Česká zemědělská univerzita v Praze

Technická fakulta

Laboratoř výpočetních aplikací

Diplomová práce

Technické a ekonomické aspekty implementace neuronových firewall

Vedoucí bakalářské práce: Ing. Zdeněk Votruba, PhD.
Vypracoval: Bc. Dominik Kolman

Česká zemědělská univerzita v Praze

Technická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Autor práce:	Dominik Kolman
Studijní program:	Obchod a podnikání s technikou
Vedoucí práce:	Ing. Zdeněk Votruba, Ph.D.
Garantující pracoviště:	Katedra technologických zařízení staveb
Jazyk práce:	Čeština
Název práce:	**Technické a ekonomické aspekty implementace neuronových firewall**
Název anglicky:	**Technical and economic aspects of neural firewall implementation**
Cíle práce:	Cílem práce je na základě literární rešerše definovat klíčové parametry pro návrh ověření funkčnosti neuronového firewall. Realizovat tento firewall a vhodným způsobem provézt testování výkonové a spolehlivostní charakteristiky firewallu. Na základě zjištěných dat definovat technické a ekonomické aspekty implementace tohoto typu firewall v porovnání s jinými typy.
Metodika:	1. Úvod 2. Cíl Práce 3. Metodika 4. Problematika firewallů 5. Možnosti neuronových sítí a implementace směrem k neuronovým firewall 6. Návrh parametrů neuronového firewall včetně realizace 7. Testování a ověření funkce 8. Ekonomické a technické zhodnocení 9. Celková analýza a závěr
Doporučený rozsah práce:	50 - 60 stánek včetně obrázků a grafů
Klíčová slova:	počítačová síť, bezpečnost, firewall, neuronová síť

Doporučené zdroje informací:

1. BISHOP C. M.: Neural Networks for Pattern Recognition. Oxford University Press, NewYork, 1995, 498 s., ISBN 0-38-731073-8
2. FOX, A., CHAPMAN Jr., D.:Zabezpečení sítí pomocí Cisco PIX Firewall, 2004, CPRESS, ISBN: 80-722-6963-1
3. Healy,R.,Odom,W.,Mehta, N.:Směrování a přepínání sítí, Cpress, 2009, ISBN: 978-80-251-2116-0
4. MAŘÍK V., ŠTĚPÁNKOVÁ O., LAŽANSKÝ J.: Umělá inteligence 4, Academia, Praha, 2003, 476 s., ISBN 80-200-1044-0
5. NOVÁK, M., et al.: Umělé neuronové sítě teorie a aplikace, C.H.Beck, 1998, Praha, 382 s., ISBN 80-7179-132-6.
6. Satrapa, P.:Internetový protokol IPv6, CZ.NIC, 2008, ISBN: 978-80-904248-0-7

Předběžný termín obhajoby: 2020/2021 LS - TF

Elektronicky schváleno: 3. 3. 2020
doc. Ing. Jan Malaťák, Ph.D.
Vedoucí katedry

Prohlášení

Prohlašuji, že svou diplomovou práci Technické a ekonomické aspekty implementace neuronových firewall jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 14. 5. 2021

Bc. Dominik Kolman

Poděkování

Rád bych touto cestou poděkoval vedoucímu práce Ing. Zdeňkovi Votrubovi, Ph.D., za výběr tématu, odborný dohled a možnost opět pracovat pod jeho vedením. Díky tomuto tématu jsem byl schopný opět rozšířit své schopnosti v oblasti bezpečnosti počítačových sítí. Dále bych také rád poděkoval své přítelkyni Markétě Vrbatové a Ing. Lumíru Kymrovi za podporu a trpělivost.

Technické a ekonomické aspekty implementace neuronových firewall

Abstrakt: Cílem této práce je technicko-ekonomická analýza implementace a provozu neuronových firewallů. Bylo navrženo vzorové řešení, které je realizováno jako příkladové řešení. V rámci této práce byly posouzeny dva typy v současnosti nejperspektivnějších architektur neuronových sítí, a to autoenkodér a vícevrstvý perceptron. Tyto neuronové sítě byly vyhodnoceny na třech typech datasetů, z toho jeden dataset byl syntetizován pro účely této práce. Výsledné řešení je potom technicky a ekonomicky rozebráno a posouzeno nejen vůči ostatním možnostem implementace, ale také ekonomickou náročností.

Klíčová slova: bezpečnost, počítačová síť, neuronové sítě, detekce anomálií, neuronový firewall, síťová bezpečnost

Technical and economic aspects of neural firewall implementation

Summary: The aim of this work is a technical and economic analysis of the implementation and operation of neural firewalls. A model solution was proposed, which is implemented as an example solution. In this work, two types of currently the most promising neural network architectures were assessed, namely Autoencoder and Multilayer perceptron. These neural networks were evaluated on three types of datasets, of which one dataset was synthesized for the purposes of this work. The resulting solution is then technically and economically analyzed and assessed not only in relation to other implementation options, but also in terms of economic complexity.

Keywords: security, computer network, neural networks, anomally detection, neural firewall, network security

Obsah

Seznam obrázků

Seznam tabulek

Seznam použitých zkratek

AE	Autoencoder
DBN	Deep Belief Network
GAN	Generative Adversarial Network
ACC	Accurancy
MLP	Multilayer perceptron
RBM	Restricted Boltzman Machines
PoC	Proof of Concept
NGFW	Next Generation Firewall
DBN	Deep Belief Network
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
CSV	Comma-separated values
PCA	Principal Component Analysis
IDS	Intrusion Detection Systems
TCP/IP	Transmission Control Protocol/Internet Protocol
HTTP	Hypertext Transfer Protocol
FTP	File Transfer Protocol
SMTP	Simple Mail Transfer Protocol
NAT	Network address translation
UDP	User Datagram Protocol

1 Úvod

I v divokém roce 2020 je stálou pravdou o kontinuálně sílící tendenci přesouvat větší část lidských fyzických činností do nehmatatelného prostoru počítačových sítích či osobních počítačů. Vytváření stále komplexnějších informačních struktur a datových toků je na jedné straně velmi přínosné pro efektivitu lidské práce. Na straně druhé tato zvětšující se komplexnost a objemy informačních struktur poskytují tím spíše ideální prostředí pro vedení mnoha moderních útoků a krytí útočníků.

Tyto skutečnosti dokazuje stále sílící objem a výnosnost kybernetického zločinu. Odhaduje se, že na přelomu roku 2019 dosáhl trh kybernetického zločinu obratů kolem 2 biliónů dolarů. Většina těchto případů se uskutečnila právě nelegálním vniknutím do systémů, které můžeme definovat jako neautorizované použití nebo vniknutí do počítačové sítě či systému.

Zde právě firewall představuje první ochranu vnějšího perimetru proti těmto hrozbám přicházejícím z vnějšího prostředí a v současné době také z vedlejší vnitřní sítě. Správně fungující firewall dokáže systematicky a efektivně tuto komunikaci filtrovat a zabránit široké škále škodlivé datové komunikace. Také lze pozorovat trend posledních let, kdy se firewally používají stále více v interních sítích pro minimalizaci škod. To neznamená, že by správci počítačových sítí vzdali ochranu vnějšího perimetru počítačových sítí, ale ukazuje to na stupňující se problematiku ochrany tohoto perimetru.

V dnešní době se již nelze spolehnout na jednorázově nastavená pravidla, základní technologie a chování firewallu. Hrozby v počítačových sítích se vyvíjejí a jsou vždy pokročilejší než systémy, kterými jsou počítačové sítě vybaveny pro svoji ochranu. Rozšiřování těchto pravidel ovšem také není cestou, jelikož bychom museli dojít dříve či později do stavu, kdy by byla pravidla spíše zatížením nežli ochranou.

Z tohoto a z mnohých dalších důvodů můžeme právem považovat monitorování počítačových sítí a jejich komunikace za čím dál tím více potřebné pro zachování fungování počítačových sítí. Také se jedná o jeden z největších úkolů současné počítačové bezpečnosti, kde se vedle zachování funkce zvyšuje důraz hlavně na ochranu dat obsažených v síti. Vedle vnějších hrozeb pro počítačové sítě existuje jako hrozba i samotný uživatel počítačové sítě. Na tyto úkoly správy bezpečnosti počítačových sítí se právě zaměřují neuronové firewally, jelikož jsou schopny řešit celou výše zmíněnou problematiku. Zároveň můžeme pozorovat dynamický vývoj tímto

směrem. Neuronové firewally se právě samy dokážou do určité míry se přizpůsobit nejen novým typům útoků a indexovat je, ale jsou i schopné se adaptovat na požadavky a potřeby jednotlivých uživatelů či datových uzlů, a podle toho o to efektivněji rozhodovat.

2 Cíle práce

Cílem práce je nastínit možnosti zabezpečení počítačových sítí pomocí různých druhů firewallů. Dále pak ověřit možná a vhodná řešení pro realizaci síťového neuronového firewallu pomocí užití různých typů neuronových sítí a jejich architektur.

V praktické části práce jsou porovnány vhodné výchozí architektury neuronových sítí vycházející z teoretické části. Vybrané modely jsou vyhodnoceny na základě předem stanovených parametrů, stavů v průběhu učení, jejich stability a úspěšnosti v detekování potencionálních útoků. Dalšími cíli práce jsou následná hodnocení v porovnání s reálnými aplikacemi jiných neuronových firewallů, a zachycení celé realizace z technologického a ekonomického hlediska.

3 Metodika

V úvodu práce bude nejprve provedena analýza všech nejčastějších typů firewallů, které budou navzájem porovnány s ohledem na práci se síťovou komunikací. V závislosti provedené analýzy firewallů budou vyhodnoceny možné umístění výsledného neuronového firewallu a určí se jeho hlavní části, které neuronové firewally odlišují od ostatních typů tzn. rule-based firewallů. V následující kapitole budou shrnuty teoretické základy neuronových sítí, určující základní parametry pro vyhodnocování neuronových sítí v jejich pozdějším testování. Následně budou představeny nejčastější architektury používané pro detekci anomálií v síťové komunikaci počítačových sítí, z nichž se vyberou dvě architektury pro pozdější zpracování, a to autoenkodér a vícevrstvý perceptron.

Tyto architektury byly předběžně zpracovány v prostředí „Playground.Tensorflow“ pro získání náhledu pozdějšího hlubšího zpracování a z tohoto předzpracování byly získány první poznatky o možnostech nasazení vybraných architektur a rozdílných výkonech při učení.

Z tohoto předběžného zpracování ovšem výsledky nebyly finální, jelikož zpracování mohlo být provedeno v prostředí „Playground.Tensorflow“ pouze z hlediska učení. Proto bylo rozhodnuto že do hlubšího zpracování šli obě architektury a byly obě dále podrobněji zkoumány.

Po získání základního přehledu se stanovil rámec zpracování praktické části, který se zaměřil hlavně na samotné jádro neuronových firewallů a jejich výkon, jakožto hlavní rozdílnou část oproti jiným typům firewallům.

Zde byly sledovány hlavně přesnosti procentuální, recall a průběh loss křivek neboli průběhu učení a jeho stabilita. V pozdějších zpracováních byla upřednostněna přesnost zpracování neoznačkovaných datasetů oproti průběhu a výkonnosti při učení. Následně se přešlo k testování samotné výsledné architektury Vícevrstvého perceptronu, která vykazovala lepší vlastnosti. Tato architektura byla rozpracována a podrobně popsána a výsledky testování byly interpretovány v použití s ostatními architekturami či použití v reálné implementaci.

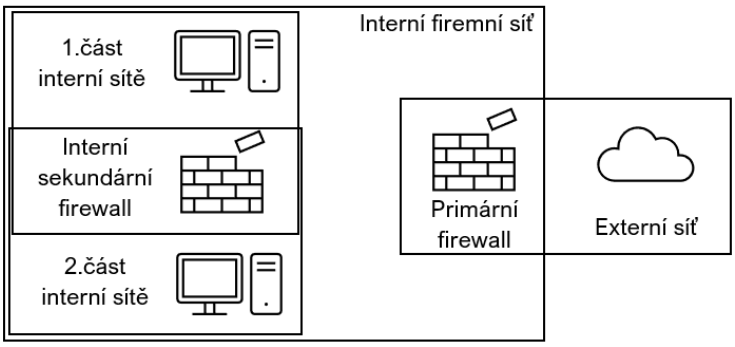
Závěrem práce rozpracovala technologické a ekonomické zhodnocení výsledné architektury, a to hlavně v návaznosti na ostatní moduly používané v kombinaci s neuronovou sítí v neuronových firewallech. Následně byl rozpracován ekonomický rámec a možnost ohodnocení vývoje a implementace neuronového firewallu a rozpracování SWOT analýzy

4 Problematika firewallů

Firewall může být softwarově i hardwarově řešený telekomunikační síťový prvek, instalovaný nejčastěji jako vstupní ochrana do interní sítě. Firewally jsou po právu považované jako jedny z nejdůležitějších bezpečnostních prvků sloužících k základní ochraně počítačové bezpečnosti. Firewall je zařízení sloužící k řízení a zabezpečení datového toku vstupující do interní sítě nebo její části. V současnosti se dá Firewall popsat jako kontrolní bod v počítačové síti, který na základě definovaných pravidel kontroluje a v některých typech firewallů i řídí datový tok a tím počítačové sítě rozděluje z bezpečnostního hlediska. Firewall v modernějším pojetí v kombinaci s některými moderními metodami kontroly síťového provozu patří do skupiny Intrusion Detection Systems (IDS). (1)

4.1 Technologie firewallů

Z historického hlediska se pravidla pro kontrolu a nakládání s datovým pakety řešila pomocí identifikace zdrojového a cílového portu, kontroly oprávněnosti přístupu. Starší firewally a bohužel podstatná část novějších firewallů poté řešili analýzu datového toku pomocí statistických metod, kde podle předem určených scénářů identifikovali nebezpečný datový tok. Jelikož tyto metody začínají být překonávány, posledních několik let dochází k prudkému rozvoji pružnějších metod, které nejsou vázány na statické datové sady, jak bylo dříve zvykem, ale opírají se o dynamické metody, konkrétně jde o využití neuronových sítí, které ke svému provozu například mohou využívat statické datové sady k naučení a posléze je aplikují v dynamické datové sadě, tzn. čili umí vytvářet nová pravidla a ta ihned implementovat. (1) (2)



Obr. 1 Možnosti umístění neuronového firewallu, Zdroj: Autor

Také ve starších případech použití, se firewall používal především jako ochrana mezi interní a externí počítačovou sítí. V současné době se ovšem využívá i jako mezi článek v interní síti, kde posléze dokáže omezit šíření spamu, či ochránit části sítě při případném průniku. Tato

umístění znázorňuje obrázek č.1. Instalace Firewallů z bezpečnostních důvodů probíhá obvykle stranou od zbytku počítačové sítě, tak aby příchozí požadavky šli právě přímo do firewallu. (3)

Obsah datové sady určující kritéria o vpuštění, či odmítnutí externích, popř. interních požadavků, záleží na typu a provedení firewallu. Obvykle se ale sada skládá z dvou základních částí. Část, kterou má firewall v sobě zapsanou od výrobce, a část, kterou firewallu nadefinuje správce počítačové sítě. Současné firewally na trhu pracují hlavně s rozpoznáváním virů a útoků pomocí signatur virů. Tyto signatury pocházejí z datové sady od výrobce. Bohužel, aby tento způsob fungoval, musí být neustále aktualizována datová sada se signaturami nových virů. Další častý zabudovaný způsob ochrany od výrobce bývá filtrování obsahu podle nežádoucích kategorií, jako např: pornografie, či obsah propagující rasismus a další obdobná závadná témata, která jsou známá svým častým výskytem podvodných a nebezpečných kódů. (3)

4.1.1 Paketové filtry

Paketový filtr je součástí každého hardwarového provedení firewallu a funguje na principu kontroly transportní a síťové vrstvy. V těchto vrstvách kontroluje informace v záhlaví každého TCP/IP paketu. Na základě oprávněnosti informací v headru TCP/IP paketu rozhoduje o jejich vpuštění nebo nevpuštění paketů do cílových IP adres. Tento typ firewallu je součástí routerů, které v sobě již standartně mají tzn. Forwarding Table Filters. (3)

Tyto filtry se stále používají pro svou vysokou rychlost zpracování probíhajícího spojení, proto se umisťují na místech s vysokým objemem dat, nebo naopak v místech, kde nepotřebujeme mít až tak sofistikované zabezpečení jako doplněk. Naproti tomu nevýhodou těchto firewallů je, že jsou poměrně jednoduché, a tudíž i úroveň kontroly je zejména u složitějších protokolů jako FTP protokol, či v dnešní době stále se rozrůstající video streaming, nedostačující. (4)

Forwarding Table Filters rozhodují o nepřijetí či přijetí packetu na základě:

- Zdrojové IP adresy
- Cílové IP adresy
- Typu použitého protokolu (ve zdroji, zprávě i cíli)

Ve filtrech jsou obvykle nastavena pravidla jako: omezení připojení komunikace zvenčí – pouze zevnitř ven, propouštění paketů pouze ve specifických portech, omezení přístupu na specifické IP adresy. Pokročilejší filtry pak zkoumají také stav všech připojení a porovnávají je mezi sebou,

čímž sledují příznaky útoků, ty mohou následně přerušit. Dalšími pravidly také mohou být nastavení, která chrání samotný firewall před přímým připojením, či naopak mohou mít nastavená pravidla povolení pouze určitých komunikací a zakázaný zbytek. (2)

4.1.2 Aplikační brány

Aplikační brány fungují na sedmé aplikační vrstvě referenčního síťového modelu OSI, někdy jsou taky zaměňované jako Proxy servery, ty jsou potom určitou pod součástí komplexnějších celků. Veškerá komunikace skrze aplikační bránu probíhá ve dvou fázích. V první fázi přijde na bránu požadavek o spojení, aplikační brána následně tento požadavek zpracuje. V druhé fázi aplikační brána vyšle za sebe požadavek o spojení na cílový server a působí tam jako klient. Datový tok, který aplikační brána potom od cílového serveru přijímá je automaticky přeposílán na koncový uzel, ze kterého vzešel požadavek na spojení. Tento princip potom funguje i opačně. To má následně automaticky schopnost fungovat jako NAT, ten je ale obvykle také součástí paketových filtrů. (2)

Pro účinně bezpečnou síťovou komunikaci potřebujeme nahlížet i do samotných přenášených informací. Zde se potom uplatňují právě Proxy servery, které pracují hlavně na aplikační úrovni retenčního modelu ISO/OSI. Proxy servery mají oprávnění prostřednictvím firewallu odmítnout datový tok protokolů na síťové úrovni a omezit provoz pouze na datový tok s protokoly vyšší úrovně například: HTTP, FTP a SMTP. Rozhodování je opět vázáno na datový set stanovený výrobcem a je upravováno na základě požadavků klientských stanic v interní síti na aplikační vrstvě. (2) (3)

Celý děj potom probíhá tak že Proxy server otevře informaci TCP/IP paketu na aplikační vrstvě a zkontroluje legitimnost informace uvnitř obsažené. Pokud server rozhodně o legitimitě, pouští paket dál, v opačném případě opět ukončí spojení. (3)

Nevýhodou těchto systémů je vysoká hardwarová náročnost a nižší výkonnost při zpracování datasetů. Každý protokol vyžaduje specifikaci skrze vlastní Proxy server. Další nevýhodou je, že Proxy servery následně nejsou technologicky víc napřed oproti paketovým filtrům, jen se specializují na jinou vrstvu referenčního modelu OSI. Přes tyto nedostatky se aplikační brány používají především pro známé protokoly s vysokou predikcí datového toku. (2) (1)

4.1.3 Stavové paketové filtry

Stavové paketové filtry neboli stavový filtr je typ firewallu, který pracuje na transportní vrstvě referenčního OSI modelu. V principu je schopný fungovat jako paketový filtr popisovaný výše, oproti tomuto filtru je však stavový paketový filtr schopen sledovat a udržovat informace o všech navázaných a probíhajících spojení TCP a UDP. Je schopen si ukládat informace o minulých spojeních a o rozhodovacím procesu již povolených spojeních. Potom následně k těmto rozhodnutím využívá informace pro zlepšení výkonu v budoucnu, jelikož nemusí celý datový tok procházet znovu rozhodovacím procesem. Zásadním posunem je u stavových paketových filtrů schopnost vytváření umělého stavu spojení, kdy lze konfigurovat otevření určitých portů pro naslouchání příchozí komunikace pro bez stavové protokoly jako UDP nebo ICMP. (1)

Největší výhoda tohoto typu řešení je právě v principu pouštění opakovaného datového spojení, což mu poskytuje vysokou rychlost. V obecné rovině poskytují stavové paketové filtry nižší úroveň zabezpečení než například aplikační brány. (1)

4.1.4 Stavové paketové filtry s kontrolou protokolů a IDS

Jedná se o moderní verze firewallů, které v sobě kombinují výše zmíněné systémy kontroly, ty potom rozvíjejí a přidávají také některé funkce antivirů. Tyto typy firewallů se také označují jako Next Generation Firewall, zkráceně – NGFW. Základem těchto typů firewallů potom často bývají neuronové sítě a jejich architektury, které provádějí samotnou detekci. Z pohledu běžných aplikačních bran, či stavových filtrů jsou stavové filtry s kontrolou protokolů a IDS schopné dynamicky kontrolovat otevření, zavření portů, a to jak pro jednoduchá, tak komplexní datová spojení. U novějších verzí se z IDS systémů potom přebírá možnost kontroly až na fyzické vrstvy a je tedy schopný odhalovat širší spektrum počítačových útoků. Kvůli své komplexnosti se také používají spíše v podobách HIDS – Host intrusion detection systems, které mají i oprávnění pracovat s daty i na koncovém uzlu. Pomocí databáze signatur, statistických modelů a neuronových sítí jsou potom schopny odhalovat vzorce chování, nebo hodnoty útoků i ve zdánlivě nesouvisejících datech, například skenování adres, portů, tunelování již povolených protokolů a dalších. (1)

V použití s neuronovými sítěmi jsou tyto typy firewallů schopné také detekovat útoky a odprosit je od běžné datové komunikace uživatele, jelikož jsou tyto typy sítě schopné se přizpůsobit síťové komunikaci daného nasazení. (1)

