نشریه علمی «پدافند الکترونیکی و سایبری»

سال هشتم، شماره ۱، بهار ۱۳۹۹؛ ص **۱۵-۱**

تولید خودکار داده آزمون در فازرهای قالب فایل

مرتضی ذاکری نصرآبادی[۱]، سعید پارسا[۲*]

۱- دانشجوی دکتری تخصصی، ۲- دانشیار، دانشگاه علم و صنعت ایران، تهران.

(دریافت: ۹۷/۱۲/۲۱، پذیرش: ۹۸/۳/۲۸)

چکیده

آزمون فازی یک فن آزمون پویای نرم‌افزار است. در این فن با تولید ورودی‌های بدشکل و تزریق پی‌درپی آن‌ها به نرم‌افزار تحت آزمون، دنبال یافتن خطاها و آسیب‌پذیری‌های احتمالی آن هستیم. ورودی اصلی بسیاری از نرم‌افزارهای دنیای واقعی فایل است. تعداد زیادی از داده‌های آزمون که برای آزمون فازی این نرم‌افزارها تولید می‌شوند در همان مراحل اوّلیه به‌علّت نداشتن قالب مورد قبول، توسط پویش‌گر فایل برنامه رد می‌شوند. در نتیجه شاهد پوشش کم کد برنامه در روند آزمون فازی هستیم. استفاده از گرامرِ ساختار فایل برای تولید داده آزمون، منجربه افزایش پوشش کد می‌گردد، اما این گرامر معمولاً به‌صورت دستی تهیه می‌شود که کاری زمان‌بر، پرهزینه و مستعد خطا است. در این مقاله روشی نو با استفاده از مدل‌های زبانی عصبی ژرف برای یادگیری خودکار ساختار فایل و سپس تولید و فاز داده‌های آزمون ارائه شده است. آزمایش‌های ما بهبود پوشش کد روش پیشنهادی را در مقایسه با دیگر روش‌های تولید داده آزمون نشان می‌دهد. برای نرم‌افزار MuPDF که قالب فایل پیچیده PDF را به‌عنوان ورودی می‌پذیرد، بیش از ۱/۳۰ تا ۱۲ درصد بهبود پوشش کد را نسبت به روش‌های هوشمند و روش تصادفی داشته‌ایم.

کلیدواژه‌ها: آزمون فازی، داده آزمون، پوشش کد، شبکه عصبی مکرر، مدل زبانی، یادگیری ژرف

۱. مقدمه

آزمون فازی [۴-۱] یک فن آزمون پویای نرم‌افزار است که در آن برنامه با ورودی‌های مختلفی اجرا می‌گردد. این ورودی‌ها توسط ابزاری به نام فازر[۱] و به‌صورت خودکار تولید می‌شوند [۵]. هرچه‌قدر این ورودی‌ها متنوع‌تر باشند، پوشش کد برنامه هنگام آزمـون افزایش داشته و کشف خطاها و آسیب‌پذیری‌های احتمالی افزایش می‌یابد. لذا، الگوریتم تولید خودکار داده آزمون در فازرها بسیار حائز اهمیت است. ورودی بسیاری از نرم‌افزارهای دنیای واقعی، به‌شکل فایل بوده و فازرهای توسعه داده شده برای آزمون فازی آنها، با مشکل اساسی آگاهی از ساختار فایل، هنگام تولید داده آزمون مواجه هستند. بایستی ابزاری داشت که قادر باشد داده‌های آزمون را با توجه به ساختار فایل تولید کند؛ تا هم محتوا و هم قالب را آن چنان تغییر دهد که بتوان کلیه مسیرهای اصلی اجرایی برنامه را بررسی و پوشش داد. در صورت تولید داده‌های آزمون به‌شکل تصادفی یا بدون در نظر گرفتن ساختار ورودی، به‌خصوص برای ساختارهای ورودی پیچیده مانند فایل‌ها، آزمون فازی پوشش کد بسیار پایینی خواهد داشت. در نتیجه امکان کشف خطا به‌شدت کاهش می‌یابد [۶]. AFL [۷] و SampleFuzz [۸] دو ابزار مطرح در زمینه آزمون فازی قالب فایل، مبتنی بر روش‌های هوشمند هستند.

فازر AFL [۷] با استفاده از یک فرایند تکاملی، جمعیتی تصادفی از دانه‌های اولیه[۲] را چنان اصلاح می‌کند که وقتی داده‌های جدید به‌عنوان ورودی، به نرم‌افزار تحت آزمون (SUT)[۳] داده شوند؛ در مجموع تعداد بیشتری از دستورالعمل‌های برنامه اجرا گردیده و پوشش یابند. چالش عمده AFL این است که چون بدون دانش پیرامون ساختار فایل، برای نمونه فایل PDF، مبادرت به ایجاد فایلی جدید، از ترکیب قبلی‌ها می‌نماید، فایل حاصل ممکن است برچسب‌ها و اشیای مورد نیاز را برای پیروی از ساختار فایل مورد نظر نداشته باشد. چنین فایلی در عمل غیرقابل استفاده است؛ زیرا، به احتمال قوی در همان مراحل اولیه اجرا، توسط تجزیه‌گرِ[۴] SUT رد شده و لذا فقط مسیرهای سطحی

*رایانامه نویسنده پاسخگو: parsa@iust.ac.ir

1- Fuzzer

2- Initial seeds

3- Software under test

4- Parser

و تکراری را پوشش می‌دهد [۶]. AFL همچنین در جهش و تولید داده‌های آزمون با حجم زیاد (بیشتر از ۱ مگابایت) بسیار کند عمل می‌کند و هفته‌ها تا ماه‌ها زمان لازم است تا شاید به پوشش کد مورد قبولی دست یابد.

برای حل مسائل بالا، داده آزمون را با استفاده از قالب یا گرامر ورودی تولید می‌کنند. اما، این قالب یا گرامر به‌صورت دستی و از روی مستندات تهیه می‌شود که با توجه به حجیم بودن آن برای ساختارهای پیچیده‌ای همچون فایل PDF، عملی زمان‌بر، پرهزینه و مستعد خطاست [۸]. همچنین ممکن است برای یک قالب فایل همواره مشخصه‌های آن در اختیار نباشد که در این حالت نوشتن گرامر بسیار سخت و تا حدی غیرممکن خواهد بود.

گودفروید و همکاران [۸]، برای نخستین بار، روشی تحت عنوان کلی *یادگیری و فاز*[۱] برای استخراج خودکار گرامر فایل با استفاده از مدل‌های یادگیری ژرف ارائه کردند. در واقع در این روش یک مدل مولد روی یک مجموعه از فایل‌های موجود، ایجاد شده و سپس با استفاده از این مدل، فایل‌های مورد نظر برای آزمون نرم‌افزار مربوطه تولید می‌گردد. اما روش یادگیری و فاز [۸] مشکلاتی دارد. از جمله مدل استفاده شده در این روش برای نگاشت توالی‌ها از دو دامنه مختلف به یکدیگر استفاده می‌شود. این روش بیشتر در مسائلی مانند ترجمه ماشینی کاربرد دارد [۱۰-۹]. برای یادگیری ساختار فایل می‌توان از مدل‌های ساده‌تری مثل مدل زبانی احتمالاتی [۱۱] استفاده کرد. همچنین روش مذکور، 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 نمونه‌برداری از مدل بوده‌اند. از طرفی فقط داده‌های متنی فایل در این روش تولید و فاز شده‌اند؛ حال آنکه یک فایل با ساختار پیچیده، در حالت کلی حاوی ترکیبی از داده‌های متنی و غیرمتنی (دودویی) است. در نهایت الگوریتم ارائه شده برای تولید داده‌های جدید، تحت عنوان SampleFuzz [۲]، ممکن است هیچ‌گاه پایان نیابد.

در روش پیشنهادی با مطالعه‌ای که روی ابزارها و الگوریتم‌های یادگیری ژرف و پردازش زبان طبیعی (NLP)[۲]، به‌عمل آمد به‌جای روش کدگذار-کدگشا[۳] (توالی‌به‌توالی[۴]) [۱۰-۹]، که پیش از این در [۸] استفاده شده بود، از مدل‌های زبانی عصبی (NLM)[۵] [۱۲]، استفاده گردیده است. در عمل شاهد بهبود میزان پوشش کد آزمون فازی، نسبت به فازرهای AFL [۷]، SampleFuzz [۸] و FileFuzz [۱۳] بوده‌ایم. همچنین مدل‌های زبانی عصبی، در یادگیری ساختار فایل به دقت بیشتری نسبت به مدل توالی‌به‌توالی استفاده‌شده در [۸]، دست یافتند. فازر ما تحت عنوان *IUST-DeepFuzz* قادر به تشخیص خودکار ساختار فایل‌های مختلف و تولید داده‌های آزمون جدید است. به‌عنوان یک نمونه پیچیده، توانستیم ساختار قالب فایل PDF [۱۴] را توسط این ابزار استخراج کرده و در پی آن، ابزار مربوطه را روی نرم‌افزار MuPDF [۱۵] با استفاده از فایل‌های تولید شده، مورد استفاده قرار دادیم. در عمل نشان داده شد که تعدادی هشدار امنیتی و استفاده از توابع ناامن، در نسخه آزمون شده از MuPDF وجود دارد. به‌طور خلاصه مهم‌ترین نوآوری‌ها و دستاوردهای این مقاله عبارتند از:

- استفاده از مدل‌های زبانی عصبی برای یادگیری ساختار فایل‌های پیچیده، مثل PDF.
- تولید و فاز (بدشکل‌سازی) خودکار و هوشمند داده‌های آزمون با استفاده از مدل‌های مولد.
- ارائه یک الگوریتم ترکیبی تولید داده آزمون، تحت عنوان NeuralFuzz، که بخش‌های متنی را با روش مبتنی بر تولید[۶] و بخش‌های دودویی را با روش مبتنی‌بر جابه‌جایی[۷] ایجاد می‌کند.
- ارائه یک فازر قالب فایل، تحت عنوان *IUST-DeepFuzz*، برای آزمون فازی نرم‌افزارهای واقعی.
- جمع‌آوری و انتشار یک مجموعه داده حاوی بیش از ۵۰۰ هزار شئ داده‌ای PDF برای یادگیری ماشینی ساختار پیچیده این قالب فایل.

