

**REPUBLIC OF TURKEY
AKDENİZ UNIVERSITY**



**EXPANDING PASSWORD DICTIONARIES BY GENERATING NEW
PROBABLE PASSWORDS USING MACHINE LEARNING TECHNIQUES**

Mehmet Görkem KESTANE

INSTITUTE OF NATURAL AND APPLIED SCIENCES

DEPARTMENT OF COMPUTER ENGINEERING

MASTER THESIS

JANUARY 2024

ANTALYA

**REPUBLIC OF TURKEY
AKDENİZ UNIVERSITY**



**EXPANDING PASSWORD DICTIONARIES BY GENERATING NEW
PROBABLE PASSWORDS USING MACHINE LEARNING TECHNIQUES**

Mehmet Görkem KESTANE

INSTITUTE OF NATURAL AND APPLIED SCIENCES

DEPARTMENT OF COMPUTER ENGINEERING

MASTER THESIS

JANUARY 2024

ANTALYA

REPUBLIC OF TURKEY
AKDENİZ UNIVERSITY
INSTITUTE OF NATURAL AND APPLIED SCIENCES

**EXPANDING PASSWORD DICTIONARIES BY GENERATING NEW
PROBABLE PASSWORDS USING MACHINE LEARNING TECHNIQUES**

Mehmet Görkem KESTANE

DEPARTMENT OF COMPUTER ENGINEERING

MASTER THESIS

This thesis unanimously accepted by the jury on 06/02/2024.

Asst. Prof. Dr. Murat AK (Supervisor)

Prof. Dr. Melih GÜNAY

Asst. Prof. Dr. Naci ER

ÖZET

MAKİNE ÖĞRENMESİ TEKNİKLERİ İLE YENİ OLASI ŞİFRELER ÜRETEREK ŞİFRE SÖZLÜKLERİNİN GENİŞLETİLMESİ

Mehmet Görkem KESTANE

Yüksek Lisans Tezi, Bilgisayar Mühendisliği Anabilim Dalı

Danışman: Dr. Öğr. Üyesi Murat AK

Ocak 2024; 136 Sayfa

Çevrimiçi hesaplar, kullanıcı adları ve farklı parola kombinasyonları kullanılarak korunmaktadır. Parolalar kullanıcıların hesaplarına ve bu hesaplarda bulunan banka hesap bilgileri, satın alma geçmişi, ev adresi gibi kişisel bilgilerine yapılan yetkisiz erişimlere karşı koruma sağlamaktadır. Şifre ne kadar güçlü oluşturulursa, kullanıcıların hesapları bilgisayar korsanı hacker’lardan ve kötü amaçlı yazılımlardan o kadar iyi derecede korunacaktır. Bu sebeple, sistem ve ağ güvenliğini sağlamak için, kullanıcıların sistemlerindeki tüm hesaplar için güçlü ve sağlam şifreler bulundurması gerekmektedir. Zayıf bir parola; sözlükteki sözcükler, özel adlar ve kullanıcı adlarından türetilen sözcükler de dahil olmak üzere tüm olası parolaların bir alt kümesini kullanan kaba kuvvet saldırısı gerçekleştirilerek tahmin edilmesi kolay olan paroladır. Ayrıca zayıf ve kolay kırılabilir bir parola, kısa, yaygın kullanılan veya sistem varsayılanı şifrelerdir. Zayıf şifrelerin zaafiyet oluşturduğu göz önünde bulundurulduğunda, yapılan hack saldırılarında zayıf şifreler önemli bir rol oynamaktadır. Bazı uygulamalar kullanıcılara kolaylık sağlamak için, kullanıcılarına her zaman güçlü şifre oluşturma ve kullandırma zorunluluğu şart koşturmamaktadır. Bu durum, kullanıcıların password, password123, Password@123, 123456, telefon numaralarından oluşan parolalar gibi zayıf parolalar oluşturmalarında önemli bir etki teşkil etmektedir. Bir parola yeterli uzunlukta olup, özel karakter içerse bile, kolay tahmin edilebilen zayıf bir parola olabilmektedir. Örneğin Gorkem@12345 parolası karmaşık bir şifre gibi görünse de tahmin edilebilmektedir. Zayıf şifreler arasında kullanıcının adını, adresini veya telefon numarasını içeren şifreler bulunmaktadır. Bu tarz bilgiler ve pattern’lar içeren şifreler human-like parolalardır. Kullanıcılar şifreleri oluştururken özel karakter içeren rastgele string’ler kullanılmalıdır ve oluşturulan bu şifrelerin

uzunlukları çok kısa olmamalıdır. Çünkü zayıf şifreler saldırganlar tarafından kaba kuvvet saldırılarıyla kırılabilir veya parolaların uzunlukları çok kısaysa tahmin edilebilir olabilmektedir. Hatırlaması zor olsa da, rastgele string veya özel karakterler içeren ve belli bir uzunluktan daha kısa olmayan parolalar güvenlik açısından oldukça güvenlidir. Bir şifre günlük hayattan ifadeler içeriyorsa zayıf bir şifre olarak nitelendirilmektedir. Bunun nedeni çoğunlukla tahmin edilebilir olmaları ve sözlük saldırılarıyla saldırıya uğrayabilir ve ele geçirilebilir olmalarından kaynaklanmaktadır. Parola sözlüğü, olası parolaları ihtiva etmekte olan bir listedir. Saldırgan, ele geçirilmiş bir şifre veri tabanından, bir şifrenin hash'ini elde ettiği zaman, bir parola sözlüğünde muhafaza edilen tüm şifrelerin hash'ini, elde edilen hash değeriyle karşılaştırabilmektedir ve bir eşleşme bulunursa hash değerine karşılık gelen şifreyi bulabilmektedir. Bu durum, sözlük saldırısı olarak adlandırılmaktadır. Sızdırılan şifreler analiz edilip incelendikten sonra en sık kullanılan şifrelerin basit sözlük kelimeleri olduğu, şifre hash'lerini kırmanın en yaygın yönteminin ise sözlük saldırıları olduğu görülmüştür. İnsanlar hatırlanması kolay şifreler oluşturma eğilimindedirler, bu nedenle genellikle şifre sözlüklerinde bulunan, günlük hayatta kullanılan yaygın ifadeleri seçmektedirler. Kelime Listeleri olarak da bilinen parola sözlükleri, sözlük saldırısı gerçekleştirerek şifreleri kırmak için kullanılmaktadır ve sızdırılmış veri tabanlarından alınan gerçek şifre dökümlerinden oluşmaktadır. Rockyou.txt bu kelime listelerinden bir tanesidir. Diğerleri ise milyonlarca şifreden oluşan daha büyük yığınlardan alınmış ve en sık tekrarlanan öğelerle oluşturulmuştur. SecLists, WordList-Compendium, Kaonashi, Crackstation Wordlist Şifre Kırma Sözlüğü, RockYou ve Top-WPA-Probable sözlük saldırısı gerçekleştirmek amacıyla oluşturulmuş olan parola sözlüklerinden bazılarıdır. Sözlük saldırısı gerçekleştirirken, saldırganın kırmak istediği şifrenin, saldırı yapmak için kullanmış olduğu parola sözlüğünde bulunan muhtemel şifrelerden biri olma ihtimali ne kadar yüksek olursa, atak yapılan şifrenin kırılma olasılığı da o kadar artmaktadır. Bu tez çalışmasında sözlük saldırıları yapılırken bu kelime listelerini kullanmak ve sözlük saldırısıyla şifre kırma işlemini bir sonraki aşamaya taşımak için makine-benzeri oluşturulmuş veya insan-benzeri oluşturulmuş gibi sınıflandırılmış parola sözlükleri oluşturmaya çalışılmıştır. Zenginleştirilmiş parola sözlükleri, makine öğrenmesi ve derin öğrenme yöntemlerinden LSTM (RNN) kullanılarak oluşturulmuştur. Bu projede, makine öğrenimi teknikleri kullanılarak mevcut parola sözlüğü listelerinin genişletilmesi, mev-

cut şifre cümlelerinin diğer permütasyonlarla birleştirilmesi ve gerçek bir insan kullanıcının oluşturabileceği tarzda daha fazla potansiyel şifre geliştirilmesi üzerine yoğunlaşmıştır. Üretilen şifrenin makine benzeri mi yoksa kullanıcı benzeri mi üretildiğine göre sınıflandırma işlemi yapıldıktan sonra kullanıcının oluşturabileceği tarzda şifreler oluşturulmakta ve yeni parola sözlük listeleri oluşturulmaktadır. İnsanların hatırlanması kolay şifreler oluşturma eğiliminde olmaları ve günlük hayatta kullanılan ortak ifadeleri seçmeleri nedeninden yola çıkılarak, insan-benzeri-üretilmiş parolalar oluşturularak ve bunlar kullanılarak zenginleştirilmiş parola sözlükleri oluşturulmuş ve bu insan kullanıcıların oluşturabileceği tarzda parolalarla oluşturulmuş sözlükler kullanılarak daha güçlü saldırılar gerçekleştirilmesinin mümkün olup olmadığı araştırılmıştır. Tezin amacı, sözlük saldırılarının kapasitesini arttırmak amacıyla gerçek insan kullanıcıların gelecekte oluşturabileceği potansiyel mantıksal şifreleri üretip zenginleştirilmiş sözlükler oluşturmaktır. Bu çalışmada parola sözlüğü saldırılarına ek güç vererek, sözlük saldırılarının aslında önceden inanıldığından daha etkili olduğunu göstermek mümkün olacaktır. Makine öğrenimi literatüründe amacımıza uygun yaygın bir yöntem RNN(LSTM)'dir. Bu nedenle Makine Öğrenmesi tekniklerinden biri olarak Yinelemeli Sinir Ağları (RNN) kullanılmaktadır. Bu çalışmada Yinelemeli Sinir Ağı (RNN) mimarisi olarak Uzun Kısa-Süreli Bellek (LSTM) kullanılmıştır. Ayrıca bu tezde şu soru ele alınmaktadır: Makine öğrenmesi tekniklerini kullanarak, mevcut şifreleri analiz edip, şifre sınıflandırma işleminden sonra yenilerini ekleyerek bir şifre sözlüğünü zenginleştirebilir miyiz? Şifre sözlüklerinin bu şekilde güçlendirilmesiyle sözlük saldırılarının aslında sanıldığından daha etkili ve efektif hale getirilmesi mümkün olacaktır.

ANAHTAR KELİMELELER: Derin Öğrenme, Doğal Dil İşleme, Makine Öğrenmesi, Parola Sözlükleri Oluşturma, Parola Tahmin Etme, Siber Güvenlik, Sözlük Saldırısı, Uzun Kısa-Süreli Bellek, Yapay Zeka, Yinelemeli Sinir Ağları

JÜRİ: Dr. Murat AK

Prof. Dr. Melih GÜNEY

Dr. Naci ER

ABSTRACT

EXPANDING PASSWORD DICTIONARIES BY GENERATING NEW PROBABLE PASSWORDS USING MACHINE LEARNING TECHNIQUES

Mehmet Gökem KESTANE

MSc Thesis in Computer Engineering

Supervisor: Asst. Prof. Dr. Murat AK

January 2024; 136 pages

By using username and password combinations, online accounts are protected. Passwords provide protection against unauthorized access to a user's accounts and personal information, such as bank account information, purchase history, or home address. The stronger the password, the more protected account of users will be from hackers and malicious software. For this reason, users should maintain solid passwords for all accounts on their systems. A weak password is one that is easy to guess by performing a brute force attack utilizing a subset of all possible passwords, including words in the dictionary, proper names, and words derived from user names. In addition, they are short, common, or system default passwords. Any hack has a significant role played by weak passwords. Applications may not always require strong passwords for the ease of the user, which leads to users using weak passwords like password, password123, Password@123, 123456, their own mobile number, etc. A weak password might also be one that is easy to guess and doesn't always refer to length and character usage. For example, Gorkem@12345 appears to be a complicated password, but it is predictable. Weak passwords include those that contain the user's name, address, or phone number. While creating the passwords, users should use random strings with special characters and their length shouldn't be too short because weak passwords can be brute-forced by attackers or be predictable if their length is too short. Although it may be challenging to remember, that is actually quite secure from a security standpoint. If a password contains daily life phrases it is called a weak password. This is due to the fact that they are mostly predictable and can be attacked by dictionary attacks. A password dictionary is a list of possible passwords. When an attacker obtains the hash of a password from a compromised password database, they can

compare the hash of all the passwords in a dictionary to an obtained hash value and if a match is found, the password that corresponds to the hash value is found. This is called a dictionary attack. After leaked passwords were analyzed, it was found that the most commonly used passwords were simple dictionary words, and the most common method to crack password hashes was dictionary attacks. People tend to create passwords that are easy to remember, so they choose common phrases that are used in daily life which usually exist in password dictionaries. The password dictionaries, also known as Wordlists are used to crack passwords and they consist of real password dumps from leaked databases, such as rockyou.txt. Others are taken from larger dumps of millions of passwords and resulted in the most commonly reoccurring items. Here are some of the more important wordlists for generic password cracking, such as SecLists, WordList-Compendium, Kanashi, Crackstation Wordlist Password Cracking Dictionary, RockYou and Top-WPA-Probable. When performing a dictionary attack, the more and more likely the existing passwords in the wordlist used by the attacker are, the more likely the hacked password is to be cracked. In this thesis work, we tried to build classified wordlists such as machine-like-generated or human-like-generated in order to use those wordlists while dictionary attacks are performed and take the password-cracking to the next level. Enriched wordlists are created using machine learning and deep learning methods, especially LSTM(RNN) method. In this project, it is concentrated on expanding the existing password dictionary lists using machine learning techniques, combining existing password phrases with other permutations, and developing more potential passwords that a user could construct. High-accuracy passwords that the user can create are generated and new wordlists are built, after classifying operation of whether the password generated is machine-like generated or user-like-generated. Due to the fact that people tend to create passwords that are easy to remember and they choose common phrases that are used in daily life, In this way, it is investigated that it is possible to perform more powerful attacks by enriching the wordlist by utilizing human-like classified passwords. The goal of the thesis is to create enriched dictionaries after generating potential logical passwords that the user might construct in the future in order to increase the capacity of the dictionary attacks. By giving password dictionary attacks additional power in this investigation, it will be feasible to demonstrate that dictionary attacks are actually more effective than previously believed. In the mac-

hine learning literature, a common method suitable for our aim is RNN(LSTM). So that, Recurrent Neural Networks (RNN) is used as one of the Machine Learning techniques. As Recurrent Neural Network (RNN) architecture, Long Short-Term Memory (LSTM) is utilized in this work. Moreover, In this thesis, we consider the following question: Can we enrich a password dictionary by analyzing the existing passwords and adding new ones after password classification operation, by using machine-learning techniques? By empowering the password dictionaries using that way, it will be possible that dictionary attacks can actually be made more effective than previously thought.

KEYWORDS: Artificial Intelligence, Building Password Dictionary Wordlists, Cyber Security, Deep Learning, Dictionary Attack, Long Short-Term Memory, Machine Learning, Natural Language Processing, Password Guessing, Recurrent Neural Networks

COMMITTEE: Asst. Prof. Dr. Murat AK

Prof. Dr. Melih GÜNEY

Asst. Prof. Dr. Naci ER

ACKNOWLEDGEMENTS

My thesis advisor, who provided me with help to establish this study, I would like to thank Asst. Prof. Dr. Murat Ak. He taught me a lot about cryptology and cybersecurity. I am really happy to work with him in this field. He is a very understanding professor, he always answered my questions whenever I needed them. He's always just a call away. He increased my interest in cryptology & cybersecurity topics.

I would like to thank my valuable professors from Akdeniz University who taught me very important knowledge during my master's degree.

I would also like to thank Asst. Prof. Dr. Joseph William Ledet, one of the professors at Akdeniz University because he taught me how to think critically and look at events from different perspectives. Additionally, he supported me a lot in terms of motivation during the course phase. He made me believe in myself at times when I thought I couldn't do it. I learned that I could do the things I said I couldn't do, and I learned to overcome obstacles. He is a very valuable professor. Today, I still see the benefits of the system administration course.

Another thank you goes to Asst. Prof. Dr. Hüseyin Gökhan Akçay, one of my professors at Akdeniz University. Thanks to him, I liked machine learning and deep learning. His lessons really helped me a lot, I really love working in this field.

I would like to thank the Tübitak family and my colleagues for supporting me in the process of writing my master's thesis. They are all truly valuable and knowledgeable people. They added incredible value to me. I would like to thank my very valuable team captain Erol Yavuz for believing and trusting me, he supported me a lot. He provided a working environment and a computer when necessary, without which I might not have been able to create this project today. Indeed, the model training phase of deep learning from scratch is a job that requires patience and a good technological infrastructure. I would also like to thank my very valuable unit chief, Fatih Acar. He is someone who follows the innovations in Europe very closely and constantly gives advice on how to learn new things and improve myself.

I would like to thank one of my professors in the Department of Computer Engineering at Pamukkale University, who was my thesis advisor during my undergraduate studies, Assoc. Prof. Dr. Kenan Karagül. Together, we created The Simulated Annealing

Approach to the Traveling Salesman Problem. He is truly a very knowledgeable person, I thank him for training me.

I would like to thank one of my professors from Medimurje, Zeljko Knok, who supported me during my time in Croatia. Zeljko is a really good scout, he helped me to discover my talents and predicted from those years that I could be successful in the fields I am working in and gave me advice and guidance. I would like to thank my professors at Medimurje Technical University.

I really thank my teacher Ahmet Yıldız from Balıkesir Cumhuriyet Anatolian High School, who taught me perseverance and patience.

I would like to thank my family for their financial and motivational support throughout my life. My father asked me to write the part where I thank my family in Turkish. So my following Turkish message is for my family:

Saygıdeğer ailem, bir çok zorluğu beraber atlattık. Bu tezi yazarken bana destek olduğunuz için çok teşekkür ederim. Uzakta da olsak, ailem yanımda, bunu biliyordum.

LIST OF CONTENTS

ÖZET	i
ABSTRACT	iv
ACKNOWLEDGEMENTS	vii
TEXT OF OATH	xi
SYMBOLS AND ABBREVIATIONS	xii
LIST OF FIGURES	xiii
LIST OF TABLES	xvii
1. INTRODUCTION	1
2. LITERATURE REVIEW	9
3. MATERIAL AND METHOD	13
3.1. Rule-Based Models	13
3.2. Markov Models	14
3.3. Probabilistic Context-Free Grammar models(PCFG)	15
3.4. Neural Network Models	17
3.4.1. Discriminative and Generative Model Types	17
3.4.2. Text Generation Through Neural Networks	18
3.4.3. Deep Neural Networks	22
3.4.4. Perceptrons	25
3.4.5. Bias Term of Perceptrons	26
3.4.6. Contributing Nonlinearity By Using Activation Functions	28
3.4.7. Constructing Neural Networks Utilizing Perceptrons	34
3.4.8. Building Deep Neural Networks	38
3.4.9. Recurrent Neural Networks	39
3.4.10. Long Short-Term Memory Architecture	41
4. RESULTS AND DISCUSSION	44
4.1. Top62WPAProbable-v3 Dataset Experiments	45
4.2. Top62WPAProbable-v4 Dataset Experiments	49
4.3. Top4800-WPA-probable-v2 Dataset Experiments	53
4.4. Top4800-WPA-probable-v3 Dataset Experiments	58
4.5. Top204Thousand-WPA-probable-v2 Dataset Experiments	63
4.6. Rocky62 Dataset Experiments	69
4.7. Rocky4800 Dataset Experiments	73
4.8. Rocky204kv2 Dataset Experiments	78

4.9. Kaonashi62 Dataset Experiments	83
4.10. Kaonashi4800 Dataset Experiments	87
4.11. Kaonashi204k Dataset Experiments	92
4.12. Machine-gen-62-dataset Experiments	97
4.13. Machine-gen-4800-datasetv2 Experiments	101
4.14. Machine-gen-204k-dataset Experiments	106
4.15. Mix62 Dataset Experiments	111
4.16. Mix4800 Dataset Experiments	116
4.17. Mix204kv2 Dataset Experiments	121
5. CONCLUSION.	130
6. REFERENCES	131

CURRICULUM VITAE

TEXT OF OATH

I declare that this study "EXPANDING PASSWORD DICTIONARIES BY GENERATING NEW PROBABLE PASSWORDS USING MACHINE LEARNING TECHNIQUES", which I present as master thesis, is in accordance with the academic rules and ethical conduct. I also declare that, I cited and referenced all material and results that are not original to this work.

