک فایل PDF ظاهر شوند بدون اینکه تأثیر بر پرداخت فایل داشته باشند. برای فایل ساده شکل آـ۱ تنها با تعویض ترتیب ظاهر شدن اشیاء می‌توان تعداد ۵۰۴۰ (برابر با !۷) فایل با ساختار فیزیکی متفاوت داشت. این درحالی است که تغییر ترتیب قرارگیری اشیا تنها یک روش برای جابه‌جایی ساختار فیزیکی فایل PDF است و روش‌های دیگری مثل تغییر محل قرارگیری جدول ارجاع متقابل نیز وجود دارد. بنابراین رابطه بین ساختار منطقی و فیزیکی فایل PDF به صورت یک به چند است.

آـ۳ بروزرسانی فایل PDF

دیدیم که چگونه ساختار منطقی فایل PDF مستقل از ساختار فیزیکی آن شکل می‌گیرد. بروزرسانی فایل PDF بر همین مبنا است. فایل‌های PDF می‌توانند به‌صورت افزایشی[1] بروزرسانی شوند. بدین ترتیب که اگر نویسنده فایل PDF بخواهد اطلاعات داخل شیء شماره ۷ را بروزرسانی کند، یک بدنه جدید آغاز می‌کند. سپس محتوای شیء جدید را داخل آن نوشته، عدد تولیدی (عدد بعد از شناسه شیء) را یک واحد نسبت به عدد تولیدی شیء قدیم افزایش داده و برای شی جدید می‌نویسد. در انتها نیز یک جدول ارجاع متقابل را که به شیء جدید اشاره می‌نماید، تولید کرده و بدنه جدید را به سند قبلی الصاق می‌کند.

[1] Incremental

پیوست ب

فازر IUST DeepFuzz

در این پیوست جزئیات پیاده‌سازی و استفاده از روش پیشنهادی خود را در قالب محصول نرم‌افزاری IUST DeepFuzz شرح می‌دهیم. کلیه کدها و مستندات این محصول از طریق صفحه پروژه در وب‌سایت GitHub[1] قابل مشاهده و دریافت است. برنامه به‌صورت مجموعه‌ای از اسکریپت‌های زبان پایتون نوشته شده است؛ لذا، روی هر سیستم عاملی پس از نصب کتابخانه‌ها و چارچوب‌های لازم قابل اجرا خواهد بود. فایل های پروژه، به‌تفکیک وظایف، در تعدادی دایرکتوری سازمان‌دهی شده است. همچنین بخشی از فازر که عملیات آزمون فازی، پایش خطا و ابزارگذاری کد SUT را انجام می‌دهد، به‌صورت یک فایل Batch در سیستم عامل ویندوز نوشته شده است که در صورت تغییر سیستم عامل یا هریک از ابزارهای پایش و ابزارگذاری SUT، بایستی تغییر داده شود. پیمانه تولیدکننده خودکار داده‌های آزمون اما نیاز به تغییر ندارد.

ب‌ـ۱ پیش‌نیازهای نرم‌افزاری و سخت‌افزاری

پیش‌نیازهای نرم‌افزاری پروژه در جدول ب‌ـ۱ آمده است. پس از نصب تمامی بسته‌ها اسکریپت‌های اصلی پروژه قابل اجرا خواهد بود. برای تسریع آموزش مدل‌ها استفاده از سیستمی با پردازش‌گر گرافیکی قوی (تعداد هسته‌های بالا) توصیه می‌شود ولی الزامی نیست. همچنین پرادزش‌گر مرکزی قوی و حافظه اصلی بالا (۸ گیگابایت و بیشتر) برای اجرای قابل قبول پروژه مورد نیاز خواهد بود.