ساختار ادامه این مقاله بدین شرح است: در بخش ۲، مفاهیم اولیه و کارهای مرتبط با آزمون فازی، پوشش کد و نیز مدل‌های زبانی عصبی را مطرح می‌کنیم. بخش ۳ را به تبیین روش پیشنهادی خود که شامل مدل‌هایی برای یادگیری ساختار فایل و نیز الگوریتمی برای تولید و بدشکل‌سازی فایل است، اختصاص داده‌ایم. در بخش ۴، آزمایش‌های انجام شده و ارزیابی روش پیشنهادی را بیان کرده و در نهایت در بخش ۵، به نتیجه‌گیری و ذکر تعدادی از کارهای آتی، خواهیم پرداخت.

1- Learn & Fuzz
2- Natural Language Processing
3- Encoder-decoder
4- Sequence-to-Sequence
5- Neural Language Model
6- Generation Based
7- Mutation Based

۲. مفاهیم اولیه و کارهای مرتبط

۲-۱. آزمون فازی

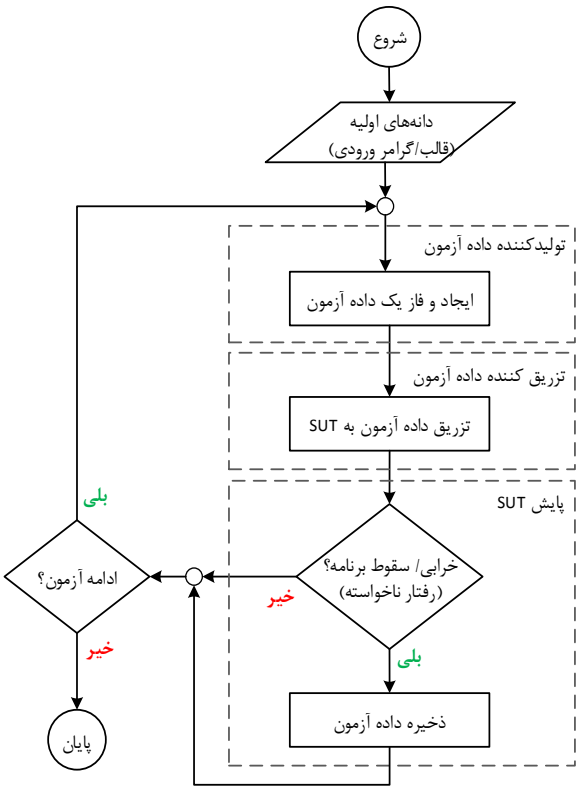
آزمون فازی فرایند ساده تولید و سپس تزریق یک ورودی ناخواسته (بدشکل شده یا نامتعارف[1]) به SUT است. چنان‌چه برنامه بر اثر پردازش این ورودی ناخواسته، دچار خرابی شود، حافظه برنامه مورد تحلیل قرار گرفته و خطای احتمالی موجود در کد آشکار می‌گردد [۴-۱]. چون SUT با تعداد ورودی‌های بسیار زیادی مورد آزمون قرار می‌گیرد، آزمون فازی را می‌توان نوعی آزمون فشار[2]، آزمون نفوذ[3] و آزمون قدرتمندی[4] هم به‌شمار آورد [۱۶،۱۳]. فرایند معمول آزمون فازی، در شکل (۱) نشان داده شده است.

۲-۱-۱. فازر

فازر ابزاری است که فرایند آزمون فازی را خودکار می‌کند. پیاده‌سازی هریک از پیمانه‌های شکل (۱) و تجمیع آنها در کنار یکدیگر یک فازر را ایجاد می‌کند [۱۶]. تمرکز اصلی در یک فازر، نحوه تولید داده آزمون است به‌گونه‌ای که می‌توان آن را وجه تمایز اصلی فازرهای مختلف دانست. فازری که برای آزمون برنامه‌های با ورودی فایل توسعه داده می‌شود، *فازر قالب فایل*[5] هم نامیده می‌شود [۵، ۱۳]. روش‌های تولید داده در فازرها قابل تفکیک به دو دسته کلی روش‌های مبتنی بر جابه‌جایی (یا جهش) و روش‌های مبتنی بر تولید هستند [۱۷].

در روش مبتنی بر جابه‌جایی، تعداد یک یا بیشتر داده ورودی معتبر به‌عنوان دانه اولیه برای تولید داده‌های آزمون بیشتر به‌کار می‌رود. دانه اولیه سپس جهش (تغییر یا جابه‌جایی ناگهانی) می‌یابد تا داده آزمون دیگری تولید شود. 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 [۱۸]. AFL [۷] یک فازر مبتنی بر جهش است.

روش مبتنی‌بر تولید، داده‌های آزمون را کاملاً تصادفی یا از روی یک توصیف صوری مانند گرامر، قالب یا مدل تولید می‌کند [۱۹]. در حالت آخر، از مشخصه‌های داده ورودی، برای ساخت یک مدل مولد استفاده می‌شود. این روش بیشتر روی قالب‌های فایلی که مستندی از مشخصه‌های آنها در دسترس است به‌کار می‌رود و معمولاً در مقایسه با فازرهای مبتنی بر جابه‌جایی، به پوشش کد بالاتری دست می‌یابد. اما همان‌طور که گفتیم زمان و هزینه زیادی باید صرف شود تا مشخصه‌های قالب فایل کاملاً فهمیده و مدل خوبی از آن تهیه شود. SAGE [۲۰] یک فازر مبتنی بر تولید است. فازر ارائه شده در این مقاله یک فازر ترکیبی است که داده‌های متنی را با روش مبتنی بر گرامر و داده‌های دودویی را با روش مبتنی بر جابه‌جایی تولید و حاصل را ترکیب می‌کند.



شکل (۱): روندنمای فرایند آزمون فازی در حالت ساده، پیمانه‌های مورد نیاز برای خودکارسازی فرایند، با مستطیل خط‌چین مشخص شده‌اند [۱۶].

کارهای متعددی در زمینه آزمون فازی قالب فایل انجام شده است. یعقوبی [۲۱] یک روش تولید داده آزمون برمبنای الگوریتم ژنتیک و میزان حافظه مصرفی برنامه حین اجرای آن ارائه داده است ولی تمرکز آن برروی ساختارهای ساده است. امینی [۲۲]

1- Mal-Formed
2- Stress Testing
3- Penetration Testing
4- Robustness Testing
5- File Format Fuzzer

سعی در شناسایی فراخوانی‌های سیستمی و اجرای مسیرهای حاوی این فراخوانی‌ها داشته است، اما روش وی نیازمند اطلاع از ساختار ورودی SUT است. رحیمی و پنجی [۲۳] عملکردهای جابه‌جایی FileFuzz [۱۳] را که یک فازر تصادفی است، بهبود داده‌اند. روش پیشنهادی آنها مبتنی‌بر جابه‌جایی است و تلاشی در راستای افزایش پوشش کد نمی‌کند.

۲-۱-۲. پوشش کد

آزمون نرم‌افزار قادر است وجود خرابی[1] را نشان دهد ولی نبود آن را تضمین نمی‌کند. به‌بیان صوری، مسئله یافتن تمامی خرابی‌ها در یک برنامه تصمیم‌ناپذیر[2] است [۲۴]. این موضوع سبب می‌شود به‌دنبال راه‌کارهایی برای اندازه‌گیری آزمون و تعیین میزان خوب بودن آن باشیم. یک معیار برای این منظور پوشش کد است. پوشش کد یعنی چه میزان از کد برنامه توسط مجموعه داده‌های آزمون پوشش داده شده است. این معیار خود در سطوح مختلفی مطرح است؛ پوشش دستور، پوشش شاخه و پوشش مسیر سه سطح شناخته شده هستند [۲۵-۲۴].