06/02/2024

Mehmet Görkem KESTANE



SYMBOLS AND ABBREVIATIONS

Symbols:

$\mathbb{N}_0$: $\mathbb{N} \cup \{0\}$

$\mathbb{Z}_0^-$: $\mathbb{Z}^- \cup \{0\}$

Abbreviations:

AI	: Artificial Intelligence
BPTT	: Backpropagation Through Time
BRNN	: Bidirectional Recurrent Neural Network
DL	: Deep Learning
DNN	: Deep Neural Network
GRU	: Gated Recurrent Unit
IT	: Information Technology
LSTM	: Long Short-Term Memory
MD5	: Message - Digest Algorithm 5
ML	: Machine Learning
NLP	: Natural Language Processing
OMEN	: Ordered Markov Enumerator
PCFG	: Probabilistic Context-Free Grammar
PIN	: Personal Identification Number
RNN	: Recurrent Neural Network
seq2seq	: Sequence To Sequence Model
SHA-1	: Secure Hash Algorithm 1
SHA-2	: Secure Hash Algorithm 2
SHA-3	: Secure Hash Algorithm 3

LIST OF FIGURES

Figure 3.1.	Comparison Between Generative and Discriminative Deep Neural Network Models	18
Figure 3.2.	The Process Of Image Production from Natural Language Text .	22
Figure 3.3.	The Relationship of Artificial Intelligence, Machine Learning and Deep Learning with each other	24
Figure 3.4.	A perceptron	25
Figure 3.5.	A Perceptron with Bias Term	27
Figure 3.6.	The Sigmoid Activation Function Drawn Using Matplotlib . . .	29
Figure 3.7.	The Data Points With Linear Separation	30
Figure 3.8.	The Data Points With Nonlinear Separation	31
Figure 3.9.	A Perceptron With Weights for Nonlinear Separation Example .	32
Figure 3.10.	The Decision Boundary Line	33
Figure 3.11.	Constructing Neural Networks Utilizing Perceptrons	35
Figure 3.12.	Perceptron Architecture That Has Multiple Outputs	36
Figure 3.13.	Neural Network Architecture with One Hidden Layer	37
Figure 3.14.	Deep Neural Network Architecture With Multiple Hidden Layers	39
Figure 3.15.	Recurrent Neural Network	40
Figure 3.16.	Long Short-Term Memory Architecture	43
Figure 4.17.	Hyperparameters Value For Top62WPAProbable-v3 Dataset . . .	46
Figure 4.18.	Hyperparameters Value For Top62WPAProbable-v3 Dataset . . .	46
Figure 4.19.	Loss Function Value For Each Iteration	46
Figure 4.20.	Cost Value For Each Iteration	47
Figure 4.21.	Learning Rate For Each Iteration	47
Figure 4.22.	Logits For Each Iteration	48
Figure 4.23.	Hyperparameters Value For Top62WPAProbable-v4 Dataset . . .	49
Figure 4.24.	Hyperparameters Value For Top62WPAProbable-v4 Dataset . . .	49
Figure 4.25.	Loss Function Value For Each Iteration	50
Figure 4.26.	Cost Value For Each Iteration	50
Figure 4.27.	Learning Rate For Each Iteration	50
Figure 4.28.	Logits For Each Iteration	51
Figure 4.29.	Hyperparameters Value For Top4800-WPA-probable-v2 Dataset .	53
Figure 4.30.	Hyperparameters Value For Top4800-WPA-probable-v2 Dataset .	53

Figure 4.31. Loss Function Value For Each Iteration	54
Figure 4.32. Cost Value For Each Iteration	54
Figure 4.33. Learning Rate For Each Iteration	55
Figure 4.34. Logits For Each Iteration	55
Figure 4.35. Hyperparameters Value For Top4800-WPA-probable-v3 Dataset .	58
Figure 4.36. Hyperparameters Value For Top4800-WPA-probable-v3 Dataset .	58
Figure 4.37. Loss Function Value For Each Iteration	59
Figure 4.38. Cost Value For Each Iteration	59
Figure 4.39. Learning Rate For Each Iteration	60
Figure 4.40. Logits For Each Iteration	60
Figure 4.41. Hyperparameters Value For Top204Thousand-WPA-probable-v2 Dataset	63
Figure 4.42. Hyperparameters Value For Top204Thousand-WPA-probable-v2 Dataset	63
Figure 4.43. Loss Function Value For Each Iteration	64
Figure 4.44. Cost Value For Each Iteration	64
Figure 4.45. Learning Rate For Each Iteration	65
Figure 4.46. Logits For Each Iteration	65
Figure 4.47. Hyperparameters Value For Rocky62 Dataset	69
Figure 4.48. Hyperparameters Value For Rocky62 Dataset	70
Figure 4.49. Loss Function Value For Each Iteration	70
Figure 4.50. Cost Value For Each Iteration	71
Figure 4.51. Learning Rate For Each Iteration	71
Figure 4.52. Logits For Each Iteration	72
Figure 4.53. Hyperparameters Value For Rocky4800 Dataset	73
Figure 4.54. Hyperparameters Value For Rocky4800 Dataset	73
Figure 4.55. Loss Function Value For Each Iteration	74
Figure 4.56. Cost Value For Each Iteration	74
Figure 4.57. Learning Rate For Each Iteration	75
Figure 4.58. Logits For Each Iteration	75
Figure 4.59. Hyperparameters Value For Rocky204kv2 Dataset	78
Figure 4.60. Hyperparameters Value For Rocky204kv2 Dataset	78
Figure 4.61. Loss Function Value For Each Iteration	79
Figure 4.62. Cost Value For Each Iteration	79

Figure 4.63. Learning Rate For Each Iteration	80
Figure 4.64. Logits For Each Iteration	80
Figure 4.65. Hyperparameters Value For Kaonashi62 Dataset	83
Figure 4.66. Hyperparameters Value For Kaonashi62 Dataset	83
Figure 4.67. Loss Function Value For Each Iteration	84
Figure 4.68. Cost Value For Each Iteration	84
Figure 4.69. Learning Rate For Each Iteration	85
Figure 4.70. Logits For Each Iteration	85
Figure 4.71. Hyperparameters Value For Kaonashi4800 Dataset	87
Figure 4.72. Hyperparameters Value For Kaonashi4800 Dataset	87
Figure 4.73. Loss Function Value For Each Iteration	88
Figure 4.74. Cost Value For Each Iteration	88
Figure 4.75. Learning Rate For Each Iteration	89
Figure 4.76. Logits For Each Iteration	89
Figure 4.77. Hyperparameters Value For Kaonashi204k Dataset	92
Figure 4.78. Hyperparameters Value For Kaonashi204k Dataset	92
Figure 4.79. Loss Function Value For Each Iteration	93
Figure 4.80. Cost Value For Each Iteration	93
Figure 4.81. Learning Rate For Each Iteration	94
Figure 4.82. Logits For Each Iteration	94
Figure 4.83. Hyperparameters Value For machine-gen-62-dataset	98
Figure 4.84. Hyperparameters Value For machine-gen-62-dataset	98
Figure 4.85. Loss Function Value For Each Iteration	98
Figure 4.86. Cost Value For Each Iteration	99
Figure 4.87. Learning Rate For Each Iteration	99
Figure 4.88. Logits For Each Iteration	100
Figure 4.89. Hyperparameters Value For machine-gen-4800-datasetv2	102
Figure 4.90. Hyperparameters Value For machine-gen-4800-datasetv2	102
Figure 4.91. Loss Function Value For Each Iteration	102
Figure 4.92. Cost Value For Each Iteration	103
Figure 4.93. Learning Rate For Each Iteration	103
Figure 4.94. Logits For Each Iteration	103
Figure 4.95. Hyperparameters Value For machine-gen-204k-dataset	107
Figure 4.96. Hyperparameters Value For machine-gen-204k-dataset	107

Figure 4.97. Loss Function Value For Each Iteration	108
Figure 4.98. Cost Value For Each Iteration	108
Figure 4.99. Learning Rate For Each Iteration	109
Figure 4.100. Logits For Each Iteration	109
Figure 4.101. Hyperparameters Value For Mix62 Dataset	113
Figure 4.102. Hyperparameters Value For Mix62 Dataset	113
Figure 4.103. Loss Function Value For Each Iteration	114
Figure 4.104. Cost Value For Each Iteration	114
Figure 4.105. Learning Rate For Each Iteration	114
Figure 4.106. Logits For Each Iteration	115
Figure 4.107. Hyperparameters Value For Mix4800 Dataset	118
Figure 4.108. Hyperparameters Value For Mix4800 Dataset	118
Figure 4.109. Loss Function Value For Each Iteration	119
Figure 4.110. Cost Value For Each Iteration	119
Figure 4.111. Learning Rate For Each Iteration	119
Figure 4.112. Logits For Each Iteration	120
Figure 4.113. Hyperparameters Value For Mix204kv2 Dataset	123
Figure 4.114. Hyperparameters Value For Mix204kv2 Dataset	123
Figure 4.115. Loss Function Value For Each Iteration	124
Figure 4.116. Cost Value For Each Iteration	124
Figure 4.117. Learning Rate For Each Iteration	124
Figure 4.118. Logits For Each Iteration	125

LIST OF TABLES

Table 3.1.	Example Rules in order for Password Guessing through HashCat .	14
Table 3.2.	List of Grammar Structure	16
Table 4.3.	Password distributions by percentages in the produced output dataset	128
Table 4.4.	Optimal Loss Value Distributions For Each Dataset	129
Table 4.5.	The Amount of Password Generated For Each Dataset	129



1. INTRODUCTION

Password is an authentication method that has been used in the field of information security for a long time. Therefore, it is still very much used. However, people tend to choose insecure ones from time to time. A password could be a string of characters utilized for confirming a user on a computer system.

For illustration, you will have an account on your computer that requires you to log in. In arrange to effectively get to your account, you must give a valid and substantial username and password. This combination is generally alluded to as a login. Whereas usernames are for the most part open data, passwords are private to each user. Most passwords are comprised of a few characters, which can ordinarily include letters, numbers, and most symbols, but not spaces. Whereas it is nice to select a password that's easy to bear in mind, you ought to not make it so easy that others can figure it. The foremost secure passwords utilize a combination of letters and numbers and don't contain real words.

In terms of information on the web, protecting the information and providing its security is concerned and quite crucial. Mostly, the databases of websites include the credentials of millions of users. Thus, any website needs to ensure security for all users whose data stored on the website. It is quite important to validate the identity of users before retrieving data and services of a web application software as well. Hence, the authentication process is an essential requirement for information security (Wang and Qin 2010).

Different authentication methods are used diversely, users can choose one of them at their convenience. For instance, security hardware tokens, PINs, biometric fingerprints, and passwords are considered as authentication mechanisms (O'Gorman 2003). Although there are abundant ways exist in order to provide user identification, the most popular method for this is still password-based authentication (Luo et al. 2019).

Even if, it is a popular method; there is the vulnerability that exists stems from its simplicity. A secure password must be hard to guess. In contrast, a poorly secured password is very easy to guess. In order to protect their passwords against hackers and stay more secure, users must be aware of how password protection important is, to provide that users need to make their password security better. Otherwise, the users might get hacked and entire passwords are generally stolen.

To avoid that, it is crucial for users need to use a unique password on each of their services. Thus, in case of one of a user's online services is hacked, the hackers are not able to reach the user's other credentials of another online services. Also, in order to generate a secure password, the users had better to create a password with at least a certain amount of characters. Passwords used by most people should not be used and, passwords that already exist in dictionaries shouldn't be used. The same password in different places shouldn't be used by users. In this case, for example, if a user's e-mail address is compromised, then the password of his/her accounts such as Instagram, Facebook, and Twitter can be easily compromised.

Due to the email of a user generally used as a password recovery mechanism for his/her other accounts, keeping the email's password security weak is like handing an intruder the key to the online identity of a user. This brings another aspect of identity theft, by having access to a few accounts of a user, especially through an email, can actually be pretty easy to steal someone else's identity, in this way.

Once a user's email has cracked and its password has been revealed by a hacker, he might be registering new credit cards to get healthcare on others' dime. In that scenario, the hacker would be obtaining someone else's information and using it to get a credit card in that person's name, and as a result, they'd be paying for the hacker's healthcare unwittingly until they found out. Not only that, but also, hackers can even register their bridge or highway tolls under other people's name.

Not only, In case of users have logged into their account from other devices which are not belong to them, they must log out securely, but also, when users log into their account on another computer, they shouldn't select the "remember me" box in the browser. Once the users know all this, they will be able to create a strong password that is easy to remember.

So as to provide a secure password, also the initials of the sentences can be combined. For example; 18aC1wDtS (I ate a chocolate, I will drink tea soon). On the other hand, instead of some letters in words, we can prefer numbers that look like letters. Such as; 1nc3Pt10n (Inception) 4kD3n12Un1v3rS1tY (Akdeniz University). Also, we are able to use special characters instead of some letters. For instance; 2645Milwaukee\$\$ (2645Milwaukee Bucks)

Apart from that, users had better not store their passwords in plain text on their computer, or they shouldn't write down their passwords somewhere. A survey of corporate password users found that they flustered by password requirements, and users are dealing with passwords by writing them down on post-it notes. (Adams and Sasse 1999). Another point is, not using username, real name, birthday, phone number, or any other easily identifiable information in the password. Also, never using an easy keyboard combination like ABCD, QWERTY, 123456 is crucial to provide password security and creating strong passwords.

There are various methods used by hackers to crack passwords. The main ones are brute-force, dictionary and rainbow table attacks. Hackers tend to use different kinds of attack types in order to achieve the input password's plain text entered by users, from its hash value. Such as; Brute-force, dictionary, and rainbow table attacks. In light of these facts, paying attention to providing the security of user passwords is quite crucial (Kumar et al. 2013).

In addition, certain password generators provide pattern-based passwords that defend themselves to be strong, solid, and secure. Hackers can define a common password pattern and generate a new pattern-based dictionary file from a regular dictionary file (Tath 2015). In these situations, pattern-based passwords are not able to durable to hacker's attacks, and they are also vulnerable to such threats. Therefore, even a very strong hash function cannot protect weak passwords from hash attacks. After leaked passwords were analyzed, it was found that the most commonly used passwords were simple dictionary words, and the most common method to crack password hashes was dictionary attacks. People like to create passwords that are easy to remember, so they choose common vocabulary from dictionaries.

Hackers generally attempt cracking passwords, in order to do that they tend to use a predefined dictionary of words. So, when an attack happened through the predefined dictionary of words by hackers, it is called a dictionary attack. However, nowadays, hackers prefer utilizing specific password-cracking software, so as to perform a dictionary attack, rather than looking up words through piles of physical books or dictionaries.

Using a password domain, an attacker can try to log in to the account by trying all possible passwords until the correct password is found. This kind of attack is called a

dictionary attack. The smaller the password domain enables the more attacks that will be able to possible by hackers. Successful dictionary attacks were performed in the past. As a case in point, hackers broke into eBay users' accounts which they are sellers who have got good reputations on eBay, so as to perform fraudulent auctions through this platform (Pinkas and Sander 2003).

In addition, due to cyber criminals perform loads of research so as to improve their possibility of cracking passwords and narrow their dictionary word-list down, the dictionary entries used by them are generally not random. Most social media users share some personal info as, they generally use their pets' name, favourite celebrity or other personal references as password. Cybercriminals know that users tend to generate passwords in this way. Therefore, they check the social media profiles of their victim's in order to receive some personal information. This is a great way to narrow the wordlist down. Apart from that, hackers may compile the dictionary list of most commonly used password phrases.

Due to the fact that, users ought to definitely avoid creating passwords such words as "password", "qwerty" or "123456" for the purpose of provide a secure password and stay safe. On the other hand, users mostly combine their phrases with diverse symbols and special characters as well, so even combinations like "\$ecurePa\$\$word" with dollar sign replacing the letters "s" are not resistant to dictionary attacks performs by hackers.

A dictionary attack is performed very similarly with a brute force attack but are not entirely the same. As a matter of fact, both of them are such aggressive attempts dedicates to cracking your passwords. However, a dictionary attack uses a list of predefined phrases, in spite of the fact that the brute force method generates every possible combination of letters, numbers and special symbols. Hence, the dictionary attacks are able to crack short passwords quickly, but it needs more time for longer ones.

Obviously, because of passwords are the main target in dictionary attacks, it is essential that users should use strong ones in order to prevent this kind of attacks. In order to generate a strong and a solid password, users had better use long passwords and lots of upper and lowercase letters and special characters. It would be better to avoid any personal names or references and easily predictable number series like 123 or the date of birth.

Changing the password on specific periods of time, protects the users from getting

used against them in view of the fact that attackers may snatch it. Even though, whether the password of a user is super-strong, there is still a possibility to someone may crack it. Because of that, changing the passwords at least every two month is essential. Reusing the same password for different accounts lead up to be inflicted huge damage with just only one leaked password by hackers. As it is mentioned before, hackers look at social media channels for feasible password hints. Should the users share their personal information too much, they assist to cyber criminals.

In most cases, clear text passwords are not stored in IT systems; only the hash output of these passwords is stored. Hashing is an algorithm and keyless one-way encryption. When users try to log in, the password they enter is hashed, and compare this hash with the hash stored in the system. The hash function is irreversible; it is not possible to change the algorithm and obtain the password from the hash. Although the hash is irreversible, an attacker can run the hash algorithm multiple times by selecting multiple possible passwords and comparing the output with the desired hash, in the hope of finding a match (and thus obtaining the original password). This is called password cracking (Conrad et al. 2017).

Hash can be defined as a value that expresses the mathematical summary of a data. In case of there is a file that have been downloaded from the internet by a user, and he would like to be sure that this file is exactly the same as the original file. In order to assure that, verification can be carried out by downloading the original file and checking each data bit one by one, but this is an extremely laborious and time-consuming process. Hash functions become useful in this situation. No matter how much data there is, these functions always generate hashes that are the same length. A different hash will be generated even if a single bit in the data has changed.