ب‌ـ۲ ساختار سطح بالای پروژه

در دایرکتوری ریشه پروژه تعدادی فایل و تعدادی زیردایرکتوری قرار دارند که بخش‌های مختلف برنامه را تشکیل می‌دهند. در انتهای نام هر فایل یک عدد ترتیبی قرار دارد که نسخه آن را مشخص می‌کند. در میان فایل‌های دارای یک نام بدین ترتیب فایل با نسخه بالاتر اجرایی است و نیازی به فایل‌های قبلی نیست.

[1] https://github.com/m-zakeri/iust_deep_fuzz

جدول ب_۱ : بسته‌های نرم‌افزاری مورد نیاز جهت اجرای صحیح برنامه IUST DeepFuzz.

ردیف	بسته نرم‌افزاری	نسخه مورد نیاز
۱	Python	نسخه ۳/۵ و بالاتر
۲	TensorFlow	نسخه ۱/۵ و بالاتر
۳	Keras	نسخه ۲/۲/۰

دایرکتوری‌ها عبارتند از:

- **دایرکتوری ریشه.** در این دایرکتوری ریشه فایل‌های اصلی پیکربندی[1]، فایل تعریف مدل‌های ژرف، فایل آموزش مدل و تولید داده از مدل و نیز فایل راهنما وجود دارد.
- **دایرکتوری batch_jobs.** این دایرکتوری کدهای مربوط به پیمانه‌های تزریق و پایش (خطا و پوشش کد) فازر را شامل می‌شود.
- **دایرکتوری binary_to_base64.** این دایرکتوری برای کارهای آتی رزرو شده و در نسخه فعلی محصول به‌کار نمی‌رود.
- **دایرکتوری dataset.** این دایرکتوری همان‌طور که از نام آن پیداست. محل قرار گیری مجموعه داده مورد آموزش و آزمون مدل‌ها است. برای قالب‌های فایل جدید کافی است مجموعه داده را زیردایرکتوری مختص به آن قرار دهیم. درحال حاضر مجموعه داده خوبی برای فایل‌های PDF و مجموعه داده کوچکی نیز برای فایل‌های XML داخل آن قرار دارند.
- **دایرکتوری generated_results.** در این دایرکتوری داده‌های آزمون جدید تولید شده توسط مدل‌های ژرف ذخیره می‌شوند.
- **دایرکتوری incremental_update .** این دایرکتوری شامل کدهای مربوط به بروزرسانی افزایشی و ساخت فایل‌های PDF جدید است. بنابراین مختص آزمون فازی قالب فایل PDF است و در قالب‌های فایل دیگر کاربردی ندارد.
- **دایرکتوری logs_csv.** در این دایرکتوری فایل‌های گزارش ضبط شده در فرایند آموزش مدل‌ها ذخیره می‌شود. این فایل‌ها شامل میزان خطا، دقت و سرگشتگی مدل در هر دوره اموزش هستند.
- **دایرکتوری logs_tensorboard.** فایل‌های گزارش مربوط به ابزار مصورسازی Tensorboard در این دایرکتوری ذخیره می‌شود. این فایل‌ها سپس توسط همین ابزار خوانده و گزارش‌های دقت، خطا، معماری مدل و غیره را به‌صورت تصویری در اختیار آزمون‌گر قرار می‌دهد.

[1] Configuration

- **دایرکتوری model_checkpoint .** در پایان هر دوره آموزش مدل، یک نمونه از مدل در این دایکتوری ذخیره می‌شود. هر نمونه مستقل از سایر نمونه‌های بوده است. آزمون‌گر بدین ترتیب پس از اتمام فرایند آموزش می‌تواند بهترین مدل را برای تولید داده آزمون انتخاب کند.
- **دایرکتوری modelpic.** این دایرکتوری شامل گراف محاسباتی مربوط به مدل‌های ژرف است که توسط کتابخانه Keras تولید شده است.
- **دایرکتوری seed.** این دایرکتوری پوشش کد تمامی فایل‌های دانه اولیه آزمون را ذخیره می‌کند.