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 بالأخره در پوشش مسیر اجرای حداقل یک‌مرتبه هر مسیر اجرایی مد نظر است. بدیهی است که پوشش مسیر، پوشش شاخه و پوشش شاخه، پوشش دستور را شامل می‌شود [۲۴]. پوشش بلوک‌پایه[3] 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 پوشش بلوک پایه سادگی اندازه‌گیری آن و عدم نیاز به وجود کد منبع برنامه است [۲۵]. یعنی از آن می‌توان در آزمون جعبه سیاه و جعبه خاکستری نیز که کد منبع در اختیار آزمون‌گر نیست، استفاده کرد. در این مقاله نیز پوشش بلوک‌پایه را محاسبه و گزارش می‌کنیم.

۲-۲. مدل زبانی

مدل زبانی[4] یک مفهوم پایه در NLP است که امکان پیش‌بینی نشانه بعدی در یک توالی را فراهم می‌کند. به‌بیان دقیق‌تر مدل زبانی عبارت است از یک توزیع احتمالی روی یک توالی از نشانه‌ها (اغلب واژه‌ها) که احتمال وقوع یک توالی داده‌شده را مشخص می‌کند. درنتیجه می‌توان بین چندین توالی داده شده برای مثال چند جمله، آن را که محتمل‌تر است، انتخاب کرد [۱۱]. برای توالی داده شده $x = < x^{(1)}, \dots, x^{(n)} >$ مدل زبانی به صورت زیر تعریف می‌شود [۱۸]:

$$p(x) = \prod_{t=1}^{n} p\left(x^{(t)} | x^{(<t)}\right) \quad (۱)$$

در رابطه (۱) هر جمله منفرد $p\left(x^{(t)} | x^{(<t)}\right)$ احتمال شرطی نشانه $x^{(t)}$ به شرط وقوع (ظاهر شدن) t نشانه قبلی است که به آن زمینه یا تاریخچه می‌گویند. در عمل محاسبه این احتمال به‌صورت رابطه (۱)، تقریباً غیرممکن است؛ زیرا، نیازمند داشتن همه توالی‌های ممکن هستیم. مدل‌های سنتی n-gram برای غلبه بر چالش‌های محاسباتی، با استفاده از فرض مارکوف، رابطه (۱) را به درنظر گرفتن تنها n-1 نشانه قبلی محدود می‌کنند. اگرچه در بسیاری از مسائل این مدل‌ها به خوبی پاسخ‌گو هستند؛ اما، برای توالی‌های طولانی (بیشتر از ۴ یا ۵ نشانه) و نیز مشاهده‌نشده مناسب نیستند؛ زیرا به توالی‌های مشاهده نشده مقدار صفر را نسبت می‌دهند.

۲-۲-۱. مدل زبانی عصبی

برای حل مشکلات n-gram می‌توان یک رده خاص از شبکه‌های عصبی ژرف به نام شبکه عصبی مکرر (RNN)[5] [۲۶] را برای ساخت مدل زبانی استفاده کرد. مدل‌های زبانی حاصل را مدل زبانی عصبی می‌گویند [۱۲]. 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 [۲۷]. RNNها رده‌ای از شبکه‌های عصبی هستند که به‌صورت یک گراف جهت‌دار دارای دور بیان می‌شوند. به‌عبارت دیگر ورودی هریک از لایه(های) پنهان یا لایه خروجی افزون بر خروجی لایه قبل، شامل ورودی از گام زمانی قبل به‌صورت بازخورد نیز می‌شود. شکل (۲) گراف محاسباتی یک RNN را نشان می‌دهد.

در هر گام زمانی t یک بردار $x^{(t)}$ از توالی ورودی پردازش می‌شود. معادلات گذرجلو شبکه در t عبارتند از [۲۶]:

$$z^{(t)} = Ux^{(t)} + Wh^{(t-1)} + b \quad (۲)$$

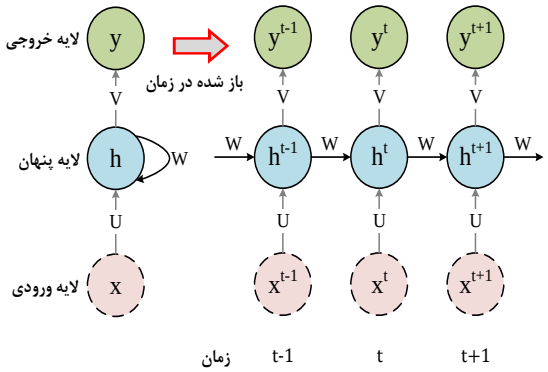
$$h^{(t)} = \sigma(z^{(t)}) \quad (۳)$$

$$y^{(t)} = Vh^{(t)} + c \quad (۴)$$

$$\hat{y}^{(t)} = softmax(y^{(t)}) \quad (۵)$$

1- Fault
2- Undecidable
3- Basic-Block
4- Language Model
5- Recurrent Neural Network

در روابط (۲) تا (۵)، b و c مقادیر بایاس و ماتریس‌های U، V و W به‌ترتیب وزن یال‌های لایه ورودی 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. برای آموزش شبکه عصبی ژرف ابتدا یک تابع هدف تعریف و سپس با استفاده از الگوریتم پس‌انتشار[2] پارامترهای شبکه که همان وزن یال‌های گراف محاسباتی هستند، تنظیم می‌شوند که به این مرحله گذرعقب می‌گویند [۲۶].



شکل (۲): گراف محاسباتی یک شبکه عصبی مکرر با یک لایه پنهان [۲۶].

مدل‌های زبانی را مدل مولّد هم می‌نامند؛ زیرا یک توزیع احتمالی روی توالی‌های یک زبان می‌دهند که با نمونه‌برداری از آن می‌توان توالی‌های جدید تولید کرد. در وظایف NLP نشانه‌های هر توالی واژه‌ها هستند، اما می‌توان این مدل را در سطح کاراکتر نیز آموزش داد. در روش پیشنهادی از این مدل‌ها در سطح کاراکتر استفاده خواهیم کرد.

۲-۲-۲. سرگشتگی

سرگشتگی[3] معیاری است برای ارزیابی مدل زبانی که به‌صورت زیر تعریف می‌شود [۲۸]:

$$pp_{LM}(x) = \sqrt[n]{\prod_{i=1}^{n} p(x^{(i)}| < x^{(1)}, \dots, x^{(i-1)} >)} = 2^{-\frac{1}{n}\sum_{i=1}^{n} \log_2 p(x^{(i)}|<x^{(1)},\dots,x^{(i-1)}>)} \quad (۶)$$

در رابطه (۶)، x توالی مورد ارزیابی است. سرگشتگی میزان اختلاف بین مدل و نمونه‌های مجموعه آزمون را نشان می‌دهد و هرچه‌قدر پایین‌تر باشد، مدل زبانی بهتر است. در بدترین حالت میزان سرگشتگی برابر با اندازه مجموعه واژگان زبان است. حال چنان‌چه از مدل زبانی استفاده کنیم، احتمال نسبت داده‌شده به وقوع هر نشانه، نسبت به حالت عدم استفاده از مدل زبانی، افزایش و در نتیجه سرگشتگی کاهش خواهد یافت.

۳. روش پیشنهادی

چالش اصلی در یادگیری ساختار فایل، تعیین قسمت‌هایی است که در قالب فایل ثابت هستند. هدف از یادگیری ساختار امکان تولید فایل‌هایی است که تاحد زیادی منطبق بر مشخصه‌های قالب فایل باشند. اما هدف از آزمون فازی در مقابل فاز یا بدشکل‌کردن فایل به عنوان ورودی برنامه است. در روش پیشنهادی ابتدا مدلی برای یادگیری خودکار ساختار فایل ارائه می‌دهیم. سپس یک الگوریتم برای فاز هدفمند بخش‌هایی از فایل همزمان با تولید آن پیشنهاد می‌کنیم. وقتی داده‌ها بر اساس گرامر تولید می‌شوند قادر به گذشتن از شرایط اولیه تجزیه‌گر بوده و درصد بیشتری از کدهای برنامه را اجرا می‌کنند.