Two separate files are very unlikely to have the same hash value. Hash functions are algorithms that transform one set of data into an other set of fixed-length data. Hash functions work deterministically. Particularly, a deterministic model tends to always produce the same output from a given input. In other words, it consistently produces the same result every time for a data. Despite the fact that, it's a beneficial approach to check whether two files are the same or not, these functions can be inefficient when privacy is at stake. Although there is no direct way to get from the hash value of the data to the original data,

the original data can be estimated by indirect methods such as Rainbow table attacks.

As a result of memory being utilized by rainbow table attack techniques in order to cut down on the overall time required to find a correct password, rainbow tables are categorized as a time-memory trade-off approach. In his A Cryptanalytic Time - Memory Trade-Off article which is published in 1980, M. Hellman proposed a cryptanalytic time-memory trade-off for block ciphers that shortens computation time by distributing the possible key solution space N between memory M and time T , using a pre-computed table (Hellman 1980). Although his method only utilized one reduction function, Martin Hellman created the first recalculated table concept which is known as Hellman's table with his work.

Philippe Oechslin developed rainbow tables as an implementation of Martin Hellman's earlier, simpler technique (Hellman 1980; Oechslin 2003). An improved version of Hellman's tables was developed by Philippe Oechslin in his paper which is Making a Faster Cryptanalytic Time-Memory Trade-off and his presentation at the CRYPTO 2003 conference (Oechslin 2003; Tasevski 2015). The name "Rainbow tables" comes from the fact that this new approach utilized a greater amount of reduction functions, which the author dubbed "rainbow."

It is important to understand what a rainbow table is. A rainbow table is a pre-computed table of hashes that are used in password cracking. The idea behind a rainbow table attack is to use the pre-computed hashes to speed up the process of cracking a password by reducing the amount of time needed to compute hashes for each possible password. Furthermore, in a rainbow table attack, the attacker first obtains a list of hashed passwords, and then uses the rainbow table to look up the corresponding plaintext passwords. This can be a very effective method for cracking passwords, especially if the password is relatively short or is a commonly used password. Additionally, rainbow tables are essentially large databases of pre-computed hashes. The goal of creating a rainbow table is to reduce the time needed to crack a password by pre-calculating the hashes for a large number of possible passwords, and then using the rainbow table to quickly look up the corresponding plaintext password.

Moreover, rainbow tables are often used in combination with other password cracking techniques, such as brute-force attacks or dictionary attacks. In a brute-force attack, the

attacker tries every possible password until the correct one is found. In a dictionary attack, the attacker uses a pre-defined list of words (or a dictionary) to try and crack the password. Rainbow tables can be used to speed up these attacks by reducing the time needed to compute hashes for each possible password.

One of the main advantages of rainbow table attacks is that they can be very fast, especially if the password is short or is a commonly used password. The disadvantage of rainbow table attacks is that they require a large amount of storage space to store the pre-computed hashes, and they can be computationally intensive to create. In order to protect against rainbow table attacks, it is important to use strong and unique passwords, and to use a password hashing algorithm that is resistant to pre-computation attacks, such as bcrypt or scrypt. It is also a good idea to use two-factor authentication, which adds an extra layer of security to your accounts by requiring you to enter a code that is sent to your phone or email in addition to your password.

In order to provide password security for the passwords which are stored in databases, the password salting method is utilized. In this technique, a string of 32 or more characters is added to passwords after that, the component is hashed by utilizing hash functions. Hackers tend to perform reverse engineering on passwords so as to reach them from the database and crack them, In such cases salting passwords provides safety.

Passwords used by users to log in to online systems are stored in databases after being hashed. For illustration, assuming the case that a password like Akdeniz0707 has been generated by a user, this password is passed through a hash function, such as MD5 and it turns out to be a value like 3cc89c5ae4f8cb0c945abd5c1c7ba995 or a06a1382ad002e97fbe6b416aed29baff50ab06d if the SHA-1 hash function is used, and then stored in the database.

When a user attempts to log into the system, the password he inputs is transformed into a hash using the same process, and the hashes are compared. If the hashes match, the user is permitted to enter the system. A malevolent hacker who has gained access to this system's database can use rainbow tables to crack existing passwords by comparing hashes of commonly known passwords. To handle this problem, a keyword called salt is utilized. The used hash function is modified using a specific string that only the system owner knows. As a result, even if the database is compromised, these passwords cannot

be converted to the original. In some systems, a unique salt is generated for each user, and these salt values are stored in the database. Hence, even the hashes of users using the same password will be different from each other. In this case, a hacker who takes over the database would have to try all known passwords individually to decrypt each user, which is a very time-consuming process. Therefore, with this method, passwords are stored securely. Algorithms such as MD5, SHA-1, SHA-2, SHA-3, bcrypt and BLAKE3 are the most popular hash algorithms.

Unlike other approaches, this study will focus on using pre-existing dictionaries containing various passwords as train datasets, generating new passwords that are not normally included in dictionaries by using machine learning techniques and thus creating more enriched dictionary lists. In this thesis, We'll concentrate on expanding the existing password dictionary lists using machine learning techniques, combining existing password phrases with other permutations, and developing more potential passwords that a user could construct.

The goal of the research is to generate brand new potential logical passwords that the user might construct in the future. And then, creating enriched dictionaries with generated passwords. Therefore, those produced dictionaries will be able to utilized in order to perform a dictionary attack. By giving password dictionary attacks additional power in this investigation, it will be feasible to demonstrate that dictionary attacks are actually more effective than previously believed. In the machine learning literature, a common method suitable for our aim is RNN(LSTM). So that, Recurrent Neural Networks (RNN) will be used as one of the Machine Learning techniques. As Recurrent Neural Network (RNN) architecture, Long Short-Term Memory (LSTM) will be utilized in this work. Also, In this investigation, we consider the following question: Can we enrich a password dictionary by analyzing the existing passwords and adding new ones by using machine learning techniques?

In addition, we will write a classifier that can analyze what percentage of passwords generated are human-generated or machine-generated in order to form enriched dictionaries. As a result, the dictionary attack will be more successful by focusing on generating passwords that can be produced by humans. In this thesis, we will examine how these dictionaries can be enhanced.

2. LITERATURE REVIEW

Vu et al., in their study in 2007 (Vu et al. 2007), mention that passwords with personal meaning are more easily remembered by users. Likewise, the study conducted by Huh et al. in 2015 supports this phenomenon (Huh et al. 2015). Both studies emphasize that user-generated passwords are much more memorable than system-generated passwords. In addition, in their 2016 study, Melicher and others speculate that more complex alphanumeric passwords are more difficult to be recollected by a user (Melicher et al. 2016). On the other hand, they also infer a correlation between utilizing various character classes and weak memorability. Mnemonic passwords, patterns, and passphrases which are generated by users, words and biographical passwords are the password generation techniques which makes easy memorable passwords' production easier.

Although Zviran and Haga in their study in 1990 (Zviran and Haga 1990), Ur et al. in their study in 2015 (Ur et al. 2015), and Biddle and Stobert in their study in 2014 drew attention to the use of strong passwords, many users tend to use methods in order to make their passwords easy memorably (Biddle and Stobert 2014). It was mentioned in a 2015 study by Pfleeger et al. that users fear the possibility that they might forget their passwords if they generate them without using easier-to-remember techniques (Pfleeger et al 2015). It is considerable and logical that utilizing those techniques while the account number which each user has to manage is increasing day by day. According to the paper which is published in 2007 by Vu et al., the memory of humanbeing is not able to handle multiple complex passwords and it is a limited source (Vu et al. 2007).

The Rule-based method is a well-liked approach in the realm of digital forensics and is frequently utilized by IT specialists in forensics. Attacks of this nature rely on password creation and reuse patterns extracted in the dataset that has previously been compromised or through user surveys (Nosenko et al. 2022). Applications designed specifically to guess passwords such as John the Ripper, Hashcat, Passware Kit Forensic, and others.

An event depends on one or more of the most recent preceding occurrences in a Markov chain, which is a model that explains a succession of potential events. The Markov model, which Narayanan et al. first developed in 2005 as a password-guessing method (Narayanan and Schmatikov 2005) , was the most popular modeling language technique.

The idea behind a Markov model is to make predictions about a character's behavior based on those of prior characters. Subsequently, with their outstanding work, Duermuth et al. developed a program known as Ordered Markov Enumerator(OMEN) using the Markov-based technique (Dürmuth et al. 2015), in which users create passwords by calculating the password's probability through the connection between characters from left to right.

Narayanan et al. were the first to employ Markov models to produce password guesses. Their method makes use of manually created password restrictions, such as determining how much of the generated passwords are made up of letters and digits. However, this method was further enhanced by Weir et al. using probabilistic context-free grammars (PCFGs). Weir et al. showed how to learn these rules from password distributions using PCFGs (Weir et al. 2009). Later developments of this first study have been made by Ma et al. (Ma et al. 2014) and Durmuth et al. (Dürmuth et al. 2015)

In their article published in 2020, Ji et al. emphasize that, text which is produced had better fulfill basic language structure. In addition, it should contain the proper message (Jin et al 2020). In as much as text production process and the evaluation of the process is open-ended, the whole generation process and forming and assess the content in grammatical, semantical and synthetical way is demanding. On the other hand, there is lack of studies about generating real user passwords and building & enriching password dictionary sets using deep learning methods in the state-of-art.

In the literature, in order to employ DL models, different DL architectures has been benefitted, such as RNN (Recurrent Neural Networks) (Koutnik et al. 2014). For the future states' input, RNN utilizes the previous states' output. RNNs are known as a class of deep neural networks. Recurrent Neural Network algorithm considered as the first algorithm which keeps past states' outputs. Because of the Vanishing Gradient, Recurrent Neural Networks are tend to forget the previous outputs across a stretch of time.

In order to perform sequential data processing, BRNN architecture is utilized. It's a neural network architecture. The neural network is able to use the data both from past and the future context in its predictions. Bidirectional RNN (BRNN) utilizes two RNN layers in order to process input sequences in both forward and backward directions. (Schuster and Paliwal 1997) This is the main difference of BRNN compared to traditional RNNs. If the current state is contingent on both the previous state and the future state, BRNN is

useful.

Long Short Term Memory (LSTM) is RNN's class. (Hochreiter and Schmidhuber 1997) It is utilized in order for keeping previous states' information over a long time interval and it forgets the information which are insignificant.

A Bidirectional Recurrent Neural Network contains noticeable two different recurrent hidden layers. One of those recurrent hidden layers is able to perform forwardly processing the sequential input. On the other hand, the other one performs the same operation in a backward way. Henceforth, results are gathered from those recurrent hidden layers. And then, they are given as input into a final layer which is the prediction-maker.

Recurrent hidden layers can be created by utilizing recurrent network cells. Namely, LSTM or GRU. Gated Recurrent Unit (GRU) and LSTM can be used in order to solve the vanishing gradient problem in Recurrent Neural Networks (RNN). Gated Recurrent Unit can be categorized as LSTM's simplified form.

Generative Adversarial Networks (GAN) are another technique in order to produce text. It consists of two important elements, such as a discriminator and a generator. With those fundamental elements, it performs with the methodology of a min-max game. Having said that, while the discriminator tries to guess whether the sample comes from training dataset or it is generated using generative network; another component, generator focuses on making the discriminator's mistakes increased.

GPT is an abbreviation of Generative Pre-trained Transformer and its first debut was in the year of 2018. GPT was revealed for the first time in 2018 by the company called OpenAI. (Radford et al. 2018). GPT models are artificial neural networks(ANN) built with transformer architecture and It was pretrained with very large datasets consisting of unlabeled texts. This model is an artificial neural network that can produce text and create content like human beings do.

In another study published in 2019, Radford and his colleagues introduced the GPT-2 model. The model is a transformer-based model trained with 40 GB of internet texts taken from 8 million different websites. The model has 1.5 billion parameters (Radford et al. 2019).

The GPT-3 model, which is 100 times larger than the GPT-2 model, contains 175 billion parameters and 96 layers. This model was developed by OpenAI in June 2020.

In order to train GPT-3 model 499 billion tokens of web data were utilized. It is able to show better performance and more production power on various task such as zero-shot and one-shot learning and production of text (Bowman 2023, Elondou et al. 2023, Brown et al. 2020). On March 15, 2022 GPT-3.5 model was launched. It has got 175 billion parameters.

In March 2023, the newest GPT-based model, the GPT-4 model, was launched by OpenAI. This model is a multi-modal and large-scale model. It can take images and texts as input and then has the ability to generate texts as outputs. It is directly accessible to users who have ChatGPT's premium version. By using the API of OpenAI it is able to be involved to other services and products. So that the developers are able to utilize it in that way. (OpenAI 2023, Bubeck et al. 2023)

3. MATERIAL AND METHOD

By repeatedly attempting various possible passwords, the password guessing attack seeks to crack and reveal the password of a user. Different password guessing techniques focused on the production of lists of potential new passwords can build dictionaries that are more resemblant to how real users generate passwords, giving them a more promising option than the brute-force approach. There are state-of-the-art password guessing techniques that exist, such as Rule-based models, Markov models, Probabilistic Context-Free Grammar models(PCFG), and Neural Network models. Markov models and Probabilistic Context-Free Grammar models(PCFG) can be categorized as statistical models as well. In this section, we will explore several methods that are used to generate textual outputs.

3.1. Rule-Based Models

The primary concept behind the Rule-based technique is the combination of several training lists, such as dictionaries, with mangling rules, which results in the generation of new possible candidate passwords from the original passwords, such as dictionary words. Most conventions provide for adding characters, flipping the items, capitalizing the initial letter, etc. Some fundamental rules that may be developed and used with the Hashcat program are shown in Table 3.1 taken from (Hashcat).

Table 3.1 demonstrates how adaptable, precise, and built-for quick possible password candidate generation is the Rule-based attack. For instance, having a password dictionary dataset comprises of a large amount of data, and in order to enrich that password dictionary, it is possible to run the append command on Hashcat in order to add digits to the end of each word. Therefore, it is possible to create larger dictionaries. By having larger dictionaries, the chance of guessing the user-generated passwords can be increased.

The structures of patterns that are revealed can be used to develop rules for efficient password guessing. Additionally, by using the specialized password-guessing applications, a multi-stage hybrid password-guessing attack that makes use of various dictionaries, rules, and even fixed brute-force attacks can be implemented. Through the Passware application, different attack settings are able to be used so as to execute rule-based password guessing.

Table 3.1. Example Rules in order for Password Guessing through HashCat

Name	Function	Description	Example Rule	Input Word	Output Word
Nothing	:	Do nothing (passthrough)	:	p@ssW0rd	p@ssW0rd
Lowercase	l	Lowercase all letters	l	p@ssW0rd	p@ssw0rd
Uppercase	u	Uppercase all letters	u	p@ssW0rd	P@SSW0RD
Capitalize	c	Capitalize the first letter and lower the rest	c	p@ssW0rd	P@ssw0rd
Invert Capitalize	C	Lowercase first found character, uppercase the rest	C	p@ssW0rd	p@SSW0RD
Reverse	r	Reverse the entire word	r	p@ssW0rd	dr0Wss@p
Duplicate	d	Duplicate entire word	d	p@ssW0rd	p@ssW0rddp@ssW0rd
Reflect	f	Duplicate word reversed	f	p@ssW0rd	p@ssw0rddr0Wss@p
Rotate Left	{	Rotate the word left	{	p@ssW0rd	@ssW0rdp
Rotate Right	}	Rotate the word right	}	p@ssW0rd	dp@ssW0r
Append Character	\$X	Append character X to end	\$1\$2	p@ssW0rd	p@ssW0rd12
Prepend Character	^X	Prepend character X to front	^1	p@ssW0rd	12p@ssW0rd
Truncate left	[	Delete the first character	[	pssW0rd	ssW0rd
Truncate right	]	Delete the last character	]	pssW0rd	pssW0r
Delete N	DN	Delete character at position N	D3	pssW0rd	psW0rd
Insert N	iNX	Insert character X at position N	i4!	pssW0rd	p@ss!W0rd
Overwrite N	oNX	Overwrite character at position N with X	o3\$	pssW0rd	ps\$W0rd
Truncate N	,N	Truncate word at position N	,6	pssW0rd	pssW0
Replace	sXY	Replace all instances of X with Y	ss\$	p@ssW0rd	p\$\$W0rd

3.2. Markov Models

Considering those of prior characters, a Markov model is able to perform predictions about a character's behaviour. This is the key factor of a Markov model. Each character in the zero-order Markov's model is created independently from the characters that have come before it and in accordance with the underlying probability distribution. Each character diagram, ordered pair, in the first-order Markov model is given a probability. By considering the prior, each character is created.

$$\mathbb{P}(x_1x_2...x_n) = v(x_1) \prod_{i=1}^{n-1} v(x_{i+1}|x_i) \quad (3.1)$$

According to the Eq. 3.1, P represents the Markovian probability distribution on character sequences, x_i emphasizes individual characters, and the function is the frequency of individual letters and diagrams in English text.

In order to create passwords through Markov-based technique, there are two phases to this process. The n-gram model is trained during the training process, and each letter's frequency is counted after a substring of length n. In the generation stage, candidate passwords are generated after the probability of a likely password is determined using the Markov chain. The size of the alphabet, the number of levels for counting passwords, and the size of the n-grams are the three parameters for this approach. The accuracy of the approach is most strongly influenced by the n-gram size parameter.

Since bigger n-grams provide a more precise approximation to the password distribution, a larger n typically produces better results. However, it also suggests greater runtime, memory, and storage needs. The runtime and memory requirements rise as an alphabet's size increases because more parameters must be estimated. A tiny alphabet, however, suggests that not every password can be produced. A third crucial factor that is used to identify potential password possibilities is the number of levels. As with the preceding parameters, a greater number of levels may boost accuracy while also lengthening the runtime.

Theoretically, the deep learning neural networks are efficient enough to generate several amount of properties which express most user passwords. In addition, we don't have to train the neural networks with any prior knowledge. However, by utilising Markov model approaches and rule-based model approaches we should have a prior knowledge to generate passwords. For example, Markov models use n-grams and every related password characteristics are able to be declared by benefiting the n-grams. On the other hand, while using rule-based approach we have to define prior rules, so that generated passwords only can be created using those available rules.

3.3. Probabilistic Context-Free Grammar models(PCFG)

Context-free grammar has been used to produce strings with a certain structure in the study of natural languages. A probabilistic variation on conventional context-free grammar, probabilistic context-free grammar integrates data on the probability distribution of user passwords. Using this data, password patterns are generated in decreasing probability order. These structures can also be word-mashing templates that can be populated with dictionary words or password guesses. An outline of probabilistic context-free grammar's

use for password cracking is given below:

In the preprocessing stage, particular patterns' frequencies linked to the password string are calculated. Here, the string of alphabetical characters, which is also called the alpha string, the digit string, and the string of non-alpha and non-digit symbols, also known as the special string, are represented by L, D, and S respectively. The base structure for the password "@password789" would be $S_1L_8D_3$. The password's structure would be SLD, and it would be similar to the base structure except that it would additionally take into account the length of strings. While the terminal structure (guess) would fill in the value of L in the preterminal structure, the preterminal structure fills in the values of S and D in the base structure.