ب_۳ پیکربندی پروژه

در حال حاضر این محصول مبتنی بر GUI نیست و کلیه تنظیمات از طریق فایل config.py در دایرکتوری ریشه مشخص می‌شود و مهمترین فایلی است که در پروژه در اختیار آزمون‌گر قرار دارد. این فایل شامل تعدادی فرهنگ لغت است. هر فرهنگ لغت تعدادی فیلد به‌صورت کلید_ مقدار دارد که کلید نام تنظیم ورودی بود که توسط برنامه مشخص شده است و مقدار آن توسط آزمون‌گر قبل از انجام آزمون تعیین می‌شود. فایل config.py کاملاً مستندگذاری شده و مقادیر قابل استفاده برای هر کلید در مقابل آن درج شده است. برای تغییر آن کافی است یکی از مقادیر معتبر را به‌جای مقدار پیش‌فرض قرار دهید. مهمترین تنظیم‌های قابل اعمال در این فایل عبارتند از:

- **file_fromat.** این کلید قالب فایل مورد آزمون را مشخص می‌کند. برای مثال PDF یا XML.
- **training_set_path.** این کلید آدرس مجموعه آموزش را مشخص می‌کند.
- **validation_set_path.** این کلید آدرس مجموعه ارزیابی را مشخص می‌کند.
- **testing_set_path.** این کلید آدرس مجموعه آزمون را مشخص می‌کند.
- **new_objects_pathh.** این کلید آدرس محل ذخیره داده‌های آزمون تولید شده را مشخص می‌کند.

افزون بر تنظیمات بالا، تعدادی از تنظیمات مختص آزمون فازی قالب PDF هستند. این تنظیمات در فرهنگ لغت iu_config تعریف شده‌اند؛ از جمله:

- **single_object_update.** این کلید مقادیر True و False را می‌گیرد و همان‌طور که از نام آن پیداست تعیین می‌کند که هنگام تولید فایل PDF جدید تنها یک شیء بروزرسانی شود و یا چند شیء.
- **portion_of_rewrite_objects.** این کلید نسبت تعداد اشیای مورد بروزرسانی را در حالت MOU مشخص می‌کند. بنابراین تنها وقتی که تنظیم single_object_update برابر False است به‌کار می‌رود.

- **update_policy.** این کلید نوع انتخاب اشیاء برای بروزرسانی را مشخص می‌کند که می‌تواند یکی از موارد random برای انتخاب تصادفی، bottom_up برای انتخاب برحسب شماره نزولی اشیاء و top_down برای انتخاب برحسب شماره صعودی اشیاء باشد.
- **raw_host_directory.** این کلید آدرس فایل‌های میزبان را مشخص می‌کند.

ب_۴ پیاده‌سازی‌ها

هر یک از اسکریپت‌های پایتون بخشی از پروژه را پیاده‌سازی می‌کند. فایل deep_models.py هریک از مدل‌های ژرف را در قالب یک تابع پایتون تعریف کرده است. فایل lstm_text_generation_pdf_objs_9datafuzz.py الگوریتم فاز عصبی داده و فایل lstm_text_generation_pdf_objs_9formatfuzz.py الگوریتم فاز عصبی فراداده را پیاده‌سازی کرده است. همچنن فایل iu_6.py بروزرسانی افزایشی PDF را پیاده‌سازی می‌کند.