Uvedený typ firewallů představuje nejlepší možnou ochranu oproti všem výše zmíněným typům firewallů. Také konfigurace těchto firewallů není o moc náročnější oproti předchozím způsobům. Další výhodou je také poměrně vysoká rychlost kontroly ve srovnání s některými nastaveními aplikačních bran, ovšem nedosahuje takové rychlosti jako paketové filtry. (1) (5)

Jedinou nevýhodou těchto typů firewallů je jejich samotná komplexnost. Většina bezpečnostních řešení pro ochranu počítačových sítí je založená na integritě, a ta se snáze udržuje v jednodušších systémech, a také mohou mít vliv na propustnost počítačové sítě. (1)

4.2 Klasifikace firewallů

Klasifikace firewallů je obsáhlá a z historického hlediska se může zdát trochu zmatečná. Některé starší technologie se již nepoužívají, a i přesto zanechávají po sobě stopu ve formě nejednoznačného rozdělování firewallů. Také dochází ke kombinacím Intrusion detection systémů a firewallů právě v podobně neuronových firewallů, kdy neuronový firewall využívá princip funkce neuronové sítě k tomu, aby odlišil anomálie v provozu. (1)

4.2.1 Podle platformy

Firewally mohou mít softwarovou či hardwarovou platformu. Nejčastějším typem jsou softwarové firewally, ovšem v průmyslovém použití nezaostávají ani hardwarová provedení. Oproti softwarovým provedením mají hardwarová provedení rychlejší pracovní rychlosti, jsou robustnější, používají více modulů zabezpečení zároveň a využívají se pro ochranu více zařízení najednou. Oproti tomu softwarová řešení fungují více na kvantitě použití a jejich jednoduché šiřitelnosti, v rámci počítačových sítí se používají hlavně pro kontrolu koncových zařízení. Také pořizovací cena tvoří značnou část úspěchu právě softwarové platformy. (1)

4.2.2 Podle umístění v síti

Podle umístění řešíme firewally hlavně podle toho, jaký typ počítačových sítí propojuje. Firewally mohou být umístěny mezi interními a externími počítačovými sítěmi, což bývá nejčastější. Ovšem i umístění mezi interními částmi počítačových sítí je v poslední době značně oblíbené. Interní umístění firewallů je logické, při revizi úniků dat u velkých společností, které vynakládají mnohonásobně větší úsilí a ekonomických prostředků na zabezpečení svých počítačových sítí, je velmi pravděpodobné, že menší firmy budou prolomeny dříve, či později. Proto ochránění v rámci interní sítě představuje jednoduchý způsob, jak mírnit následky

průniku do počítačové sítě a v mnohých případech zajišťuje účinnou obranu při striktním nastavení a důsledném dodržování pravidel obsažených v datových sadách. (6)

Další pohled na umístění v počítačové síti může být podle místa jejich implementace. Toto rozdělení se může převzít z rozdělení systémů pro detekci anomálií neboli IPS a IDS systémů. Například zde se také projevuje prudký vývoj v oblasti neuronových sítí a jejich implementací, spojením Firewallů a IDS systémů. Tyto systémy mohou být rozděleny na dvě kategorie:

- **Síťově řešený systém (Network intrusion detection system – NIDS)**

  Takto řešený systém je obvykle implementován přímo ve strategicky významných síťových prvcích, jako jsou switche nebo routery. V současnosti je používán také samostatný termín přímo pro systémy, které se využívají pro detekci anomálií. Tyto systémy se potom nazývají Anomaly-based Network Intrusion Detection Systems (A-NIDS)

- **Klientsky řešený systém (Host intrusion detection system – HIDS)**

Klientsky řešený systém byl první typ IDS systémů, je nainstalovaný přímo v koncovém uzlu sítě. Od síťově řešeného systému je toto řešení odlišné především hloubkou, s jakou dokáže na daném koncovém uzlu pracovat s datovými toky. Dokáže již kontrolovat co přesně programy dělají na daném uzlu, jako například jestli textový editor nezasahuje do systémové databáze pro hesla a obdobně. (7)

4.2.3 Podle metody vytváření pravidel

Podle metodologie vytváření pravidel pro nakládání s pakety můžeme typy firewallů dělit na dva typy:

1. **Blacklist** – Firewall pouští veškerou komunikace a zakazuje průchod pouze paketům, specifikovaným v datových sadách, popřípadě definovaným administrátorem.

2. **Whitelist** – Firewall naopak pouští pouze specifikovanou komunikaci a ostatní komunikaci automaticky zamítá.

4.3 Metody detekce útoků

Detekcí útoků vedených skrz běžný síťový provoz existuje několik druhů s různými metodologiemi a způsobu identifikace. Většina těchto systémů ovšem funguje na principu

hledání určitých znaků v záznamu síťového provozu, a to jako systémy založené na detekci signatur, systémy založené na detekci anomálií, nebo jako systémy založené na statistické analýze síťového provozu. (8)

- **Detekce signatur**: Systém vychází ze známých průběhů útoků, tzn. retrospektivně. Může detekovat pouze útoky, které se již staly v minulosti a pomocí aktualizací jsou doplňovány do datové sady každého bezpečnostního softwaru, či zařízení. Již z principu funkce vyplývá, že tyto systémy jsou relativně náchylné na nové typy útoků. Zároveň jsou ale velmi robustním řešením, ve kterém bude docházet k malému množství falešných poplachů. (8)

- **Detekce anomálií**: Řeší datový tok přímo jak přichází a nemusí ho porovnávat s datovou sadou závislou pouze na vývojáři softwaru. Datovou sadu si aktualizuje a doplňuje na základě v průběhu času stráveného v provozu konkrétní počítačové sítě. V těchto případech může systém dostávat datovou sadu zčásti od vývojáře v podobě poskytnutí „normálních“ neboli referenčních dat (trénovací fáze) a následně si je systém doplňuje anebo některé druhy systémů jsou si schopny je vytvořit samy. (8)

- **Statistické metody:** V těchto metodách systémy vytvářejí statistické profily chování daného uzlu. Popisují ho pomocí statistických veličin, jako například frekvence výskytu informací, odchylky od normativních hodnot, střední hodnoty a další. Vytvoří se aktuální profil, který je neustále obnovován, a podle něj je odvozováno skóre anomality. Pokud skóre anomality překročí prahovou hodnotu, provede program zásah do síťové komunikace. (8) (9)

Všechny tyto systémy tedy musí procházet přípravnou fází a posléze fází testovací. Systémy jsou nejdříve naučeny pomocí datových sad vložených vývojářem. Jsou to datové sady sesbírané za běžného provozu nebo jejich kombinace. V trénovací fázi se do systému vkládá určitá znalost ohledně datového provozu a utváří se model normálního chování, a to buď s pomocí učitele, nebo si je systém schopen tento model utvářet sám. V prvním případě můžeme mluvit o standartních systémech detekce útoků na počítačovou síť, kde učitele můžeme chápat jako vývojáře, který pomocí statistických nebo hodnotových znaků vytváří datovou sadu. Prostřednictvím této sady systém následně určuje závadnost probíhající datové komunikace nebo komunikace nahrané ze záznamu. Další možností je, že si je systém schopen sám vytvořit

vlastní model normálního chování (datového toku) na základě neupravených provozních dat. (3)

V testovací fázi program testuje nám známou datovou sadu. Firewall porovnává naučený model datové sady „normálního“ chování počítačové komunikace, nalezené odchylky potom neuronová síť označuje jako anomálie. Zjednodušený zápis matematického vzorce, podle kterého program určuje, zdali se jedná o anomálii, je:

$$A = (M + s; D)$$

- Kde M je míra referenčního modelu datové sady normálního provozu
- S odchylka nastavená vývojářem pro nastavování citlivosti modelu
- D je míra určující odchylku vstupující datové sady od referenčního modelu (10)

4.3.1 Metoda detekce signatur

Metoda je jednou ze základních druhů metod detekcí síťových útoků a v dnešní době ustupujících na úkor metod detekcí anomálií. Tato metoda detekce předpokládá znalostní dataset, který popisuje průběh jednotlivých útoků. Z toho vyplývá, že tyto systémy jsou pouze retrospektivní a nejsou schopny detekovat neznámé a nové útoky, což je také jejich největší nevýhodou. Datasety mají v sobě zapsané charakteristické rysy a vlastnosti jednotlivých útoků, ty jsou potom pomocí diferenčních metod porovnávány s předkládaným datasetem z provozu. Tyto systémy jsou i v dnešní době z části využívány hlavně jako hrubý filtr před analýzou jiným systémem, vzhledem k jejich hlavní vlastnosti, a tou je procento falešných detekcí. (5) (11) (6)

Klasifikační proces obsahuje tyto kroky:

1. Rozdělení předkládaného datasetu
2. Předzpracování dat a určení hlavních vlastností
3. Naučení modelu z trénovacích dat.
4. Použití naučeného datasetu ke klasifikaci škodlivých dat.

Stavová detekce značek a protokolových anomálií

Jedním ze základních systémů pro výkonnou a jednoduchou detekci útoků je stavová detekce značek a protokolových anomálií. Metoda zvyšuje výkon ostatních systémů a snižuje procento chybných hlášení. Systém nahlíží a kontroluje stav spojení a nahlíží do síťového provozu, který

potom porovnává s datasetem. Kontroluje, jestli nenajde vzor řetězce uložených v datasetech v těle zprávy, headru nebo v profilu chování jednotlivých komunikačních spojení. (6)

Back-doors

Systémy detekce signatur umí detekovat a chránit proti útokům typu back-door. Nejznámějším příkladem je Trojský kůň, kdy útočníci mohou rozesílat například emaily s nebezpečnými přílohami. Systémy podle signatury chování tohoto typu útoku umí poslat např. email rovnou do virové truhly. (6)

4.3.2 Metoda detekce anomálií

Pro detekci anomálií v datasetu síťového provozu existuje několik řešení a rozdělení je stále nejednoznačné. Všechny tyto metody ale mají společnou fázi učení a fázi testování. Fáze učení potom může být ve třech variantách:

- **S učitelem** – Referenční datová sada s normálním provozem, či naopak s provozem obsahujícím útoky, pro vytvoření referenčních datasetů a natrénování neuronových sítí.
- **Bez učitele** – převážně využívané v metodách založených na strojovém učení, kdy je systém vložen do síťového provozu a vytvoří si vlastní dataset prahových hodnot
- **Zpětnovazebné** – někdy taky označované jako hybridní systémy, jsou metody založené na zpětné vazbě od učitele či jiného systému. Systém podle reakce od učitele detekuje nebo značkuje data, a podle reference od učitele zdokonaluje svoje detekční schopnosti. (6)

Metody s využitím strojového učení

Metoda strojového učení je založena na vytvoření systému s algoritmem schopným se učit neboli zlepšovat svoji efektivitu a výkonnost v proměnlivém okolním prostředím. Cíl strojového učení je obdobný jako u statistické metody, tedy zlepšování profilu uživatele na základě jeho chování. Tento systém zlepšuje svoji výkonnost na základě předchozích detekcí. S postupem vývoje počítačových útoků a jejich stupňující se komplexností jsou tyto systémy od přelomu tisíciletí na vzrůstu. V těchto systémech se využívají způsoby učení s učitelem i bez učitele. Při těchto metodách se využívají hlavně Neuronové sítě, dále pak Bayesovské sítě, algoritmus k-nejbližších sousedů a další. (8)

Metody založené na dolování dat – DATAMINIG

Metoda je založená na vytváření znalostních sad, kdy systém vyhledává vztahy mezi jednotlivými položkami ve velkých objemech dat. Tyto metody jsou určitým mezistupněm mezi metodami detekcí signatur a detekcí anomálií. Do těchto metod patří například systémy založené na klasifikaci, shlukové analýze anomálií a asociačních pravidlech. (5)

Klasifikační metody rozdělují datasety na normální a abnormální na základě pravidel a vzorců. Následně pomocí algoritmů pro určení pravidel rozdělování můžeme potom vyhledávat vztahy mezi veličinami, které by lidským okem byly jen těžko rozpoznatelné. Těmito algoritmy potom mohou být rozhodovací stromy, neuronové sítě, fuzzy logika a další. (5)

Shlukování a detekce abnormalit v metodách založených na dolování dat je technika strojového učení bez učitele. Slouží pro rozdělení množiny clusterů (jinak řečeno indexů síťové komunikace) do skupin podle jejich vlastností. Ve skupinách jsou clustery s nejméně jednou společnou vlastností. Jde o nalezení vzorů v neoznačkovaných datech s mnoha atributy (jednodušeji řečeno vlastnostmi) Výhodou této metody je samostatnost a schopnost fungovat bez předchozího předložení vzorců chování. Příkladem může být metoda LOF (local outliner factor), která přiřazuje každému datovému indexu LOF scóre. (7)

4.3.3 Statistické metody

Při této metodě systém generuje tzv. profily chování (statistický model) pro jednotlivé uživatele a tyto profily potom popisuje pomocí několika proměnných parametrů, jako jsou klouzavé průměry prosté i vážené, dále pak zaznamenáváním časových řad, či užitím ASTUTE (A Short-Timescale Uncorrelated-Traffic Equilibrium) metody, která je založena na multiplexování určitého datového toku, kde by se změny velikostí v krátkém časovém okamžiku měly téměř vyrušit. Tento typ metody potom může být parametrický a neparametrický. Dataset v profilu uživatele je neustále aktualizován a obnovován ze vzniklých detekcí anomálií, a to pro zvyšování jednoznačnosti uživatele. Tyto systémy by taky neměly obsahovat celé balíky dat, ale měly by pracovat pouze se statistickými údaji (střední hodnoty, frekvence, odchylky a další…). Jedny z vůbec prvních systémů založených na statistických modelech byly vytvořeny v Stanford Research Institute již v roce 1988. V současné době je tato metoda formy detekce převážně využívaná v hybridním použití s některou další metodou detekce anomálií. (12)

Detekce anomálií pak vychází z diference mezi jednotlivými profily. Systém kontroluje a porovnává jednotlivé proměnné mezi profily, a čím více je předkládaný dataset odlišný od profilu uživatele, tím je větší pravděpodobnost, že jde o anomálii, a systém spouští poplašnou zprávu. Jednotlivé proměnné v profilu uživatele odráží určité chování uživatele v síti. Jedná se například o frekvenci výskytů packetů jednotlivých protokolů nebo portů, čase připojení a další. Tyto paramenty předkládaného datasetu jsou porovnávány s datasetem profilu uživatele. Podle velikosti odchylek či rozdílu středních hodnot jsou porovnávány finální diference. (12)

Výhodou těchto systémů je, že nepotřebují znát konkrétní průběh útoků, ale soustřeďují se na vybočení celkového chování z profilu uživatele. Tato skutečnost dává systému relativní možnost reagovat i na neznámé útoky. Na druhé straně nevýhodou této metody detekce je relativně vysoká statistická nepřesnost. Také některé útoky nejdou zachytit a statisticky modelovat pomocí takto „hrubé“ kontroly chování na počítačové síti. V současné době je také známá možnost vytrénovat statisticky založené systémy na vyšší úroveň hranice tolerance. Pokud se hranice po detekci určí špatně, nedetekuje posléze nic podstatného. (12)

5 Neuronové sítě a implementace v neuronových firewallech

Struktura a funkce umělých neuronových sítí jsou inspirovány v lidském mozku konkrétně u neuronových sítí člověka. Počítačové neuronové sítě mají svůj počátek kolem 50.let 20. století, kdy začaly přicházet první zprávy o úspěšném počítačovém napodobení funkce biologického neuronu lidského mozku. Od té doby neuronové sítě pokročily v technologiích a dnes jsou běžnou součástí většiny pokročilých systémů. Neuronové sítě v počítačové bezpečnosti nacházejí své místo v oblasti detekce anomálií, tyto systémy jsou schopny detekovat jak jednu skupinu, tak i více skupin najednou. Pokud jsme schopni neuronové síti předložit dostatek dat, je schopna zařadit data do některé ze skupin, nebo hlásí anomálii. Neuronové sítě byly ze začátku používané převážně k detekci DDoS útoků, ale velmi rychle začalo docházet k jejich implementaci do jiných systémů a útoků v počítačových sítích. (13) (6)

5.1 Metrika neuronových sítí

TP – True positive: Stavy označení indexu z datasetu v případech, kdy došlo ke správnému označení pozitivního datového indexu (správné odhalení anomálie).

TN – True Negative: Stav označení indexu z datasetu v případech, kdy došlo ke správnému označení negativního datového indexu (správné označení normálního datového indexu).

FP – False positive: Stav označení indexu z datasetu v případech, kdy došlo k nesprávnému označení negativního datového indexu (normální datový index program označil jako anomálii).

FN – False negative: Stav označení indexu z datasetu v případech, kdy došlo k nesprávnému označení pozitivního datového indexu (anomálie byla označena jako normální datový index).

Zkombinováním těchto stavů lze získat matici záměn, která se využívá jako podklad pro určování výkonu tzv. Recommender systems (predikčních neuronových sítí). Matice řeší indexy datasetu ve dvou sloupcích, a to predikované hodnoty a hodnoty z testovacího datasetu. Snažíme se pak úpravami získat model s co nejnižšími hodnotami False positive a False negative, model s nulovými hodnotami potom dosahuje maximální výkonnosti. (8) (14)

5.1.1 Loss křivky

Jedná se o možnosti (způsob) výpočtu chyby mezi hodnotami z neuronové sítě a mezi očekávanými hodnotami z trénovacího datasetu. Často se používá v rámci metody nejmenších čtverců (MSE – Mean Squared Error) pro výpočet plochy pod křivkami, a určení jejich diference. Tato metoda se v praxi používá spíše ojediněle a většinou se sledují hodnoty a trendy křivek přímo v grafech. Nejlepším teoretickým výsledekem MSE je pak s nulovou hodnotou. (15)

5.1.2 Procentuální přesnost

Přesnost modelu nám určuje, jak moc se predikce vytvořené modelem mýlily od skutečnosti. Vypočítá se jako podíl správných predikcí na celkovém počtu provedených predikcí. I když je dále této metrice kladech důraz hlavně v oblasti před zpracováním, měřit výkonnost neuronové sítě pouze na základě této metriky by bylo nedostatečné, protože u této metriky není brána v potaz struktura dat, na kterých je přesnost určena. Pokud bude mít zkoumaný dataset přesnost 98 % a 100 indexů síťové komunikace, z toho 98 normálních a 2 anomálie, bude sice procentuální zastoupení správné, procenta se budou zdát dostačující, ale faktický výkon neuronové sítě může být nedostačující, protože model mohl označit všechny indexy jako normální a přesnost by stejně zůstala 98 %. (14)

5.1.3 Preciznost

Uvádí velikost správně označeného podílu pozitivně označených dat neuronovou sítí. Neboli kolik anomálií bylo správně označené neuronovou sítí. Preciznost je obdobný způsob jako procentuální přesnost, ale opět opomíjí část faktického výkonu neuronové sítě. (14)

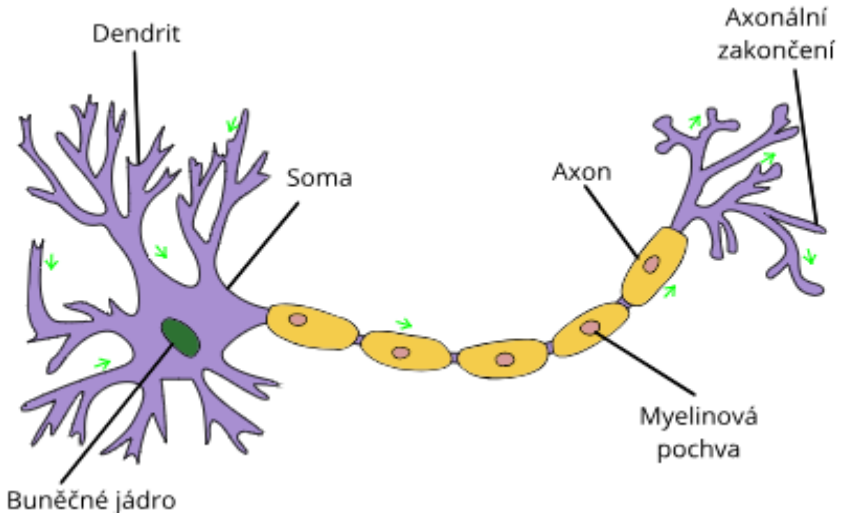
5.1.4 Recall

Jedná se o zásadní parametr právě pro kontrolu neuronových sítí implementovaných v predikčních systémech (Recommender systems). Vyznačuje, jaká část pozitivně označkovaných (detekovaných anomálií) byla správně identifikována. Zde již v rámci této práce nedochází k žádnému úniku faktických dat. Maximální teoretická hodnota je 1 neboli 100 %. (14)

5.2 Biologický neuron

Neuron je základní funkční a anatomickou buňkou nervové soustavy vedle glie (gliální buňky). Jsou to vysoce specializované buňky, které jsou schopny přijmout informaci ze speciálních signálů k němu přiváděných, převést, zpracovat a odpovídat na základě informací uložených v paměti. Zpracovávají informace jak z vnějšího, tak i z vnitřního prostředí a vytváří plán reakce organismu. Těchto neuronů se v lidském mozku vyskytuje přibližně 40-100 miliard. Samozřejmě existuje i několik druhů neuronů a každý z nich je propojený s 10-100 tisíci ostatních. (1)

Biologický neuron se skládá z několika částí: z těla (soma), to rozhoduje o vzniku a šíření vzruchu. Je ohraničeno plazmatickou membránou buněčného jádra, přijímá informace od ostatních neuronů a rozhoduje o vybuzení (excitaci) nebo utlumení (inhibici) neuronu. Dále se neuron sestává z dendritů, které fungují jako přijímače signálů (vzruchů) od ostatních neuronů a smyslových buněk, z axonu, který po vybuzení neuronu přenáší signál k axonálnímu zakončení. Myelinová pochva ta zajišťuje přenos a zrychlení posílaného signálu. Axonálního zakončení, představuje výstupní periferie. Schématické znázornění s průběhem signálu znázorňuje obrázek č. 2. (1)



Obr. 2 Znázornění funkce biologického neuronu, Zdroj: (16) upraveno autorem

5.3 Umělý neuron

Umělý neuron má obdobnou architekturu jako biologický. Je zde také používána funkce neuronu jako funkční jednotka neuronové počítačové sítě. Dále pak jsou vstupy, které chápeme jako vstupní hodnoty do neuronové sítě u biologického neuronu v tomto místě přijímáme vstupy z okolních neuronů nebo smyslových buněk, u uměle vytvořených neuronů se jedná o dataset. Výstupy, ty představují výsledné hodnoty, ty mohou být binární nebo kategorické a v neuronových sítích využívané přenosové váhy můžeme chápat jako axonální zakončení, popř. synapse. Somu (tělo buňky) lze chápat jako sumační prvek, tuto podobu srovnává tabulka č.1. Každý vstup, váha, výstup v umělém neuronu je reprezentován číselnou hodnotou. Váhy umělého neuronu reprezentují citlivost, s jakou neuronová síť propojuje jednotlivé neurony. Každý vstup je vynásoben hodnotou váhy spojení. Výsledná suma všech spojení v sumačním prvku (somě) je porovnávána s prahovou hodnotou nastavenou vedle samotné neuronové sítě. Pokud hodnota neuronu přesáhne prahovou hodnotu vyvolá to excitaci neuronu a následnému přenosu na výstup neuronu ve formě aktivační funkce.