Table 3.2. List of Grammar Structure

Structure	Example
Simple	SLD
Base	$S_1L_8D_3$
Pre-Terminal	@ L_8 789
Terminal (Guess)	@wordpass789

A context-free grammar is capable of being defined mathematically as $G = (V, \Sigma, S, P)$. In this formula, the variables' finite set, the terminals' finite set, and the start variable are represented by V , Σ , and S respectively. On the other hand, P shows the productions' finite set of the form $\alpha \rightarrow \beta$. At this point, α denotes a single variable and β denotes a string made up of variables or terminals.

Each production in probabilistic context-free grammars has a probability so that, for a given left-side variable, the sum of all the related productions equals 1. Sentential form refers to a string that is derived from the start symbol. The sentential form's probability is just the sum of the probabilities of the productions employed to derive it.

The derivation of sentential forms achieves optimal results by employing alpha variables that encompass terminal digits and special characters, owing to the insufficiency of production rules in converting alpha variables to alpha strings. These pre-terminal structures correspond to the aforementioned sentential types. At the heart of this concept lies the establishment of preterminal structures, which delineate malleable procedures. When

these procedures are supplied to a distributed system for populating the alpha variables with dictionary terms and subsequently hashing the conjectures, they may be promptly deployed in a distributed password-cracking experiment.

Subsequent investigations regarded semantic patterns as components strategically integrated into the lexicon, with the aim of optimizing the efficacy of Probabilistic Context-Free Grammar (PCFG). (Veras et al. 2014; Li et al. 2014). Houshmand et al. directed their attention towards augmenting the predictive capabilities concerning keyboard-walk structures, employing a refined methodology known as smoothing. (Houshmand et al. 2015). Through their empirical endeavors, they attained commendable results in accurately guessing passwords that followed the keyboard-walk pattern.

3.4. Neural Network Models

There are types of artificial intelligence that can produce images, sounds, music, videos, and text. These types of artificial intelligence, which focus on producing content, are called generative artificial intelligence, and they differ significantly from classical artificial intelligence (discriminative artificial intelligence).

3.4.1. Discriminative and Generative Model Types

Statistical models can be examined under two headings: discriminative and generative. While discriminative models have been at the forefront for many years with the advantages they provide, the popularity of generative models has been rapidly increasing recently. So, what are these discriminative and generative models and what do they do, let's explain with a simple example:

Let's assume that we have millions of photos of cats and dogs. But all the photos are mixed together and we cannot know which photo belongs to the cat and which one belongs to the dog unless we open and look at it. Our task is to separate and organize cat and dog photos. Doing this one by one with human power will take a lot of time. At this point, artificial intelligence can step in and provide you with great convenience. You just label some photos as cat or dog. Then, you feed these labeled photos to the AI model.

The neural network model begins to learn the difference between cats and dogs through these photos and labels. When the training phase is completed, it is possible to present

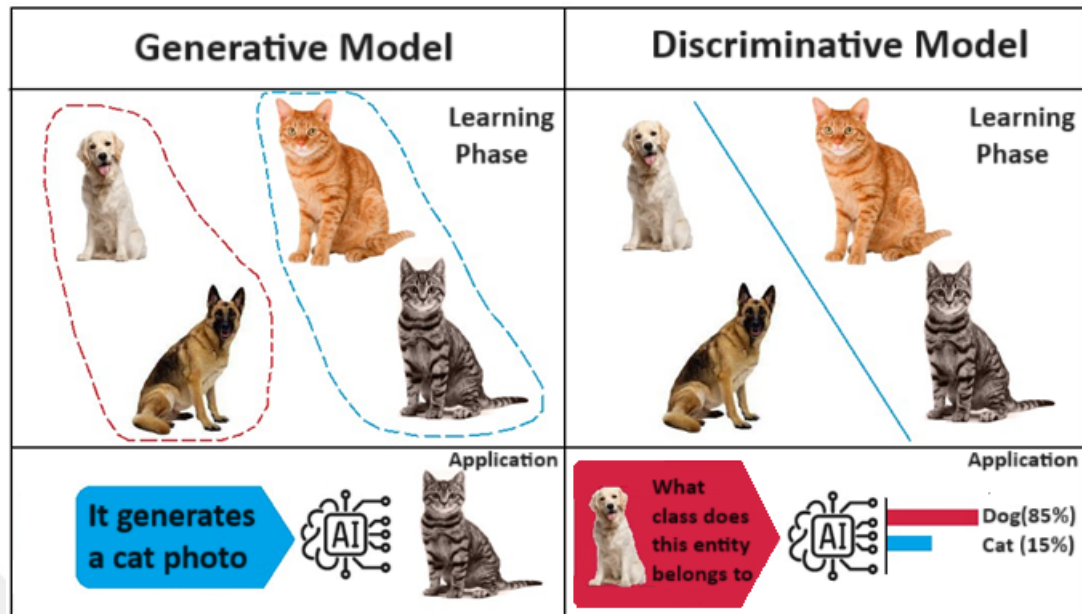


Figure 3.1. Comparison Between Generative and Discriminative Deep Neural Network Models

the remaining photos to the model and ask it to classify them. The model which is trained and utilized in this process is called the discriminative model. The discriminative model tries to learn the features that distinguish cats and dogs from photographs. For example, ears, noses, etc. These models are trained through supervised learning on labeled data, as in the example.

However, if the goal is a little unusual, such as creating a photo of a dog that never existed, the process works differently. In this case, millions of dog photos are given to the generative model without any labeling. The model learns the distributions of data in dog photos. After the training is completed, when it is asked to create a dog photo, it can present a dog photo that has never existed, like an artist.

3.4.2. Text Generation Through Neural Networks

In order to produce brand-new content such as text, computational linguistic techniques and AI is utilized together. The technique which is used so as to combine them is called Natural Language Processing in the state-of-art. On the other hand, NLP is a sub-branch of Artificial Intelligence. Therefore, Text Generation is a branch of NLP. During this generation phase, to form appropriate content is essential in semantical and gramma-

tical way. The mechanism is consisted of training a model which takes data as input in order to train the model, evaluating the input and learning the context from it and produce brand new string content related with the domain of the input set.

The text can be generated at varying complexity such as sentence, word and character level so as to suit the specific needs. Sentence-level text generation analyzes the text in detail and determines the relationship between each sentence and the context of those sentences. The goal of word-based text creation is to study sequence structure and forecast the likelihood of next word in a text given. Text generation in character level is a similar approach, when it's performed, the character is recognized by the model instead of identifying the complete document.

Deep learning is a sub-branch of machine learning and it benefits from representation learning and the neural networks. Loads of enhancements have been done on the methods of deep learning. Not only the methods of the deep learning, but also there are a lot of improvements have been done in computational sources, because of that reasons, text generation by machine learning is became doable. Thus, models of deep learning in order for performing text generation can be examined under three subheadings, which are Sequence to sequence model, Sequence Vector Model and Vector Sequence model.

Among those three models the most suitable one for text generation purpose is Sequence to Sequence model. While generating passwords through the program we write, we also use this Sequence to Sequence models. Seq2seq is an abbreviation of this model. This model doesn't contain any fixed size of input and output, they are variable. In addition, seq2seq model comprehends Language translation (Dusek et al. 2020).

A model may contain fixed-size vector either on its input or output. Supposing that the model contain fixed size vector on its input, it is called Vector Sequence Model and its output varies, if it has got the fixed size vector on its output it is called as Sequence Vector Model and its input varies. (Toshevskaa et al. 2020, Elmaghrabhy and Abdelwahad 2018). It is possible to generate caption for images through Vector Sequence Model. On the other hand, classifying can be provided through benefitting Sequence Vector Model.

In order to perform different tasks of natural language processing (NLP) and generation of texts; deep neural networks(DNN), so, deep learning(DL) have been used. Examples can be given for those various tasks of NLP, such as; generating texts, text classifica-

tion, question answering, script writing, machine learning (ML) translation, analysis of sentiments, suggesting texts and predicting next words during chatting, poetry writing, summarizing texts, balancing the datasets.

Especially with the development of deep-learning architectures and the effective methods in order to train massive volumes of training text data, in the field of machine learning based text generation outstanding breakthroughs have been accomplished. The mentioned effective methods simply pick up knowledge from input training text's structure. Therefore, any external knowledge on the data's structure or the domain are not required. Those methods are known as fully data driven methods.

Development in several key directions, including the following, has propelled significant improvements in the area of study:

Firstly, the development of the attention mechanisms is extremely crucial. A token may be a letter, a word or a sentence. In addition, a textual environment can be words, sentences, and paragraphs. Assuming that if there are tokens in such kinds of environments, the main purpose is to provide a flexible understanding of the meaning by connecting the tokens with textual environment's other elements. So, if the flexible notion of context is provided, the learning model would be able to understand the structure of the text such as semantic meaning, structure of the words and the grammar. Therefore, it causes enhancements on producing text, classifying text and interpreting text as a result. Several types of Transformers architecture, GPT models, ELMO, BERT are considered as leading attention-based models.

Secondly, the other essential thing is the architectures of models and their abilities of representation. By way of illustration, in Convolutional Neural Network based ResNets, Wide-Resnets, RNNs, and Generative Adversarial Network(GAN) models, significant improvements have been made in deep learning structures. So that, it became possible to form deep learning structures more flexibly. The prosperity of deep neural networks (DNN) depends significantly on their representation ability. In other words, deep neural networks (DNN) are able to build data's proper transformation which makes the data possible to process easily and figure out the specific problem. Extracting the semantic meaning and compression can be given as an example in order for the data's proper transformation.

In this context, thanks to autoencoders, a central class of deep learning models is provided. The task of autoencoders is to perform both encoding and decoding operations. Encoding process is to make a meaningful and useful data representation and transformation. The decoding process means going back and recreating the initial data from the representation. As a result, new data can be created by sampling the points in the representation space and then returned by decoding.

Another important point is the developments regarding how the training operation will be carried out. The methods mentioned in the first two items above would not be as efficient as they are now if there were no existing methods for choosing the parameters and weights of neural networks. These contains, among others, appropriate momentum and annealing-driven stochastic gradient descents, Wasserstein regularization and variational approaches.

In this study, we will carry out our investigation into data-driven deep learning text generation techniques in order for generating and enriching password-dictionaries which are also known as wordlist.

In order to reach out the system of a machine or accounts of them various techniques are able to be used by users. On the other hand, users tend to configure solid passwords so as to provide it. The reason why the users prefer using password authentication is applying this method is practical and easy to use. Another reason is developers and users of a systems are familiar with password-authentication technique. Therefore it is also easy to carry out and set up this technique into the systems.

According to the findings of the research of Dai et al., Chanda and Dell'Amico et al., while the users forming their password credentials, they have habit of utilize strings or dates which are memorable and having importance for them, such as birth dates and names. On the other hand, users form same password in order to connect different websites (Dai et al. 2019, Chanda 2016, Dell'Amico et al. 2010). It is against the security of systems and it may causes vulnerabilities in the systems.

There are two crucial notions of passwords guessing which is a progressive research area. Those are evaluating the huge password leak dumps statistically and building the algorithm so as to produce passwords. Passwords have formed by the users of a system contains particular structure and patterns. Consequently, it raises the question in minds

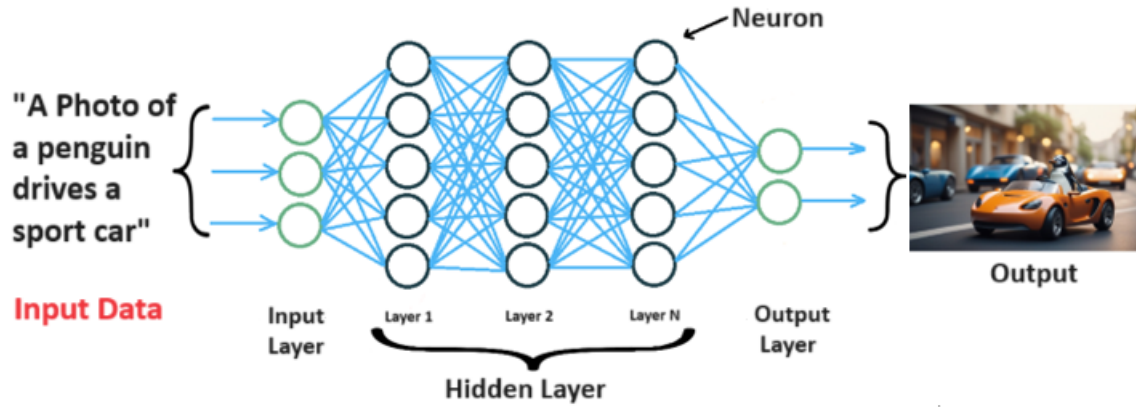


Figure 3.2. The Process Of Image Production from Natural Language Text

that can a machine form and simulate the structures and patterns generated by users and produce the passwords which a human-being may have selected in the real life.

3.4.3. Deep Neural Networks

It should be pointed out that, the problems which a decade ago they were considered as not able to be solved in the near future, have started to be solved by applying deep learning much more easily, thanks to the developments in the deep learning and AI technology. Through the year 2022, improvements have been made in deep learning progress, as well as generative deep learning. By implementing deep learning methods in order for producing new types of data which never existed in reality before has also become possible. For instance, using deepfake, it's been able to generate video content that never existed actually.

The deep learning method accelerates at a rate faster compared to the past and this technology is innovative and making improvements in several fields such as robotics and medicine and many more. On the other hand, deep learning techniques are capable of not only producing faces' images, but also it is able to be generated fully synthetic environments where autonomous vehicles are possible to be trained totally in simulation. And then, in the real world implementing them on full-scale vehicles without any problem is the key factor. Using natural language and human speech patterns as input, deep learning can produce content directly from human speech prompts which are conveyed to the model.

Deep learning models are capable of reasoning about the prompts in natural language such as English. And then, the model guides and controls what is generated according to what is specified as an input. Those models are known as text-to-image models.

Even though text-to-image models has been started to develop in the 2010s, the outcomes of the models started to resemble authentic and real photos' and hand-drawn photos' quality in the year 2022, thanks to the enhancements in deep neural networks. Midjourney Inc.'s Midjourney model, Stable Diffusion developed by StabilityAI, DALL-E 3 by OpenAI and Imagen which is advanced by Google Research's Brain Team are considered as text-to-image models in the state of art (Hanna 2023, Wang and Vastola 2022, Sasijaran et al. 2023, Betker et al. 2022, Saharia et al. 2022).

Text-to-image models associates a language model with a generative image model. In this compound, the language model transform the text input to a latent representation, while the generative image model generates image which is formed according to that latent representation. As a matter of fact, the most efficient models have been trained using vast volumes of text and image data that have been collected from the internet. Furthermore, images that have never existed in real life are able to be generated by applying those models.

Figure 3.2 highlights that, when an input text, which is a photo of a penguin drives a sports car, is given into the deep neural network model. This input transforms to a latent representation and the generative model will produce a matched image depending on the latent representation. The model can actually imagine and hallucinate what this photo may look like even though the photo has never occurred before. Assuming that, there isn't any realistic photo related to a penguin drives a sport car because the penguins are not able to drive a car, so it is not possible to be happen in real life. So, even if there isn't any training data exist which is able to be utilized, the model imagines and hallucinates from other pictures such as penguin pictures and car pictures separately in the training dataset, and produces the final output image which is a penguin drives drives a sports car. The examples mentioned above highlight how far deep learning methods improved.

Figure 3.3 represents the relationship of artificial intelligence, machine learning and deep learning with each other. The competence of processing information which is able to be utilized in order for notifying future decision or the action which is going to be taken,

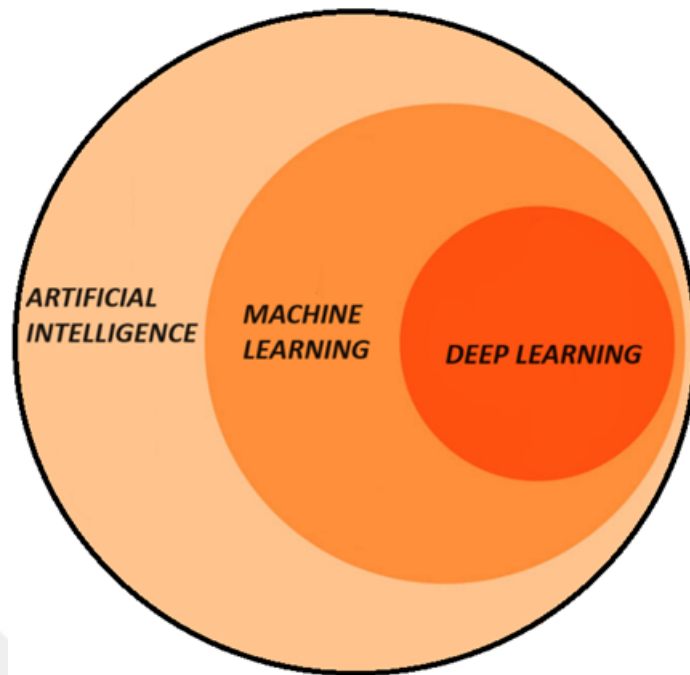


Figure 3.3. The Relationship of Artificial Intelligence, Machine Learning and Deep Learning with each other

is known as intelligence. It is possible to build AI algorithms which is the information is processed in order for informing future decision. Machine learning is a subset of AI that concentrates on how a machine can be built and taught in order to perform this process from some experiences or data.

On the other hand, deep learning is subset of machine learning that aims decidedly on neural networks. And it targets constructing neural networks which are able to extract features in the data. Those features are able to be considered as patterns that appears within the data. Therefore, it is capable of learning so as to complete tasks as well. Not only this, but also it teaches computers the way of learning a task from raw data directly. Building and teaching computers to learn many different type of tasks directly from raw data is essential. When deep learning projects are developed and very large neural networks are trained which may be able to require loads of computing power, GPU is a hardware which plays a crucial role.

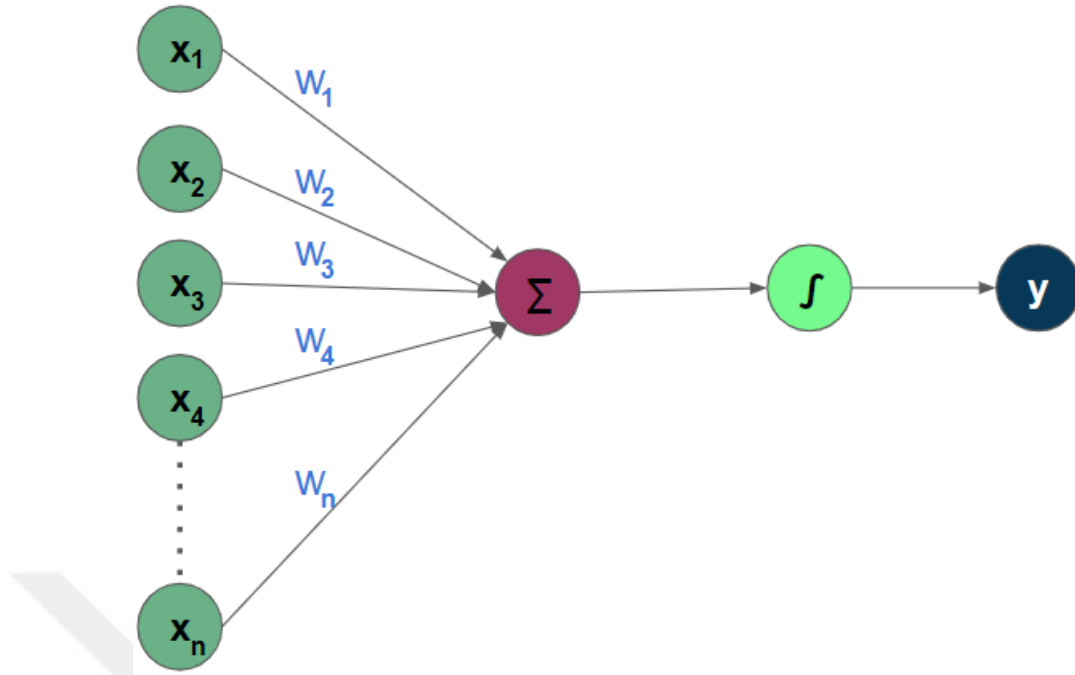


Figure 3.4. A perceptron

3.4.4. Perceptrons

As the name suggests, artificial neural networks(ANN) consist of neurons. Therefore, neurons are the essential building block of neural networks. A single neuron is referred to as a perceptron in the literature. Perceptrons are known as structural building blocks of deep learning architecture.