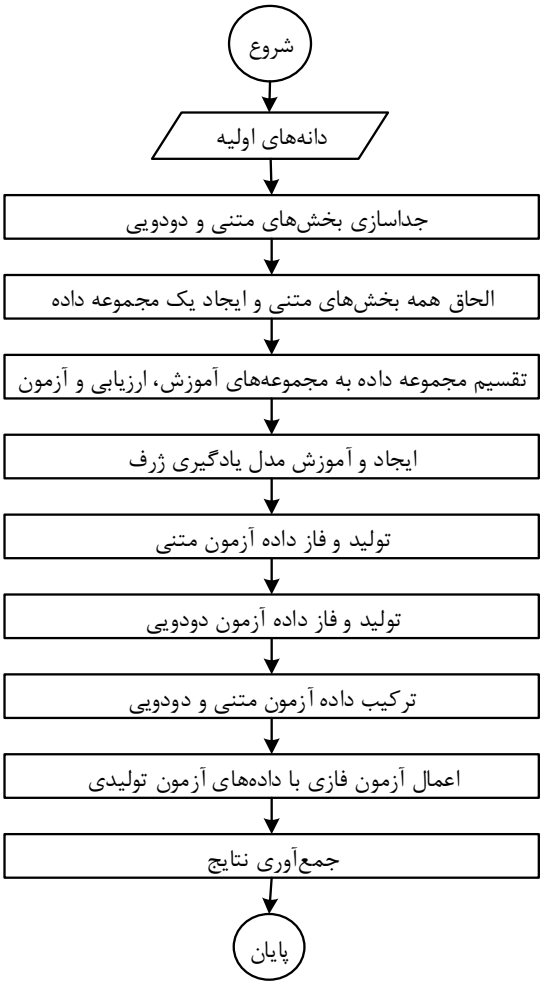
۳-۱. کلیات

شکل (۳)، یک روندنمای کلی از مراحل روش پیشنهادی ما را نشان می‌دهد. در ساختار قالب‌های فایل پیچیده مانند PDF، بسیاری از قسمت‌ها به‌صورت متنی (کدهای ascii) هستند؛ اما، ممکن است قسمت‌های دودویی (کدهای غیر ascii) هم وجود داشته باشد. در این مقاله، منظور از قسمت‌های متنی، کاراکترهای ascii و منظور از قسمت‌های دودویی هر نشانه غیر ascii به‌کار رفته در یک فایل است. در فایل PDF، قسمت‌هایی دودویی مابین کلیدواژه‌های *stream* و *endstream* ظاهر می‌شوند و در واقع مبیّن بخش‌هایی مانند متن‌ها و تصاویر هستند؛ در مقابل، بخش عمده فایل، به‌شکل متنی بوده و ساختار اصلی آن را توصیف می‌کند [۱۴]. این تفکیک، حین خواندن فایل PDF، قابل انجام است. در سایر قالب‌ها نیز تفکیک قسمت‌های متنی و دودویی به‌آسانی، با یک مقایسه ساده، بر اساس تعریف مفهوم دودویی و متنی، امکان‌پذیر خواهد بود.

در روش پیشنهادی خود، پس از جمع‌آوری تعدادی فایل با ساختار کاملاً درست، از یک قالب مشخص مانند PDF، به‌عنوان دانه اولیه، کلیه قسمت‌های متنی فایل‌های موجود در مجموعه دانه اولیه را استخراج کرده و سپس این قسمت‌ها را به یکدیگر الحاق می‌کنیم. حاصل این کار ایجاد یک پیکره متنی بزرگ است که در بطن خود مشخصه‌های قالب یک فایل را دارد. قسمت‌های

1- Softmax
2- Back-Propagation
3- Perplexity

دودویی را نیز به‌صورت مجزا ذخیره کرده و در محل وقوع آنها در قسمت متنی، یک نشانه متنی یکتا، قرار می‌دهیم. ما این نشانه را *توکن دودویی*[1] می‌نامیم. مجموعه داده در این مرحله، شامل داده‌های متنی بوده که از نمونه فایل‌های با ساختار درست حاصل شده است؛ زیرا هدف یادگیری هرچه بهتر ساختار فایل است. در ادامه، این مجموعه را مطابق با ضوابط فنون یادگیری ماشینی به سه مجموعه آموزش، ارزیابی و آزمون با نسبت‌های مشخص تقسیم‌بندی می‌کنیم. سپس یک مدل مولد را ایجاد نموده و آن را روی داده‌های مجموعه آموزش، با تعریف یک روش یادگیری، آموزش می‌دهیم. برای این‌که فرایند آموزش به‌خوبی انجام شود، مطابق یک حساب سرانگشتی در وظایف NLP، بایستی به میزان $O(v^2)$ نشانه در پیکره متنی وجود داشته باشد که v اندازه مجموعه واژگان است [۱۱]. براین اساس می‌توان تعداد دانه‌های اولیه را نیز تعیین کرد.



شکل (۳): مراحل روش پیشنهادی

در مرحله بعد، از این مدل، برای ایجاد داده‌های آزمون جدید استفاده خواهیم کرد. از آن‌جایی‌که مدل روی داده‌های متنی آموزش می‌بیند، صرفاً قادر به تولید داده‌های متنی است که مطلوب نیست. در نتیجه پس از تولید و فاز داده آزمون متنی، در مرحله بعدی، قسمت‌های دودویی را که قبلاً تفکیک و ذخیره کرده بودیم، با روش مبتنی بر جابه‌جایی فاز و بازتولید کرده، سپس به داده آزمون متنی تزریق می‌نماییم. توکن دودویی، مکان تزریق داده‌های دودویی در داخل داده آزمون متنی را، به‌شکل احتمالی، برای ما مشخص می‌سازد. بدین ترتیب یک روش ترکیبی برای تولید داده‌های آزمون حاصل می‌گردد که داده‌های متنی را با روش مبتنی بر تولید، براساس یک مدل مولد، تولید کرده و داده‌های دودویی را با روش مبتنی بر جابه‌جایی، براساس تصادف، تولید کرده و در نهایت هر دو قسمت را ترکیب می‌کند.

علّت تفکیک اولیه داده‌های متنی و دودویی از یکدیگر، در این روش، دشواری یادگیری قسمت‌هایی دودویی فایل است. قسمت‌های دودویی، معمولاً در سطح بیت تفسیر می‌شوند، برخلاف قسمت‌های متنی که در سطح بایت (کاراکتر) به‌کار گرفته می‌شوند. یادگیری وابستگی‌های بین بیت‌ها در قسمت‌های دودویی، با مدل‌های ژرف امیدبخش نبود. در واقع هنگامی که ورودی و خروجی مدل زبانی را در سطح بیت تعریف کردیم، شبکه به‌خوبی آموزش ندید و همگرا نشد. یعنی قادر به تولید یک توالی معنادار از بیت‌ها نبود که در نتیجه نمی‌توان داده‌های آزمون دودویی مناسب از این طریق تولید کرد. از طرفی روش‌های مبتنی بر جهش در آزمون فازی ساختارهای دودویی ساده، بسیار مؤثر واقع شده‌اند [۸-۶]. بنابراین، ایده یک روش ترکیبی می‌تواند برای ساختارهای پیچیده مناسب‌تر باشد. در مرحله پایانی روش پیشنهادی، آزمون نرم‌افزار، با استفاده از داده‌های آزمون تولیدی، صورت می‌پذیرد.

۳-۲. مدل

برای ایجاد مدل‌های زبانی در اینجا دو مدل با معماری مختلف بر مبنای RNN می‌سازیم. در هسته عصب‌های RNN هر دو مدل از سلول LSTM[2] [۲۹] استفاده می‌کنیم که قابلیت یادگیری توالی‌های با طول زیاد را دارد. مدل اول LSTM یکسویه نام دارد که معماری آن مشابه شکل (۲) است. مدل دوم LSTM دوسویه است. LSTM دوسویه توالی ورودی را به دو صورت روبه‌جلو و روبه‌عقب می‌بیند. این مدل از دو LSTM یک‌سویه تشکیل شده است. یکی از آنها توالی را از چپ به راست پردازش می‌کند و

1- Binary Token

2- Long Short-Term Memory

دیگری از راست به چپ. در نتیجه هر گذرجلو LSTM دو خروجی خواهد داشت. معمولاً از یک تابع برای ادغام هریک از خروجی‌ها استفاده می‌شود. تابع ادغام می‌تواند هر تابعی که قادر به ترکیب دو بردار با طول‌های یکسان است، درنظر گرفته شود؛ از جمله: جمع، ضرب و الحاق. ما از تابع جمع برای مدل پیشنهادی خود استفاده می‌کنیم که درایه‌های نظیر به نظیر هریک از بردارهای خروجی را با یکدیگر جمع می‌زند.

علاوه‌بر نوع مدل، برای LSTMهای یک‌سویه مدل‌هایی با تعداد عصب‌های متفاوت و نیز تعداد 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 (۱)، مدل‌های طراحی شده برای استفاده در آزمایش‌ها و مشخصه‌های هریک را نشان می‌دهد. یکی از مهم‌ترین مشخصه‌ها تعداد پارامترهای هر مدل است که پیچیدگی آن را مشخص می‌کند.

۳-۲-۱. آموزش مدل

فرایند آموزش همه مدل‌های جدول (۱) 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 دربردارنده قالب 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. جداسازی مجموعه آزمون در روش پیشنهادی ما، یکی از ایده‌های نو به‌شمار می‌رود که ارزیابی بهتر مدل‌های یادگیری را فراهم می‌آورد. 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 هستند برچسب‌گذاری می‌شود. بسیاری از قالب‌های فایل چنین توکن‌هایی به‌عنوان بخشی از ساختار خود دارند. برای مثال فایل‌های HTML، XML و PHP، چنین هستند.

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 $S =< s^{(1)}, \dots, s^{(n)} >$ ایجاد می‌کنیم. سپس این توالی را به تعدادی توالی ورودی کوچکتر با طول ثابت d می‌شکنیم طوری که iامین توالی ورودی برابر است با:

$$x_i = S[i * j : (i * j) + d]) \qquad (۷)$$

در رابطه (۷)، S[l..u] برشی از توالی S بین شاخص‌های l و u بوده و j گام پرش است که میزان پرش به جلو در انتخاب توالی ورودی بعدی از توالی اصلی را نشان می‌دهد. 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 به‌عنوان برچسب برای هر توالی ورودی نسبت می‌دهیم. بنابراین، برچسب خروجی برای i امین توالی ورودی برابر خواهد بود با:

جدول (۱): مدل‌های یادگیری ژرف طراحی شده و مشخصه‌های آنها.