ب_۴_۱ تعریف مدل‌ها

در Keras به دو صورت می‌توان مدل‌های یادگیری را تعریف کرد. یک روش با استفاده از پشته‌کردن لایه‌ها مختلف که برای مدل‌های ترتیبی به‌کار می‌رود و یک روش با استفاده از کلاس Model و API‌های آن، که برای ساخت مدل‌هایی با چندین ورودی و خروجی و به‌طور کلی معماری‌های پیچیده‌تر استفاده می‌شود. تعریف مدل‌های ما با روش اول انجام شده است. در برنامه ب_۱ تعریف مدل‌ها را آورده‌ایم.

```
1 from keras.models import Sequential, load_model
2 from keras.layers import Dense, Activation, Dropout
3 from keras.layers import LSTM, Bidirectional
4
5 # Unidirectional LSTM (Many to One)
6 def model_1(input_dim, output_dim):
7     print('Build model...')
8     model = Sequential()
9     model.add(LSTM(128, input_shape=input_dim))
10    model.add(Dense(output_dim))
11    model.add(Activation('softmax'))
12    return model, 'model_1'
13
```

```
14 # Unidirectional LSTM (Many to One)
15 def model_2(input_dim, output_dim):
16     model = Sequential()
17     model.add(LSTM(128, input_shape=input_dim,
    return_sequences=True))
18     model.add(LSTM(128, input_shape=input_dim,
    return_sequences=False))
19     model.add(Dense(output_dim))
20     model.add(Activation('softmax'))
21     return model, 'model_2'
22
23 # Unidirectional LSTM (Many to One)
24 def model_3(input_dim, output_dim):
25     model = Sequential()
26     model.add(LSTM(256, input_shape=input_dim,
    return_sequences=True, recurrent_dropout=0.1))
27     model.add(Dropout(0.3))
28     model.add(LSTM(256, input_shape=input_dim,
    return_sequences=False, recurrent_dropout=0.1))
29     model.add(Dropout(0.3))
30     model.add(Dense(output_dim))
31     model.add(Activation('softmax'))
32     return model, 'model_3'
33
34 # Bidirectional LSTM (Many to One)
35 def model_4_1(input_dim, output_dim):
36     model = Sequential()
37     model.add(Bidirectional(LSTM(256, return_sequences=
    False), input_shape=input_dim, merge_mode='sum'))
38     model.add(Dense(output_dim))
```

```
39      model.add(Activation('softmax'))
40      return model, 'model_4_1'
41
42  # Bidirectional Deep LSTM (Many to One)
43  def model_4_2(input_dim, output_dim):
44      model = Sequential()
45      model.add(Bidirectional(LSTM(128, return_sequences=True
    ), input_shape=input_dim,merge_mode='sum'))
46      model.add(Bidirectional(LSTM(128, return_sequences=
    False), merge_mode='sum'))
47      model.add(Dense(output_dim))
48      model.add(Activation('softmax'))
49      return model, 'model_4_2'
50  # End of model definition
```

برنامه ب_۱ : پیاده‌سازی مدل‌های یادگیری در Keras

ب_۴_۲ آموزش و تولید داده

کدهای مربوط به فرایند آموزش مدل و تولید داده از مدل در یک فایل قرار دارند به ترتیب اجرا می‌شوند. در اینجا از ذکر کدها امتناع می‌کنیم؛ زیرا طولانی بوده و فایل کامل در مخزن پروژه وجود دارد. اجرای نهایی برنامه پس از انجام تنظیمات گفته شده از طریق فراخوانی هر فایل در خط‌فرمان سیستم عامل در مسیر پروژه انجام می‌شود. برای آموزش مدل می‌توان تعداد دوره محدودی درنظر گرفت و سپس با بررسی میزان خطا آموزش را متوقف ساخت.

ب_۴_۳ پیاده‌سازی آزمون فازی

آزمون فازی همان‌طور که گفتیم توسط یه برنامه Batch روی سیستم عامل ویندوز نوشته شده است. برنامه ب_۲ یک پیاده‌سازی اولیه از آن را نشان می‌دهد.

```
1  REM IUST DeepFuzz version 0.2
2  REM @ECHO off
```

```
SET PATH=%PATH%;"C:\Program Files (x86)\Microsoft Visual
    Studio 14.0\Team Tools\Performance Tools"
CD D:\SUT_DIRECTORY
CLS

START VSPerfMon /coverage /output:fuzzing_code_coverage.
    coverage
timeout /t 5

FOR %%i IN (.\TEST_DATA\*.pdf) DO (
  mupdf %%i
  timeout /t 10
  taskkill /IM mupdf.exe
  timeout /t 2
  )

timeout /t 5
VSPerfCmd /shutdown
```