In the Figure 3.4 a perceptron is illustrated. It consists of five main elements such as input values, weights, summation operation, providing non-linearity, and the output. In order to process data, a set of inputs is given into the perceptron. It is possible to observe that the perceptron takes the amount of n various inputs from x_1 to x_n . The mentioned inputs are represented with green circles in the Figure 3.4. And every input has its own weight, at the Figure 3.4 above, those are denoted as w . There are n weights from w_1 to w_n that exist in the figure.

Mentioned weights are shown with blue colors. During the summation operation, each of those inputs are multiplied by the related weight. And the results of multiplications are summed. To put it another way, x_1 are going to be multiplied by w_1 , x_2 are going to be multiplied by w_2 , x_3 are going to be multiplied by w_3 , x_4 are going to be multiplied by w_4

and x_n are going to be multiplied by w_n . After a while, all of those multiplication results are going to be summed. That's how the summation operation works in a perceptron.

After summation operation has been performed, it should be provided the non-linearity for the deep neural network. So as to perform that, the result of summation pass through a non-linear activation function for the purpose of providing the non-linearity. After that step, the activation function creates the final output of the perceptron y .

$$y = g \left(\sum_{i=1}^n x_i w_i \right) \quad (3.2)$$

In Eq. 3.2, the summation operation expression in parentheses represents the linear combination of the inputs, while the g expression indicates a non-linear activation function. When linear combinations of inputs and the non-linear activation function specified with g are multiplied, it will give an output. This output value is equal to the output denoted by y in the mathematical equation above.

3.4.5. Bias Term of Perceptrons

An artificial neuron is supplied with a number of inputs within the neural network, and each input has its own corresponding weights. Weight escalates the activation function's steepness. Therefore, a weight determines how quickly an activation function is going to be triggered and go into operation, while bias term is utilized so as to slow it down and delay activation function's triggering.

The weight indicates how effective a specific input is. The network will be impacted more by input with a higher weight. Conversely, bias is comparable to the addition of the intercept in a linear equation. It is an extra parameter in the neural network that is used to modify the output in addition to the neuron's weighted sum of inputs. As a result, bias is a constant that aids in the model's optimal fit to the provided data.

In Figure 3.5, it is represented that along with multiplying all the inputs with their corresponding weights, a bias term was added additionally which is denoted as w_0 . As stated in the Figure 3.5, w_0 is the bias term and is subjected to summation operation after being multiplied by 1, due to the fact that w_0 is a scalar weight and it comes with an input of number 1.

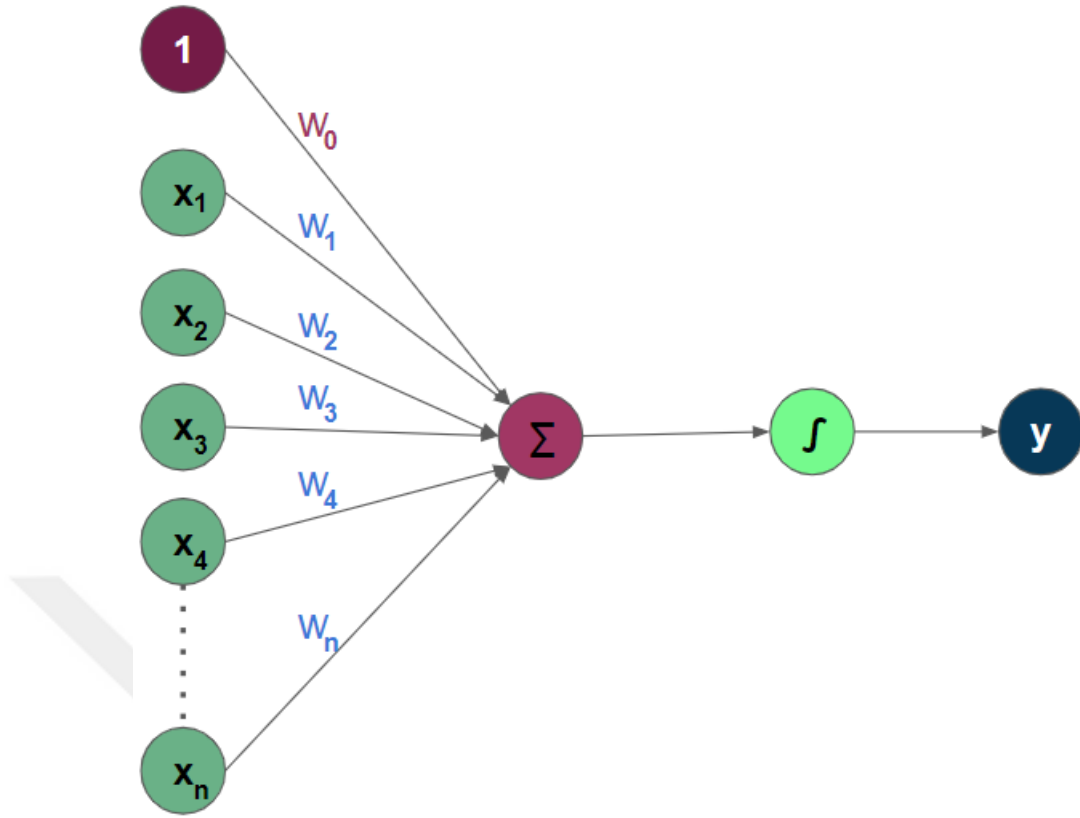


Figure 3.5. A Perceptron with Bias Term

By adding a bias term into the perception, the network is enabled to shift the nonlinear activation function in a non-linear way when it sees its inputs.

The Figure 3.5 above represents a perceptron with a bias term is added. The bias term and its corresponding weight value both are represented by burgundy color. A total value is obtained by multiplying the input values with the relevant weight values, then the value w_0 is added to this total value. In this section, the summation operation is completed. After the bias value and linear combination of inputs are collected, they are subjected to a non-linear activation function. In the next step, an output value is obtained from here. If it is wanted to transform how the perceptron works, shown in Figure 3.5, into a mathematical equation, the following equation can be formulated.

$$y = g \left(w_0 + \sum_{i=1}^n x_i w_i \right) \quad (3.3)$$

In Eq. 3.3, w_0 denotes the bias term, and the linear combination of input is the summa-

tion operation process. g is a non-linear activation function. Finally, y denotes the output.

The linear equation mentioned in Eq. 3.3 can be rewritten using linear algebra methods such as vectors and dot production. For the sake of example, let's assume that all input values from x_1 to x_n are defined as large vector X . On the other hand, let all weight values from w_1 to w_n be large vector W . It can be written as the matrix multiplication of Large vector X and Large vector W as follows. These two matrices are subjected to the dot product process. The dot product operation not only multiplies the matrices but also adds the resulting terms together. After the dot production process is completed, the bias term specified by w_0 is added. In the next stage, the result is subjected to a nonlinear activation function denoted by g in the Eq. 3.3. As a result of these operations, output y is obtained. This process, described in Eq. 3.4, is expressed mathematically.

$$y = g (w_0 + X^T W)$$

$$\text{where : } W = \begin{bmatrix} w_1 \\ \dots \\ w_n \end{bmatrix} \text{ and } X = \begin{bmatrix} x_1 \\ \dots \\ x_n \end{bmatrix} \quad (3.4)$$

3.4.6. Contributing Nonlinearity By Using Activation Functions

The non-linear function to which the outputs of a neural network must be passed before sending them as input to the next neuron in the line is called the activation function. The sigmoid function, which is a well-known nonlinear function in deep neural networks, can be given as an example. This can be thought of as a kind of continuous version of the threshold function. All the activation functions are considered nonlinear functions.

Figure 3.6 delineates the formulation of a sigmoid function, subsequently generating a set of x values through the utilization of NumPy's `linspace` function. The corresponding y values are then computed by applying the Sigmoid function. Ultimately, Matplotlib is employed to visualize the graph, with the red dashed line serving as a representation of the threshold value at $y=0.5$.

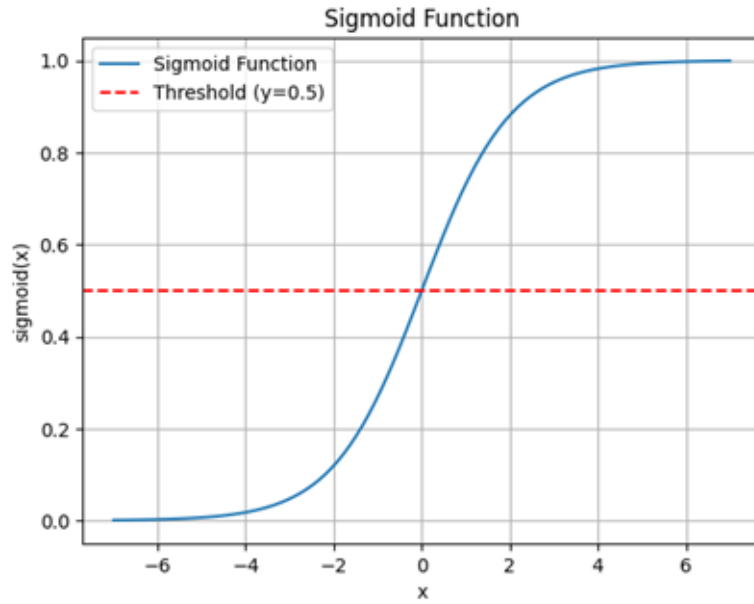


Figure 3.6. The Sigmoid Activation Function Drawn Using Matplotlib

As can be seen from the Figure 3.6, the Sigmoid function can take any real number value on the x-axis as input. The value taken as input is passed through in the Eq. 3.5 and produces an output on the y-axis. The output values of the sigmoid function are between 0 and 1, as seen in the Figure 3.6. As a matter of fact, in deep neural networks, there are several types of nonlinear activation functions such that the Sigmoid Function, Rectified Linear Unit(ReLU), and Hyperbolic Tangent exist.

Since sigmoid activation functions produce outputs between 0 and 1, they are very useful when working with probability distributions because probabilities are between 0 and 1.

$$g(z) = \sigma(z) = \frac{1}{1 + e^{-z}} \quad (3.5)$$

Due to the fact that it's a piecewise linear activation function in deep neural networks, ReLU is immensely efficient in terms of computing compared to computing against the derivatives of it. Besides, one non-linear ideal zero, its derivatives are the constants.

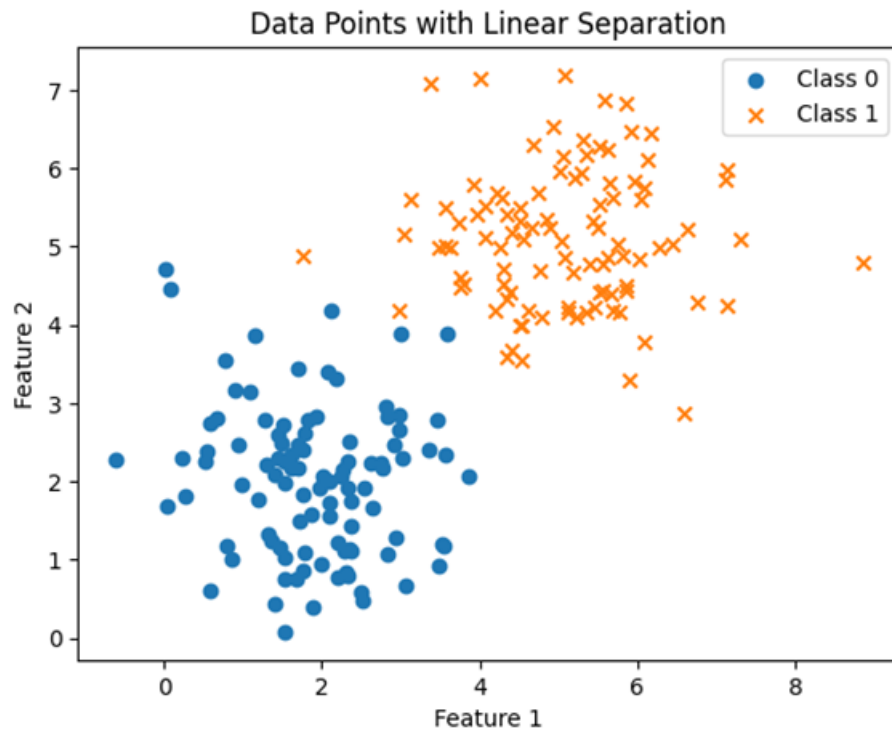


Figure 3.7. The Data Points With Linear Separation

The role of the activation function is to provide non-linearity to the network.. A case is presented so as to illustrate this concept. Assume that, it is asked to form a neural network with the purpose of separating blue and orange point from each other. Figure 3.7 represents the data points with linear separation, on the other hand Figure 3.8 represents the data points with nonlinear separation.

The purpose of employing a nonlinear activation function is to introduce nonlinear elements to the data which are used. Due to the fact that almost all real world data exhibits a high degree of nonlinearity, nonlinear activation functions are required. To process those kinds of datasets effectively, nonlinear models are required. Thus, they will be able to capture same kinds of patterns.

The datasets given in Figure 3.7 and Figure 3.8 are intended to be classified and separated from each other. Assume that the orange and blue points are divided into two different data point types in the given data sets. However, imagine that this classification process will be done using only a single line. For Figure 3.7, separation can be done with a single single line without any problems. However, performing this operation poses a problem for Figure 3.8. This cannot be done effectively when a single single line is used.

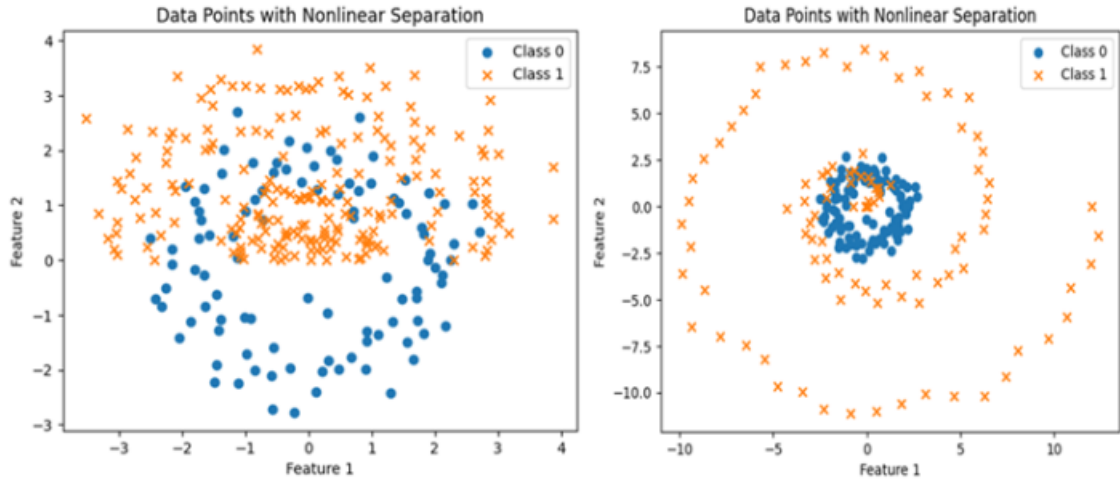


Figure 3.8. The Data Points With Nonlinear Separation

However, if a nonlinear activation function is used to break linearity, this problem can be overcome and classification can be performed. Nonlinear activation functions allow the processing of data of nonlinear data type, which makes neural networks inherently very powerful. Figure 3.7 shows the activation functions that are linear in nature yield decisions that remain linear, irrespective of the network's size while Figure 3.8 illustrates the introduction of non-linearities enables to approximate functions of arbitrarily high complexity.

So as to grasp the idea behind it, a perceptron example is given on the Figure 3.9. On the figure, there is a trained neural network with weights exist again, but this time numbers are given to the weights rather than w symbols on the edges. In order to represent a 2-dimensional line example on the Cartesian coordinate system, this time, only 2 input values and bias values should be given to the perceptron. The trained weight w_0 is set to 1, and W is represented as a vector with elements -4 and 8. As previously mentioned, this neural network consists of two inputs, namely x_1 and x_2 . To obtain the output, it is crucial to emphasize that the process involves three essential steps.

As a first step, in order to be calculated, the input values x_1 and x_2 are subjected to multiplication operation with their corresponding weights. After the operation ends, their results of multiplication, and bias term are added to each other. And then, the non-linearity is calculated through an activation function. Applying those three essential operations is known as the forward propagation of information through a perceptron.