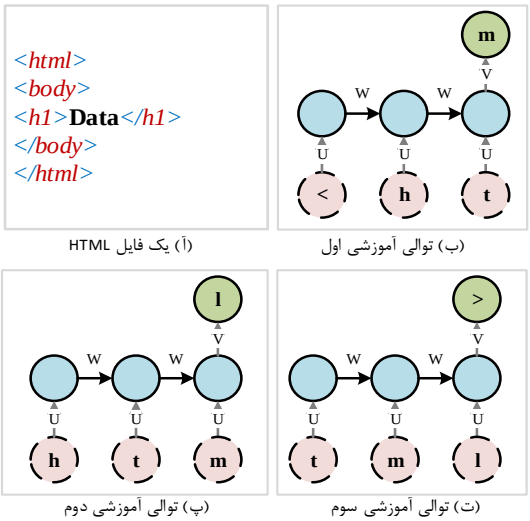
شماره مدل	نوع مدل	تعداد لایه(ها)ی پنهان	تعداد عصب(ها) در هر لایه	تعداد پارامترها
۱	Unidirectional LSTM	۱	۱۲۸	۱۲۷۵۸۴
۲	Unidirectional LSTM	۲	۱۲۸	۲۳۸۶۵۶
۳	Unidirectional LSTM	۲	۲۵۶	۸۷۰۴۶۴
۴	Bidirectional LSTM	۲	۱۲۸	۴۶۹۰۵۶

$$Y_{x_i} = S[(i * j) + d + 1]) \qquad (۸)$$

پس از تولید همه نمونه‌های آموزشی و برچسب‌های متناظر آنها، مدل به‌صورت انتهابه‌انتها[1] آموزش داده می‌شود؛ یعنی ویژگی‌ها از داده‌های خام بیرون کشیده می‌شوند. در این حالت مدل ایجاد شده، قادر به پیش‌بینی احتمال شرطی $p(x^{(i+d+1)} | < x^{(i)}, \dots, x^{(i+d)} >)$ خواهد بود. شکل (۴)، یک فایل HTML و سه‌توالی آموزشی اول تولید شده برای آن، توسط الگوریتم شرح داده شده در بالا را به همراه موقعیت قرارگیری آنها در شبکه نشان می‌دهد. توالی ورودی کلی در این مثال، همه

1- End-to-End

کاراکترهای فایل HTML و پارامترهای d و j به‌ترتیب برابر ۳ و ۱ قرار داده شده‌اند. در هر مرحله یک توالی برای آموزش به شبکه داده می‌شود.



شکل (۴): نحوه آموزش مدل‌های چندبه‌یک پیشنهادی

۳-۲-۲. تولید داده‌های آزمون 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 خورانده می‌شود. شبکه توزیع احتمالی کاراکتر d+1ام را به‌عنوان خروجی تولید می‌کند که شامل یک بردار 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+1 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+2 حاصل شود. این روند تا زمان تولید توکن پایانی که خاتمه‌یافتن یک توالی را پیش‌بینی می‌کند، ادامه خواهد داشت.

در روش یادگیری و فاز [۸] سه راهبرد حریصانه، نمونه‌برداری[1] و نمونه‌برداری- فاصله، برای تولید داده جدید از توزیع احتمالی پیش‌بینی شده توسط مدل، معرفی شده است در اینجا ما از راهبرد دوم یعنی نمونه‌برداری که روشی مرسوم برای تولید داده از یک توزیع آماری است، استفاده می‌کنیم؛ یعنی هربار از توزیع احتمالی پیش‌بینی شده به عنوان یک توزیع چندجمله‌ای، نمونه‌برداری می‌نماییم. برخلاف روش یادگیری و فاز، چون پیشوندهای ما متفاوت است (هر بار از مجموعه آزمون انتخاب می‌گردد)، می‌توانیم از راهبرد حریصانه نیز استفاده کنیم. در هر دو حالات تنوع داده‌های تولید شده نسبت به روش یادگیری و فاز بیشتر خواهد بود.

۳-۳. الگوریتم فاز عصبی

هنگامی که داده‌های آزمون را با استفاده از مدل‌های جدول (۱) و راهبردهای معرفی شده در بخش ۳-۲-۲ تولید می کنیم یک تنوع ذاتی در داده‌ها وجود خواهد داشت و در هر صورت داده‌های آزمون جدید محسوب می‌شوند. اما همانطور که در ابتدای بخش ۳ اشاره کردیم، اهداف یادگیری و آزمون فازی قالب فایل در تضاد با یکدیگر هستند. هدف الگوریتم یادگیری ایجاد داده‌های خوش‌شکل[2] است که بتواند از تجزیه‌گر اولیه عبور و مسیرهای عمیق را اجرا کند و هدف آزمون فازی بدشکل کردن ورودی است به‌نحوی که تجزیه‌گر و دیگر قسمت‌های کد اجرا شده را به حالت خرابی ببرد. در این بخش، ما الگوریتم فاز عصبی را با هدف ایجاد یک مصالحه بین دو هدف قبلی و تولید نهایی داده آزمون مناسب برای یک فازر قالب فایل، معرفی می‌کنیم.

گفتیم فاز کردن داده آزمون همزمان با تولید آن از روی مدل صورت می‌گیرد و مرحله مجزایی نیست. الگوریتم پیشنهادی ما با نام *فاز عصبی*، در شکل (۵) نشان داده شده است. این الگوریتم به‌عنوان ورودی مدل آموزش داده شده M، پیشوند شروع توالی P (حاوی توکن شروع)، نرخ فاز FR، توکن پایان ET و توکن دودویی BT را پذیرفته و داده آزمون TD را به‌عنوان خروجی باز می‌گردانند. در یک حلقه تکرار هر بار از مدل نمونه‌برداری شده و یک نشانه از داده آزمون تولید می‌شود. حلقه اصلی الگوریتم، 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. بدین‌ترتیب مطمئن می‌شویم که الگوریتم همواره خاتمه خواهد یافت.

در هر تکرار پس از نمونه‌برداری احتمال کاراکتر نمونه‌برداری شده با یک حد آستانه β مقایسه می‌شود. چنان‌چه این احتمال بزرگتر از β بود، بدین مفهوم است که کاراکتر فعلی در این موقعیت بارها تکرار شده و لذا بخشی از قالب فایل است. در این حالت کاراکتر پیش‌بینی‌شده توسط مدل با کم احتمال‌ترین کاراکتر، با هدف بدشکل‌سازی آن، جایگزین می‌شود. البته جایگزین شدن برای همه کاراکترهای قالب اتفاق نمی‌افتد و فقط در حالتی رخ می‌دهد که عدد p_fuzz که از یک توزیع تصادفی

1- Sampling

2- Well-Formed

یکنواخت انتخاب می‌شود از عدد FR که توسط آزمون‌گر تعیین گردیده است، کوچکتر باشد. به این ترتیب آزمون‌گر روی درصد و شدت کارکترهای فاز شده کنترل خواهد داشت. پیش از خروج از حلقه تکرار یا بازگشت به ابتدای آن کاراکتر تولید شده به داده آزمون اضافه می‌شود. همچنین کاراکتر اولیه صرف‌نظر از اینکه تغییر یافته باشد یا خیر، به پیشوند اضافه می‌شود و پیشوند را بروزرسانی می‌کند. چون نمی‌خواهیم مدل از فاز کردن کاراکترها آگاهی داشته باشد، آن را به پیشوند انتشار نمی‌دهیم؛ زیرا، هدف ما فاز تنها یک کاراکتر است و با انتشار فاز به پیشوند، ممکن است که مدل کاملاً گمراه شود. همچنین در آخرین دستور حلقه بررسی می‌شود که طول TD از یک عدد تصادفی MaxLen بیشتر نباشد. در غیر این صورت الگوریتم نتوانسته است تا بدین جای کار ET را پیش‌بینی کند که آن را به‌صورت دستی به TD اضافه کرده و از حلقه بیرون می‌آییم.