برنامه ب_۲: پیاده‌سازی آزمون فازی

ب_۵ ملاحظات عملی

در پیاده‌سازی همواره با برخی از پیچیدگی‌ها روبه‌رو می‌شویم که در تئوری مسئله وجود ندارد. در این پروژه تبدیل تمامی توالی‌های آموزشی به بردارهای عددی نیازمند حافظه بالای ۲۰ گیگابایت بود. به عبارت دیگر امکان بارگذاری یک‌باره همه مجموعه آموزش در حافظه اصلی نبود. برای حل این مشکل، هر بار یک دسته از توالی‌ها را انتخاب، آنها را از حافظه جانبی وارد حافظه اصلی کرده و پس از شرکت دادن در فرایند آموزش، این دسته را خارج و دسته بعدی را می‌خوانیم. این امکان با نوشتن از یک تابع از نوع Generator به کد برنامه اضافه گردید. مصرف حافظه به‌این ترتیب با تعیین اندازه دسته، در اختیار برنامه‌نویس و آزمون‌گر خواهد بود.

واژه‌نامه فارسی به انگلیسی

ا

آزمون ... Test
آزمون امنیت ... Security Testing
آزمون فازی ... Fuzz Testing
آزمون فشار ... Stress Testing
آزمون قدرت‌مندی ... Robustness Testing
آزمـون نفوذ ... Penetration Testing
آسیب‌پذیری ... Vulnerability
ابَر پارامتر ... Hyper-parameter
ابزارگذاری ... Instrumenting
اتکاپذیری ... Dependability
اختلال ... Noise
ارزیابی بیرونی ... Extrinsic Evaluation
ارزیابی درونی ... Intrinsic Evaluation
اشکال ... Error
اعتبارسنجی ... Validation
افراز فضای ورودی ... Input Space Partitioning
افزایشی ... Incremental
افزوده ... Augmented
الگوی آسیب‌پذیر ... Vulnerable Pattern
انتهابه‌انتها ... End to End
انفجار ... Explosion
اینترنت چیزها ... Internet of Things

ب

باج‌افزار ... Ransomware
بازنمایی ... Representation
بانظارت ... Supervised
بایاس ... Bias
بایت جادویی ... Magic Bytes
بدشکل ... Malformed
بدنه ... Body
برچسب ... Label
برنامه شناسنده ... Recognizer Program
برنامه مولد ... Generator Program
بلوک پایه ... Basic Block
بهره‌برداری ... Exploit
بیش‌برازش ... Overfitting
بیشینه هموار ... Softmax

پ

پاک‌ساز ... Filter
پایش ... Monitor
پرداخت ... Render
پردازه پس‌زمینه ... Background Process
پس‌انتشار ... Backpropagation
پوشش بلوک پایه ... Basic Block Coverage
پوشش خط ... Line Coverage
پوشش خط جزئی ... Partial Line Coverage
پوشش دامنه ... Domain Coverage
پوشش دستور ... Statement Coverage
پوشش ساختار نحوی ... Syntax-Based Coverage

پوشش شاخه ..... Branch Coverage
پوشش کد ..... Code Coverage
پوشش گراف ..... Graph Coverage
پوشش مسیر ..... Path Coverage
پوشش منطق ..... Logic Coverage
پیشوند ..... Prefix
پیکربندی ..... Configuration
پیکره ..... Corpus
پیمانه ..... Module

ت

تابع ادغام ..... Merge Function
تابع انگیزش ..... Activation Function
تابع خطا ..... Error Function
تابع هزینه ..... Cost Function
تاریخچه ..... History
تجزیه ..... Parse
تجزیه‌گر ..... Parser
تصمیم‌ناپذیر ..... Undecidable
تطبیق‌پذیر ..... Adaptive
تعمیم‌پذیر ..... Generalization
تنوع ..... Diversity
توالی ..... Sequence
توزیع چندجمله‌ای .... Multinomial Distribution
توکن ..... Token
توکن دودویی ..... Binary Token