Jedním z nejstarších neuronových sítí je perceptron, který byl vyvinut Frankem Rossenblattem v roce 1958. Je to zároveň také nejpoužívanější a nejvíce rozvinutá metoda vytváření neuronových sítí, samozřejmě v komplexnější podobě, jako například vícevrstvý perceptron.

Nejznámější a nejpoužívanější neuronové sítě jsou vícevrstvé neuronové sítě, ty se používají v 83 % všech případů. Tento model je zobecněním sítě perceptronů, což jsou jedny z prvně vytvořených neuronových sítí vůbec, a i přes některé nevýhody tohoto modelu je díky své rozšířenosti dobře technologicky popsaný a jeho architektura dobře prozkoumaná. Také se některé nedostatky vícevrstvých sítí V následujících kapitolách bude rozebrán umělý neuron z matematického hlediska, to má kopírovat výše popsanou funkci. (16)

Biologický neuron	**Umělý neuron**
Neuron – buňka	Neuron – Jednotka
Synapse	Synaptická váha
Dendrid	Vstup
Axon	Výstup
Soma	Sumační prvek

Tab. 1 Porovnání biologického a umělého neuronu, Zdroj: Autor

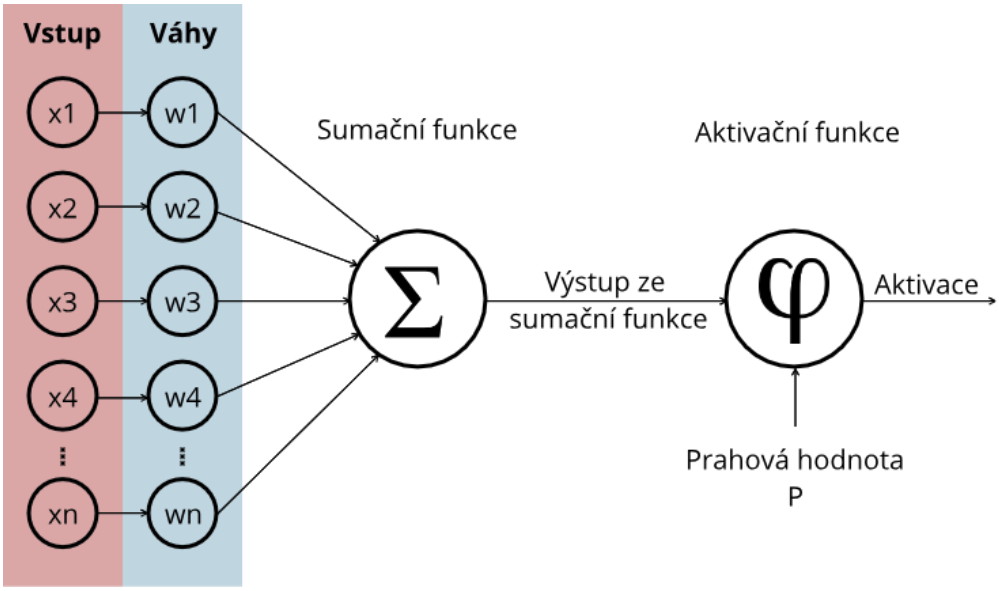
5.3.1 Matematický zápis neuronu

Obecný matematický model umělého neuronu a pochopení činnosti modelu neuronu z matematického hlediska je nezbytná pro pochopení celé neuronové sítě. Základní schéma funkce ukazuje obrázek č.3 (1)

- **X1-Xn** – vstupní hodnoty(vstup), číselná charakteristika v oboru reálných čísel
- **W1-Wn** – Synaptická váha, je číselná charakteristika vyjadřující citlivost spojení neuronu a vstupu. Jsou obvykle také charakterizována reálnými čísly a bývají rovnou navázány na vstupní hodnoty. Z počátku jsou tyto hodnoty náhodné a postupem tréninku se upravují.
- **Σ** – Sumační prvek představuje součet všech vstupů vynásobených příslušnými synaptickými vahami.
- **G** – je matematický zápis funkce a výstup ze sumačního prvku, ten potom přechází do aktivační funkce, a pokud tento součin potom překročí prahové hodnoty vybudí aktivační funkci.

$$G = \sum_{i=1}^{n} (Xi * Wi)$$

- **φ** – Aktivační funkce zpřesňuje a provádí diferenci mezi prahovými hodnotami a výstupem ze sumační funkce. Pokud je hodnota neuronu vyšší, než prahová hodnota provede aktivační funkce excitaci umělého neuronu, a pokud je nižší, provede jeho inhibici.
- **P** – prahová hodnota, která se určuje před zpracováním dat paralelně a vstupuje do aktivační funkce
- **Oj** – hodnota aktivace je charakterizována jako reálné číslo a je počítána jako diference mezi výstupem G a prahovou hodnotou P (1)



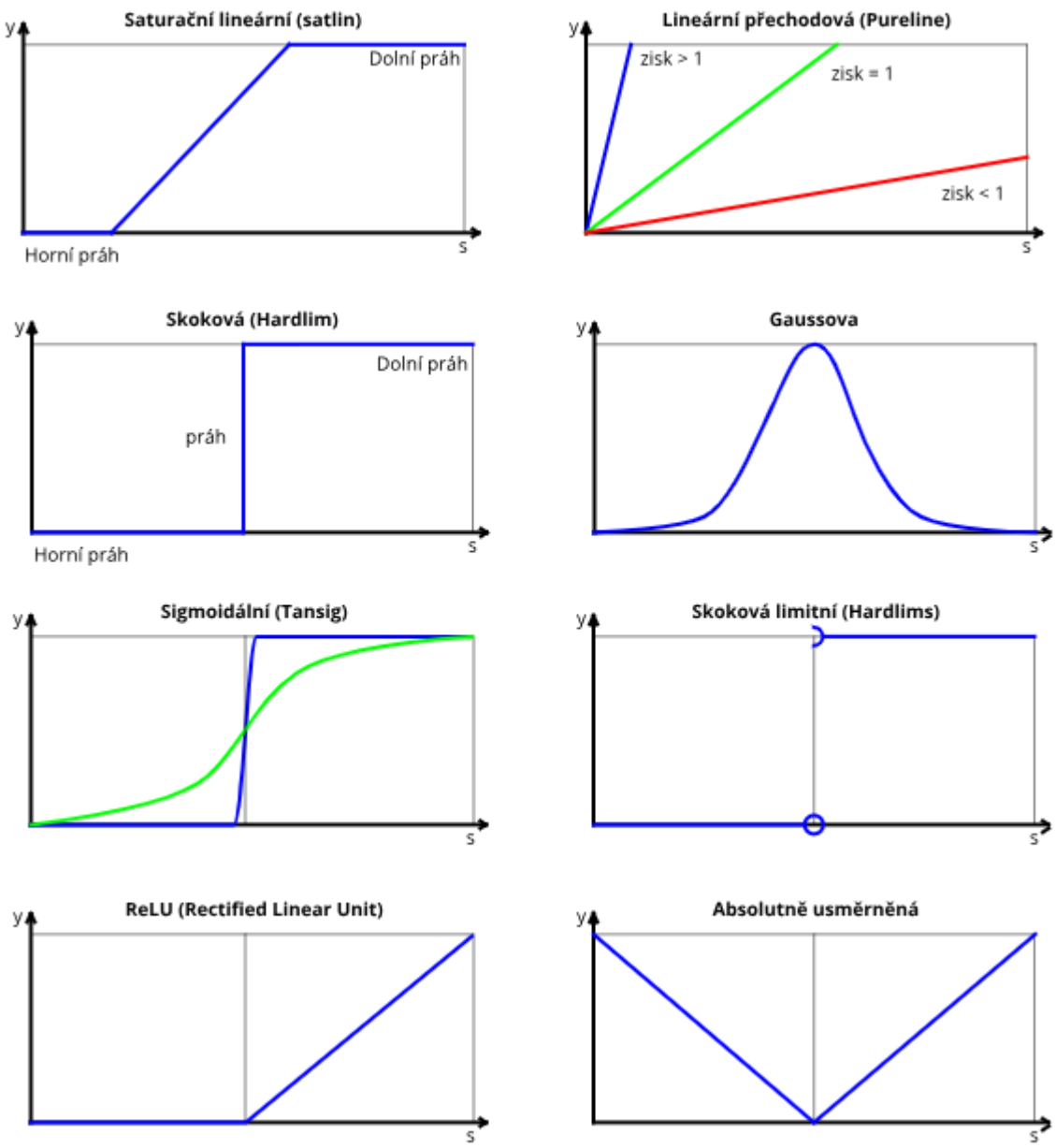
Obr. 3 Znázornění funkce umělého neuronu, Zdroj: Autor

5.3.2 Aktivační funkce neuronu

Aktivační funkce umělých neuronů slouží k převedení vnitřní hodnoty Oj na výstupní hodnotu, která je potom přiváděná buď na další neurony, nebo slouží jako výstup z celé neuronové sítě. Jedná se o přenosovou funkci. Průběhy těchto funkcí bývají různorodé a mění se dle potřeb jednotlivých systémů. Aktivační funkce se také využívá pro limitaci hodnot, například (-1;1), nebo častěji využívaný interval (0;1). Jsou tedy využívány aktivační funkce linerární, skokové a gaussovy. Méně používané, avšak ne zanedbatelné, jsou aktivační funkce hardlim, satlin, logsig, tansig, které jsou výsledkem lineárních kombinací vstupů a prahů rozhodování mezi dvěma stavy. Celkově nejpoužívanější aktivační funkce jsou: 1.) ReLU 2.) Tanh 3.) Sigmoid 4.) Linear. Průběhy těchto funkcí zobrazuje obrázek č.4.

Základní rozdělení aktivačních funkcí:

- Podle spojitosti:
    - Spojité
    - Diskrétní
- Podle křivky průběhu:
    - Lineární
    - Nelineární

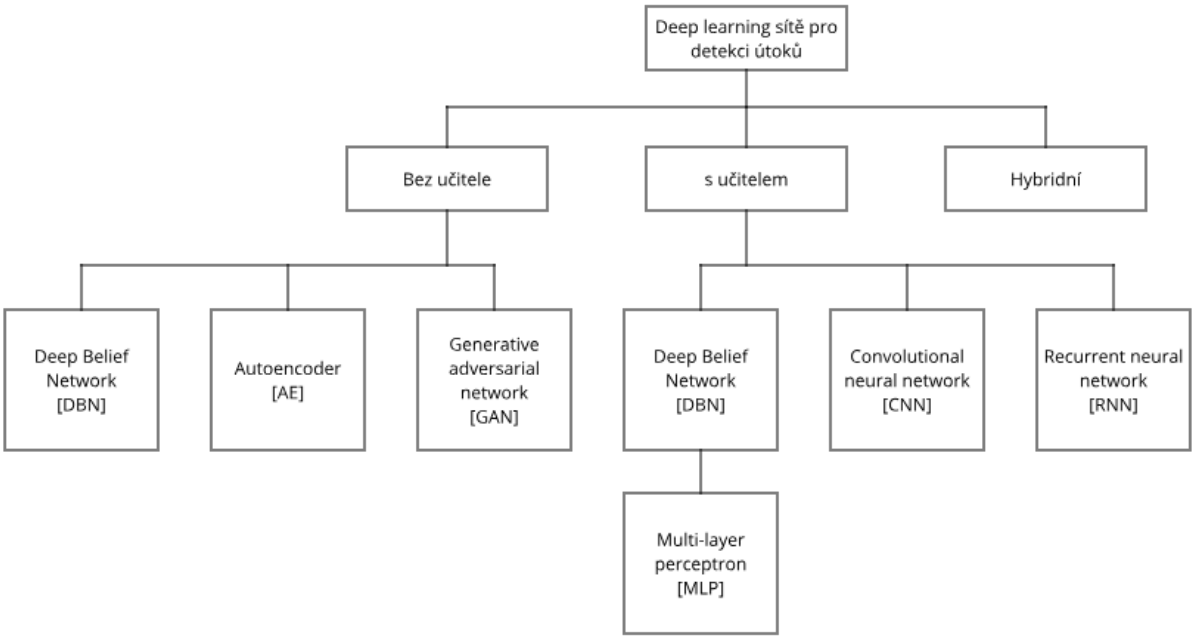


Obr. 4 Grafy aktivačních funkcí, Zdroj: Autor

Při výběru a implementaci aktivačních funkcí je potřebné zvolit takovou funkci která je schopná skloubit v kompromisu několik požadavků. Především požadovaný výstup, který chceme po neuronové síti jako celku (rychlost, spínání mezi neurony, atd…), složitost implementace, zde se řeší, zdali je možné funkci zakomponovat přímo do programu nebo se musí vypočítávat v externí třídě. V občasných případech také řešíme použití náhledových tabulek. Důležité je také při výběru zohlednit implementaci derivace pro případné další učení obvodu. Nejpoužívanější funkce ReLU při záporných hodnotách nereaguje a nechává umělý neuron v inhibici při kladných hodnotách vstupu řeší sílu synaptické váhy v násobku se vstupem. (5) (1)

5.4 Klasifikace neuronových sítí

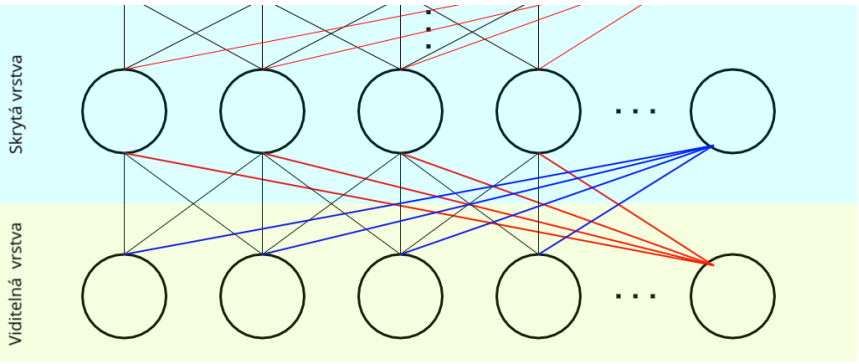
Základní klasifikací, jak již bylo výše řečeno, je rozdělení na sítě s učitelem, bez učitele nebo hybrid, který představuje všechny upravené verze a využívá zbylých dvou dohromady, Obrázek č. 5 znázorňuje jejich základní rozdělení při využívání v neuronových sítích pro detekci útoků vedených přes počítačové sítě. (5) (6) (8)



Obr. 5 Rozdělení neuronových sítí používaných v počítačové bezpečnosti, Zdroj: Autor

5.4.1 Deep belief network – DBN

Deep belief networks, dále jen DBN architektura, pracuje obdobně jako vícevrstvý perceptron. S tím rozdílem, že poslední dvě vrstvy jsou nejčastěji tvořeny pomocí RBM neboli Restricted Boltzman Machines. Jedná se o neuronové sítě stochastického typu, které jsou schopny pracovat bez učitele. Jsou specifické tím, že se skládají z několika vrstev, ale neurony nejsou spojené v rámci vrstev mezi sebou, tak jak znázorňuje obrázek č.6. Na vstupu i výstupu používají tyto architektury binární, směrodatné vektory. Vyjma zbylých dvou vrstev je zbytek těchto sítí tvořen Bayesovou sítí, kde jednotlivé záznamy parametrů pravděpodobnosti jedné vrstvy závisí na parametrech pravděpodobnosti předešlé vrstvy. Trénování těchto sítí je potom náročnější než u jednoduššího vícevrstvého perceptronu, to hlavně díky nastavování meta parametrů, jako jsou počty skrytých vrstev, inicializační hodnoty vah a struktury datasetu pro jednu epochu trénování. (4) (5)



Obr. 6 Znázornění architektury DBN neuronových sítí, Zdroj: Autor

Specifické pro tuto architekturu je také závislost výkonu modelu na vstupní inicializaci modelu a nastavení vzorkování. Nejčastěji se používají při klasifikaci a rekonstrukci poškozených dat. Úvodní zpracování probíhá ve většině DBN architekturách obdobně v několika krocích:

1. Nejprve se v první vrstvě RBN provede aproximace vstupních dat.

2. Druhá vrstva RBM potom využívá výstupy první vrstvy a parametry nižší se přitom nemění. Z první RBM vrstvy se převedou transponované váhy, a výstup z první vrstvy se zpracuje v druhé vrstvě.

3. Dále se pak postup přizpůsobuje konkrétní implementaci. (9)

V poslední vrstvě se obdobně jako u vícevrstvých perceptronů využívá softmax funkce pro klasifikaci datasetu. Takto vytvořené váhy se potom převádí do klasické architektury, např.: vícevrstvého perceptronu. V tomto bodě se potom může provádět i zpětná propagace. Při rekonstrukcích dat je potřeba zavést trénovací datasety s požadovanou finální strukturou a zpracovat je v prvních dvou vrstvách RBN. Na základě zpracování bude architektura na výstupní vrstvě schopná vygenerovat ztracená data za použití Gibbsova vzorkování. Tato fáze je potom hlavně časově náročná a obtížně aplikovatelná třeba v systémech reálného času. (9)

5.4.2 Autoencoders – AE

Autoenkodéry jsou jedny z jednodušších architektur obdobné vícevrstvým perceptronům. Tato architektura se snaží natrénovat své váhy tak, aby výstup odpovídal co nejlépe vstupu. Autoenkodéry mají dvě části, první částí je enkodér, který zakóduje vstupní data podle kódové abecedy do určité šifry. V druhé části je potom dekodér, ten si načítá z enkodéru šifrovací abecedu, aby mohl data rozšifrovat. Kódovou abecedou jsou v případě autoenkodéru myšleny parametry neuronové sítě. Pokud enkodér pracuje s určitými váhami, tyto váhy budou muset

být logicky přivedeny také do druhé části, tedy do dekodéru, aby mohl zašifrovaná data rozklíčovat. Tato architektura se potom vyznačuje tím, že:

1. Je ztrátová, neboť zde dochází v podstatě ke kompresi a dekompresi datasetu.
2. Má pevně daný vstupní formát dat.
3. Parametry se určují při zpracování datasetu. (11)

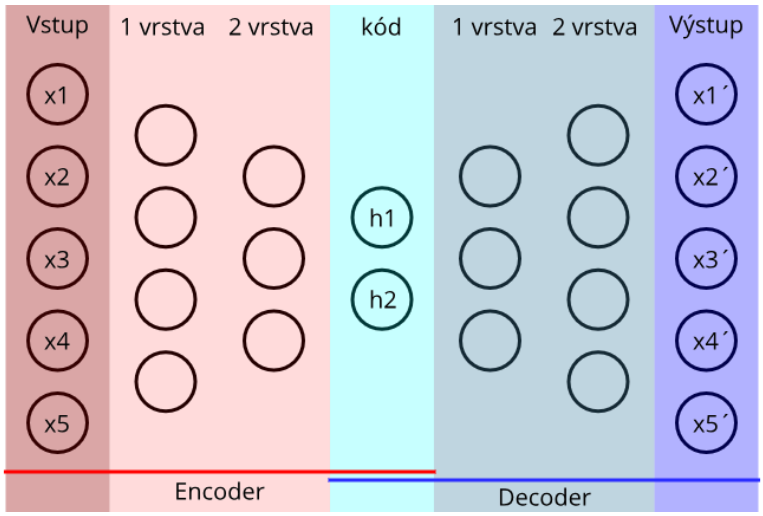
Architektura autoenkodéru je od středu symetrická a vstup i výstup musí mít stejný počet neuronů, viz. obrázek č.7. Výkonnost šifrování se potom počítá obdobně jako u MSE – Mean Squared Error, ze které se potom vypočítá plošný rozdíl. (10) (8)

S autoenkodéry lze pracovat dvěma způsoby:

- Skrytá vrstva má méně neuronů nežli vstupní vrstva, a využijí se lineární operace, ty potom přes společné vlastnosti vykazují vztahy ve skrytých vrstvách. V zásadě autoenkodér provádí analýzu hlavních komponent (PCA).