Algorithm NeuralFuzz

Input: LearnedModel M, SequencePrefix P, FuzzingRate FR, EndToken ET, BinaryToken BT

Output: TestData 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)      /* Sample c from
        the learned 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}  /* Replace c
            by c' (lowest likelihood) */
9      end
10     TD ← TD + c'
11     P ← P[1 :] + c   /* Do not propagate fuzzing
        to the next prefix */
12     if Len (TD) > MaxLen then
13         TD ← TD + ET
14         Break
15     end
16 end
17 if BT ∈ TD then
       /* fuzz binary data (mutation based) */
18     TD ← AddBinaryPart (TD)
19     TD ← MutateBinaryPart (TD)
20 end
21 Return TD
```

شکل (۵): الگوریتم فاز عصبی. تولید ترکیبی و بدشکل‌سازی همزمان داده‌های آزمون از روی مدل‌های زبانی عصبی.

پس از خروج از حلقه while، در صورتی که مدل حضور یک داده دودویی در TD را پیش‌بینی کرده باشد؛ یعنی نشانه خاص BT در توالی TD یافت شود (خط ۱۷)، ابتدا یک قسمت دودویی از قسمت‌های قبلاً ذخیره شده، جایگزین BT شده (خط ۱۸) و سپس این بخش با روش‌های جابه‌جایی تصادفی یا هر روش دلخواه دیگر، به‌صورت مجزا فاز می‌شود (خط ۱۹) و در نهایت داده آزمون نهایی توسط الگوریتم بازگردانیده می‌شود (خط ۲۱). همانطور که قبلاً هم اشاره شد، برای قالب‌های فایل ساده مثل jpeg (قالب رایج عکس) و به‌طور کلی داده‌های دودویی روش‌های جابه‌جایی مناسب‌تر به نظر می‌رسد [۸-۶].

الگوریتم فاز عصبی در نگاه اول مشابه با الگوریتم یادگیری و فاز [۸] است. اما چندین تفاوت و بهبود مهم دارد که عبارتند از:

- الگوریتم تضمین می‌کند که همواره پایان یابد. درعین‌حال قادر است تا داده‌های آزمونی با طول متغیر را تولید نماید.
- پرس‌وجوی مدل هربار با یک پیشوند متفاوت صورت می‌گیرد که منجربه تنوع بیشتر داده‌های آزمون تولیدی می‌گردد.
- عملیات فاز در پیش‌بینی‌های بعدی مدل و درهم ریختگی کنترل نشده داده آزمون، تأثیر گذار نیست.

ایده ما برای فاز در سطح کاراکتری با احتمال بیشتر از β، این است که در اغلب مواقع یک تغییر کوچک، مثلاً تغییر یک بایت در قسمتی که قالب فایل را بیان می‌کند سبب گمراه‌شدن تجزیه‌گر فایل می‌شود. حد آستانه MexLen که بیشینه طول داده آزمون تولیدی را در الگوریتم شکل (۵) مشخص می‌کند نیز با یک عدد صحیح تصادفی در بازه‌ی [a,b) تعیین می‌گردد.

۳-۴. پیاده‌سازی

برای پیش‌پردازش و تفکیک قسمت‌های متنی و دودویی فایل PDF اسکریپت‌هایی به زبان پایتون نوشته شد. برای پیاده‌سازی مدل‌های یادگیری ژرف از کتابخانه سطح بالای یادگیری ژرف Keras [۳۰] بر روی بستر Tensorflow [۳۱] استفاده کردیم. در آموزش مدل‌ها نیز تابع خطای آنتروپی متقاطع[1] و بهینه‌ساز Adam [۳۲] را به‌کار بردیم. هدف این مقاله ارائه یک روش تولید خودکار داده آزمون است. اما تولید داده آزمون به‌تنهایی کافی نیست و برای ارزیابی لازم است تا یک فازر قالب فایل با همه پیمانه‌های شکل (۱) در اختیار داشته باشیم. برای این منظور نیاز

1- Cross Entropy

به دو پیمانه تزریق‌کننده مورد آزمون و پایش SUT نیز داریم. اغلب نرم‌افزارهایی که یک فایل را به‌عنوان ورودی می‌پذیرند یک واسط خط فرمان نیز دارند که می‌توان از این طریق یک فایل را به آنها داد.

برای پایش هم ابزارهای مستقلی وجود دارد که می‌توان از آنها استفاده کرد. فازر پیشنهادی از ابزار Application Verifier [۳۳] مایکروسافت برای پایش SUT استفاده می‌کند. Application Verifier فایل اجرایی SUT را می‌گیرد و در هر بار اجرای آن، 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 غیرمجاز و غیره) شود، نوع خطا در این فایل ثبت و ضبط می‌گردد. همچنین برای اندازه‌گیری پوشش کد از ابزار VSPerfMon در محیط توسعه مجتمع ویژوال استادیو، استفاده کردیم که قادر به گزارش پوشش کد در سطح دستورات و بلوک‌های پایه است.

۴. ارزیابی روش پیشنهادی

ما روش پیشنهادی خود را روی قالب فایل PDF [۱۴] به‌عنوان یک قالب فایل با ساختار پیچیده و ترکیبی و نرم‌افزار متن‌باز و رایگان MuPDF [۱۵] مورد ارزیابی قرار دادیم. MuPDF مجموعه غنی از کتابخانه‌ها، APIها و ابزارهای کاربردی برای کار با فایل‌های PDF است. فایل PDF ساختار پیچیده‌ای دارد. مشخصات کامل این فایل در [۱۴] آمده است. بخش عمده فایل‌های PDF را اشیای داده‌ای تشکیل می‌دهند که ساختار آنها متنی است. لذا ما ساختار این اشیاء را مورد یادگیری و سپس تولید خودکار قرار داده‌ایم. در این بخش به تشریح معیارهای ارزیابی، چیدمان آزمایش‌ها، یافته‌های حاصل و نتیجه‌گیری از آنها می‌پردازیم.

۴-۱. معیارهای ارزیابی

در تولید داده آزمون به روش یادگیری ژرف تعداد زیادی پارامتر وجود دارد که می‌توان تأثیر آنها را در فرایند آزمون فازی سنجید. مهم‌ترین هدف در فرایند آزمون فازی، همان‌طور که قبلاً هم اشاره شد، یافتن خطا در SUT است. هدف مهم بعدی افزایش میزان پوشش در راستای نیل 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 آزمایش‌های خود در این بخش لحاظ کرده‌ایم:

- خطا و دقت مدل‌های یادگیری ژرف: شامل خطا و دقت هر مدل روی مجموعه‌های آموزش و ارزیابی، میزان و نحوه تغییر آنها با گذشت زمان که توسط تابع هدف به‌دست می‌آید.
- سرگشتگی: برای ارزیابی میزان خوب بودن مدل زبانی در پیش‌بینی نشانه بعدی توالی داده شده. معیار سرگشتگی از رابطه ۶ قابل محاسبه است.
- پوشش کد: 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.
- خطا و آسیب‌پذیری: خطاهای گزارش‌شده از SUT توسط ابزار پایش Application Verifier [۳۳].

۴-۲. مجموعه داده

برای یادگیری آماری ساختار اشیای فایل PDF پیکره بزرگی از فایل‌های PDF را جمع‌آوری کردیم. چنین پیکره‌ای از قبل وجود نداشت. بخشی از این پیکره شامل مجموعه داده‌های آزمون مرورگر وب Mozilla [۳۴] است. بخش دیگری از آن PDFهای استفاده شده در فازرهای مثل AFL [۷] است و بالأخره بخشی نیز از طریق وب جمع‌آوری شد به‌نحوی که تنوع خوبی از لحاظ اندازه و محتوا داشته باشند. از این فایل‌ها بیش از ۵۰۰ هزار شئ داده‌ای با طول‌های گوناگون استخراج شد که به عنوان مجموعه داده جهت یادگیری ساختار اشیای داده‌ای در نظر گرفته شده‌اند. بخش‌های دودویی این اشیاء که نماینده تصاویر در فایل PDF هستند، با توکن *stream* (نمایان‌گر توکن دودویی در الگوریتم شکل (۵))، جایگزین شدند.

۴-۳. انتخاب فایل‌های میزبان

در روش پیشنهادی اشیای PDF را استخراج، یادگیری و تولید می‌کنیم اما MuPDF یا هرPDF خوان دیگری فایل‌های کامل PDF را به‌عنوان ورودی می‌پذیرد. به همین دلیل لازم است تا اشیای تولید شده را به یک فایل تبدیل کنیم. برای این منظور اشیای جدید را با استفاده از سازوکار به‌روزرسانی افزایشی [۱۴] به یک فایل معتبر از پیش موجود، که آن را میزبان می‌نامیم، اضافه می‌کنیم. به‌روزرسانی افزایشی روشی استاندارد برای تغییر اشیای PDF است که در مستندات آن آمده است. در نتیجه یک فایل PDF جدید در اختیار خواهیم داشت که ساختار آن معتبر بوده و برخی اشیای آن بازنویسی شده‌اند. برای فایل‌های میزبان ما سه فایل از پیکره فایل‌های جمع‌آوری شده، به ترتیب با

بیشترین میزان پوشش کد (host1_max)، کمترین میزان پوشش کد (host2_min) و پوشش کد میانگین (host3_avg) را انتخاب کردیم. هدف از این کار بررسی تأثیر روش پیشنهادی بر بهبود میزان پوشش کد در فایل‌های با اختلاف پوشش کد زیاد و سپس انجام آزمون فازی براساس بهترین میزبان است.

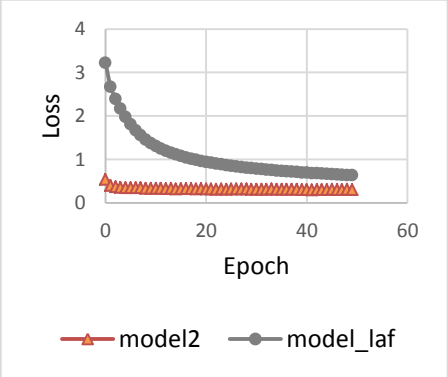
در تولید داده آزمون، اشیای جدید را به دو صورت به فایل‌های میزبان اضافه می‌کنیم: ۱- اضافه کردن و بروزرسانی یک شیء (SOU)[1]، ۲- اضافه کردن و بروزرسانی چندین شئ (MOU)[2]. هدف بررسی این فرضیه است که آیا تغییر بیشتر ساختار فایل پوشش کد بیشتری را نتیجه می‌دهد یا خیر؟

۴-۴. سرگشتگی، خطا و دقت مدل‌ها