ج

جدول ارجاع متقابل ..... Cross-reference Table
جریان دودویی ..... Binary Stream
جعبه خاکستری ..... Gray Box
جعبه سفید ..... White Box
جعبه سیاه ..... Black Box

ح

حریصانه ..... Greedy

خ

خرابی ..... Failure
خطا ..... Fault
خطای آنتروپی متقاطع ..... Cross Entropy Error
خطای مطلق میانگین ..... Mean Absolute Error
خطای میانگین مربعات .... Mean Squared Error
خوش‌شکل ..... Well-formed

د

داده آزمون ..... Test Data
دانه اولیه ..... Initial Seed
درستی‌یابی ..... Verification
درهم‌آمیزی ..... Shuffle
دوره ..... Epoch
دوسویه ..... Bidirectional

ر

روبه‌جلو ..... Forward
روبه‌عقب ..... Backward

ز

زمینه ..... Context

س

سازگار ..... Consistent
سرگشتگی ..... Perplexity
سروش ..... Oracle

ش

شبکه عصبی ..... Neural Network
شبکه عصبی روبه‌جلو ..... Feed-forward Neural Network
شبکه عصبی ژرف ..... Deep Neural Network
شیء ..... Object
شیء داده ..... Data Object

ض

ضد فازینگ ..... Anti-fuzzing
ضریب انشعاب ..... Branching Factor

ط

طبقه‌بندی ..... Classification
طراحی ..... Design

ع

عصب ..... Neuron

ف

فازر ..... Fuzzer
فازر قالب فایل ..... File Format Fuzzer
فراداده ..... Metadata
فرهنگ لغت ..... Dictionary
فساد حافظه ..... Memory Corruption

ق

قابلیت حمل ..... Portability

ک

کد منبع ..... Source Code
کدگذار ـ کدگشا ..... Encoder-Decoder
کمینه‌سازی دانه ..... Seed Minimization
کنترل استثنا ..... Exception Handling

گ

گام ..... Step
گام زمانی ..... Time Step
گذرِجلو ..... Forward Pass

ل

لایه پنهان ..... Hidden Layer

م

مبتنی بر تولید ..... Generation Based
مبتنی بر جابه‌جایی ..... Mutation Based
مجموعه آزمون ..... Test Set
مجموعه آموزش ..... Training Set
مجموعه ارزیابی ..... Validation Set
مدل مولد ..... Generative Model

مشخصه ..... Specification
معتبر ..... Valid
معیارهای پوشش ..... Coverage Criteria
مفهوم ..... Concept
منظم‌سازی ..... Regularization
مؤلفه ..... Component
مهاجم ..... Attacker
مهندسی معکوس ..... Reverse Engineering
میانگیر ..... Buffer
میزبان ..... Host

ن

ناپدید شدن ..... Vanishing
نرخ یادگیری ..... Learning Rate
نرمال‌سازی ..... Normalization
نشانگر ..... Markers
نیازمندی ..... Requirement

و

واحد ..... Unit
واژگان ..... Vocabulary
وظیفه ..... Task

ه

هموارسازی ..... Smoothing

ی

یادگیری بانظارت ..... Supervised Learning
یادگیری ژرف ..... Deep Learning
یادگیری و فاز ..... Learn&Fuzz
یک‌سویه ..... Unidirectional

واژه‌نامه انگلیسی به فارسی

A

تابع انگیزش ..... Activation Function
تطبیق‌پذیر ..... Adaptive
ضد فازینگ ..... Anti-fuzzing
مهاجم ..... Attacker
افزوده ..... Augmented