- Skrytá vrstva má větší počet neuronů než viditelná vrstva, a používá se ke zpracování nelineární operace k získání rekonstrukce vstupu. V podstatě jde o opačný způsob práce architektury. (10) (8)

Model tedy vytvoří šifrovací abecedu, která bude pouze šifrovat zašifrovaná vstupní data. V některých případech se ale potom může vstup pouze kopírovat na výstup, a tím dojde k nefunkčnosti neuronové sítě. Tomu se zabraňuje zašuměním vstupních dat. (8)

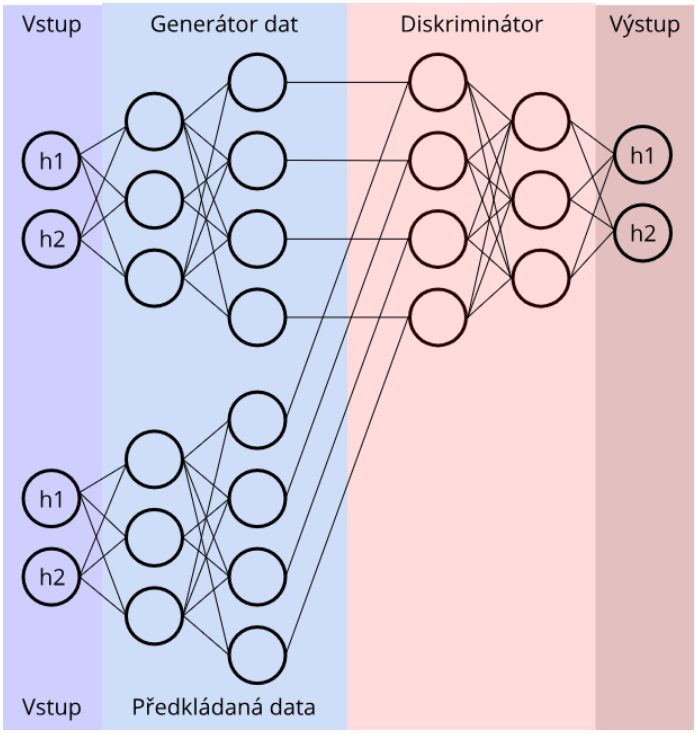


Obr. 7 Znázornění architektury AE, Zdroj: Autor

Autoenkodéry jsou po vícevrstvých perceptronech jedny z nejvíce používaných, a to především pro zpracování velkých datasetů u kterých byla nebo bude v nějaké podobně provedena komprese. Tyto typy neuronových sítí jsou také často používané a kladně hodnocené právě v tzv. Recommender systems. (9)

5.4.3 Generative adversarial network – GAN

Neuronové sítě s architekturou GAN využívají dvě nervové sítě stojící proti sobě, aby vytvářely nová data tak podobná reálným datům, jak jen to je možné. V podstatě tyto sítě využíváme k tomu, aby nasimulovaly data vyskytující se v našem síťovém provozu, a tak vytváříme nový dataset ze syntetického datasetu. Oba datasety potom porovnáváme mezi sebou a učíme neuronovou síť tyto datasety odlišovat. Největší výhodou GAN sítí je, že téměř nepotřebují syntetická, či jiná data, umí si plně poradit bez učitele, protože si architektura



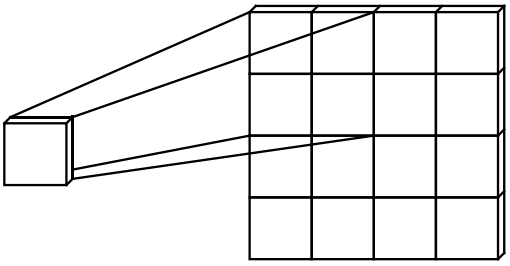
Obr. 8 Znázornění architektury GAN neuronových sítí, Zdroj: Autor

trénovací data umí dovytvořit sáma. Na druhou stranou jsou tyto neuronové sítě náročné na učící fázi. Díky zdlouhavé učící fázi se tedy nehodí na větší datasety nebo na real-time implementace. Obě nervové sítě Generátor i Diskriminátor proti sobě soupeří, a z jejich soupeření se učí diskriminátor následně rozpoznávat simulovaný provoz od reálného. (17) (9)

5.4.4 Convolutional neural network – CNN

Konvoluční nervové sítě používají princip vícevrstvého perceptronu, ale implementují navíc konvoluční vrstvy. Tyto nervové sítě se nejvíce používají pro práci s pozičně závislými daty (videa a fotky), kde obraz tvoří mřížku a jednotlivé buňky mají v sobě obsáhlou informaci, jak ukazuje obrázek č.9. Každá buňka a její informace vypovídá o intenzitě svícení daného pixelu. Pokud tuto skutečnost potom využijeme při zpracování jako datasetu, může nervová síť hledat určité trendy (křivky) a další charakteristiky obrázků. Konvoluční sítě tedy ke zpracování těchto datasetů využívají tzv. Filtry. (7)

Filtry jsou definované velikostí mřížky z M x N, a ty jsou potom násobeny několikrát, a v podstatě dochází ke kompresi dat zapsaných v mřížce. Filtr takto zpracuje celou mřížku a vytvoří nový „kompresovaný“ dataset. Tyto hodnotové charakteristiky potom dokážou lépe reprezentovat právě určité tvary a křivky.



Obr. 9 Funkce CNN neuronových sítí, Zdroj: Autor

Konvoluční vrstva

Tato vrstva konvoluje zpracovaný dataset z M x N, a počítá skalární součin mezi filtrem a vstupem. Touto úpravou vzniká tzv. 2D aktivační mapa filtru. Již po průchodu první skrytou vrstvou je program schopen začít rozpoznávat základní grafické vlastnosti, například rohy, barvy a další. (7)

Pooling vrstva

Obdobně jako konvoluční vrstva rovněž provádí určitou kompresi dat po výstupu z jedné skryté vrstvy tak, aby data byla abstraktnější, a zároveň nebyla tak obsáhlá. Existuje několik druhů poolingu, jako například: Max pooling, kdy vrstva započítává jen maximální hodnoty z vrstvy předchozí, a ty pak odesílá do další vrstvy nebo Average pooling, kdy vrstva bere jenom průměry do dalších vrstev ke zpracování. (7)

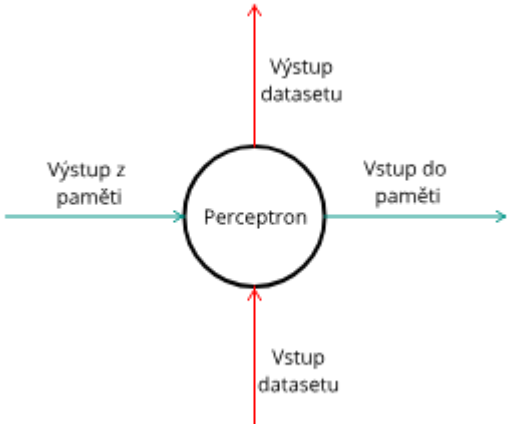
Ztrátová vrstva

Tato vrstva provádí jistou abstraktní kontrolu výstupu a určuje, jako moc se po zpracování liší výstup konvoluční nervové sítě od očekávání.

5.4.5 Recurrent neural network – RNN

Zatím zmíněné architektury neuronových počítačových sítí řešily problémy buď se syntetickými datasety, vytvořenými na míru k učení, nebo si umělé datasety vytvořily samy na základě předkládaných anebo vytvářených hodnot. Využití datasetu pro předpoklad pokračování nebo předpovědi budoucího vývoje žádné z předchozích neřešily. K tomuto účelu se právě využívají RNN sítě, neboť tyto sítě právě vynikají v analýze předkládaného datasetu, a na základě analýzy dokážou rovnou předpovědět pokračování trénovacího datasetu s dočasnými indexy. RNN ukládá dočasné indexy předchozích stavů do stavových matric. Tyto matrice fungují jako normální vrstva ve vícevrstvém perceptronu, a používájí stavové matrice jako vstup pro nový výstup.

Long Short Term Memory (LSTM) architektura RNN sítě tuto techniku dále rozšiřuje a přidává navíc další vrstvu stavových matric k ukládání druhého dočasného indexu. Tento typ architektury potom ukládá dva dočasné indexy, jeden dlouhodobý a druhý krátkodobý. Pokud se stav od předchozího stavu nezmění, stane se z něho dlouhodobý index a bude vstupovat do dalšího zpracování s větší synaptickou váhou. (18)

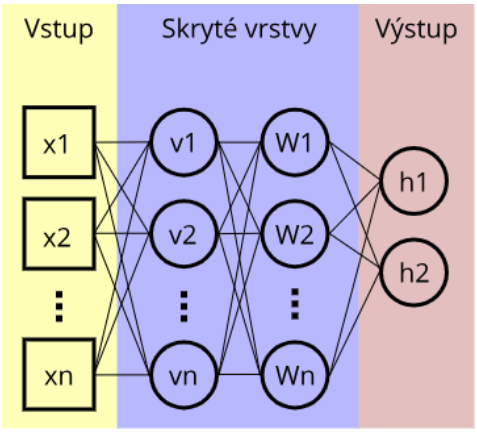


Obr. 10 funkce RNN neuronových sítí, Zdroj: Autor

Tento princip funkce již předpovídá využívání této architektury, nejvíce je implementován v predikčních modelech na trzích s cennými papíry a komoditami, v časově závislé predikci, aj.

5.4.6 Multilayer perceptron – MLP

Vícevrstvý perceptron si své prvenství získal mezi neuronovými sítěmi množstvím implementací (přibližně 80 % všech implementací) díky své jednoduchosti natrénování. Vzhledem k tomu, že se jedná o jednoduchý perceptron (jedna z prvních neuronových sítí vůbec), který je znám už od 50.tých let minulého století, a že existuje již řada ověřených a



Obr. 11 Znázornění architektury MLP Neuronových sítí, Zdroj: Autor

popsaných implementací, zůstává i přes své chyby a díky své vysoké referenčnosti je stále velmi populární. To platí i v případech internetové bezpečnosti, kde se jeho architektura využívá k detekci síťových anomálií. Architektura vícevrstvého perceptronu viz. Obrázek č.11, se obvykle skládá ze vstupní vrstvy, kde jsou hodnoty přivedeny, určitého počtu skrytých vrstev, které provádí vlastní analýzu, a výstupní vrstvy neuronů, z které se data načítají. (19) (17)

Vstupní vrstva

Vstupní vrstva se využívá, aby generalizovala data, která vstupují přímo k neuronům. V některých případech implementace jsou vstupní data nenumerická a vstupní vrstva je zapotřebí, aby standardizovala vstup datasetu k neuronům. Tento proces se obvykle nazývá „data processing“ a je obvykle fází, která zabírá nejvíce času v celé implementaci. (19) (17)

Skryté vrstvy

Skryté vrstvy se skládají převážně z neuronů a jsou hlavním jádrem úprav a manipulace s datasety. Ty jsou prohnány skrz skryté vrstvy a jsou upravovány pomocí synaptických vah. Skryté vrstvy se nazývají proto, že vývojáři nepracují přímo s těmito vrstvami, ale pouze je nechávají vyvolávat. Naopak upravují hlavně vstupy a výstupy do těchto vrstev. (19) (17)

Výstupní vrstvy

Výstupní vrstva je finální výstup celé neuronové sítě, obvykle reprezentuje různé charakteristicky, objekty a indexy. (19) (17)

Dopředný princip

Dopředný princip funguje na systému posouvání již zpracovaných dat z první skryté vrstvy neuronové sítě do další skryté vrstvy k další numerické operaci. Tímto posouváním dat dopředu ve skrytých vrstvách lze dosáhnout vždy potřebného výsledku v závislosti na tom, jaký očekáváme výstup od neuronové sítě. Počtem takto provedených výpočtů nervovou síť trénujeme a zpřesňujeme její výstup. Trénování neuronových sítí je potom jedna z nejdůležitějších fází implementace nervových sítí. (19) (17)

Vlastnosti dopředných vícevrstvých neuronových sítí jsou velmi dobře prozkoumány a existuji pro ně spolehlivé učící algoritmy. Proto se toto uspořádaní neuronových sítí často používá v průmyslových aplikacích při zpracování jak obrazových datasetů, tak hodnotových trendů. Zejména díky důkladnému zkoumání této sítě se používá jako jádro nebo částečná fáze pro ostatní nervové sítě. (19) (17)

Zpětný princip

Tento typ architektury vícevrstvých perceptronů obsahuje část neuronů, které jsou tzv. stavové. Ty mají za úkol zachytit vnitřní váhy v síti a přivést je zpět na vstup. Tímto se u neuronových sítí zajišťuje návaznost vstupních dat na výstupních datech.

Nejpoužívanější zpětnovazební vícevrstvý perceptron je Elmanova síť. Zde se výstupy (váhy jedné skryté vrstvy) ukládají do paralelní stavové vrstvy. Ta potom při další epoše přivádí váhy na vstup a stává se součástí vstupu. V závislosti na tom, kde síť odebírá hodnoty pro zpětnovazební působení, máme několik dalších modifikací, jako například Jordanova síť, která potom odebírá výstup z výstupní vrstvy a přivádí ho zpět do skryté vrstvy. (19) (17)

6 Návrh a realizace neuronového firewall

V rámci praktické části bude realizována zjednodušená verze neuronového firewallu. Jak bylo uvedeno již v kapitole 4.1.4 Stavové paketové filtry s kontrolou protokolů a IDS, jádrem neuronových firewallů jsou neuronové sítě, aplikace tedy bude vyhodnocována na základě výkonnosti jádra (modelu neuronové sítě) neuronových firewallů a jeho výkonnostních parametrech, na syntetických označkovaných datech a reálných datech, sesbíraných pomocí programu pro analýzu provozu v počítačových sítích Wireshark od společnosti The Wireshark team. Aby mohla být praktická část realizována, je potřeba vzít v potaz, že neuronový firewall musí odpovídat svojí efektivní velikostí účelu této práce, tedy hlavně jako ověření funkčnosti a zároveň jako podklad pro stanovení ekonomického a technického rámce jeho implementace. Proto bude v rámci práce navržena hlavně neuronová síť, a zbylé moduly budou rozebrány čistě teoreticky v návaznosti na vlivy, které tyto moduly mohou mít na výkonnost jádra neuronového firewallu, tedy samotné neuronové sítě. Problematika efektivní velikosti bude potom rozebrána dále v technologickém a ekonomickém rozboru. Pro účely této práce byly vypracovány dvě verze programu:

- **Pack&Go verze** – jednodušší a sekvenčně řešený zdrojový kód, který bude přílohou této práce. Tato verze je kromě rychlosti a objemu plnohodnotnou verzí Fullstack verze. Pack&Go verze je finální verze, která je přílohou této práce, jelikož se jedná o finální produkt této práce.

- **Fullstack verze** – Tato verze byla zpracována pro ověření zlepšení rychlosti a byla zpracovaná objektově pro lepší výkonnost architektur. Byla vypracována pouze jako ověření, zda v této části návrhu a implementace neuronových sítí neleží lepší řešení oproti identickému řešení v sekvenčním typu programování.

Z teoretické části vyplývá, že v současné době jsou nejvíce používané základy architektur pro detekci anomálií v rámci strukturovaných dat a v tzn. Recommender systems (doporučovacích systémech) jsou architektury neuronových sítí, jako jsou autoenkodéry a vícevrstvé perceptrony. Tyto architektury byly porovnány v „playground.tensorflow.org“, online aplikaci pro zevrubnou analýzu architektur neuronových sítí. Pro širší využití a v rozličnějších parametrech vycházela architektura vícevrstvého perceptronu jako stabilnější. (20)

Následně byly vybrány dvě datové referenční sady, KDD CUP 99 a KDD-NSL pro tréning modelu, a to jako jedny z nejvíce popsaných a reprezentativních datasetů pro analýzu anomálií ve strukturovaných datech. Použití dvou open source datasetů potom bylo zvoleno pro větší referenci a možnost naučení vybraného modelu na jednom datasetu a dále otestování přesnosti na druhém datasetu. Tímto bylo možné zajistit vhodné určování přesnosti a porovnání s již vydanými publikacemi, které se zabývají využitím těchto datasetů.

Zavaděč dat lze potom realizovat dvojí způsobem:

- **Dynamicky** – Dynamické řešení si načítá s určitým zpožděním data přímo z probíhající komunikace a funguje s real-time daty.

- **Staticky** – Načtení záznamu je prováděno z CSV souboru, tento soubor byl stažený jako KDD-NSL a KDD CUP 99, nebo byl prořízen přímo ze síťového provozu.

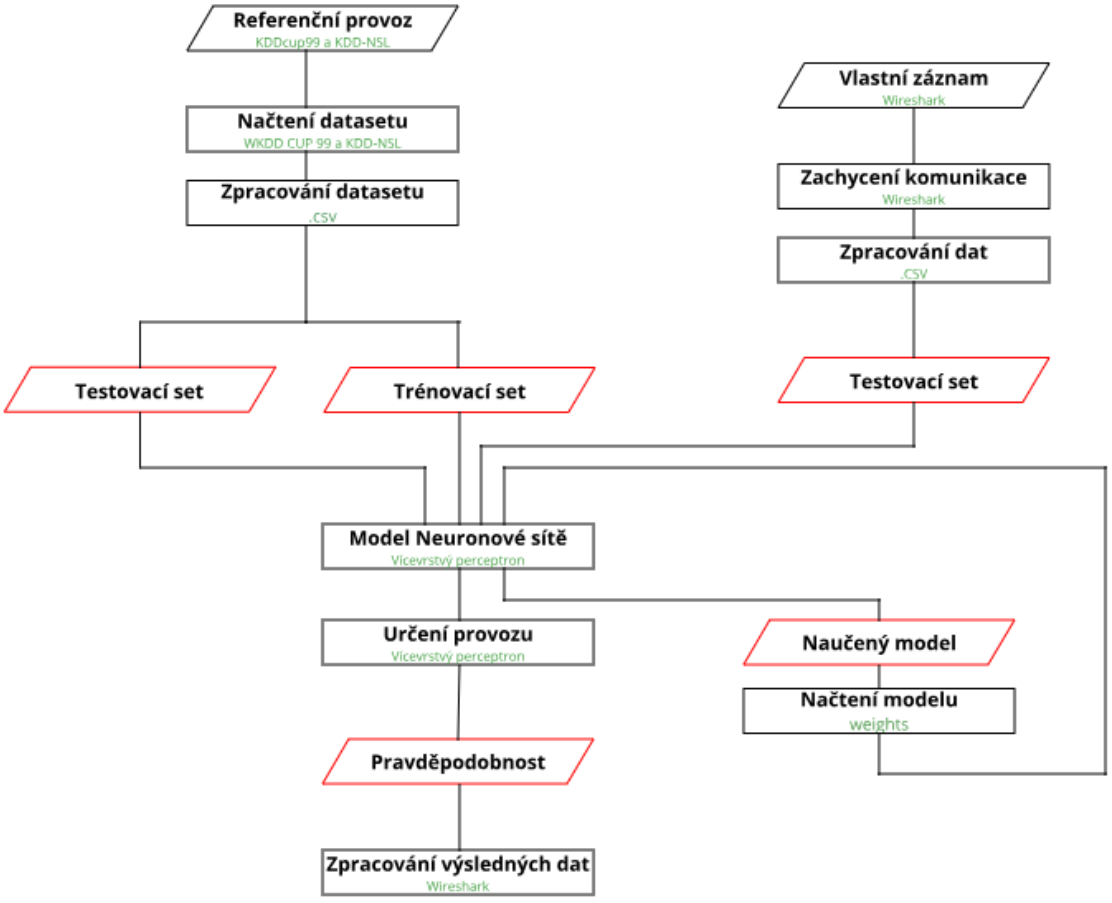
Celý návrh funkce obou verzí architektur neuronového firewallu a jejich funkčnost je potom znázorněna na obrázku č.12. Na začátku fungování program vybírá, se kterým datasetem bude pracovat, program následně stáhne, uloží a zavede vybraný dataset a připraví ho ke zpracování. Program již v této části začne sledovat některé parametry a připraví složky pro ukládání informací o výkonnosti, a zapíše informace o probíhaném testu.

Ve všech třech případech ukládaných datasetů jsou data zapsaná ve své „nejsurovější“ podobě a je tedy potřeba je statisticky upravit, aby byla porovnatelná a měla stejnou měrnou hodnotu pro model neuronové sítě. Zde začne program připravovat datasety a začne docházet k vytváření dvou typů hodnot v rámci každého datasetu, tzn. dummy proměnné a z-score. Stavové nečíselné veličiny se převedou na dummy proměnné, a z číselných hodnotových veličin se vypočítají poměrné z-score. Model neuronové sítě nerozděluje jednotlivé sloupce podle jejich významu v rámci síťové komunikace a referenčního modelu OSI podobně jako člověk. Neuronová síť proto toto zpracování provádí, aby nemusela řešit význam jednotlivých sloupců, a mohla porovnávat váhu výskytu. Po zpracování datasetů se opět celý krok zvlášť uloží a program bude připraven k další fázi.

V další fázi je dle schématu znázorněném na obrázku č.12 již statisticky upravený dataset rozdělen na dva datasety. Tj. trénovací, což je původní dataset pouze upravený statisticky, a testovací který se syntetizuje z trénovacího datasetu. Z původního trénovacího datasetu se odebere procentuální část a je následně zbavena sloupce „outcome“, neboli označení každého

indexu záznamu síťové komunikace, určující o jaký typ provozu jde. Tento krok byl ozkoušen od 10-100 %, a ovlivňuje výsledek nejen ve výkonnost, ale hlavně v přesnosti zpracování výsledků. Pro výkonnost testování byla stanovena optimální velikost odebrání jako 25 % z celkového počtu indexů při vyšších hodnotách, aby již nedocházelo k výrazným změnám v přesnosti, a bylo tak možno zachovat rozumnou výkonnost při práci s datasetem. V kapitole 7.1 Interpretace dosažených výsledků bude tato část následně více rozebrána.