جدول (۲) سرگشتگی، خطا و دقت مدل‌های چهارگانه پیشنهادی و مدل یادگیری و فاز را بعد از گذشت ۵۰ دوره، روی مجموعه‌های آموزش و ارزیابی نشان می‌دهد. خطا و دقت توسط Keras برای هر مدل حساب شده‌اند. در ردیف‌های سرگشتگی و خطا کمترین مقدار و در ردیف‌های دقت بیشترین مقدار هر ردیف پررنگ شده‌اند. همچنین شکل (۶) 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 آبرپارامترها با مدل laf دارد.

جدول (۲). ارزیابی مدل‌های یادگیری ژرف. اعداد بدون واحد هستند و بهترین عدد در هر سطر پررنگ شده است.

معیار / مدل	۱	۲	۳	۴	laf [۲]
سرگشتگی	۱/۴۴۰	۱/۳۹۱	**۱/۳۳۵**	۱/۳۵۰	۱/۸۶۳
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 (۶): نمودار میزان تغییر خطای مدل‌های ۲ و laf در دوره‌های ۱ تا ۵۰.

مشاهده‌ها:

- سرگشتگی و خطای همه مدل‌های پیشنهادی از مدل laf کمتر و دقت آنها از laf بیشتر است. یعنی مدل زبانی عصبی از مدل کدگذار-کدگشا [۹، ۱۰] بهتر عمل کرده است.
- بیشترین دقت در بین تمامی مدل‌ها مربوط به مدل ۴ است. مدل ۴، LSTM دوسویه است. این مدل هنگام آموزش علاوه‌بر کاراکترهای قبلی، کاراکترهای بعدی را نیز در نظر می‌گیرد. بنابراین، توانسته است بـه دقت بیشتر و در نتیجه سرگشتگی کمتری دست یابد.
- در شکل (۶)، منحنی خطای مدل ۲ در همه دوره‌ها، از منحنی خطای مدل laf پایین‌تر است که نشان می‌دهد آموزش مدل بهتر و سریع‌تر است.

۵-۴. پوشش کد مدل‌ها

برای مقایسه پوشش کد با روش یادگیری و فاز [۸] ابتدا مدل کدگذار-گشای پیاده‌سازی شده را ۵۰ دوره آموزش دادیم. سپس ۱۰۰۰ فایل PDF [۱۴] 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. همین کار را توسط ۴ مدل پیشنهادی خود نیز انجام دادیم. در نهایت دو حالت SOU و MOU را به‌صورت مجزا مورد ارزیابی و مقایسه قرار دادیم. نتایج در شکل (۷) نشان داده شده است.

1- Single Object Update
2- Multiple Object Update

مشاهده‌ها:

- 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 که نشان می‌دهد 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 توصیه می‌کنیم.
- در نهایت مشاهده می‌شود که مدل ۲ توانسته است به پوشش کد بیشتری دست یابد؛ لذا، آن را برای تولید داده آزمون در آزمون فازی MuPDF [۱۵] انتخاب می‌کنیم..

شکل (۷). نمودار پوشش کد مدل‌های مختلف در مقایسه با مدل laf برحسب میزبان‌ها در دو حالت SOU وMOU. host123 اجتماع پوشش کد سه میزبان را نشان می‌دهد.

۴-۶. آزمون فازی عصبی

در این آزمایش نرم‌افزار MuPDF [۱۵] را مورد آزمون فازی قرار دادیم. برای این منظور الگوریتم‌های فاز عصبی را که در بخش ۳-۲-۳ معرفی کرده بودیم، پیاده‌سازی و با استفاده از آن تعداد ۱۰،۰۰۰ فایل PDF مجزا را تولید کردیم. جدول (۳)، مشخصات مقادیر هریک از ورودی‌ها و ثابت‌های به‌کار برده شده در الگوریتم فاز عصبی را در آزمایش انجام شده نشان می‌دهد. ما همچنین آزمون فازی را با فازر FileFuzz [۱۳] که یک فازر قالب فایل تصادفی است و نیز با الگوریتم ارائه شده در [۸]، انجام داده و پوشش کد را اندازه‌گیری کردیم. درضمن در همه آزمایش‌ها، از host1_max به‌عنوان میزبان همچنین به‌عنوان دانه اولیه برای روش تصادفی استفاده شد. نتایج حاصل از پوشش کد روش‌های مختلف در جدول (۴) آمده است. در جدول (۵) نیز اختلاف پوشش کد الگوریتم پیشنهادی با هریک از الگوریتم‌های دیگر جدول (۴) و نیز با فازر قالب فایل AFL [۷] ذکر شده است.

مشاهده‌ها:

- الگوریتم فاز عصبی پوشش کد بهتری نسبت به الگوریتم ارائه شده در [۸] داشته است که نشان از عملکرد بهتر مدل‌های پیشنهادی در یادگیری ساختار فایل و نیز عملکرد بهتر الگوریتم پیشنهادی در فاز کردن داده‌های آزمون دارد.
- روش ترکیبی ارائه شده توسط الگوریتم فاز عصبی نسبت به فازر AFL [۷] هم بهتر است که نشان می‌دهد برای ساختارهای پیچیده استفاده از تولید مبتنی بر گرامر منجربه افزایش پوشش کد می‌شوند.

- به‌وضوح می‌توان برتری روش‌های هوشمند تولید داده آزمون را نسبت به روش‌های تصادفی مشاهده کرد. پوشش کد الگوریتم فاز عصبی بیش از ۳ برابر الگوریتم فاز تصادفی یعنی FileFuzz [۱۳] است.

جدول (۳): تنظیمات و مقادیر ورودی‌ها و ثابت‌های الگوریتم فاز عصبی

پارامتر	مقدار
Learnt model M	۲
Sequence prefix P	انتخاب تصادفی از مجموعه آزمون
Fuzzing rate FR	۰/۱
End token ET	endobj
Binary token BT	stream
(a, b)	(۴۵۰, ۵۵۰)
β	۰/۹۵

جدول (۴): پوشش کد حاصل از آزمون فازی MuPDF با الگوریتم‌های مختلف.

الگوریتم / معیار	پوشش بلوک پایه	درصد
NeuralFuzz	۲۲۸۵۳	۱۸/۴۳
SampleFuzz [۸]	۲۰۹۵۷	۱۷/۱۰
RandomFuzz (FileFuzz) [۱۳]	۷۵۳۶	۶/۱۷

جدول (۵): میزان بهبود پوشش کد آزمون فازی MuPDF (بر حسب درصد) با الگوریتم فاز عصبی در مقایسه با الگوریتم‌های دیگر.

الگوریتم موجود / الگوریتم پیشنهادی	NeuralFuzz
SampleFuzz [۸]	+۱/۳۳
AFL [۷]	+۶/۸۰
RandomFuzz (FileFuzz) [۱۳]	+۱۲/۲۶

۴-۷. خطا و آسیب‌پذیری

در بررسی گزارش‌های تولید شده توسط Application Verifier [۳۳]، پس از هر آزمون، خطای حافظه مشاهده نکردیم. باتوجه به‌اینکه ما نسخه نهایی نرم‌افزار MuPDF [۱۵] 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 چندین مورد استفاده از توابع ناامن را شناسایی کرد که Application Verifier آنها را درقالب هشدارهای امنیتی اعلام کرده است.

۵. نتیجه‌گیری

مولد داده آزمون، مهمترین پیمانه در فازرها است. ایجاد یک روش برای تولید خودکار داده‌های آزمون که در نرم‌افزارهای پیچیده به پوشش کد بالایی دست یابد، برای یافتن خطاها، امری ضروری است. در این مقاله روشی بر مبنای مدل‌های زبانی عصبی و یادگیری ژرف برای یادگیری خودکار ساختار فایل ارائه شد. نتایج ارزیابی‌ها بهبود پوشش کد و دقت مدل‌های پیشنهادی نسبت به روش‌هایی مثل الگوریتم‌های تکاملی و تصادفی را تأیید می‌کنند. برای اثبات درستی روش پیشنهادی ما قالب فایل PDF [۱۴] را که یک قالب فایل ترکیبی و بسیار پیچیده است مورد یادگیری و سپس از آن برای تولید فایل‌های PDF جدید و آزمون فازی نرم‌افزار شناخته شده MuPDF [۱۵] استفاده کردیم. افزون بر این نتیجه‌گیری کلی آزمایش‌های مختلف ما چندین واقعیت تجربی دیگر را مشخص می‌کند که مهم‌ترین آنها عبارتند از:

- روش‌های ترکیبی تولید داده آزمون مانند الگوریتم‌های فاز عصبی، که بخش‌های دودویی را نیز در فرایند آزمون فازی شرکت می‌دهند منجربه پوشش کد بیشتری می‌شوند.
- مدل‌های یادگیری ژرف ساده‌تر مانند مدل ۲، در میان مدل‌های پیشنهادی، مدل‌های پیچیده‌تر مثل مدل کدگذار-کدگشا [۹، ۱۰] و مدل LSTM دوسویه را در معیار پوشش کد شکست می‌دهند. زمینه نتیجه‌گیری مشابهی در همین زمینه، در [۲۶] گزارش شده است.
- فایل‌های PDFای که پوشش کد بیشتری دارند، هنگام تغییر از طریق بروزرسانی افزایشی نیز اختلاف پوشش کد بیشتری نسبت به دیگر فایل‌ها فراهم می‌کنند.
- تغییر هرچه بیشتر ساختار فایل ضمن پیروی از چارچوب گرامری آن، پوشش کد بیشتری ایجاد می‌کند، که امکان کشف خطاها و آسیب‌پذیری‌های احتمالی را افزایش می‌دهد.

چندین کار مرتبط برای آینده مدنظر هستند؛ از جمله استفاده از دیگر مدل‌های یادگیری ژرف مانند GAN [۳۵] که

نوع جدیدی از مدل‌های مولد هستند. همچنین استفاده از روش پیشنهادی در فازرهای دیگر مانند آزمون فازی پروتکل‌های شبکه جالب به‌نظر می‌رسد. فازر پیشنهادی در این مقاله دارای حلقه بازخورد نیست و در نتیجه از اطلاعات زمان اجرا برای تولید داده آزمون‌های بعدی استفاده نمی‌کند که این محدودیت نیز با افزودن سازوکار بازخورد و استفاده از دیگر روش‌های یادگیری ماشینی مثل یادگیری تقویتی، قابل حل است.

در نهایت روش پیشنهادی در این مقاله را می‌توان نخستین استفاده از مدل‌های زبانی در تولید خودکار داده‌های آزمون دانست. این مدل‌ها پیش از این در وظایف پردازش زبان طبیعی، مورد استفاده قرار گرفته و در مدل‌سازی پدیده‌های مبتنی بر توالی، بسیار موفق ظاهر شده‌اند. بنابراین، می‌توانند در همه وظایف این طیف استفاده شوند. به‌عنوان مثال، در حالت جعبه سفید که کد منبع برنامه دردسترس است، از این مدل می‌توان برای کشف توالی فراخوانی‌های سیستمی [۳۶] و توابع آسیب‌پذیر مورد استفاده در یک برنامه، نیز استفاده کرد. برای این منظور لازم است تا یک مجموعه داده از فراخوانی‌های سیستمی آسیب‌پذیر در اختیار داشته باشیم.