B

پردازه پس‌زمینه ..... Background Process
پس‌انتشار ..... Backpropagation
روبه‌عقب ..... Backward
بلوک پایه ..... Basic Block
پوشش بلوک پایه ..... Basic Block Coverage
بایاس ..... Bias
دوسویه ..... Bidirectional
جریان دودویی ..... Binary Stream
توکن دودویی ..... Binary Token
جعبه سیاه ..... Black Box
بدنه ..... Body
پوشش شاخه ..... Branch Coverage
ضریب انشعاب ..... Branching Factor
میانگیر ..... Buffer

C

طبقه‌بندی ..... Classification
پوشش کد ..... Code Coverage
مؤلفه ..... Component
مفهوم ..... Concept
پیکربندی ..... Configuration
سازگار ..... Consistent
زمینه ..... Context
پیکره ..... Corpus
تابع هزینه ..... Cost Function
معیارهای پوشش ..... Coverage Criteria
خطای آنتروپی متقاطع ..... Cross Entropy Error
جدول ارجاع متقابل ..... Cross-reference Table

D

شیء داده ..... Data Object
یادگیری ژرف ..... Deep Learning
شبکه عصبی ژرف ..... Deep Neural Network
اتکاپذیری ..... Dependability
طراحی ..... Design
فرهنگِ لغت ..... Dictionary
تنوع ..... Diversity
پوشش دامنه ..... Domain Coverage

E

کدگذار_کدگشا ..... Encoder-Decoder
انتهابه‌انتها ..... End to End

دوره ..... Epoch
اشکال ..... Error
تابع خطا ..... Error Function
کنترل استثنا ..... Exception Handling
بهره‌برداری ..... Exploit
انفجار ..... Explosion
ارزیابی بیرونی ..... Extrinsic Evaluation

F

خرابی ..... Failure
خطا ..... Fault
شبکه عصبی روبه‌جلو ..... Feed-forward Neural Network
فازر قالب فایل ..... File Format Fuzzer
پاک‌ساز ..... Filter
روبه‌جلو ..... Forward
گذرِجلو ..... Forward Pass
آزمون فازی ..... Fuzz Testing
فازر ..... Fuzzer

G

تعمیم‌پذیر ..... Generalization
مبتنی بر تولید ..... Generation Based
مدل مولد ..... Generative Model
برنامه مولد ..... Generator Program
پوشش گراف ..... Graph Coverage
جعبه خاکستری ..... Gray Box
حریصانه ..... Greedy

H

لایه پنهان ..... Hidden Layer
تاریخچه ..... History
میزبان ..... Host
ابَر پارامتر ..... Hyper-parameter

I

افزایشی ..... Incremental
دانه اولیه ..... Initial Seed
افراز فضای ورودی .... Input Space Partitioning
ابزارگذاری ..... Instrumenting
اینترنت چیزها ..... Internet of Things
ارزیابی درونی ..... Intrinsic Evaluation

L

برچسب ..... Label
یادگیری و فاز ..... Learn&Fuzz
نرخ یادگیری ..... Learning Rate
پوشش خط ..... Line Coverage
پوشش منطق ..... Logic Coverage

M

بایت جادویی ..... Magic Bytes
بدشکل ..... Malformed
نشان‌گر ..... Markers
خطای مطلق میانگین ..... Mean Absolute Error
خطای میانگین مربعات .... Mean Squared Error
فساد حافظه ..... Memory Corruption
تابع ادغام ..... Merge Function
فراداده ..... Metadata

پیمانه ..... Module
پایش ..... Monitor
توزیع چندجمله‌ای .... Multinomial Distribution
مبتنی بر جابه‌جایی ..... Mutation Based

N

شبکه عصبی ..... Neural Network
عصب ..... Neuron
اختلال ..... Noise
نرمال‌سازی ..... Normalization

O

شیء ..... Object
سروش ..... Oracle
بیش‌برازش ..... Overfitting

P

تجزیه ..... Parse
تجزیه‌گر ..... Parser
پوشش خط جزئی ..... Partial Line Coverage
پوشش مسیر ..... Path Coverage
آزمون نفوذ ..... Penetration Testing
سرگشتگی ..... Perplexity
قابلیت حمل ..... Portability
پیشوند ..... Prefix