Po zpracování dat, neuronová síť vytvoří a naučí model, nebo si umí načíst již naučený model. Při načtení je model pouze načten a neuronová síť přechází okamžitě k určení neoznačkovaného datasetu. Z tohoto postupu jsou potom určovány výsledné přesnosti uvedené dále v praktickém zpracování práce. Při volbě uživatele pro vytvoření nového modelu dochází k načítání trénovacího datasetu s označenými indexy síťové komunikace do modelu (jádra) neuronové sítě, a tím dochází k vlastnímu učení neuronové sítě. V tomto kroku je samozřejmě zaznamenáván čas učení tak, aby nebyly započítávány časové prostoje ostatních částí. Následně program rovnou začne značkovat testovací dataset a v obou případech práce s modelem



Obr. 12 Zobrazení funkce programu, Zdroj: Autor

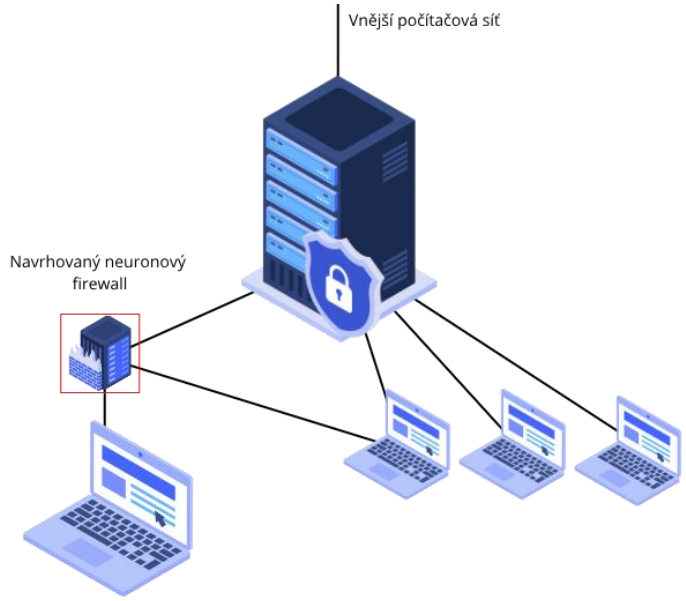
neuronové sítě dojde k uložení označkovaného datasetu s pravděpodobnostmi určení jednotlivých typů komunikace.

Ve fázi zpracování výsledných dat se porovnají oba datasety, tedy nově označkovaný testovací dataset a trénovací dataset, a jsou porovnány indexy ze sloupce „outcome“. Z tohoto porovnání jsou potom určovány výsledné přesnosti neuronové sítě.

Pro tvorbu neuronové sítě a celého neuronového firewallu byl využíván textový editor ATOM ve verzi v1.55.0 (2021-03-09) od společnosti GitHub Inc., verze Pythonu. Byla použita 3.8 v rámci celé aplikace, a to jak kvůli kompatibilitě s Tensorflow modulem, tak také zabezpečení a hladší kompatibilitě se zbytkem modulů jako Pandas, Sklearn a další, které s modulem Tensorflow pracovaly stabilně hlavně ve starších verzích. Virtuální prostředí a celou správu package managementu potom bylo nutné zajistit přes správce modulů Anaconda od společnosti Anaconda, Inc., ve své mini verzi. V následujících kapitolách je popsán průběh praktické části a jsou rozebrány jednotlivé části, dále jsou zpracovány výsledky a ekonomického a technologického zhodnocení implementace na základě realizace neuronové sítě.

6.1 Umístění v síti

V rámci této práce se předpokládá z výkonnostního hlediska a vzhledem k obsáhlosti probírané problematiky, že uvažovaný navrhnutý firewall bude teoreticky sloužit k ochraně koncového uzlu, a to vzhledem k jeho jednoduchosti a předpokládaným dosahovaným rychlostem. Uvažovaný neuronový firewall by mohl pracovat v interní počítačové síti za rozvinutějším



Obr. 13 Možnost teoretického umístění, Zdroj: Autor

průmyslově řešeným firewallem. V tomto umístění by se mohla vyplatit jednoduchost a rychlost, se kterými v této práci navržená architektura vícevrstvého perceptronu pracovala.

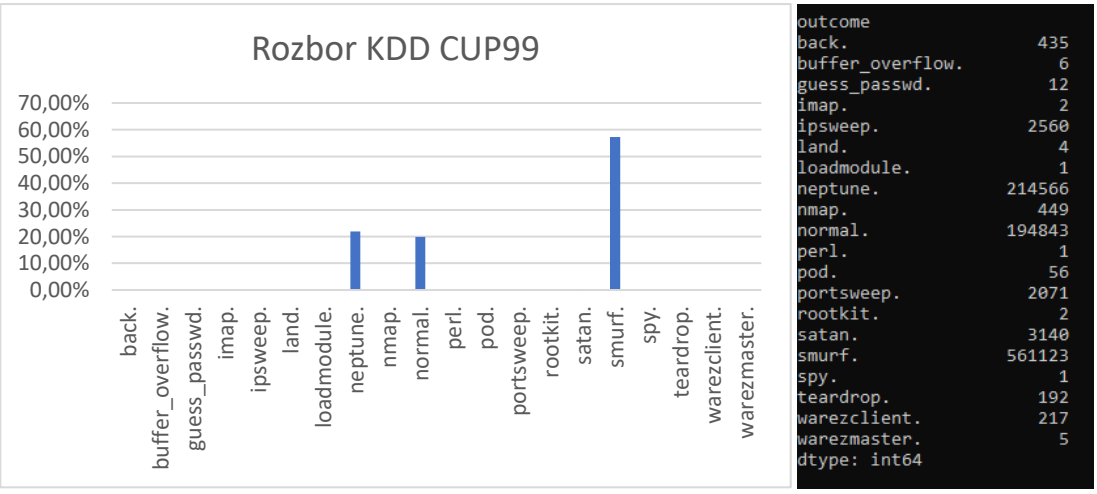
Obrázek č.13 zobrazuje schéma implementace v rámci sítě, kde je znázorněn předpoklad, že navrhovaný neuronový firewall bude také provádět kontrolu datového toku v rámci interní počítačové sítě. Teoreticky by se mohlo jednat o symetrické, mnohem modernější a výkonnější řešení ke v současnosti používaným aplikačním branám.

6.2 Výběr datasetu pro testování

Tak jak strojové učení a neuronové sítě získávají na významu v oblasti detekce anomálií, vyvstává také otázka, čím tyto neuronové sítě učit, případně podle jakých dat tyto sítě kontrolovat. Tato fáze testování a v některých architekturách hlavně i učení je nezbytná, jelikož ovlivňuje následný výkon celé neuronové sítě, a špatný výběr datasetů představuje vážné ohrožení bezpečnosti a výkonosti celého výsledného systému. Právě reálnost a vhodnost výběru vstupních dat při učení posléze vypovídají o kvalitě neuronové sítě. V zásadě existují tři typy, jak lze získávat datasety, a to: (12)

- Zachycením reálného provozu
- Syntetizací umělého datasetu
- Attack launching.

V rámci možností této práce byla snaha o zastoupení všech tří výše zmíněných způsobů. Zachycení reálného provozu je ovšem z ekonomických a situačních důvodů nahrazeno a zkombinováno s metodou attack launching. Byl vytvořen umělý datový provoz, který byl

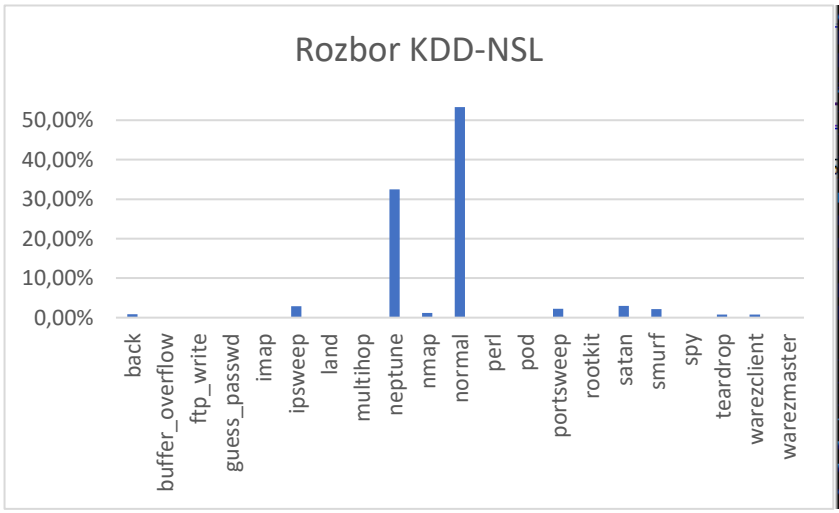


Obr. 14 Rozbor KDD CUP 99, Zdroj: Autor

napaden ve virtuálním prostředí programu VirtualBox 6.1.20. V tomto prostředí byl vytvořen server a ten byl otevřen k naslouchání příchozí komunikace protokolu SSH. Běžný provoz zachycoval přístupy na sociální sítě, sledování videí na YouTube, přihlašování a chatování. V rámci toho byl provoz posléze napaden pomocí programu Hydra v OS Kali Linux. Zde je potom zastoupení vzorku minoritní, v obsahu 3 % pouze pro ověření funkčnosti. Zbylých 97 % datasetu je běžná síťová komunikace. Tento kratší vzorek dat je posléze testován v již naučeném modelu neuronové sítě pro ověření v reálném provozu na koncovém uzlu.

V rámci využití syntetizovaného umělého datasetu byly vybrány datasety KDD CUP 99 a KDD-NSL. Od roku 1999 si na tomto poli bezpečnosti počítačových sítí získal důvěru právě dataset KDD CUP 99, který v dnešní době tvoří referenční základ, a to i přestože vzniklo několik studií, které zpochybňovaly referenční sílu tohoto datasetu vzhledem k jeho stáří a pravděpodobné zastaralosti obsažených útoků. Výběr datasetu pro ověření funkčnosti a analýzy výkonu navržené neuronové sítě byl tedy vybírán tak, aby představoval dostatečný referenční základ jak po podkladové, tak i po technické stránce. (12)

Dále byly možnosti použití datasetů posuzovány podle jejich stáří, redundance dat a podílů zastoupení jednotlivých vzorků. Právě kvůli zastoupení jednotlivých minoritních typů útoků byl k datasetu KDD CUP 99 vybrán ještě jako doplněk KDD-NSL dataset. I přestože redundance některých útoků je stále nadbytečná, tak jak je vidět na obrázcích č.15 a 14, v datasetu KDD-NSL je zastoupení minoritních typů útoků lepší nežli v datasetu KDD CUP 99. (12)




```
outcome
back              206
buffer_overflow     5
guess_passwd        9
imap                2
ipsweep           718
land                3
loadmodule          2
multihop            1
neptune          8208
nmap              299
normal          13484
pod                38
portsweep         582
rootkit             1
satan             741
smurf             523
spy                 1
teardrop          192
warezclient       175
warezmaster         5
dtype: int64
```

Obr. 15 Rozbor KDD-NSL, Zdroj: Autor

Obecně lze rozdělit obsažené útoky v obou datasetech do čtyř kategorií:

1. **DoS (denial of service) útoky** – snaha o přehlcení a vypnutí poskytované služby, například: back doors, land, neptune, pod, smurf, teardrop.

2. **R2L (remote to local) útoky** – neautorizovaný pokus o přístup ze vzdáleného zdroje, například: Guess_passwd, warezmaster, warezclient, multihop, phf, imap, spy, ftp_write.

3. **U2R (user to root) útoky** – neautorizovaný pokus o přístup na správcovský účet, například: loadmodule, perl, buffer_overflow, rootkit.

4. **Skenování/sondování (Probing)** – pokus o sledování a dolování informací, například: ipsweep, nmap,portsweep, satan. (21) (12)

Oba datasety také sledují u každého indexu 41 vlastností, jejich přehled je potom obsažený v příloze č.1

6.2.1 Rozbor datové sady

Rozbor datové sady se provádí před zpracováním datasetu pro model neuronové sítě. Tento záznam program automaticky ukládá pro budoucí referenci. Rozbor ukazuje zastoupení jednotlivých typů dat v datasetu a bude vypovídat o schopnostech naučeného modelu neuronové sítě. Rozbor datové sady se posléze ukládá do souboru s daty o provedeném testu.

Vzhledem k tomu, že program využívá pro vytvoření testovacího datasetu pouze 25 %, výsledky se mohou lišit. V případě použití 25 % původního trénovacího datasetu k vytvoření testovacího datasetu pak ovšem může docházek k malým rozílům v absolutním zastoupení jednotlivých útoků. Právě proto je rozbor datové sady vždy uložen pro zajištění zpětné reference, a to i přes velmi malé rozdíly. Rozbor absolutního zastoupení je potom automaticky zobrazen, aby mohl být ověřen průřez, s jakými daty program pracuje.

6.3 Realizace zavaděče dat

Zavaděč dat pro účely této diplomové práce bude realizován hlavně staticky ze souboru pro celkově tři datové sady. Při reálném použití je nicméně nezbytné použít dynamickou metodu zavádění dat a sledovat přímo přicházející síťovou komunikaci. Při realizaci této práce je však možné využít statickou metodu a data načítat a analyzovat retrospektivně ze souboru .CSV

a .arff, ve kterých jsou dostupná data z datasetů KDD CUP 99 a KDD-NSL, popřípadě data vytvořená v rámci této práce a syntetizovaná z programu Wireshark. Tato metoda má také lepší kontrolu nad vstupními daty.

6.3.1 Statická metoda zavádění dat

Statickou metodou se rozumí realizace zavaděče dat, která již byla nahraná a uložená. Prakticky jde o běžnou metodu retrospektivního zavádění dat do většiny programů. Výhodou tohoto systému je větší přehled a kontrola nad zaváděnými daty, a není zde potřeba řešit časovou náročnost systému. Při této metodě je možnost více rozpracovat neuronovou síť a soustředit se na její výkonnost a vyhodnocení získaných výsledků. Zavaděč v přiloženém programu má ovšem své specifikum, a to z důvodů možnosti úspory dat, je zavaděč dat realizován tak, aby si data potřebná k naučení a otestování z datasetů KDD CUP 99 a KDD-NSL stáhl sám z internetu, a samostatně je zavedl do „dataframe“, a následně přivedl ke statistickému zpracování. Příklad tohoto zavaděče zobrazuje obrázek č.16.

```
57        try:
58            path = get_file('kddcup.data_10_percent.gz', origin=
59            'http://kdd.ics.uci.edu/databases/kddcup99/kddcup.data_10_percent.gz')
60        except:
61            print('Error downloading')
62            raise
63
64        df = pd.read_csv(path, header=None,)
```

Obr. 16 Kód pro získávání datasetu online, Zdroj: Autor

6.3.2 Dynamická metoda zavádění dat

Dynamická metoda využívá real-time zavádění dat, neboli jinak řečeno jsou to systémy reálného času. Při tomto způsobu realizace pracuje jádro neuronové sítě s datovou sadou, která je aktuálně přijímána. Tento způsob zavádění dat je z hlediska ochrany počítačové sítě nejúčinnější, ale má několik požadavků specifických právě pro tento typ práce s datovými toky. Těmi nejdůležitějšími parametry pak jsou časová, výkonová a logická náročnost takovýchto systémů. Systémy reálného času musí velmi rychle provést analýzu a případně zasáhnout v řádech milisekund do síťové komunikace. Systémy reálného času potom mají vysoké nároky spíše na časovou přesnost než výkonnost. Výsledky musejí být správně vyhodnoceny, ale především je posléze nutné ověřovat časovou správnost. Jednotlivé indexy z datasetů jsou různě datově objemné, a bylo by potřeba zařídit stejnou časovou náročnost. V tomto směru je potom navrhnutý program připravený, jelikož používá jednoduchý před-rozbor datasetu. To umožnuje

detekovat anomálie v datovém objemu indexu, a následně je model potom díky své jednoduchosti a robustnosti zpracování zaváděných dat schopnen provádět zpracování s minimálními časovými rozdíly v rámci jedné architektury. To je do značné míry výsledkem použití uceleného modulu Tensorflow. Také navržení jádra je odolné vůči chybám v kódu a chování takového modelu je dobře předvídatelné pro využití dynamického zavádění dat. (9)

Navrhnutý program je i přes svoji jednoduchost a provoz testovacích úloh v domácí síti schopný zpracovat a ohodnotit 494 tisíc řádků (indexů) dat v upravené podobně o velikosti 78 Mb, což odpovídá zhruba 4,5 sekundám datového provozu dle programu Wireshark. V nejlepším možném nastavení celého programu bez datového rozboru a součástí potřebných pro splnění zadání této práce. Dosahoval výkonu kolem 0,3 sekundy což odpovídá 9 milisekundách na 78Mb. To odpovídá výkonnosti 0,052 Gbps na domácím notebooku s veřejně dostupným datasetem. To sice není průmyslová výkonost firewallů, ta se pohybuje až kolem 5-10 Gbps, ale naznačuje to určitou schopnost rozvedení a použití tohoto modelu v kombinaci s dynamickou metodou zavádění dat, popřípadě jeho praktického využití v menších objemech dat, či použití před koncovými uzly počítačové sítě. (9)

6.3.3 Statistická úprava dat

Statistickou úpravu dat provádí program před přivedením dat k modelu neuronové sítě. Před zavedením dat do neuronového firewallu je potřeba sledované data převést (standardizovat) v rámci datasetu tak, aby je mohla neuronová síť posuzovat vzájemně. Informace o datovém toku neboli datasetu dělíme na dva typy podle toho, jak je statisticky zpracujeme, tak, aby s nimi byla neuronová síť schopna pracovat. Neuronová síť nerozeznává jednotlivé informace podle jejich významu. Nerozlišuje sloupce v datasetu a jejich vztazích v referenčnímu modelu síťové komunikace OSI jako člověk. To je její největší výhodou a možná lehkou nevýhodou. Výhoda této určité „lhostejnosti“ k významu jednotlivých dat umožňuje vidět v datech souvislosti a návaznosti, které není člověk schopen postřehnout bez velmi pracného a náročného zkoumání. Mírnou nevýhodou může a nemusí být potřeba přepracování dat na poměrné veličiny. To právě umožňuje neuronové síti výše zmíněný hlubší pohled na propojení jednotlivých veličin.

Program tedy tyto veličiny dělí podle:

- **Spojité numerické informace** – jako například čas a velikost dat.
- **Diskrétní textové informace** – jako označení špatného paketu či ohlášení chybného přihlášení.

Proto spojité informace v datasetu program statisticky upraví na standardizované skóre, a to konkrétně na z-skóre. Program v jednoduchosti provede lineární transformaci v rámci datasetu, tudíž škálu rovnoměrně posunuje a mění měřítko, nedochází však ke ztrátám datových informací, jelikož z-skóre nedeformuje vzdálenosti mezi jednotlivými hodnotami. Tuto operaci program provede u těch veličin, které jsou spojité, a zbylé ignoruje. Statistická úprava pomocí z-skóre má nejčastěji průměr 0 a směrodatná odchylka vychází 1. Jelikož z-skóre může vyjít i záporně, bude potřeba na konci při zpracování výstupních informací provést datovou úpravu absolutní hodnotou. Záznam kódu je zobrazený na obrázku č.17.

```
34  def encode_numeric_zscore(df, name, mean=None, sd=None):
35      if mean is None:
36          mean = df[name].mean()
37
38      if sd is None:
39          sd = df[name].std()
40
41      df[name] = (df[name] - mean) / sd
42
43
44  def encode_text_dummy(df, name):
45      dummies = pd.get_dummies(df[name])
46      for x in dummies.columns:
47          dummy_name = f"{name}-{x}"
48          df[dummy_name] = dummies[x]
49      df.drop(name, axis=1, inplace=True)
```

Obr. 17 Statistické předzpracování dat, Zdroj: Autor

Zde se potom nachází nevýhoda neuronových sítí s učením za pomocí učitele, jelikož je potřeba neuronové síti připravit tato data před zavedením. Musíme znát obsah datové řady podle které učíme abychom mohli říct které sloupce v datasetu se mají počítat tímto způsobem a která ne.

Diskrétní informace, jinak řečeno informace vyjádřené jinak, než číselně v datasetu následně potom mění na tzn. dummy proměnné. Každé písemně vyjádřené hodnotě vytvoří sloupec a indexy kde se písemná informace vyskytovala přidělí hodnotu jedné a kde byla jiná informace či se nevyskytovala žádná informace přidělí nulu. (21)

6.4 Model neuronové sítě

Jak bylo již řečeno v úvodu této kapitoly, v praxi se v současné době používají nejvíce dva typy architektur pro realizaci doporučovacích neuronových sítích. Autoenkodér a vícevrstvý perceptron. Tyto dvě architektury byly nejdříve zevrubně nasimulovány v prostředí „Playground.Tensorflow“, kde vycházel lépe autoenkodér, co se rychlosti učení týče. Jak se později v rámci realizace praktické části ukázalo, autoenkodéry dokázaly díky své architektuře dosahovat zajímavých hodnot v rámci učení, a to kolem řádů jednotek setin, ale nedosahovaly stejných kvalit následně při vytváření predikcí a hodnocení jejich úspěšnosti (přesnosti) určení, a to jak základní procentuální přesností, tak i při měření metriky Recall. Tak jak zobrazují grafy E.) a F.) na obrázku č.24. Co se týče doby naučení tak, autoenkodéru stačil mnohem menší počet epoch, průměrně kolem 12 epoch k naučení na maximální hodnotu, ale také mnohem rychlejší byla strmost křivek Loss. To znamená, že autoenkodér byl schopen se velice rychle naučit, ale také docházelo k rychlému přeučení již po 50 pevně nastavených proběhnutých epochách, jak ukazuje graf F.) na obrázku č.24. Naproti tomu byl vícevrstvý perceptron mnohem robustnější a dokázal dosahovat mnohem lepší konečné přesnosti, tak jak zobrazuje tabulka č.4. Tabulku úvodních nejvyšších dosahovaných přesností před laděním zobrazuje tabulka č.2.

		KDD CUP99	**KDD-NSL**	**My_dataset**
MLP	% - AUC	91,54 %	90,65 %	70,6 %
50-50-50-50-50	R – AUC	0,96894211	0,957832	0,8501
AE	% - AUC	86,59 %	79,93 %	63,2 %
64-32-16-32-64	R – AUC	0,90896	0,869463	0,8778

Tab. 2 Tabulka úvodních přesností neuronových sítí, Zdroj: Autor

Jelikož u bezpečnostních systémů, a hlavně u takto řešených systémů, probíhá fáze učení a návrhu obvykle před ostrou implementací do počítačové sítě, byl tedy z tohoto hlediska zvolen vhodnější vícevrstvý perceptron, jelikož si dokázal lépe poradit s rozličnými datasety se zachováním relativně dobré spolehlivosti, tak jak zobrazuje tabulka č.2. Také při dalších testech se ukázalo, že zbytečně velký počet neuronů ve skrytých vrstvách, jednak u autoenkodéru pak u vícevrstvého perceptronu, nejen snížil rychlost učení, ale také zvýšil náchylnost k přeučení u obou typů architektur. Jak zobrazují grafy C.) a D.) na obrázku č.24.