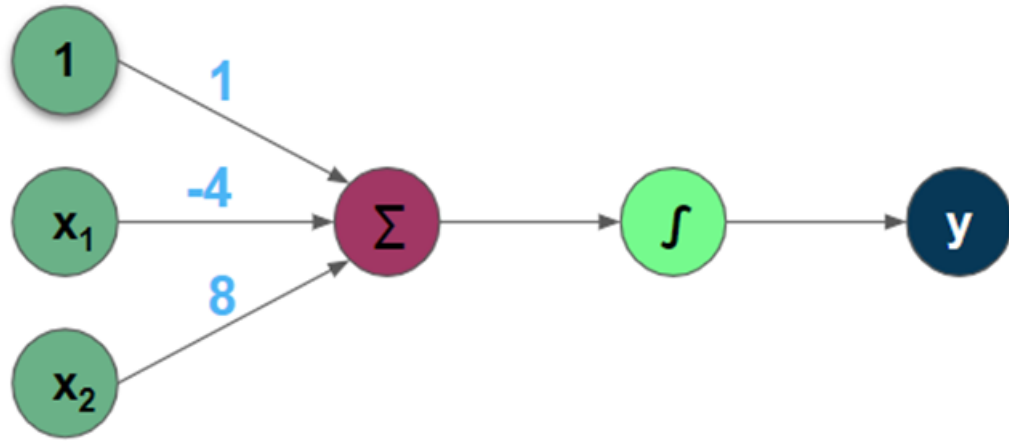


Figure 3.9. A Perceptron With Weights for Nonlinear Separation Example

If the numerical values of the weights are substituted into the equation, they are added with a bias term (w_0) after applying the dot production process for the inputs (x_1, x_2) and weights (w_1, w_2). Then, they are multiplied by an activation function (g) to achieve nonlinearity. For example, let's use sigmoid as the activation function g . If the sigmoid activation function formula in Eq. 3.5 is used instead of the g activation function in Eq. 3.4, the process can be carried out as follows.

$$y = \frac{1}{1 + e^{-(w_0 + X^T W)}}$$

$$\text{where : } W = \begin{bmatrix} -4 \\ 8 \end{bmatrix} \text{ and } X = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \text{ hence : } z = \left(1 + \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T \begin{bmatrix} -4 \\ 8 \end{bmatrix} \right)$$

$$y = \frac{1}{1 + e^{-(1 - 4 \cdot x_1 + 8 \cdot x_2)}} \quad (3.6)$$

The component to be multiplied by the nonlinear activation function actually represents a two-dimensional line. With two inputs, if it is considered that the space encompassing all potential inputs the neural network may encounter, it can be graphically represented this as a decision boundary. In a two-dimensional context, this line functions as a decision boundary or plane, effectively distinguishing between two regions within the input space. Furthermore, not only this is a single plane represented in the Figure 3.10,

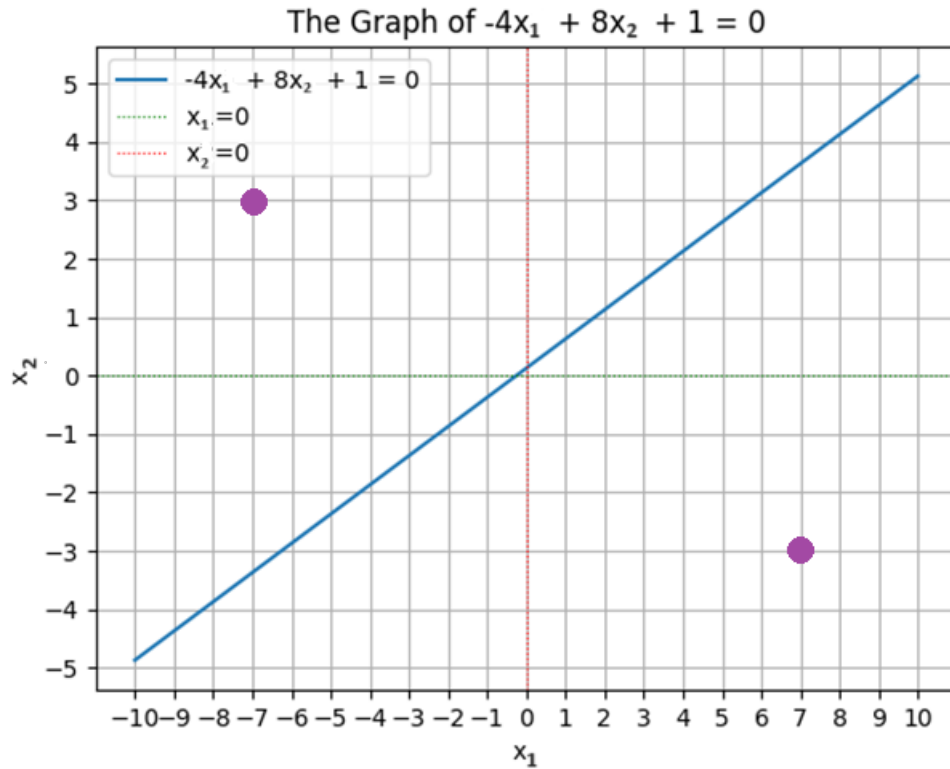


Figure 3.10. The Decision Boundary Line

but also it is considered as a directionality component in terms of which side of the plane that a point was deployed.

Assuming that, there is an input where $X=(-7,3)$. The location of this point corresponds to the 2nd region on the coordinate system. When these input values are passed through the activation function, there will be an output of certain type. This certain type of output will be positive. Positive input values are given to the activation function. To understand this situation, when these components are written in their places in Eq. 3.6, a positive number passing through the nonlinear component is obtained. If the input point is on the other side of the space, an opposite result is obtained. Here g activation function is sigmoid, and that gets propagated through as well

$$y = g(1 - 4 \cdot x_1 + 8 \cdot x_2)$$

$$y = g(1 + 28 + 24)$$

$$y = g(53)$$

$$z = 53, z > 0$$

$$y = \frac{1}{1 + e^{-53}}$$

$$y = 1, y > 0.25$$

For example, let $X=(7,-3)$ be given as input from the fourth region of the Cartesian coordinate system. For this time, negative input values are passed through the sigmoid activation function. When the specified X input is calculated for Eq. 3.6, the results are as follows:

$$y = g(1 - 4 \cdot x_1 + 8 \cdot x_2)$$

$$y = g(1 - 28 - 24)$$

$$y = g(-51)$$

$$z = -51, z < 0$$

$$y = \frac{1}{1 + e^{51}}$$

$$y = 7.095474162157617e - 23$$

$$y = 0.000000000000007095474162, y < 0.25$$

Moreover, the thresholding function essentially resides at this decision boundary. Consequently, depending on the region within the space, the sigmoid function, serving as the thresholding function, regulates the directional transition to either side.

In the previous calculation example, full space could be easily visualized because it was a two-dimensional space. However, not all problems that will be encountered in the future are two-dimensional, so the data points are not two-dimensional either. Therefore, Neural Network models are used instead of perceptrons to deal with multidimensional problems.

3.4.7. Constructing Neural Networks Utilizing Perceptrons

It was mentioned that a more complex model, a neural network structure, should be used to solve larger and more complex problems. Perceptrons, also known as neurons, are

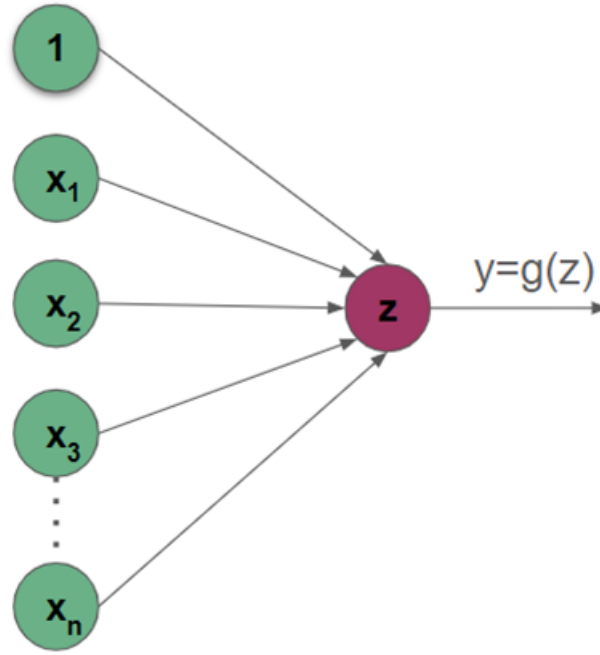


Figure 3.11. Constructing Neural Networks Utilizing Perceptrons

used to build a more complex model, a neural network. The neural networks consist of a lot of neurons, also known as perceptrons

A perceptron works and propagates information to its decision. In a neural network, it consists of perceptrons, and those perceptrons provides forward propagation by doing dot product firstly, adding bias to it secondly and providing non-linearity operation by applying activation function to the result and this mentioned process is being repeated for each perceptron in the neural network. Figure 3.11 represents a perceptron which is used for constructing more complex neural networks. Each line in this context is linked to an associated weight scalar. Furthermore, every line corresponds to an input, and it incorporates a specific weight associated with the input along the line itself, z represents the result of a summation of the dot production and bias. The result z is given to the non-linear activation function. At this point, the final output will be the result of g . Here, g is z 's activation function. The dot product plus bias result is denoted as z . On the other hand, z is been able to be considered as the state of the neuron. In Eq. 3.7, z is calculated by using that way.

$$z = w_0 + \sum_{i=1}^n x_i w_i \quad (3.7)$$

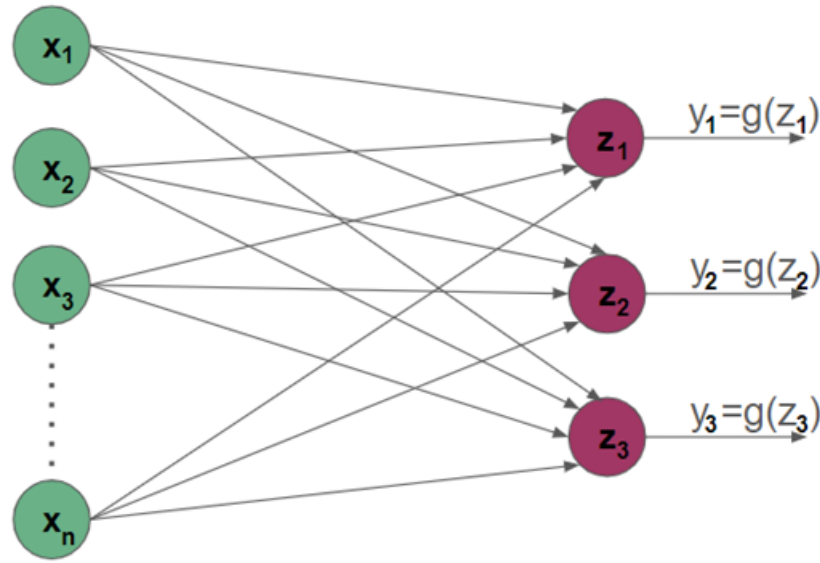


Figure 3.12. Perceptron Architecture That Has Multiple Outputs

In order to create a neural network, the output layers consisting of z 's must be multiplied. At this stage, a multi-layered output neural network structure will be created. For example, assuming that it is asked to create a structure with two outputs by updating the structure given in Figure 3.11. Two perceptrons, that is, two neurons, are used to create this structure.

Each perceptron is responsible for determining the output for its associated segment. Consequently, with two outputs, each functioning as a standard perceptron. Perceptron take all of the inputs, so that, those two perceptrons both take the same inputs. With this understanding, neural network architecture is constructed from perceptrons. All inputs are densely connected to all outputs. For this reason, these layers are called Dense Layers.

The perceptron structure with two outputs is represented in Figure 3.12. To establish the specified structure, the vector of weights is first initialized. During this process, dot production is performed for the input and weight values. After this process, the bias vector is collected with the previous dot production of all the inputs by the weights process. In the next step, the activation function is applied and the result is returned. By doing this, the forward propagation of information is performed. This defines single layer neural network layer. These operations are performed using Eq. 3.8 given below.

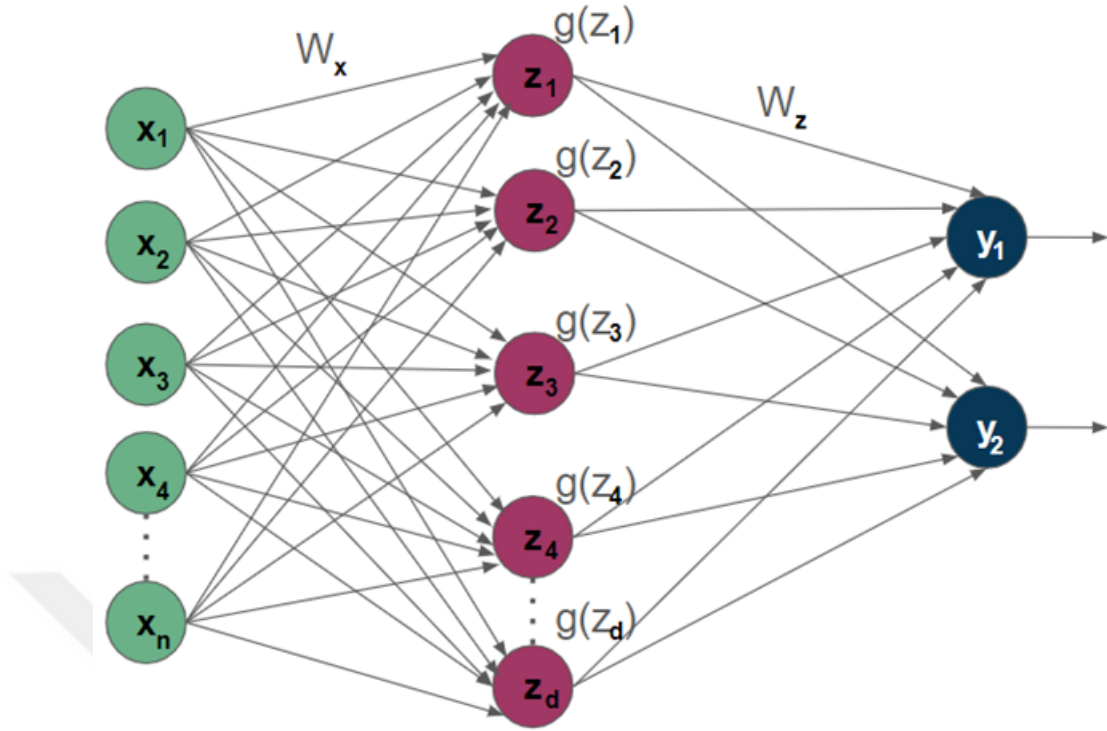


Figure 3.13. Neural Network Architecture with One Hidden Layer

$$z_i = w_{0,i} + \sum_{j=1}^n x_j w_{j,i} \quad (3.8)$$

With this comprehension, it is explained the construction of a single layer. However, it is important to note that it is able to be extended this concept by considering how to systematically stack multiple layers in the network.

Let's assume that in the perceptron structure given in Figure 3.12, propagation of information is carried out from inputs to hidden outputs, rather than directly from inputs to multioutput nodes. And the outputs from the nodes in this hidden layer will go to the nodes in the output layer and there will be output values in the output layer. In this way, a hidden layer, that is, a new dimensional space, was added. In order for the perceptron structure to create a neural network, there must be at least one hidden layer. The neural network architecture containing a single hidden layer is visualized in Figure 3.13. The model is going to be learned by the training process of the model, and this phase enables the model to understand how to map the inputs to the desired outputs.

If any single node in the middle layer is examined in the single hidden layer neural network model given in Figure 3.13, let's take node z_1 as an example. The output of the z_1 node is being calculated by applying dot production, adding bias and initialize non-linear activation. function between all of its inputs. If z_4 , another node in the middle layer, is considered, the same inputs will be received for this node as well. But this time, node z_4 will have a different weight matrix set compared to node z_1 , and dot production will be performed by applying this weight set to the inputs. Thus, it will have different output values from the other node z_1 . Although different numerical values must be calculated for each node, the following mathematical formula suits for this process.

$$z_i = w_{x_0,i} + \sum_{j=1}^n x_j w_{x_j,i}$$

$$y = g \left(w_{z_0,i} + \sum_{j=1}^d g(z_j) w_{z_j,i} \right) \quad (3.9)$$

3.4.8. Building Deep Neural Networks

In order to provide learning process, the deep neural network architecture utilizes various layers for information processing stages within hierarchial structures. Along with the input and the output layers, deep neural networks (DNN) have got multiple hidden layers. If there is only one hidden layer in the neural network, it is called as Shallow network. On the other hand, if the amount of the hidden layers in the neural network equals two or more than two, it is considered as a deep neural network (DNN). There are essential four main learning tasks exist in the scope of learning process such as Unsupervised, Semi-supervised, Supervised and Reinforcement learning. Unsupervised learning works with unlabeled dataset, while supervised learning operates with training data which is labeled one (Sarker 2021).

If many hidden layers are added side by side to the intermediate layer, one after the other, forward propagation of information is provided not only at the neuron level, but also at the layer level. All layers are fully connected to the next layer. And the output of the previous hidden layer is the input of the next hidden layer.

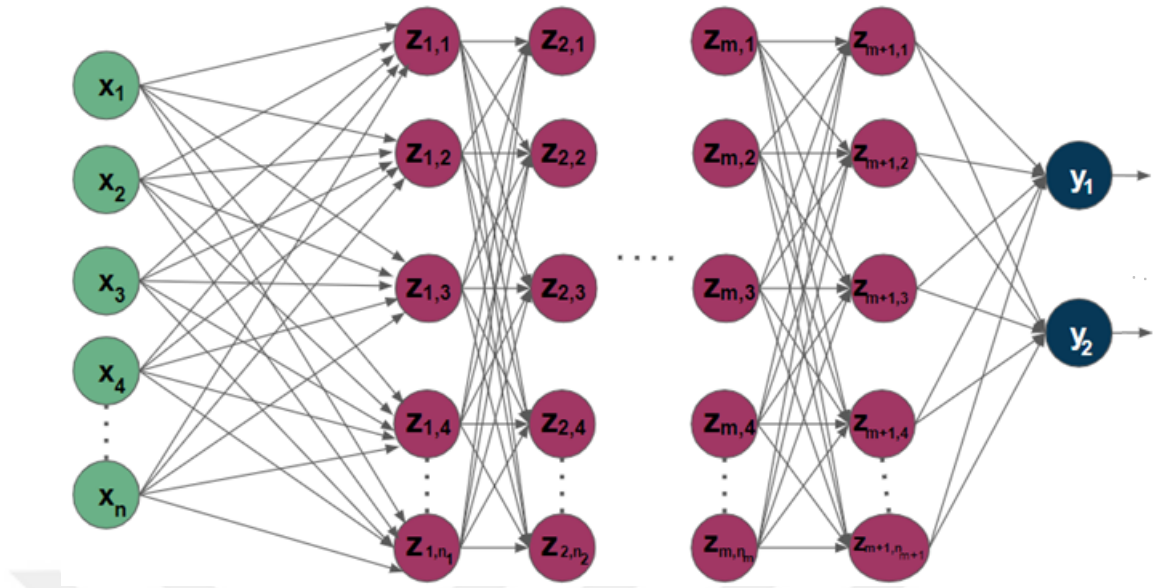


Figure 3.14. Deep Neural Network Architecture With Multiple Hidden Layers

So as to construct a very deep neural network, the number of hidden layer units in the intermediate layer must be increased. These hidden layers produce output. By applying forward propagation of information from the input layers through the hidden layers in the intermediate layer, it is reached deeply to the last layer, that is, the output layer. Output layer has the final prediction it is requested to output. It is able to construct more deep neural networks by increasing the amount of hidden layer units and creating more hierarchical models. Figure 3.14 illustrates the deep neural network (DNN) architecture with multiple hidden layers. The nodes in a hidden layer unit can be increased by adding new nodes from top to bottom. At the same time, a new hidden layer unit can be added to the middle layer to add depth to the model from left to right.

3.4.9. Recurrent Neural Networks

Deep Neural Networks(DNN) and Convolutional Neural Networks (CNN) are not suitable in order to process the sequential also known as temporal inputs of the dataset. If it is required to work with the temporal data, Recurrent Neural Networks (RNN) are applicable. For instance, audio, video, and text can be considered as sequential data. Recurrent Neural Networks are able to update their current state by considering the past states in the network and the current input data, thanks to their cyclic connection.