R

باج‌افزار ..... Ransomware
برنامه شناسنده ..... Recognizer Program
منظم‌سازی ..... Regularization
پرداخت ..... Render
بازنمایی ..... Representation
نیازمندی ..... Requirement
مهندسی معکوس ..... Reverse Engineering
آزمون قدرت‌مندی ..... Robustness Testing

S

آزمون امنیت ..... Security Testing
کمینه‌سازی دانه ..... Seed Minimization
توالی ..... Sequence
درهم‌آمیزی ..... Shuffle
هموارسازی ..... Smoothing
بیشینه هموار ..... Softmax
کد منبع ..... Source Code
مشخصه ..... Specification
پوشش دستور ..... Statement Coverage
گام ..... Step
آزمون فشار ..... Stress Testing
بانظارت ..... Supervised
یادگیری بانظارت ..... Supervised Learning
پوشش ساختار نحوی .. Syntax-Based Coverage

T

وظیفه ..... Task
آزمون ..... Test
داده آزمون ..... Test Data
مجموعه آزمون ..... Test Set
گام زمانی ..... Time Step
توکن ..... Token
مجموعه آموزش ..... Training Set

U

تصمیم‌ناپذیر ..... Undecidable
یک‌سویه ..... Unidirectional
واحد ..... Unit

V

معتبر ..... Valid
اعتبارسنجی ..... Validation
مجموعه ارزیابی ..... Validation Set
ناپدید شدن ..... Vanishing
درستی‌یابی ..... Verification
واژگان ..... Vocabulary
آسیب‌پذیری ..... Vulnerability
الگوی آسیب‌پذیر ..... Vulnerable Pattern

W

خوش‌شکل ..... Well-formed
جعبه سفید ..... White Box

Abstract:

Fuzz testing (Fuzzing) is a dynamic software testing technique. In this technique with repeated generation and injection of malformed test data to the software under test (SUT), we are looking for the possible faults and vulnerabilities. To this goal, fuzz testing requires varieties of test data. The most critical challenge is to handle the complexity of the file structures as program input. Surveys have revealed that many of the generated test data in these cases follow restricted numbers and superficial paths, because of being rejected by the parser of SUT in the initial stages of parsing. Using the grammatical structure of input files to generate test data lead to increase code coverage. However, often, the grammar extraction is performed manually, which is a time consuming, costly and error-prone task. In this thesis, we proposed an automated method for hybrid test data generation. To this aim, we apply neural language models (NLMs) that are constructed by recurrent neural networks (RNNs). The proposed models by using deep learning techniques can learn the statistical structure of complex files and then generate new textual test data, based on the grammar, and binary data, based on mutations. Fuzzing the generated data is done by two newly introduced algorithms, called neural fuzz algorithms that use these models. We use our proposed method to generate test data, and then fuzz testing of MuPDF complicated software which takes portable document format (PDF) files as input. To train our generative models, we gathered a large corpus of PDF files. Our experiments demonstrate that the data generated by this method leads to an increase in the code coverage, more than 7%, compared to state of the art file format fuzzers such as American fuzzy lop (AFL). Experiments also indicate a better learning accuracy of simpler NLMS in comparison with the more complicated encoder-decoder model and confirm that our proposed models can outperform the encoder-decoder model in code coverage when fuzzing the SUT.

Keywords: Fuzz Testing, Test Data, Code Coverage, Deep Learning, Recurrent Neural Network.

Iran University of Science and Technology
School of Computer Engineering

Automatic Test Data Generation in File Format Fuzzers

A Thesis Submitted in Partial Fulfillment of the Requirement for the Degree of Master of Science in Computer Engineering

By:

Morteza Zakeri Nasrabadi

Supervisor:

Dr. Saeed Parsa

September 2018