Všeobecně řečeno, čím jednodušší byly architektury, tím lépe si následně vedly. U jednodušších datasetů bylo dokonce nezbytné, aby byla architektura jednoduchá, což znamenalo tři skryté vrstvy s 20-ti neurony v každé. Tuto architekturu nakonec zobrazuje obrázek č.18. U autoenkodéru byla nejvhodnější architektura 32-16-32. Obě tyto architektury měly 25 % trénovací sady z celkového datasetu. Při vyšších poměrech potom docházelo opět ke zvýšení náchylnosti vůči přeučení. Kód finální architektury vícevrstvého perceptronu je potom uvedena v příloze č.2.

V obou případech architektur byla také použitá funkce „EarlyStopping“, která zastavuje trénování monitorovaných metrik. V jednoduchosti zastavuje načtení další epochy v době, kdy je absolutní změna v měřených metrikách rovná min_delta. Zde byla uplatňována hodnota $1*10^{-3}$, ta se ukázala jako nerozumnější v rámci datasetů. Při vyšších hodnotách byl potom problém se zvýšením citlivosti k přeučení, jelikož funkce neukončila proces dostatečně včas.

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25, random_state=42)
print(len(x_test))

start_2 = time.process_time()

model = Sequential()
model.add(Dense(20, activation='relu'))
model.add(Dense(20, activation='relu'))
model.add(Dense(20, activation='relu'))
model.add(Dense(y.shape[1],activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam')
model.compile(loss='hinge', optimizer="adam", metrics=['val_accuracy'])
model.compile(loss='mean_squared_logarithmic_error', optimizer="adam", metrics=['mse'])
model.compile(loss='hinge', optimizer="adam", metrics=['Recall'])
monitor = EarlyStopping(monitor='val_loss', min_delta=1e-3, patience=5, verbose=1, mode='auto', restore_best_weights=True)

history = model.fit(x_train,y_train,validation_data=(x_test,y_test), verbose=2,epochs=40)

cas_2 = time.process_time() - start_2
file1 = open(umisteni + "\Test" + test + "\zaznam\\" + "score_" + test + ".txt","a")
file1.write(r"čas naučení neuronové sítě:" + str(cas_2) + " \n")
file1.close()

np.savetxt(umisteni + "\Test" + test + "\zaznam\\" + "data_x.csv", x, delimiter=",")
np.savetxt(umisteni + "\Test" + test + "\zaznam\\" + "data_y.csv", y, delimiter=",")
model.save_weights(umisteni + "\Test" + test + "\zaznam\weights" + "\weights" + test)
model.save(umisteni + "\Test" + test + "\zaznam" + "\model" + test)
```

Obr. 18 Příklad zdrojového kódu modelu neuronové sítě. Zdroj: Autor

Před zavedením datasetu bylo potřeba, aby program dokázal rozdělovat datasety na trénovací a testovací. Zde se ukázala nejvhodnější funkce train_test_split u sekvenčních typů modelů. Poměr objemu testovacích a trénovacích dat byl zkoušen na vícevrstvých perceptronech. Grafy A.) a B.) na obrázku č.24, ukazují grafy, při kterých byl zkoumán právě tento poměr na architektuře o 5-ti skrytých vrstvách, graf A.) zobrazuje užití 30 % objem testovacích dat z celkového datasetu v použití s vícevrstvým perceptronem. V použití na všech případech datasetů se vícevrstvý perceptron dokázal s tímto poměrem dostat přes 90 %, vyjma my_datasetu, to je pravděpodobně zapříčiněno surovostí dat. Na grafu B.) byl použit 40 %

objem dat. Je na něm vidět jasné zhoršení výkonnosti vícevrstvého perceptronu, a to jak z hlediska kvality učení, ale také z hlediska výsledné přesnosti predikovaných dat.

6.4.1 Vícevrstvé neuronové perceptrony

Ačkoliv se dříve architektura vícevrstvých perceptronů využívala ve většině aplikací, v současné době je architektura vícevrstvého perceptronu z technologického hlediska používána hlavně k jednodušším úkolům práce s datovými řadami, i v současnosti se také jedná o nejvíce implementovanou architekturu. (19)

Z teoretické části této práce vycházeli dvě architektury, které by měli být vhodné pro zpracování datových řad a pro implementaci v neuronových firewallech, a to autoenkodér a vícevrstvý perceptron. Ve výše uvedené praktické části bylo také zmíněno, že v předběžných zpracováních v aplikaci „Playground.Tensorflow“ vycházela lépe z hlediska učení architektura autoenkodéru, ovšem v rámci referenčního ověření této predikce v reálném použití vycházel vícevrstvý perceptron jako mnohem univerzálnější typ architektury. (19)

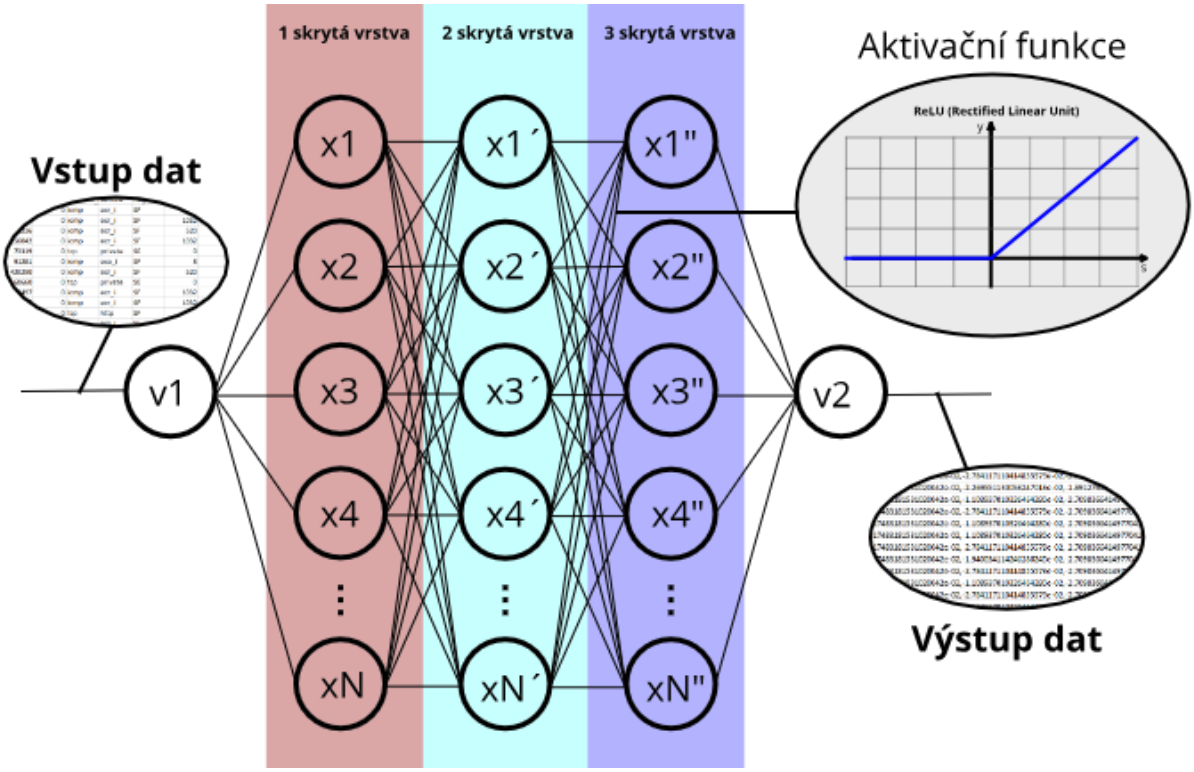
Vícevrstvý perceptron byl zpočátku relativně komplexní a předkládané datasety bylo potřeba začít vhodně upravovat pro použití právě v této architektuře. Výsledkem několika úprav, provedených od počátečního designu architektury k finálnímu modelu neuronové sítě, to byl model, který si dokázal poradit nejen s nejvyšší přesností, (s chybou pouhé dvě setiny při komparaci trénovacích a testovacích datasetů), ale také vykazoval nejvyšší přesnost a schopnost zpracovat i rozličné datasety. Proto hned v úvodu realizace praktické části a kvůli uvažované finální implementaci byl zvolen právě vícevrstvý perceptron. (19)

Vícevrstvý perceptron byl zkoušen v pěti vrstvách a 50-ti neuronech v každé vrstvě. Základním předpokladem této úvahy bylo použití v několika příkladech přímo na stránkách tensorflow.org. To se později ukázalo jako nedostatečné a spíše předimenzované řešení. Nejenom že rychlost učení byla pomalejší, ale neuronová síť byla rovněž velmi náchylná k přeučení již po 30-ti epochách. Tento stav přetrvával i přes úpravu vstupních dat a úpravu poměru datasetů. Přesnost tohoto modelu se pohybovala v průměru kolem 90 % a rychlost učení se pohybovala v optimu kolem 27 epoch. Následně se model rychle stával náchylný k přeučení a stal se nestabilním v použití na jiných datasetech.

Dále bylo zjištěno, že snížením počtu neuronů v jednotlivých vrstvách, došlo k výraznému posunu procesu učení neuronové sítě. Neuronová síť byla schopná dosahovat mnohem

vyváženějších a stabilnějších výsledků s průměrnou přesností kolem 95 %, a to lépe v rámci použití na různých datasetech opět vyjma My_dataset, ale i zde byl jasný posun. Od této úpravy byla architektura vícevrstvého perceptronu výrazně lepší, než autoenkodéru, i přes podobné úpravy, proto bylo zde již ukončeno další referenční ověřování autoenkodéru a pozornost byla věnovaná právě vícevrstvému perceptronu.

Jak zobrazuje obrázek č.19, jako výsledný počet vrstev byly zvoleny tři skryté vrstvy. Vzhledem k předchozím úpravám, které převážně zjednodušovali architekturu, bylo také vyzkoušeno 6 a 2 skryté vrstvy modelu. U 6 vrstev, byly modely pravděpodobně poslední 2 až 3 skryté vrstvy zbytečné a výsledek zkreslovali, jelikož se výsledná přesnost snížila o průměrně 6 %. Naopak při dvou vrstvách se neuronová síť stala velmi rychle náchylnou k sebemenším změnám a ztratila stabilitu ve výsledcích. Proto byly pro zachování stability a zároveň zmírnění zkreslování výsledku použity tři vrstvy. Tato architektura začala dosahovat již zajímavých přesností, a to i 98 %, a vykazovala velmi stabilní a předvídatelné chování jak v rámci epoch, (testováno bylo 3-100 epoch), tak v rámci různých poměrů testovacích a trénovacích datasetů, či rozdílných zdrojových datasetů. Maximální dosažené výsledky jsou potom více rozebrány v kapitole 7.1 Interpretace dosažených výsledků.



Obr. 19 Detailní vyobrazení funkce MLP neuronových sítí, Zdroj: Autor

Jako loss_function byla vybrána categorical_crossentropy, to vyplývá z obsahu datasetu, kdy neurčujeme binární hodnoty jako u funkce binary_crossentropy, ale určujeme několik možných výsledků (kategorií) ve sloupci „outcome“.

Z hlediska matematického pojetí hodnoty, s jakou určujeme data ve výsledném sloupci „outcome“, tedy konverzi reálných dat na data pravděpodobností výskytu každého možného výsledku, vychází pro použití funkce softmax, ta nabývá hodnot od nuly do jedné, a součet všech hodnot je roven 1. Každý výsledný vektor je potom matematickým zápisem pravděpodobnosti a stačilo jen posléze vybrat vektor s největší pravděpodobností, a podle jeho pozice určit výslednou předpovězenou hodnotu.

6.4.2 Učení neuronové sítě

Jak zobrazuje obrázek č.20, v předběžných zkoumáních vycházela lépe architektura autoenkodéru z pohledu průběhu učení. Ovšem vzhledem k tomu, že nebyla možnost v prostředí „Playground.Tensorflow“ více podrobně zkoumat vstupní datasety a jejich struktura, nebylo ještě ustoupeno od architektury Vícevrstvého perceptronu. Ve fázi učení oba datasety pracují s původním datasetem pouze statisticky upraveným a označkovanými popisky, o jaký typ síťové komunikace se jedná.

Při tomto porovnávání byl autoenkodér v předběžném zpracování lepší a stabilnější tak jak zobrazuje právě obrázek č.20. V levém sloupci jsou hodnoty autoenkodéru, který měl architekturu složenou z vrstev v počtech neuronů 64-32-16-32-64, s funkcí ReLU a používal 30 % vstupních trénovacích dat jako validační. Na obrázku je vidět, že v rámci předběžného zkoumání vycházel autoenkodér lépe o skoro čtyři desetiny lépe, a i průběh byl na stejný počet epoch stabilnější. U všech grafů v obrázku č.20 bylo provedeno 80 epoch.

Obr. 20 Počáteční zpracování původních architektur, Zdroj: Autor

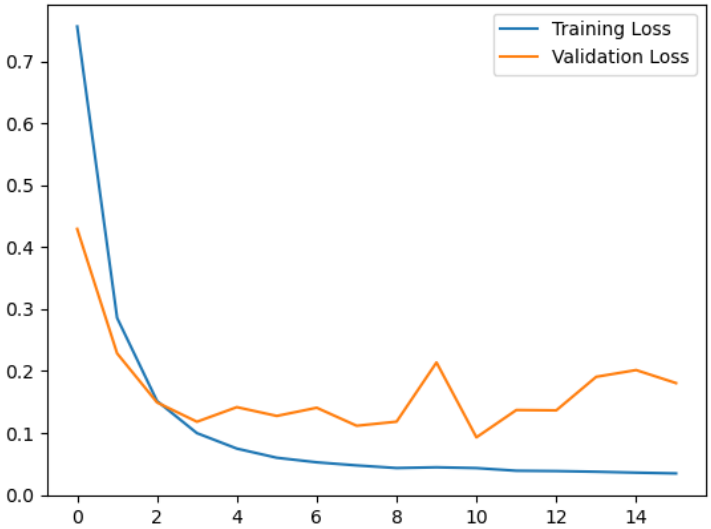
V průběhu učení ve full stack verzi programu dosahoval nejlepších výsledků autoenkodér, který se dostával pravidelně pod Training_loss 0,016 a Testing loss pod 0,017. Z obrázku č. 20

v levém sloupci je vidět, že při stejných podmínkách obdobně komplexní architektura vícevrstvého perceptronu nedosahovala ani podobného výkonu, ani takové stability. Jak ovšem znázorňují grafy E.) a F.) na obrázku č.24, byl následně po úpravách vícevrstvý perceptron schopný překonat autoenkodér hlavně v přesnosti jeho predikcí, což byla vlastnost, která vzhledem k charakteru implementace je přednější nežli rychlost a výkonnosti při učení.

6.4.3 Přeučení neuronové sítě

V mnoha pracích se lze dočíst, že největším problémem neuronových sítí je určení optimální úrovně naučení z poskytnutých dat, a „overfitting“ nebo „underfitting“ jsou problémy stejně velké důležitosti, jelikož oba zasahují a zásadně snižují výkonost a přesnost neuronových sítí. Tyto jevy sledujeme právě na vytvářených grafech a tvarech křivek Training_loss a Validation_loss. Křivka Training_loss zobrazuje počty chyb v trénovací množině datasetu a chceme od ní, aby šla co nejstrměji dolů a co nejníže. Validation_loss potom zobrazuje chyby ve značkování testovaných datasetů určených k označkování neuronovou sítí. U této křivky v grafu požadujeme, aby byla hladká, a po úvodním snížení se maximálně lehce zvedala. Tak jako je zobrazeno na grafech E.) a F.) v obrázku č.24

Datasety, které byly zvoleny v úvodu této kapitoly, jsou co se objemu týče dostatečně velké, aby navrhovaná neuronová síť neměla problém s podučením (underfitting) ze strany datasetu. Přeučení neuronové sítě sledujeme hlavně na křivce Validation_loss, která popisuje, jak se dataset chová při značkování testovaného datasetu. Softwarově můžeme ovlivnit přeučení neuronové sítě pomocí funkce „EarlyStopping“, kde si v modulu této funkce můžeme nastavit



Obr. 21 Graf zachycující přeučení neuronové sítě, Zdroj: Autor

parametr „min_delta“, tak jak je zobrazeno na obrázku č. 21. Ten ukazuje samotný model s jeho architekturou a zobrazuje právě také funkci „EarlyStopping“. Podučení a přeučení neuronových sítí lze také významně ovlivnit pomocí architektury. V rámci praktické části této práce bylo zjištěno, že vícevrstvý perceptron s deseti a více neurony ve vrstvách, začínal být náchylnější k přeučení velmi brzo a začal se snižovat počet epoch, při kterých se již začínalo projevovat přeučení. Obrázek č.21 zobrazuje vícevrstvý perceptron s 20 neurony ve vrstvách a poměru trénovací množině vůči testovacím datům 60 %.

7 Testování a ověření funkce

Přiložený program má několik funkcí, pomocí kterých zajišťoval řádné otestování a zaznamenávání průběhu každého testu. Některé tyto funkce programu jsou nepotřebné ve fázi implementace, ovšem v rámci této práce a také ve fázích učení, je nezbytné zachytit některé veličiny hodnotově, či graficky. V průběhu této práce byly ověřovány hlavně rozlišné typy architektur a jejich reakce na změny struktury a typu vloženého datasetu. Dalším důvodem pro aplikaci těchto funkcí je množství provedených testů, a při zpracování zadání bylo potřeba ověřit několik různých kombinací, kdy se provedlo celkově přes 200 typů testů.

- **Vytváření záznamových archů** – podstatnou část zpracovatelské výkonnosti hned na začátku programu vyžaduje vytvoření požadovaných složek pro uložení záznamů. V případech, kdy je potřeba provést několik desítek až stovek testů, je tato část kódu téměř nezbytná pro zachování opakovatelnosti testů.

- **Zaznamenání grafů** – V grafech byly hlavně řešeny průběhy tzn. „Validation_loss“ a „Training_loss“. Na těchto křivkách sledujeme hlavně rozdíly mezi křivkami, a to jak skalární, tak vektorové. Také bylo sledováno, jestli křivka Validation_loss neuhýbá nebo se zbytečně moc nevzdaluje od křivky Training_loss. Pomocí dynamiky těchto dvou křivek bylo možné u učících se architektur rozpoznat velmi snadno kvalitu datasetu, porovnávat výkonnost architektury a další parametry, podle kterých lze jednotlivé neuronové sítě porovnávat.

- **Automatické uložení naučených modelů** – pro referenci a pro možnost vrácení program automaticky uloží i skalární hodnotu v synapsích modelu. To slouží také následně pro otestování naučených modelů na jednom datasetu a ověření jejich funkčnosti na druhém či dalším odlišném datasetu. Na základě tohoto postupu byly potom určovány zmiňované maximálně dosažené přesnosti.

- **Výpočet Recall hodnoty** – pomocí modulu tf.keras.metrics.recall lze jednoduše vypočítat jednu z nejzákladnějších metrik pro hodnocení a kontrolu neuronových sítí, implementovaných v predikčních systémech. Tato hodnota je zobrazena přímo při spočítání každé epochy, a je zobrazována v rámci zpracování, a ukládá se automaticky do složky. Dál je tato hodnota označovaná jako R-ACC.

- **Uložení jednotlivých fází datasetu** – program opět v rámci snahy o zachycení co nejdetailnějšího průběhu uloží zvlášť všech 6 stavů datasetů, kterými vstupní datasety v rámci zpracování prochází.

- **Měření času průběhu jednotlivých fází** – aby mohlo následně po provedení testů dojít k objektivnímu zhodnocení do kódu, byl zapracován kód pro měření času jednotlivých fází programu. Tak mohlo dojít k odlišení času spotřebovaného na podpůrné části a na času nezbytný pro provoz modelu.

- **Záznam přesnosti jednotlivých modelů** – druhým hlavním a ukazatelem je potom empirické porovnání diferencí. Neuronová síť ohodnotí každý jednotlivý záznam a přidělí každému záznamu sadu pravděpodobnostních čísel, kde každé číslo v pořadí reprezentuje jeden typ záznamu v datasetu. Tato čísla pak reprezentují, s jakou mírou neuronová síť předpokládá, že je daný záznam konkrétní typ záznamu. Vezme maximální číslo z této řady a porovnává ho s totožnými záznamy z ověřovací sady dat. Funkce sklearn.metrics.accuracy_score potom porovnává shodu a určuje celkovou procentuální sadu. Kód pro zjišťování tohoto záznamu přesnosti je potom zobrazen na obrázku č.22.