۶. مراجع

[1] B. P. Miller, L. Fredriksen, and B. So, “An empirical study of the reliability of unix utilities,” Commun. ACM, vol. 33, no. 12, pp. 32–44, 1990.

[2] B. P. Miller et al., “Fuzz revisited: a re-examination of the reliability of unix utilities and services,” 1995.

[3] J. E. Forrester and B. P. Miller, “An empirical study of the robustness of Windows NT applications using random testing,” Proc. 4th USENIX Wind. Syst. Symp., no. August, pp. 59–68, 2000.

[4] B. P. Miller, G. Cooksey, and F. Moore, “An empirical study of the robustness of MacOS applications using random testing,” Proc. 1st Int. Work. Random Testing, RT’06, vol. 2006, no. March 2017, pp. 46–54, 2006.

[5] G. Evron and N. Rathaus, Open source fuzzing tools. 2007.

[6] 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), 2017.

[7] M. Zalewsky, “American fuzzy lop,” 2013. [Online] Available: http://lcamtuf.coredump.cx/afl/. [Accessed: 11-Oct-2017].

[8] 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, 2017, pp. 50–59.

[9] 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. Curran Associates, Inc., 2014, pp. 3104–3112.

[10] K. Cho et al., “Learning phrase representations using RNN encoder-decoder for statistical machine translation,” in Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2014, pp. 1724–1734.

[11] D. Jurafsky and J. H. Martin, Speech and language processing (second edition). Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2009.

[12] T. Mikolov, M. Karafiát, L. Burget, J. Cernocký, and S. Khudanpur, “Recurrent neural network based language model,” in Proceedings of the 11th Annual Conference of the International Speech Communication Association, INTERSPEECH 2010, 2010, vol. 2, pp. 1045–1048.

[13] M. Sutton, A. Greene, and P. Amini, Fuzzing brute force vulnerability discovery, 1st ed. Addison-Wesley, 2007.

[14] A. S. Incorporated, “PDF reference, version 1.7,” no. November. Adobe, 2006.

[15] “MuPDF,” 2018. [Online]. Available: https://mupdf.com/. [Accessed: 27-Jul-2018].

[16] A. Takanen, J. D. Demott, and C. Miller, Fuzzing for Software Security Testing and Quality Assurance, 2nd ed. Norwood, MA, USA: Artech House, Inc., 2018.

[17] C. Chen, B. Cui, J. Ma, R. Wu, J. Guo, and W. Liu, “A systematic review of fuzzing techniques,” Comput. Secur., vol. 75, pp. 118–137, 2018.

[18] A. Kettunen, “Test harness for web browser fuzz testing,” University of Oulu, 2014.

[19] R. Mcnally, K. Yiu, and D. Grove, “Fuzzing : the state of the art,” DSTO Def. Sci. Technol. Organ., p. 55, 2012.

[20] P. Godefroid, A. Kiezun, and M. Y. Levin, “Grammar-based whitebox fuzzing,” ACM SIGPLAN Not., vol. 43, no. 6, p. 206, 2008.

[21] S. M. Yaghoubi, “Design and implementation fuzzer to determine web browser vulnerabilities,” Iran University of Science and Technology, School of Computer Engineering, 2013 (In Persian).

[22] S. Amini, “Design and implementation of test data generation method for software vulnerability detection,” Iran University of Science and Technology, School of Computer Engineering, 2016 (In Persian).

[23] S. Rahimi and H. R. Panji, “A novel approach for design and implementation a tool for finding vulnerabilities by fuzzing and black-box testing in {PDF} viewers,” in Proceedings of the First International Conference on New Research Achievements in Electrical and Computer Engineering, 2016 (In Persian).

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

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

[26] I. Goodfellow, Y. Bengio, and A. Courville, Deep learning. MIT Press, 2016.

[27] M. T. Luong, “Neural machine translation,” Stanford university, 2016.

[28] T. Mikolov, “Statistical language models based on neural networks,” Brno University of Technology, 2012.

[29] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” Neural Comput., vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[30] F. Chollet, “Keras,” 2015. [Online]. Available: https://keras.io/. [Accessed: 11-Oct-2017].

[31] “Tensorflow,” Google Brain. [Online]. Available: https://www.tensorflow.org/. [Accessed: 20-Sep-2018].

[32] D. P. Kingma and J. Ba, “Adam: a method for stochastic optimization,” in 3rd International Conference on Learning Representations, {ICLR} 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings, 2015.

[33] “Application verifier,” Microsoft. [Online]. Available: https://docs.microsoft.com /en-us/windows-hardware/drivers/debugger/application-verifier. [Accessed: 18-Jul-2018].

[34] “PDF test corpus from mozilla.” [Online]. Available: https://github.com/mozilla/pdf.js/tree/master/test/pdfs.

[35] I. Goodfellow et al., “Generative adversarial nets,” in Advances in Neural Information Processing Systems 27, 2014, pp. 2672–2680.

[36] S. Parsa, H. Seifi, and M. H. Alaeian, “Providing a new approach to discovering malware behavioral patterns based on the dependency graph between system calls,” J. Electron. Cyber Def., vol. 4, no. 3, pp. 47–60, 2016 (In Persian).

Journal of Electronical & Cyber Defence
Vol. 8, No. 1, 2020, Serial No. 29

Automatic Test Data Generation in File Format Fuzzers

M. Zakeri Nasrabadi1, S. Parsa*

*Iran University of Science and Technology

(Received: 12/03/2019, Accepted: 18/06/2019)

ABSTRACT

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 errors and vulnerabilities. Files are significant inputs to most real-world applications. Many of test data which are generated for fuzzing such programs are rejected by the parser because they are not in the acceptable format and this results in a low code coverage in the process of fuzz testing. Using the grammatical structure of input files to generate test data leads to increase code coverage. However, often, the grammar extraction is performed manually, which is a time consuming, costly and error-prone task. In this paper, a new method, based on deep neural language models (NLMs), is proposed for automatically learning the file structure and then generating and fuzzing test data. Our experiments demonstrate that the data produced by this method leads to an increase in the code coverage compared to previous test data generation methods. For MuPDF software, which accepts the PDF complex file format as an input, we have more than 1.30 to 12 improvement in percent code coverage than both the intelligence and random methods.

Keywords: Fuzz testing, Test Data, Code Coverage, Recurrent Neural Network, Language Model, Deep Learning.

*Corresponding Author Email: parsa@iust.ac.ir