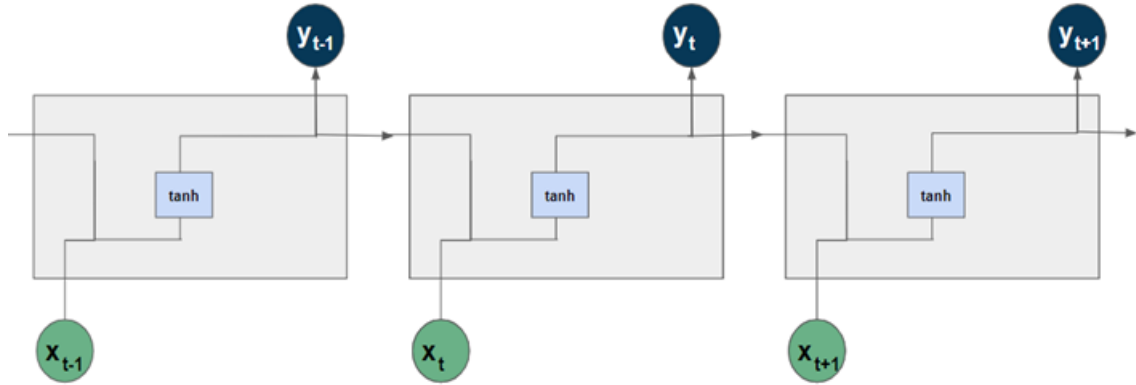


Figure 3.15. Recurrent Neural Network

On the other hand, RNN architecture is not suitable for reasoning and relate the pieces of information which are relevant to each other in case of another piece of information related to one piece of information is located far away in the dataset, RNN is not competent in associating these pieces of information with each other. In their paper which was published in 1997, Hochreiter and Schmidhuber offered Long Short-Term Memory (LSTM) in order to solve this problem. This problem is known as long-term dependency in the literature (Hochreiter and Schmidhuber 1997).

The Recurrent Neural Network architecture has input layer, hidden layer and output layer. The hidden layer is known as recurrent layers in this architecture and it is composed of recurrent cells. A recurrent cell unit's states are controlled and influenced according to the information of past states and in terms of the current input with feedback connections. The Recurrent cells forms the Recurrent neural network. On the other hand there is an activation function exists inside of the cell and it is used in order for breaking the linearity and providing non-linearity into the network model. Figure 3.15 above illustrates the recurrent neural network model with $\tanh$ activation function exists inside the recurrent cell unit.

In the given Figure 3.15 above, the recurrent cell unit takes input x_t at an existing time, and the dot production is applied for that input with its corresponding weights. In addition to that, there is also recurrent information which comes from the previous cell exists and it is denoted by h_{t-1} . It is also a dot production is applied for the recurrent information h_{t-1} with its related weights. After a while, the first dot production of x_t and the second dot production of h_{t-1} is put into the summation operation.

After that, a bias term is added to the result of summation. For the next step, an activation function is applied to the result of summation. Here in the Figure 3.15, the result of the $\tanh$ function is going to represent y_t which is the output of the recurrent cell unit at that mentioned existing time t . Furthermore, that result is also represent the recurrent information h_t . It is going to be sent to the next recurrent cell unit in the recurrent neural network. Assuming that the weight vector of input x_t is represented as W_x and the weight vector of input h_{t-1} is denoted as W_h . The mathematical formula for that operation would be like given below in Eq. 3.10:

$$h_t = \tanh(x_t W_x + h_{t-1} W_h + b)$$

$$h_t = y_t \quad (3.10)$$

On the other hand, the standard recurrent neural networks which include those recurrent cells are not feasible when they encounter with long-term dependencies. In their works, Hochreiter (Hochreiter 1991) and Bengio et al. (Bengio et al. 1994) investigated the main causes of the problem of occurring long-term dependencies. In the recurrent neural network architecture, when the error signal flows backwardly in time, it may cause a blow-up or vanishing.

3.4.10. Long Short-Term Memory Architecture

In order to follow up, control and take care of with long-term dependencies in the data, it is necessary to use more complex and detailed version of the recurrent neural unit. By applying in that way, an effective solution for the vanishing gradient is provided. In the year 1997, Hochreiter and Schmidhuber came up with the LSTM cell notion, they added gate into the recurrent cell and modified it in order to enhance the capacity of remembering of the traditional recurrent cell, the main purpose of adding these mentioned extra gates is to provide better results in case of the long term dependency problem. (Hochreiter and Schmidhuber 1997)

The problem is fixed by applying gating operation. Gating operation enables neural network to selectively control the information flow into the neural unit. The main purpose

of the gating operation is filtering out non-essential information as long as the crucial information is retained.

The pivotal and widely used type of recurrent unit implementing this gated computation operation is known as Long Short-Term Memory network. Long short-term memory is briefly expressed as LSTM. In order to trace and keep information along the whole of numerous time steps, LSTM networks build upon a gated cells. And the LSTM architecture is efficient at following up long-term dependencies while working with sequential data. LSTM architecture is comprised of four main component which they are Forget, Store, Update and Output part.

In addition, the model of LSTM is capable of governing the flow of information. LSTM operates by preserving and sustaining a cell state, similar to a traditional RNN, and this cell state remains distinct from what is directly outputted. As it is mentioned before, LSTM governs the information flow by utilizing gates. Therefore, the update process of the cell state is being done depending on the Gates.

By utilizing the forget gate of the LSTM architecture, non-essential and irrelevant information is dropped out and relevant and crucial information from current input is stored. The LSTM Network updates the cell state sequentially and subsequently filters this updated cell state in order to generate the predicted output, similar to the standard RNN. And the output gate in the LSTMS network returns a filtered version of the cell state. An LSTM model is able to be trained by utilizing the backpropagation through time (BPTT) algorithm. The mathematical formulation of the LSTM enables an uninterrupted flow of gradients, significantly mitigating the vanishing gradient problem. There are different types of LSTM architectures are being used while they are deployed into the deep learning models, such as LSTM with forget gate, LSTM without forget gate and LSTM which has a connection of peephole.

In their study in 2000, Gers et al. found that a forget gate could be added to the LSTM cell. (Gers et al. 2000) An LSTM architecture with the forget gate is visualized in Figure 3.16. The forget gate plays a crucial role in order to determine whether the information will be discarded from the cell state or not. The information is retained in case the forget gate's value, f_t , is 1. On the contrary, a value of 0 indicates the removal of all the information.

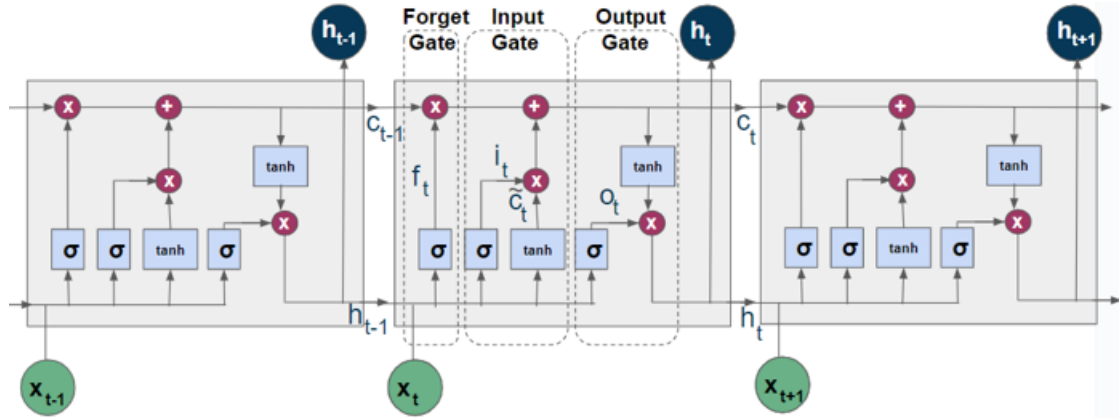


Figure 3.16. Long Short-Term Memory Architecture

The functions in the LSTM structure with the forget gate can be represented mathematically as given below in Eq. 3.11.

$$f_t = \sigma(W_{f_h} h_{t-1} + W_{f_x} x_t + b_f)$$

$$i_t = \sigma(W_{i_h} h_{t-1} + W_{i_x} x_t + b_i)$$

$$\check{c}_t = \tanh(W_{\check{c}_h} h_{t-1} + W_{\check{c}_x} x_t + b_{\check{c}})$$

$$o_t = \sigma(W_{o_h} h_{t-1} + W_{o_x} x_t + b_o)$$

$$c_t = f_t c_{t-1} + i_t \check{c}_t$$

$$h_t = o_t \tanh(c_t) \quad (3.11)$$

4. RESULTS AND DISCUSSION

When wordlists containing a large number of passwords are considered as sequential data, if the model developed using the RNN LSTM architecture is trained with these sorts of datasets, it can create many generated passwords as output, and a wordlist can be created by saving this large number of passwords into a txt file. When thought about it, this wordlist created would actually be sequential data. Within the scope of this thesis project, many-to-many RNN LSTM structure, seq2seq structure, and tensors were utilized in order to process sequential data as both input and output. Password dictionary generation experiments have been carried out with many different datasets. These datasets are: Top62WPAProbable-v3, Top62WPAProbable-v4, Top4800-WPA-probable-v2, Top4800-WPA-probable-v3, Top204Thousand-WPA-probable-v2, Rockyou62, Rockyou4800, rockyou204kv2, Mix62, Mix4800, Mix204kv2, Kaonashi62, Kaonashi4800, Kaonashi204k, machine-gen-62-dataset, machine-gen-4800-datasetv2 and machine-gen-204k-dataset.

Some of the datasets here are human-based wordlists created by cracking the passwords that were created by real human users. First, it was examined how the RNN LSTM model performed in the password dictionary generation stage when the model was trained with the mentioned human-based wordlists. The datasets used for this investigation are: Top62WPAProbable-v3, Top62WPAProbable-v4, Top4800-WPA-probable-v2, Top4800-WPA-probable-v3, Top204Thousand-WPA-probable-v2, Rockyou62, Rockyou4800, rockyou204kv2, Kaonashi62, Kaonashi4800 and Kaonashi204k.

In the next stage, when the model was trained using dictionary wordlists enriched by blending the human-generated passwords generated by the RNN LSTM model and the machine-generated passwords generated by the model, it was examined how the model performed in the password dictionary generation phase. In order to perform this experiment, the following mixed datasets were utilized during the model training phase: Mix62, Mix4800, Mix204kv2

Finally, if the developed RNN LSTM password dictionary generation model is trained only with machine-generated passwords, it was examined how it would perform in the password dictionary enhancement stage. For this purpose, the model was trained with dic-

tionary wordlists created from the password sets produced by the model and classified as machine-generated. The datasets are employed so as to conduct this experiment are as follows: machine-gen-62-dataset, machine-gen-4800-datasetv2, machine-gen-204k-dataset

4.1. Top62WPAProbable-v3 Dataset Experiments

TopXWPAProbable password datasets are real WPA-length passwords which is the format that is utilized by the routers protected by WPA/2 security. The passwords in this kind of wordlist consist of 8-40 characters long and they are created by removing entries from real password files that didn't suit the length requirements. Their length is denoted in their title. For example, Top62WPAProbable-v3 wordlist dataset contains the amount of 62 different passwords in it.

In the experimental scenario where the Top62WPAProbable-v3 dataset was used to train the RNN LSTM model, a total of 331 passwords were generated. While 314 of these generated password outputs were classified as human-like passwords, 17 of them were classified as machine-like passwords.

For the Top62WPAProbable-v3 dataset, the model was trained by running a total of 6000 iterations. The model was run for a total of 6000 iterations, and the training process was stopped at the 6000th iteration, just after the 5000th iteration, which is the optimal value where the loss function is closest to 0. Here, after the 5000th iteration, it was stopped at the 6000th iteration because it was wanted to check whether the loss value would be optimized in the next step. The loss value in the 6000th iteration is higher than the value in the 5000th iteration. However, the value where the loss function is closest to 0 is considered optimal. Therefore, the closer the loss gets to the optimal, the better the model will perform. The model was run with a total of 833 epoch values. The model was run with the hyperparameter values specified in Figure 4.17 and Figure 4.18.

Figure 4.19 indicates that the loss function value approaches optimal while the model is running. The most optimal loss value reached for this data set is 0.0039016827940940857.

While the percentage value of the humanlike password produced in the wordlist as an output set created by the model is 94.9% , the percentage value of the machine-like password in the generated output set is 5.1%

```
# Hyperparams
BATCH_SIZE = 2
SEQUENCE_LENGTH = 10
LEARNING_RATE = 0.002
DECAY_RATE = 0.97
HIDDEN_LAYER_SIZE = 1024
CELLS_SIZE = 3

TEXT_SAMPLE_LENGTH = 500
SAMPLING_FREQUENCY = 1000
LOGGING_FREQUENCY = 1000
```

Figure 4.17. Hyperparameters Value For Top62WPAProbable-v3 Dataset

```
Selected dataset: C:/Users/asuss/Desktop/crhacker/crhackDataset/Top62WPAProbable-v3/
Batch size: 2
Sequence length: 10
Learning rate: 0.002
Decay rate: 0.97
Hidden layer size: 1024
Cells size: 3
Tensor size: 120
Batch size: 2
Sequence length: 10
Batches size: 6
```

Figure 4.18. Hyperparameters Value For Top62WPAProbable-v3 Dataset

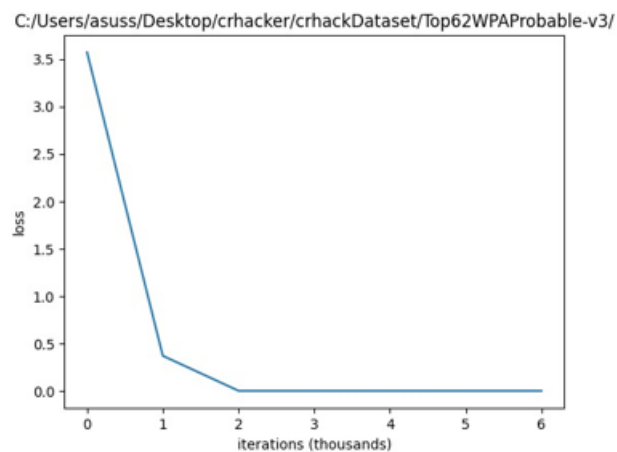


Figure 4.19. Loss Function Value For Each Iteration

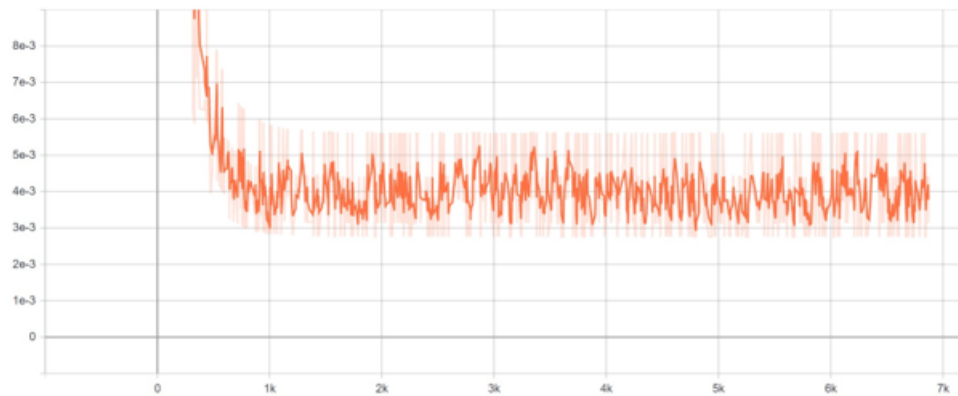


Figure 4.20. Cost Value For Each Iteration

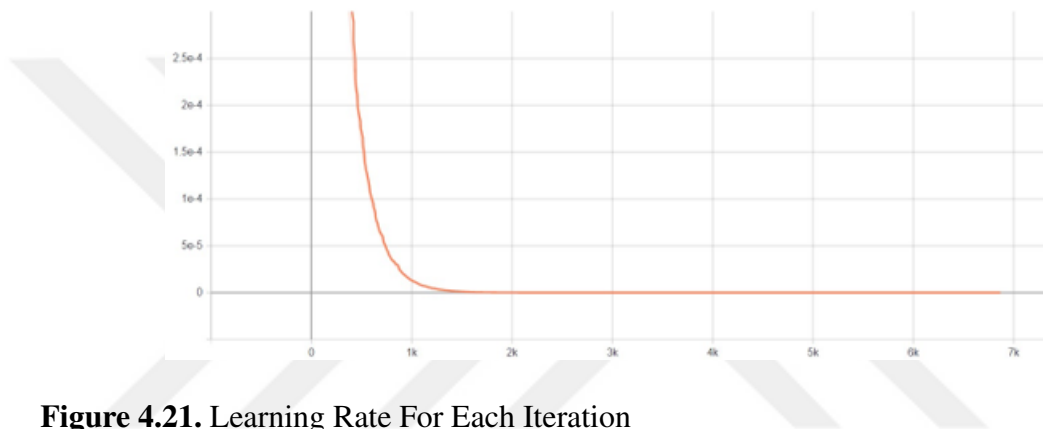


Figure 4.21. Learning Rate For Each Iteration

Figure 4.20 illustrates cost value for each iteration. So, Figure 4.21 represents learning rate for each iteration. On the other hand, Figure 4.22 shows logits for each iteration.

As can be observed from the results, the percentage of human-like passwords in the generated dictionary is higher than the percentage of machine-like passwords. When the model is trained with input wordlists created by real people, the output wordlist produces a higher percentage of passwords that real people may be able to create, compared to machine-like passwords.

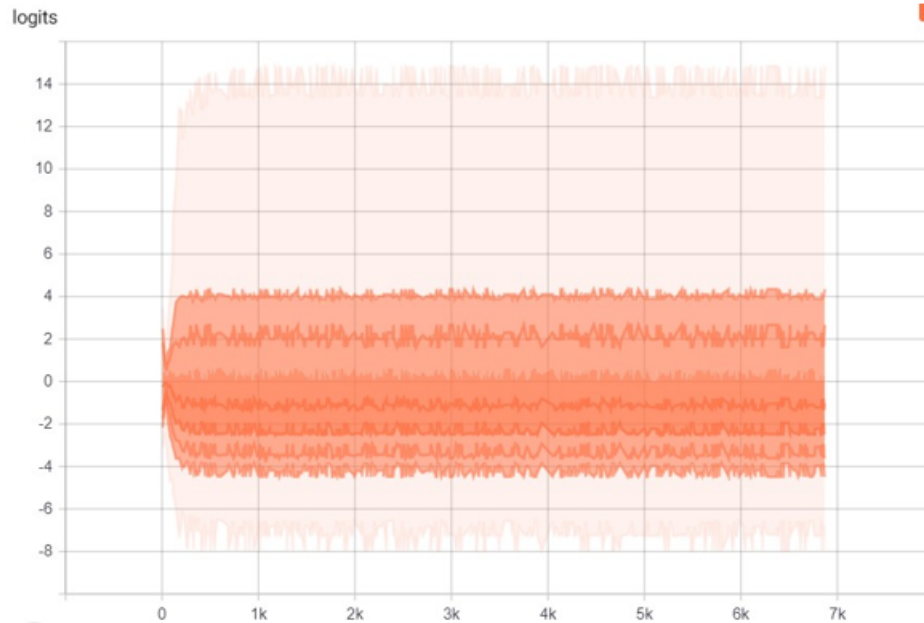


Figure 4.22. Logits For Each Iteration

Furthermore, when the model is trained using the Top62WPAProbable-v3 dataset, the following passwords can be given as examples of the generated passwords obtained as output:

Human Generated Password: 1q2w3456789

Human Generated Password: 12345678

Human Generated Password: 1q2w3e4r

Human Generated Password: sunshine

Human Generated Password: football

Human Generated Password: lootball

Human Generated Password: slnshale

Human Generated Password: llovball

Human Generated Password: llov4all

Machine Generated Password: dwd1u2p77p0m2b7a1axiv8813

Machine Generated Password: zlu6fsi4sn62x5bazh21rmoeli

Machine Generated Password: 2xtrx9fbrqh23e57m3msf3b4qq8x3wt

```
# Hyperparams
BATCH_SIZE = 2
SEQUENCE_LENGTH = 10
LEARNING_RATE = 0.002
DECAY_RATE = 0.97
HIDDEN_LAYER_SIZE = 1024
CELLS_SIZE = 3

TEXT_SAMPLE_LENGTH = 500
SAMPLING_FREQUENCY = 1000
LOGGING_FREQUENCY = 1000
```

Figure 4.23. Hyperparameters Value For Top62WPAProbable-v4 Dataset

```
Selected dataset: C:/Users/asuss/Desktop/crhacker/crhackDataset/Top62WPAProbable-v4/
Batch size: 2
Sequence length: 10
Learning rate: 0.002
Decay rate: 0.97
Hidden layer size: 1024
Cells size: 3
Tensor size: 500
Batch size: 2
Sequence length: 10
Batches size: 25
```

Figure 4.24. Hyperparameters Value For Top62WPAProbable-v4 Dataset

4.2. Top62WPAProbable-v4 Dataset Experiments

Top62WPAProbable-v4 wordlist dataset contains the amount of 62 different passwords in it. In the experimental scenario where the Top62WPAProbable-v4 dataset was used to train the RNN LSTM model, a total of 1236 passwords were generated. While 1216 of these generated password outputs were classified as human-like passwords, 20 of them were classified as machine-like passwords.