```
33  pred = model.predict(x_test)
34  pred = np.argmax(pred,axis=1)
35  y_eval = np.argmax(y_test,axis=1)
36  score = metrics.accuracy_score(y_eval, pred)
37  print("Validation score: {}".format(score))
```

Obr. 22 Příklad zdrojového kódu pro zpracování procentuální přesnosti, Zdroj: Autor

Celkové testování probíhalo v několika fázích. V první fázi byla proměřena a vytvořena architektura autoenkodéru, která se dle předběžného zkoumání zdála jako nejvýkonnější. To se později také prokázalo, ovšem pouze v oblasti počáteční práce s datasetem a v oblasti kvality a rychlosti, naučení kde autoenkodér vycházel o řád lépe než architektura vícevrstvého perceptronu. Po tomto zjištění byl vytvořen pro referenční účely také model s architekturou vícevrstvého perceptronu. V oblasti učení se vícevrstvý perceptron nedostal výkonností ve většině použitých kombinací architektury k architektuře autoenkodéru, nicméně co se týká přesnosti následného zpracování ověřovacího datasetu, dosahoval výrazně lepších procentuálních přesností při určování neoznačkovaných dat.

V obou použitých architekturách byly použity architektury v několika úrovních i řádech co se týká počtu vrstev, a to z důvodu zjištění extrémů při návrhu neuronových sítí v použití na vybrané datasety. Tabulka č.3 u zkoušených velikostí jednotlivých architektur ukazuje přehled architektur s významnějším výkonnostním posunem a zobrazuje maximální hodnoty dosažených přesností v průřezu zpracování všech datasetů.

Autoenkodér			Vícevrstvý perceptron		
neurony – vrstvy	%-ACC	R-ACC	Neurony – vrstvy	%-ACC	R-ACC
64-32-16-32-64	98,68 %	89,7 %	50-50-50-50-50	98,24 %	98,30 %
32-16-32	98,55	98,12 %	50-50-50	99,04 %	99,05 %
16-8-4-8-16	96.22 %	98,21 %	10-10-10-10-10	98,70 %	98,60 %
4-2-4	70,64 %	66,58 %	10-10-10	96,78 %	96,51 %
			20-20-20	99,98 %	99,08 %

Tab. 3 Tabulka srovnání architektur AE a MLP, Zdroj: Autor

Sloupec neurony-vrstvy – sloupec ukazuje jednoduchý záznam použité architektury, jinými slovy počet vrstev a počet neuronů.

Sloupec %-ACC – zobrazuje maximální dosahovanou procentuální přesnost při testování dané naučené architektury na jiném datasetu.

Sloupec R-ACC – zobrazuje hodnoty maximální dosahované recall přesnosti popisované v kapitole 5.1.4 Recall.

Z tabulky č.3, je patrné, že do určité úrovně při snížení počtu neuronů docházelo k většímu zlepšování u obou funkcí, hlavně ve výkonnosti obou přesností, zejména hodnot R-ACC. A to platilo konkrétně až k finální použité architektuře u Vícevrstvého perceptronu s třemi vrstvami po dvaceti neuronech v každé skryté vrstvě. V případě snížení počtu vrstev posléze docházelo ke zlepšení přesnosti, ale u autoenkodéru se ovšem v případech použití méně než 5 vrstev začalo objevovat rychlé přeučení. Počet vrstev byl optimální při 3 skrytých vrstvách u Vícevrstvého perceptronu, tak jako bylo předpokládáno z teoretické části práce, ovšem pro ověření byly vyzkoušeny také jedna a pět vrstev pro ověření. U jedné vrstvy docházelo k velmi rychlému přeučení a průběhy loss křivek vykazovali velmi nestabilní charakter, a to ve všech případech počtu neuronů. U třech vrstev vícevrstvého perceptronu bylo zjištěno, že nejoptimálnějším počtem neuronů ve vrstvách je dvacet. U menšího počtu začalo opět docházet ke zhoršení výkonu neuronové sítě.

Rychlost zpracování datasetu nebyla rozdílná v případech úpravy počtu neuronů ve skrytých vrstvách a v případě změny počtu skrytých vrstev byla změna v řádech tisícin maximálně setin. Tento parametr by hrál ovšem větší význam, pokud bychom neuronovou síť implementovali v rámci programu pracujícím s daty v reálném čase (systémy reálného času).

Rychlost statistické úpravy datasetů byla neměnná a zde záleží hlavně na velikosti zpracovávaného datasetu. V rámci práce byla snaha o zpracování co největšího možného datasetu. Velikostně byl největší dataset KDD CUP 99, ten obsahoval téměř 5 miliónů indexů (řádků v excelu) síťové komunikace. Zpracování a naučení z tohoto datasetu probíhalo několik desítek minut a velikosti tohoto datasetu by potom u testování odpovídala zhruba 4 minutám datového provozu.

7.1 Interpretace dosažených výsledků

7.1.1 Rychlost zpracování datasetu

V rámci práce byla při zkoumání použita varianta syntetizovaného 10 % datasetu z KDD CUP 99 a 20 % datasetu KDD-NSL, v obou případech datasetů jde o strukturované indexy síťové komunikace, která obsahuje z 80 % indexy abnormální povahy (útoky). V případě KDD-NSL datasetu, jak již bylo zmíněno výše, jde o odlišení v rámci hodnot a zastoupení jednotlivých útoků.

Přiložený program si ve své výsledné podobě dokázal poradit s 10 % syntetizovaným vzorkem datasetu KDD CUP 99 o velikosti 78Mb za 0,3 sekundy. To odpovídá 0,052 Gbps. V případě ostatních datasetů byla rychlost stejná, jelikož byly datasety strukturovány obdobně.

Tato dosažená rychlost testování tedy předpokládá, že program by byl schopný sloužit jako podklad pro implementaci na koncovém zařízení. V případě úvahy použití na koncovém uzlu, např. notebooku, kde se v rámci mobilních připojení (například pomocí USB modemu) rychlosti pohybují při zpracování této práce stále ještě v řádech jednotek Mbit/s, je pravděpodobné, že by si navrhnutá architektura vícevrstvého perceptronu po úpravách dokázala poradit rychlostně s běžnou síťovou komunikací na koncovém uzlu počítačové sítě.

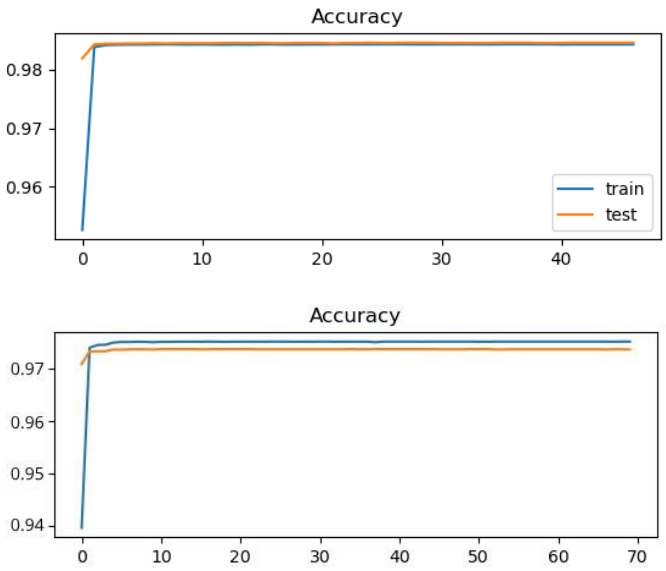
7.1.2 Přesnosti značkování indexů

Přesnost značkování byla vybrána jako jedna z nejdůležitějších metrik určování schopnosti a výkonnosti architektury výsledného programu. V případě uvažování reálné aplikace jsou neuronové firewally posuzovány především z hlediska jejich schopnosti zachytit případné počítačové útoky, proto i v rámci testování byla směřována pozornost hlavně na výslednou přesnost značkování indexů.

		KDD CUP 99	KDD-NSL	My_dataset
MLP	% - AUC	99,98 %	98,52 %	89,6 %
	R – AUC	0,98464	0,97783	0,9338
AE	% - AUC	98,68 %	96,93 %	82,6 %
	R – AUC	0,89700	0,83946	0,8868

Tab. 4 Výsledně maximální přesnosti architektur, Zdroj: Autor

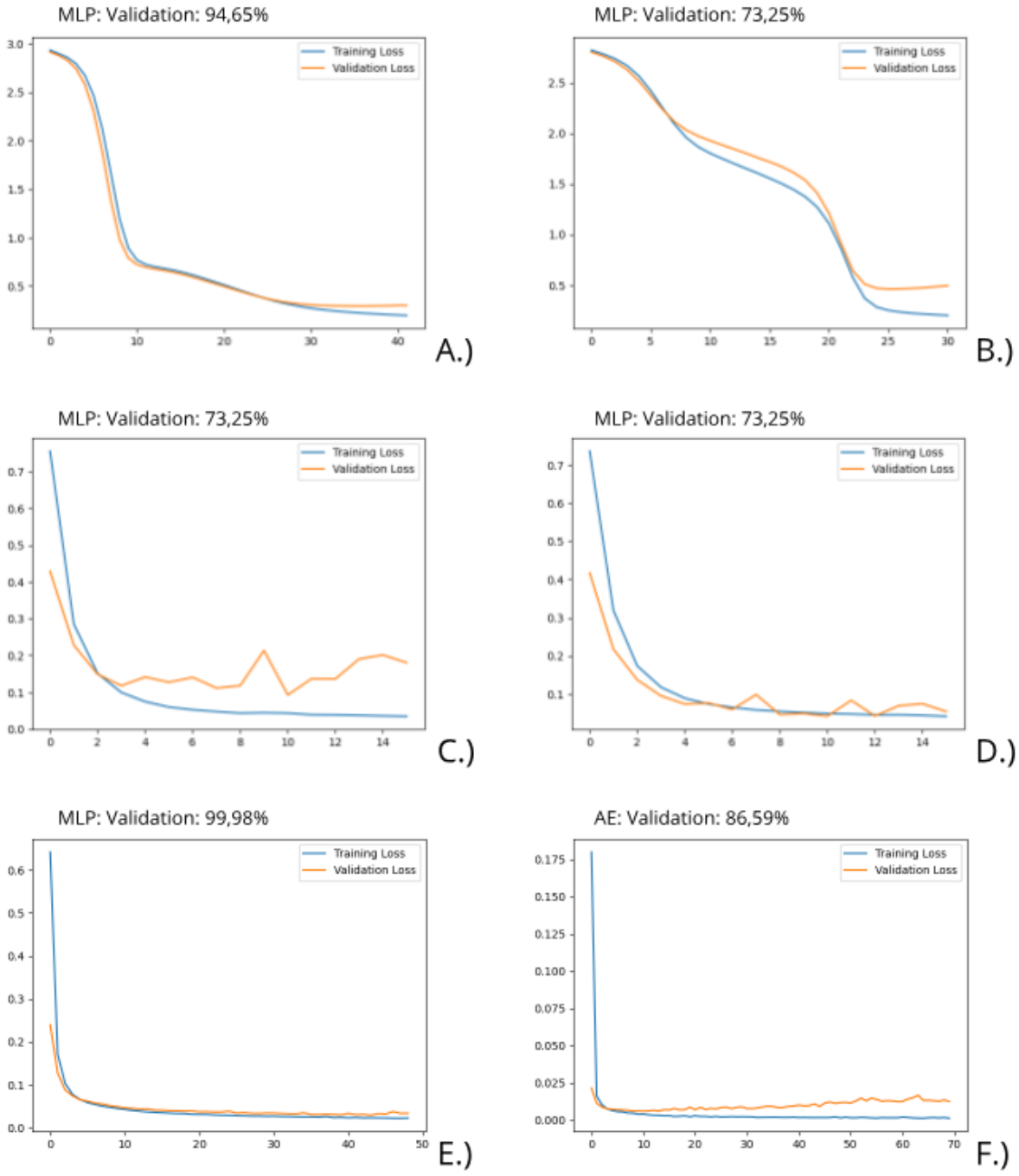
Přesnost ale nemusí být brána jen jako schopnost zachytit počítačovou hrozbu, ale také aby provoz firewallu zasahoval co nejméně do standartního provozu na koncovém uzlu. Ve standartní síťové komunikaci se u TCP/IP v literatuře udává případná náhodná ztráta paketů méně než 0,1 %. Zde se ve své finální podobě architektura Vícevrstvého perceptronu pohybovala s přesností na setiny viz. Tabulka č.4 kdy byl nejpřesněji vyhodnocen dataset KDD CUP 99 s přesností 99,98 % modelem naučeným na KDD-NSL. To by znamenalo 0,01 % případné ztráty normálních paketu, pokud budeme uvažovat obdobnou velikost jednotlivých paketů v datasetech. Průběhy s vyobrazením přesností konečných MLP a AE jsou zobrazeny na obrázku č.23



Obr. 23 Vyobrazení recall funkce pro AE a MLP, Zdroj: Autor

7.1.3 Průběh učení neuronové sítě

Průběh, rychlost a úroveň stabilizace loss křivek (Validation_loss a Training_loss) učení neuronové sítě je důležitým parametrem v oblasti zpracování datasetů, a v případě implementace byl brát v potaz hlavně z důvodů přizpůsobení se danému provozu. Na obrázku č.24 grafy A.) až F.) jsou vyobrazeny grafy s významnými rozdíly v průběhu ladění architektury vícevrstvého perceptronu. Je zde zachyceno i odlišné chování oproti autoenkodéru, a to



Obr. 24 Zobrazení významných průběhů loss křivek, Zdroj: Autor

konkrétně na grafech E.) a F.). Grafy A.) a B.) pak zachycují průběh vícevrstvého perceptronu s architekturou o pěti skrytých vrstvách, kde byly architektury náchylné již při malém počtu epoch k přeučení. Na těchto grafech byly zkoumány architektury a jejich náchylnost k objemu trénovacích a testovacích indexů dat. Ovšem pokud porovnáme grafy A.) a B.), kde byla použita architektura o 5-ti vrstvách, a grafy E.) a F.) na stejném obrázku, při kterých byla zkoumána maximální dosažená přesnost obou architektur při 3 skrytých vrstvách. Vypovídá to o zřejmé závislosti počtu skrytých vrstev na průběhu učení a je zde vidět, že nejoptimálnější byly právě tři skryté vrstvy. Ty vycházely jako možné finální řešení nejen z teoretické části, ale byly potvrzeny v této části praktické realizace. To vypovídá o minimálně správném postupu při ladění architektury. Průběhy byly po všech úpravách schopny se naučit v řádech jednotek epoch a dosahovat velmi stabilních průběhů i při podmíněném probíhání nadbytečných epoch. To vypovídá o připravenosti programu na implementace z hlediska stability chování, která je následně důležitá pro užití v systémech reálného času.

7.1.4 Rychlost zpracování datasetů

Sledování rychlosti zpracování jednotlivých datasetů bylo sledováno hlavně pro zajištění kontroly při zpracování různých datasetů a jejich indexů. Rychlost zpracování datasetů programem, ať již statistické předzpracování, nebo samotné zpracování před vložením datasetu, vypovídalo hlavně o struktuře předkládaného datasetu. Pokud by se v datasetu objevily významné anomálie hlavně z hlediska objemu indexu, byly by zachyceny již v rámci předpřípravy a projevily by se zpomalením statistického zpracování. Jelikož byly předkládané datasety strukturovány a syntetizovány ručně a byly zaváděny statickou metodou, byly časové záznamy téměř identické. Z hlediska následné implementace na koncovém uzlu v systémech reálného času se potom opět jedná o předpřípravu a referenční kontrolu připravenosti programu na případné výkyvy při zavádění reálné síťové komunikace.

8 Technické zhodnocení

Při implementaci neuronových firewallů nebude nikdy fungovat neuronová síť a její výše testovaná architektura.vícevrstvého perceptronu samostatně, ale v rámci zadání a obsáhlosti tématu neuronových firewallů se tato práce zaměřila na samotnou podstatu neuronových firewallů, a to na jejich jádro (model) neuronové sítě. Při následném vývoji je potřeba rozhodnout několik bodů, které definují výslednou výkonnost a podobu. Jedno z těchto důležitých rozhodnutí je umístění neuronového firewallu, to bylo probráno v kapitole 6.1 Umístění v síti, jelikož zásadně ovlivňuje požadované moduly a jejich kombinaci. V této fázi vývoje se již jedná o NGFW. Nejčastěji se NGFW také umisťují buď jako vstupní bod mezi vnější a vnitřní síť, nebo jako mezičlánek ve vnitřní síti. Prakticky v obou případech bude muset firewall pracovat dohromady s šifrovacími moduly na aplikačních úrovních, kdy při současných trendech v používání šifrované komunikace 70 % celkové komunikace na internetu, vypovídá o tom, že v budoucnosti bez tohoto modulu nebude moct žádný ochranný systém fungovat. Při použití formou mezičlánku také nebude potřeba vyvíjet některé ze základních funkcí stavových a paketových typů firewallů, jako například URL filtraci, či SSL VPN, ale bude soustředěna pozornost na propustnost a důraznější kontrolu na aplikační úrovni.

Další z důležitých bodů, které bude potřeba rozhodnout, je velikost požadované propustnosti a rychlost zpracování. Některé z běžně implementovaných modulů potom mohou zasahovat přímo do zavaděče dat neuronové sítě, jako například výše zmíněné šifrovací moduly, nebo přímo do samotné neuronové sítě, a bude potřeba těmto budoucím nástavbám upravit výkonnost a způsob práce neuronové sítě s předkládanými datasety. Tak, aby byl zachován požadovaný celkový výkon neuronové sítě.

Příklady dalších modulů užívaných v kombinaci s firewally NGFW:

- Ochrana před nevyžádanou poštou – antispam
- Ochrana před malware – antispyware
- Řízení aplikací – application control
- Ochrana před únikem dat – DLP
- Grafické rozhraní a moduly pro správu – reporting, user interface, online správa

8.1 Míra rozšiřitelnosti

Při vývoji neuronové sítě byla již od návrhu směřována pozornost hlavně na samotný základ Neuronového firewallu, a tím je jádro. Zbylé moduly programu se obvykle upravují posléze již pro konkrétní nasazení, a může se lišit dle použitého typu architektury. Z těchto a z důvodu obsáhlosti tématu je možné navrhnutou architekturu využít ve většině implementací. Vícevrstvý perceptron je také nejstarší a dodnes nejvíce používanou architekturou, z toho vyplývá, že míra rozšiřitelnosti navrhovaného jádra do dalších struktur neuronových firewallů je značně vysoká. Také bude potřeba zajistit funkci samoučení, která by měla syntetizovat běžnou komunikaci, vhodným způsobem provést označkování zaznamenaných indexů a ty potom předkládat neuronové síti k samostatnému doučení, popřípadě vytvořit modul pro získávání aktualizací.

8.2 Udržení přesnosti

Vzhledem v dosahovaných přesností je pouze nutné zajistit schopnost zpracování složitějších datových vstupů a upravení zavaděče dat. Zde bude hrát roli hlavně objem provozu jeho redundanci, variabilitě jednotlivých vlastností jak horizontálně (zde bude důležitou veličinu představovat střední hodnota) tak vertikálně, jako je počet a typy zastoupených vlastností. To je pravděpodobně také důvod nižších přesností při zpracování my_datasetu. Zde na druhou stranu přichází největší výhoda neuronových firewallů, a tou je do určité úrovně přizpůsobitelnost konkrétnímu datovému toku, a stačí zachytit dopředu síťovou komunikaci v dané počítačové síti. Jádro neuronového firewallu jednoduše přeučit nebo ho nechat se přeučit samotný. Ve výše navrhnutém programu pak bude potřeba jednoduše odchytit síťovou komunikaci a tu předložit neuronové síti před samotnou ostrou implementací.

8.3 Testování PoC – Proof of concept

Po fázi vývoje a distribuce NGFW by měla standardně přijít fáze testování v reálné aplikaci. Tato fáze se často zanedbává, ovšem jak byl již řečeno výše, pro počáteční implementaci bude hrát roli užitý dataset, ten ovlivní do určité míry právě funkci jádra. Ovšem abychom zajistily funkci celé aplikace jako celku bude potřeba provést určitou přípravu, potom nechat systém nasadit vyškoleného specialistu, vytvořit scénáře (zachytit datovou komunikaci, udělat soupis nejčastějších činností na síti) a ty potom před uvedením do ostrého provozu otestovat a připravit na ně neuronovou síť. Důležité je také při testování řádně otestovat všechny moduly, a to nejlépe v paralelním chodu, z důvodu ovlivnění průchodnosti datových toků.

9 Ekonomické zhodnocení

Cenové zachycení vývoje jakéhokoliv software je ekonomicky náročné, jelikož se nedá řádně určit přesná normohodina, či referenční program pro ocenění. Všeobecně se finální ceny vývoje počítačových programů mohou určovat třemi způsoby:

- **Srovnávací** – kdy se cena programu odhaduje s konkurenčním řešením na trhu. Zde lze usoudit, že cena implementace neuronových firewallů přesahuje cenu standartních firewallových řešení minimálně o řády desítek tisíc korun. Příkladem může sloužit porovnání HW obdobných řešení od společnosti Cisco viz. Tabulka č.5

Cisco MX64-HW	**Cisco FPR1120-NGFW-K9**
klientů: 50	klientů: 75
propustnost VPN: 100 Mbps	propustnost NGFW: 650 Mbps
8 057 Kč	33 880 Kč

Tab. 5 Porovnání cenových hladin HW Firewallů, Zdroj (22), (23) upraveno Autorem

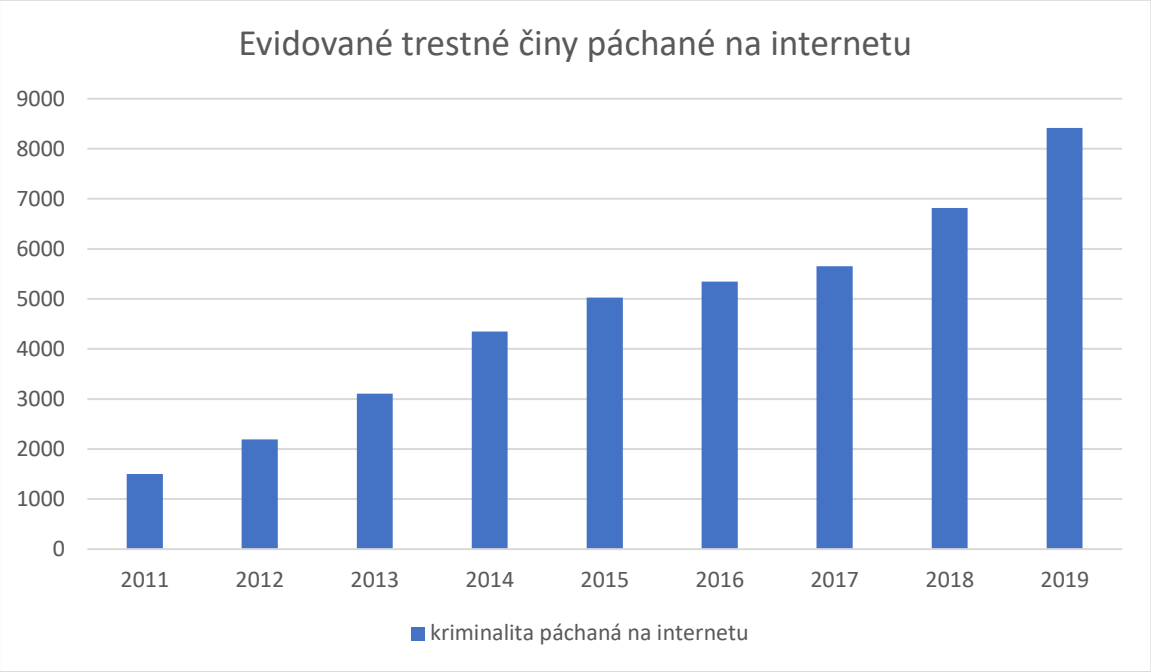
- **Výnosový** – způsob, kdy se hodnota programu určuje na základě možné ztráty, či zabránění ztrátě ekonomického aktiva (peněžní nebo nehmotné vlastnictví). Tímto způsobem to nelze přímo určit vzhledem k rozličné povaze právě nehmotných aktiv a všeobecné povaze zpracování praktické části.

- **Nákladový** – v rámci nákladového oceňování se předpokládá, že zájemce prokazuje chování dobrého hospodáře a nezaplatí na program více než by musel. Jedná se tedy o metodu, kdy se sečtou nákladové položky a jednotlivě se ocení. Výsledná suma je pak brána jako nákladová cena, ke které se připočítá marže firmy. V průběhu zpracování práce byl zaznamenán potřebný čas ke zhotovení finálního programu. Obvyklá hodinová cena ICT specialisty se pohybuje kolem 1200 Kč a programátora 1100 Kč

	Popis nákladu	**Počet hodin**	**Celková cena**
1	Analýza softwarových požadavků	7 hodin, 48 min.	8 525 Kč
2	Návrh softwaru a jeho modulů	25 hodin, 32min.	28 050 Kč
3	Programování jádra	38 hodin 15 min.	42 075 Kč
4	Implementace potřebných modulů	23hodin, 20min.	25 597 Kč
5	Testování	64 hodin, 10min.	79 432 Kč

Tab. 6 Procentuální zastoupení stráveného času

Pořízení moderních typů firewallů může být od 30 000,- Kč, až do řádů milionů korun. Záleží na typu provedení a parametrech požadované implementace. Nejedná se tedy o levnou ani zanedbatelnou položku v rámci firemních financí. Z grafu č.1 je ale patrné, že počet trestních šetření kriminality páchané na internetu rok od roku stoupá, tím nepřímo nabývá na cenně a významu technologie neuronových firewallů a jejich schopnost učit se samostatně novým útokům, případně se přizpůsobovat síťové komunikaci na kterých jsou provozovány.



Graf 1 Evidované trestné činy páchané na internetu, Zdroj: (24)

Obvykle se pak v praxi řeší počítačová bezpečnost a ochrana počítačových sítí jako doplňkový náklad pro firmy při ukončování fiskálního roku. Tak aby firmy vyčerpali svoje rozpočty, to potom předesílá, i jak se následně přistupuje k jejímu výběru.

9.1 Ekonomický rozbor při implementaci

Z ekonomického hlediska se program může posuzovat podle toho, jak je schopný usnadnit či se přizpůsobit k používání v daném nasazení. Vzhledem k tomu, že se jedná o jednu ze základních předností neuronových firewallů, Je zde na vyšší komfortabilitu pro správce počítačové sítě i koncového uživatele. Jelikož je uvedené řešení relativně rychlé za předpokladu použití na koncovém uzlu, tak nebude ani rušit uživatele instalacemi aktualizací. Úspora neprovedenými aktualizacemi, odstávka systémů, některých jeho uzlů a částí, sebou nesou také náklady z nefunkčnosti systému, či koncového uzlu, a tudíž z ušlého zisku. V tomto případě je

navrhované řešení možné naprogramovat pro plynulý chod učení a aktualizace nebudou alespoň z důvodu neuronové sítě potřebné.

Vývoj šedé ekonomiky roste stejně rychle jako provázanost všedního dne s počítači. Graf č.1 ukazuje vývoj trestných činů jen na území České republiky, a je zde jasně vidět jaký trend bude mít do budoucna šedá ekonomika počítačových sítí a jejich bezpečnosti.

Možné ekonomické ztráty se potom špatně vyčíslují, ztráty při penetraci obranného systému mohou mít nezřídka likvidační povahu pro podniky. Většinou jsou na serverech uchovávána totiž mnohem cennější aktiva, než jsou nominální hodnoty samotných serverů, a vzhledem k rozličnosti možného uchovávaného nehmotného majetku je téměř nemožné zachytit tento aspekt ekonomického zhodnocení.

I když se ekonomické aspekty počítačové bezpečnosti těžko zachycují, lze je popsat z pohledu nechtěných stavů:

- Ztráta integrity sítě
- Ztráta důvěryhodnosti počítačové sítě.
- Ztráta propustnosti počítačové sítě.

9.2 SWOT analýza a porovnání s ostatními typy firewallů

	Příležitosti	**Hrozby**
Vnitřní původ	Adaptabilita k síťovému provozu Schopnost samoučení Strengths Relativní rychlost	Stabilita předkládaných datasetů Výrazná změna v objemu indexů Weaknesses Falešné určení
Vnější původ	Informační zdroje o MLP architektuře Vývoj kybernetické bezpečnosti Opportunities Současný trend	Uživatel Znalost MLP architektury Threats Možnost naučit model na chybu

Tab. 7 SWOT analýza neuronových firewallů, Zdroj: Autor

10 Závěr

V rámci praktické části byl navržen postup pro realizaci a ověření funkčnosti neuronového firewallu v základním nejjednodušším provedení, a to neuronovou sítí se statickým zavaděčem datasetů. Neuronová síť byla vytvořena v „home office“ podmínkách, a práce tím prokázala, že k vývoji a následné implementaci neuronových firewallů, či konkrétně k práci s rozebíranými neuronovými firewally, není zapotřebí žádných větších investic nežli investic časových a mzdových. Technologie použité v rámci zpracování zadání byly všechny s open source licencí, a práce tím prokázala, že není potřeba investovat vysoké prostředky pro realizaci středně složitých neuronových sítí.

Na základě testování v praktické části práce vícevrstvého perceptronu a autoenkodéru byl ověřen a potvrzen závěr z teoretické části této práce, a to že optimální struktura pro většinu jednodušších až středně složitých implementací je vícevrstvý perceptron s třemi skrytými vrstvami. Nejlepších parametrů dosahovala architektura vícevrstvého perceptronu při třech skrytých vrstvách, v použití s ReLU aktivační funkcí, softmax funkcí a 20-ti neurony v každé vrstvě.

V průběhu zpracování práce se také neúmyslně prokázala potřeba řešit měření a kontrolu výkonnosti neuronových sítí složitějšími metrikami než jen s klasickou procentuální přesností a empirickým zkoumáním loss křivek. Přestože v předběžném zpracování se zdála architektura autoenkodéru jako lepší, je po aplikaci recall přesnosti zřejmé, že se začala posuzovat nejenom obyčejná přesnost vyvedená z podílu správně označkovaných indexů, ale také struktura indexů, které byly správně označkovány. Zde získala práce přínos díky pokročilým metodám evaluace neuronových sítí, jako jsou například metody nejmenších čtverců (MSE – Mean Squared Error), a použitím recall přesnosti, úplně nový pohled na výkonnost. A ve finále vyšla nejlépe právě architektura vícevrstvého perceptronu.

Výsledkem práce je tedy doporučení využit vícevrstvý perceptron tak, jak je uvedeno v příloze č.2 , a k implementaci do neuronových firewallů. Navržený model je připravený k implementaci v reálném provozu a navázání na ostatní moduly Next Generation Firewallů.

11 Použitá literatura

1. NOVÁK, Mirko. *Umělé neuronové sítě: teorie a aplikace.* V Praze : C.H. Beck, 1998.

2. České vysoké učení technické v Praze. Firewall. *Komponenty síťového bezpečnostního systému.* [Online] České vysoké učení technické v Praze, 2021. [Citace: 21. 5 2021.] http://techpedia.fel.cvut.cz/html/frame.php?oid=76&pid=1018&finf=.

3. Sophos Ltd. Endpoint Protection Tech Specs. *Sophos.* [Online] Sophos Ltd. [Citace: 1. 3 2021.] https://www.sophos.com/en-us/products/endpoint-antivirus/tech-specs.aspx.

4. upGrad. Neural Network: Architecture, Components & Top Algorithms. *upGradblod.* [Online] upGrad Education Private Limited., 6. 5 2020. [Citace: 11. 4 2021.] https://www.upgrad.com/blog/neural-network-architecture-components-algorithms/.

5. Feng, Yirui Wu | Dabao Wei | Jun. Network Attacks Detection Methods Based on Deep Learning. *Security and Communication Networks.* https://doi.org/10.1155/2020/8872923, 2020, Sv. Volume 2020, 8872923.

6. PANG, GUANSONG, a další. Deep Learning for Anomaly Detection: A Review. *ACM Comput. .* 2020, Sv. Surv. 1, Article 1.

7. Ertoz, L., a další. TheMINDS - Minnesota intrusion detection system. *CiteSeerx.* [Online] 2004. [Citace: 9. 2 2021.] https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.138.3245&rep=rep1&type=pdf.

8. *Anomaly detection.* CHANDOLA, Varun, BANERJEE, Arindam a KUMAR, Vipin. 41(3), místo neznámé : ACM Computing Surveys, 2009, Sv. 3. 10.1145/1541880.1541882..

9. *Real-time Analysis of Flow Data for Network Attack Detection.* Carle, Georg a Munz, Gerhard. místo neznámé : IEEE Xplore, 2007. 10.1109/INM.2007.374774.

10. Park, Jung-Min a Patcha, Animesh. An overview of anomaly detection techniques: Existing solutions and latest technological trends. *Computer Networks.* 2007, Sv. Volume 51, 12.

11. G., HINTON, a další. The wake-sleep algorithm for unsupervised neural network. *Dept. of Computer Science.* [Online] http://www.gatsby.ucl.ac.uk/ dayan/papers/hdfn95.pdf.

12. *"Benchmarking datasets for Anomaly-based Network Intrusion Detection: KDD CUP 99 alternatives," 2018 IEEE 3rd International Conference on Computing.* Divekar, Abhishek, a další. místo neznámé : Communication and Security (ICCCS), 2018 IEEE 3rd International Conference on Computing. 0.1109/CCCS.2018.8586840.

13. Mangrulkar, Nikhil S., Patil, Arvind R. Bhagat a Pande, Abhijit S. Network Attacks and Their Detection Mechanisms: A Review. *International Journal of Computer Applications.* 0975 – 8887, 2014, Sv. Volume 90, 9.

14. Classification: ROC Curve and AUC. *Machine Learning Crash Course.* [Online] Google.com, 10. 02 2020. [Citace: 1. 5 2021.] https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc.

15. How to interpret loss and accuracy for a machine learning model. *Stackoverflow.* [Online] 02 2016. [Citace: 29. 04 2021.] https://stackoverflow.com/questions/34518656/how-to-interpret-loss-and-accuracy-for-a-machine-learning-model.

16. Vojtěch Dostál. Neuron. *Wikipedie.* [Online] Wikimedia foundation. [Citace: 26. 1 2021.] https://cs.wikipedia.org/wiki/Neuron.

17. *Towards Principled Methods for Training Generative Adversarial Networks.* Arjovsky, M. a Bottou, L. místo neznámé : 5th International Conference on Learning Representations, 2017.

18. lopez, dProgrammer. RNN, LSTM & GRU. *dProgrammer lopez.* [Online] 6. 5 2019. [Citace: 15. 3 2021.] http://dprogrammer.org/rnn-lstm-gru.

19. Multi-Layered-Perceptron Models on MNIST Dataset : Say No to Overfitting! *Sayed Athar.* [Online] 1. 9 2018. [Citace: 1. 15 2021.] https://medium.com/@sayedathar11/multi-layered-perceptron-models-on-mnist-dataset-say-no-to-overfitting-3efa128a019c.

20. (Kyle), Korkrid Akepanidtaworn. Best Place to Learn Neural Network: Interactive Tensorflow Playground. *Korkrid Akepanidtaworn (Kyle).* [Online] https://kyleake.medium.com, 15. 1 2019. [Citace: 6. 1 2021.] https://kyleake.medium.com/technical-demo-visualize-neural-network-with-tensorflow-playground-9f6a1d8eb57a.

21. Irvine. KDD Cup 1999 Data. *KDD Cup 1999 Data.* [Online] University of California, Irvine, srpen. 28 1999. https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.

22. Router-switch Ltd. FPR1010-NGFW-K9. *Router-switch.com.* [Online] Router-switch Ltd. [Citace: 8. 4 2021.] https://www.router-switch.com/fpr1010-ngfw-k9.html.

23. Bechtle ag. Firewall Cisco FPR1120-NGFW-K9. *Bechtle.* [Online] Bechtle ag. [Citace: 2. 5 2021.] https://www.bechtle.com/be-cs/shop/firewall-cisco-fpr1120-ngfw-k9--4369759-73-p.

24. Policie ČR. Kyberkriminalita. *Policie ČR.* [Online] Policie ČR. [Citace: 12. 5 2021.] https://www.policie.cz/clanek/kyberkriminalita.aspx.

12 Přílohy

Příloha 1 ……. Základní vlastnosti indexů v datasetech

Příloha 2 ……. Program Pack&go_v3.3

Základní vlastnosti indexů v datasetech

	Vlastnost indexu	Popis vlastnosti	Datový typ
1	duration	Délka fungujícího spojení v sekundách	Spojité
2	protocol_type	Typ používaného protokolu (FTP,ICMP,..)	Diskrétní
3	service	Typ používané služby (HTTP, FTP..,)	Diskrétní
4	src_bytes	Počet doručených byte	Spojité
5	dst_bytes	Počet odeslaných byte	Spojité
6	flag	Typ ukončení spojení	Diskrétní
7	land	Duplikace spojení z host IP i do host IP	Diskrétní
8	wrong_fragment	Počet poškozených fragmentů	Spojité
9	urgent	Počet urgujících paketů	Spojité
10	hot	Počet „hot“ indikátorů	Spojité
11	num_failed_logins	Počet neúspěšných pokusů o přihlášení	Spojité
12	logged_in	Počet přihlášení	Diskrétní
13	num_compromised	Počet kompromitujících stavů	Spojité
14	root_shell	Úspěšné přihlášení na root	Diskrétní
15	su_attempted	Počet pokusů o využití su root	Diskrétní
16	num_root	Počet úspěšných přihlášení na root	Spojité
17	num_file_creations	Počet vytvořených složek	Spojité
18	num_shells	Počet otevřených terminálů.	Spojité
19	num_access_files	Počet operací se složkami	Spojité
20	num_outbound_cmds	Počet odcházejících příkazů	Spojité

Program Pack&go_v3.3

```
# Vytvoření poměrných hodnot (z-score) a dummy proměnných
def encode_numeric_zscore(df, name, mean=None, sd=None):
    if mean is None:
        mean = df[name].mean()

    if sd is None:
        sd = df[name].std()

    df[name] = (df[name] - mean) / sd

def encode_text_dummy(df, name):
    dummies = pd.get_dummies(df[name])
    for x in dummies.columns:
        dummy_name = f"{name}-{x}"
        df[dummy_name] = dummies[x]
    df.drop(name, axis=1, inplace=True)

start_3 = time.process_time()
encode_numeric_zscore(df, 'duration')
encode_text_dummy(df, 'protocol_type')
encode_text_dummy(df, 'service')
encode_text_dummy(df, 'flag')
encode_numeric_zscore(df, 'src_bytes')
encode_numeric_zscore(df, 'dst_bytes')
encode_text_dummy(df, 'land')
encode_numeric_zscore(df, 'wrong_fragment')
encode_numeric_zscore(df, 'urgent')
encode_numeric_zscore(df, 'hot')
encode_numeric_zscore(df, 'num_failed_logins')
encode_text_dummy(df, 'logged_in')
encode_numeric_zscore(df, 'num_compromised')
encode_numeric_zscore(df, 'root_shell')
encode_numeric_zscore(df, 'su_attempted')
```

```
encode_numeric_zscore(df, 'num_root')
encode_numeric_zscore(df, 'num_file_creations')
encode_numeric_zscore(df, 'num_shells')
encode_numeric_zscore(df, 'num_access_files')
encode_numeric_zscore(df, 'num_outbound_cmds')
encode_text_dummy(df, 'is_host_login')
encode_text_dummy(df, 'is_guest_login')
encode_numeric_zscore(df, 'count')
encode_numeric_zscore(df, 'srv_count')
encode_numeric_zscore(df, 'serror_rate')
encode_numeric_zscore(df, 'srv_serror_rate')
encode_numeric_zscore(df, 'rerror_rate')
encode_numeric_zscore(df, 'srv_rerror_rate')
encode_numeric_zscore(df, 'same_srv_rate')
encode_numeric_zscore(df, 'diff_srv_rate')
encode_numeric_zscore(df, 'srv_diff_host_rate')
encode_numeric_zscore(df, 'dst_host_count')
encode_numeric_zscore(df, 'dst_host_srv_count')
encode_numeric_zscore(df, 'dst_host_same_srv_rate')
encode_numeric_zscore(df, 'dst_host_diff_srv_rate')
encode_numeric_zscore(df, 'dst_host_same_src_port_rate')
encode_numeric_zscore(df, 'dst_host_srv_diff_host_rate')
encode_numeric_zscore(df, 'dst_host_serror_rate')
encode_numeric_zscore(df, 'dst_host_srv_serror_rate')
encode_numeric_zscore(df, 'dst_host_rerror_rate')
encode_numeric_zscore(df, 'dst_host_srv_rerror_rate')

df.dropna(inplace=True,axis=1)
input("Data zpracovaná, mám pokračovat?")
cas_1 = time.process_time() - start_3
file1 = open(umisteni + "\Test" + test + "\zaznam\\" + "score_" + test + ".txt","a")
file1.write(r"čas statistického zpracování datasetu:" + str(cas_1) + " \n")
file1.close()
```

```
start = time.process_time()
x_columns = df.columns.drop('outcome')
x = df[x_columns].values

dummies = pd.get_dummies(df['outcome'])
outcomes = dummies.columns
num_classes = len(outcomes)
y = dummies.values

df.reset_index(inplace=True, drop=True)
df.groupby('outcome')['outcome'].count()
df.reset_index().to_csv(umisteni + "\Test" + test + "\zaznam\\" + "data_set_po_zpracovani_"+
test + ".csv", index=False, header=True, decimal=",",sep = ";", float_format=' %.3f' )

cas_3 = time.process_time() - start
file1 = open(umisteni + "\Test" + test + "\zaznam\\" + "score_" + test + ".txt","a")
file1.write(r"čas spotřebný na přípravu dat:" + str(cas_3) + " \n")
file1.close()

# Neuronový model

print("Data jsou připravena k práci")
print("1 = Vytvořit nový model")
print("2 = Načíst již vytvořený model")
typ_prace = input("Vytvořit nový model nebo načíst již hotový?")

if typ_prace == "1":

    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25, random_state=42)
    print(len(x_test))

    start_2 = time.process_time()

    model = Sequential()
    model.add(Dense(20, activation='relu'))
```

```
    model.add(Dense(20, activation='relu'))
    model.add(Dense(20, activation='relu'))
    model.add(Dense(y.shape[1],activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='adam')
    model.compile(loss='hinge', optimizer="adam", metrics=['val_accuracy'])
    model.compile(loss='mean_squared_logarithmic_error', optimizer="adam", metrics=['mse'])
    model.compile(loss='hinge', optimizer="adam", metrics=['Recall'])
    monitor = EarlyStopping(monitor='val_loss', min_delta=1e-3, patience=5, verbose=1, mode='auto', restore_best_weights=True)

    history = model.fit(x_train,y_train,validation_data=(x_test,y_test), verbose=2,epochs=40)

    cas_2 = time.process_time() - start_2
    file1 = open(umisteni + "\Test" + test + "\zaznam\\" + "score_" + test + ".txt","a")
    file1.write(r"čas naučení neuronové sítě:" + str(cas_2) + " \n")
    file1.close()

    np.savetxt(umisteni + "\Test" + test + "\zaznam\\" + "data_x.csv", x, delimiter=",")
    np.savetxt(umisteni + "\Test" + test + "\zaznam\\" + "data_y.csv", y, delimiter=",")
    model.save_weights(umisteni + "\Test" + test + "\zaznam\weights" + "\weights" + test)
    model.save(umisteni + "\Test" + test + "\zaznam" + "\model" + test)

if typ_prace == "2":

    model = create_model()

    model_load = input("cesta k načtení modelu")
    model.load_weights(model_load)
    model.predict(x)

else:
    print("nerozumím")
```

```
# Zobrazení a zápis výstupu z modelu

nazev = str(input("vlož název grafu: "))
slozka = umisteni + "\Test" + test

plt.plot(history.history["loss"], label="Training Loss")
plt.plot(history.history["val_loss"], label="Validation Loss")
plt.legend()
plt.savefig(r'{}\{}{}'.format(slozka, nazev + test, ".png"))
plt.show()

# Zobrazení výstupu z modelu – výběr hodnoty

markerline, stemlines, baseline = plt.stem(df.index, df["duration"], linefmt='-', markerfmt='none', bottom=1.1)
markerline.set_markerfacecolor('none')
graf_2 = "duration_parametr"
plt.savefig(r'{}\{}{}'.format(slozka, graf_2 + test, "_loss_fce.png"))
plt.show()

# zobrazení výstupu a spočítání přenosti model.

pred = model.predict(x_test)
model.summary()

np.savetxt(umisteni + "\Test" + test + "\zaznam\\" + "data_x_test.csv", x_test, delimiter=",")
np.savetxt(umisteni + "\Test" + test + "\zaznam\\" + "data_predikce.csv", pred, delimiter=",")
pred = np.argmax(pred,axis=1)
np.savetxt(umisteni + "\Test" + test + "\zaznam\\" + "data_y_pokud2.csv", pred, delimiter=",")
y_eval = np.argmax(y_test,axis=1)
np.savetxt(umisteni + "\Test" + test + "\zaznam\\" + "data_y_pokud2.csv", y_eval, delimiter=",")
score = metrics.accuracy_score(y_eval, pred)
print("zkouska" + str(score))
```

```
plt.subplot(212)
plt.title('Recall')
plt.plot(history.history['Recall'], label='train')
#plt.plot(history.history['val_accuracy'], label='test')
graf_3 = "accurancy_"
plt.savefig(r'{}\{}{}'.format(slozka, graf_3 + test, "zkouska.png"))
plt.legend()
plt.show()

file1 = open(umisteni + "\Test" + test + "\zaznam\\" + "score_" + test + ".txt","a")
file1.write("Přesnost: ")
file1.write(str(score))
file1.close()
```