For the Top62WPAProbable-v4 dataset, the model was trained by running a total of 25000 iterations. The model was run for a total of 25000 iterations, and the optimal loss was found at the 20000th iteration, the training process was stopped at the 25000th iteration. The model was run with a total of 800 epoch values. The model was run with the hyperparameter values specified in Figure 4.23 and Figure 4.24.

Figure 4.25 indicates that the loss function value approaches optimal while the model is running. The most optimal loss value reached for this data set is 0.0010261926800012589.

While the percentage value of the humanlike password produced in the wordlist as an output set created by the model is 98.4% , the percentage value of the machine-like password in the generated output set is 1.6%

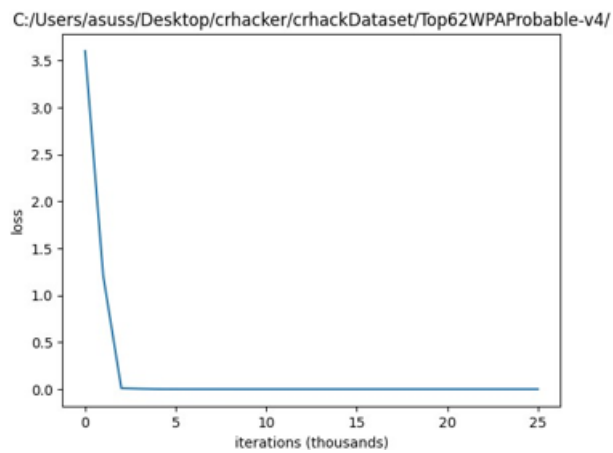


Figure 4.25. Loss Function Value For Each Iteration



Figure 4.26. Cost Value For Each Iteration

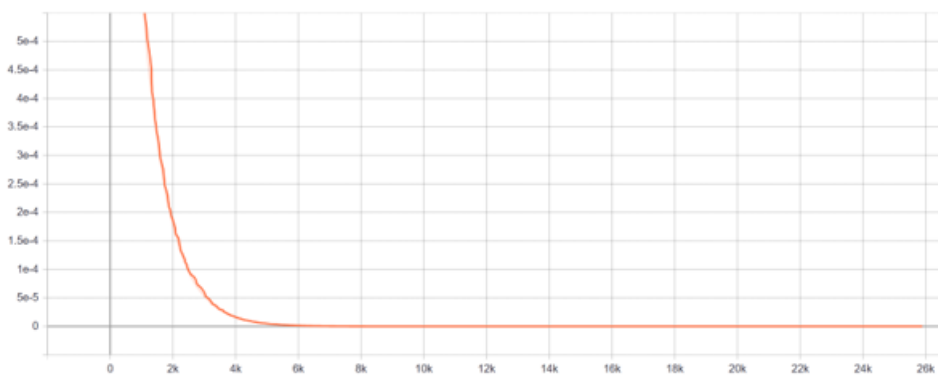


Figure 4.27. Learning Rate For Each Iteration

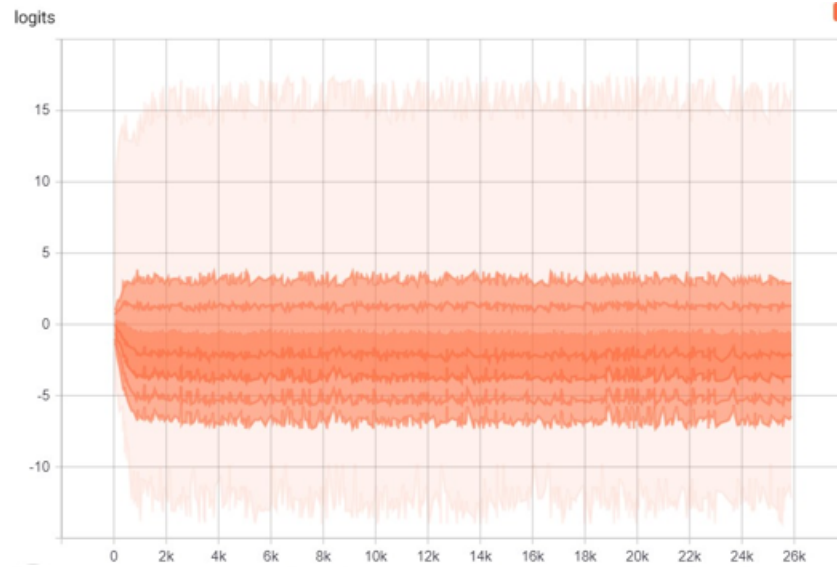


Figure 4.28. Logits For Each Iteration

Figure 4.26 illustrates cost value for each iteration. So, Figure 4.27 represents learning rate for each iteration. On the other hand, Figure 4.28 shows logits for each iteration.

As can be observed from the results, the percentage of human-like passwords in the generated dictionary is higher than the percentage of machine-like passwords. When the model is trained with input wordlists created by real people, the output wordlist produces a higher percentage of passwords that real people may be able to create, compared to machine-like passwords.

Furthermore, when the model is trained using the Top62WPAProbable-v4 dataset, the following passwords can be given as examples of the generated passwords obtained as output:

```
Human Generated Password: vefra9
Human Generated Password: qwerty123
Human Generated Password: passw0rd
Human Generated Password: blink182
Human Generated Password: a8deonp21
Human Generated Password: asdf1234
Human Generated Password: 1234qwer
Human Generated Password: qwertyuip
Human Generated Password: 1qaz2wsx
```

Human Generated Password: abcd1234
Human Generated Password: starwars
Human Generated Password: trustno1
Human Generated Password: password1
Human Generated Password: iloveyou
Human Generated Password: 1qaze9on
Human Generated Password: 1qaz2wsx
Human Generated Password: baseball
Human Generated Password: 1234567890
Human Generated Password: vq2ictoria
Human Generated Password: scorpionperl
Human Generated Password: benjamin
Human Generated Password: pissw0rd
Human Generated Password: Liverpool
Human Generated Password: slink182
Human Generated Password: qwerty123
Human Generated Password: cocacola
Human Generated Password: passw0rd
Human Generated Password: 1111lozedanat
Machine Generated Password: 76ju8559yytipmxdeg2tfr
Machine Generated Password: 752y4kud2zvjtvt5hkycoex9z
Machine Generated Password: 10b6eh2e5ob3oi4jffzeug2

```
# Hyperparams
BATCH_SIZE = 32
SEQUENCE_LENGTH = 25
LEARNING_RATE = 0.002
DECAY_RATE = 0.97
HIDDEN_LAYER_SIZE = 1024
CELLS_SIZE = 3

TEXT_SAMPLE_LENGTH = 500
SAMPLING_FREQUENCY = 1000
LOGGING_FREQUENCY = 1000
```

Figure 4.29. Hyperparameters Value For Top4800-WPA-probable-v2 Dataset

```
Please select a dataset.
Usage: python text_predictor.py <dataset>
Selected dataset: C:/Users/asuss/Desktop/crhacker/crhackDataset/Top4800-WPA-probable-v2/
Batch size: 32
Sequence length: 25
Learning rate: 0.002
Decay rate: 0.97
Hidden layer size: 1024
Cells size: 3
Tensor size: 9600
Batch size: 32
Sequence length: 25
Batches size: 12
```

Figure 4.30. Hyperparameters Value For Top4800-WPA-probable-v2 Dataset

4.3. Top4800-WPA-probable-v2 Dataset Experiments

Top4800-WPA-probable-v2 wordlist dataset contains the amount of 4800 different passwords in it.

In the experimental scenario where the Top4800-WPA-probable-v2 dataset was used to train the RNN LSTM model, a total of 528 passwords were generated. While 503 of these generated password outputs were classified as human-like passwords, 25 of them were classified as machine-like passwords.

For the Top4800-WPA-probable-v2 dataset, the model was trained by running a total of 10000 iterations. The model was run for a total of 10000 iterations, and the optimal loss was found at the 8000th iteration, the training process was stopped at the 10000th iteration. The model was run with a total of 666 epoch values. The model was run with the hyperparameter values specified in Figure 4.29 and Figure 4.30.

Figure 4.31 indicates that the loss function value approaches optimal while the model is running. The most optimal loss value reached for this data set is 0.05504848435521126.

While the percentage value of the humanlike password produced in the wordlist as

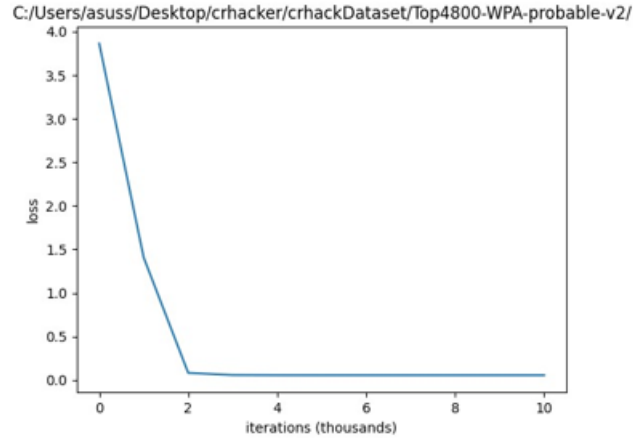


Figure 4.31. Loss Function Value For Each Iteration

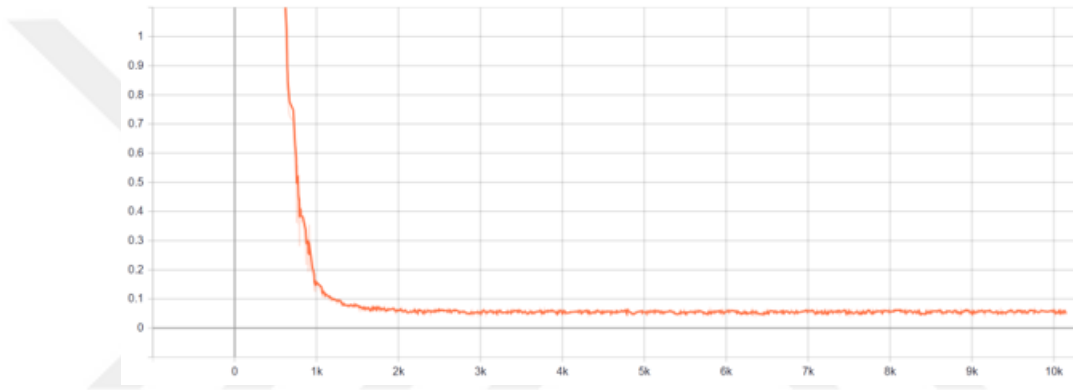


Figure 4.32. Cost Value For Each Iteration

an output set created by the model is 95.3% , the percentage value of the machine-like password in the generated output set is 4.7%

Figure 4.32 illustrates cost value for each iteration. So, Figure 4.33 represents learning rate for each iteration. On the other hand, Figure 4.34 shows logits for each iteration.

As can be observed from the results, the percentage of human-like passwords in the generated dictionary is higher than the percentage of machine-like passwords. When the model is trained with input wordlists created by real people, the output wordlist produces a higher percentage of passwords that real people may be able to create, compared to machine-like passwords.

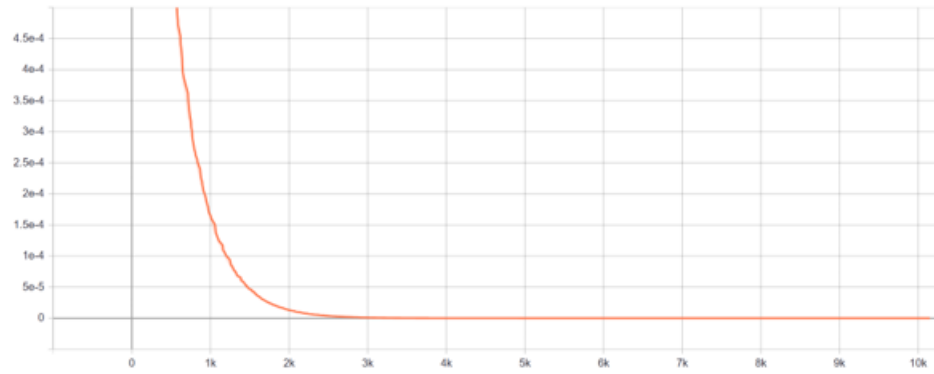


Figure 4.33. Learning Rate For Each Iteration

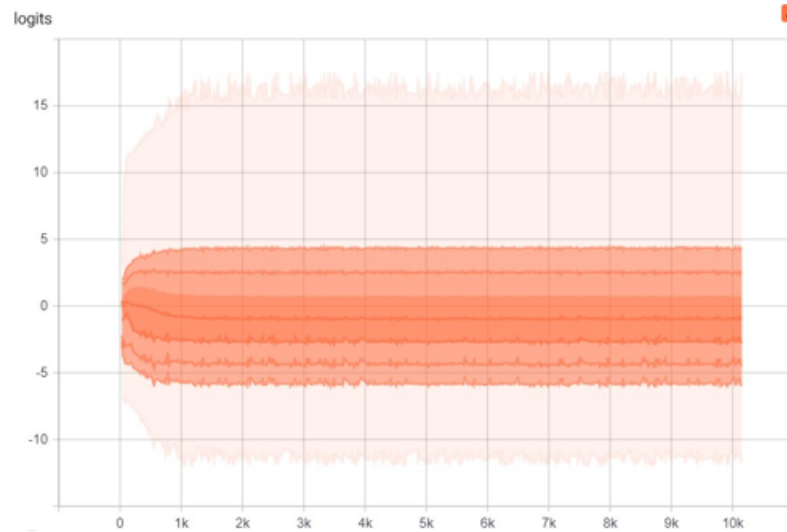


Figure 4.34. Logits For Each Iteration

Furthermore, when the model is trained using the Top4800-WPA-probable-v2 dataset, the following passwords can be given as examples of the generated passwords obtained as output:

Machine Generated Password: 6xtenntdsPAbpOr6aPs29

Machine Generated Password: dq6t9xsWlpm3rxar2j8a6cge50mRlz

Human Generated Password: wakejess

Human Generated Password: afcin123

Human Generated Password: voonkball

Human Generated Password: pheenn123

Human Generated Password: jusverm2

Human Generated Password: heather1

Human Generated Password: hallo123

Human Generated Password: qwerty1234

Human Generated Password: phoenix2

Human Generated Password: 1q2w3e4r5t

Human Generated Password: 8asabers

Human Generated Password: qwern113

Human Generated Password: satsy123

Human Generated Password: presoir1

Human Generated Password: Pqssword123

Human Generated Password: nfer12

Human Generated Password: pass1234

Human Generated Password: password10

Human Generated Password: 0wer1234

Human Generated Password: poteyou1

Human Generated Password: abc12345

Human Generated Password: als22344

Human Generated Password: qwert183

Human Generated Password: lan5boll

Human Generated Password: 12345Rwet

Human Generated Password: jeathon1

Human Generated Password: dysming1

Human Generated Password: godzilla

Human Generated Password: hello123

Human Generated Password: qwerty1234

Human Generated Password: phoenix1



```
# Hyperparams
BATCH_SIZE = 32
SEQUENCE_LENGTH = 25
LEARNING_RATE = 0.002
DECAY_RATE = 0.97
HIDDEN_LAYER_SIZE = 1024
CELLS_SIZE = 3

TEXT_SAMPLE_LENGTH = 500
SAMPLING_FREQUENCY = 1000
LOGGING_FREQUENCY = 1000
```

Figure 4.35. Hyperparameters Value For Top4800-WPA-probable-v3 Dataset

```
Selected dataset: C:/Users/asuss/Desktop/crhacker/crhackDataset/Top4800-WPA-probable-v3/
Batch size: 32
Sequence length: 25
Learning rate: 0.002
Decay rate: 0.97
Hidden layer size: 1024
Cells size: 3
Tensor size: 39200
Batch size: 32
Sequence length: 25
Batches size: 49
```

Figure 4.36. Hyperparameters Value For Top4800-WPA-probable-v3 Dataset

4.4. Top4800-WPA-probable-v3 Dataset Experiments

Top4800-WPA-probable-v3 wordlist dataset contains the amount of 4800 different passwords in it.

In the experimental scenario where the Top4800-WPA-probable-v3 dataset was used to train the RNN LSTM model, a total of 992 passwords were generated. While 979 of these generated password outputs were classified as human-like passwords, 13 of them were classified as machine-like passwords.

For the Top4800-WPA-probable-v3 dataset, the model was trained by running a total of 20000 iterations. The model was run for a total of 20000 iterations, and the optimal loss was found at the 19000th iteration, the training process was stopped at the 20000th iteration. The model was run with a total of 387 epoch values. The model was run with the hyperparameter values specified in Figure 4.35 and Figure 4.36.

Figure 4.37 indicates that the loss function value approaches optimal while the model is running. The most optimal loss value reached for this data set is 0.0011434524785727262.

While the percentage value of the humanlike password produced in the wordlist as an output set created by the model is 98.7% , the percentage value of the machine-like

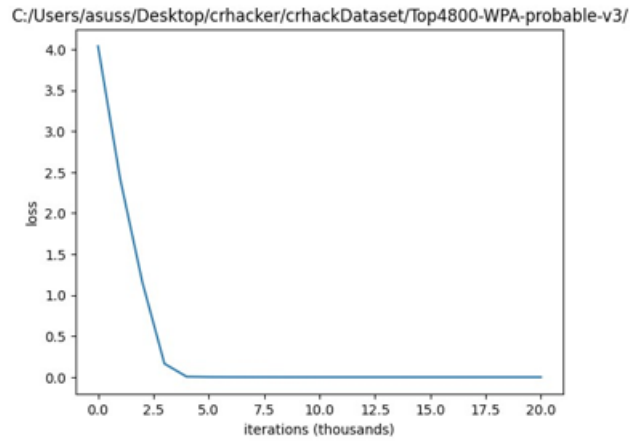


Figure 4.37. Loss Function Value For Each Iteration

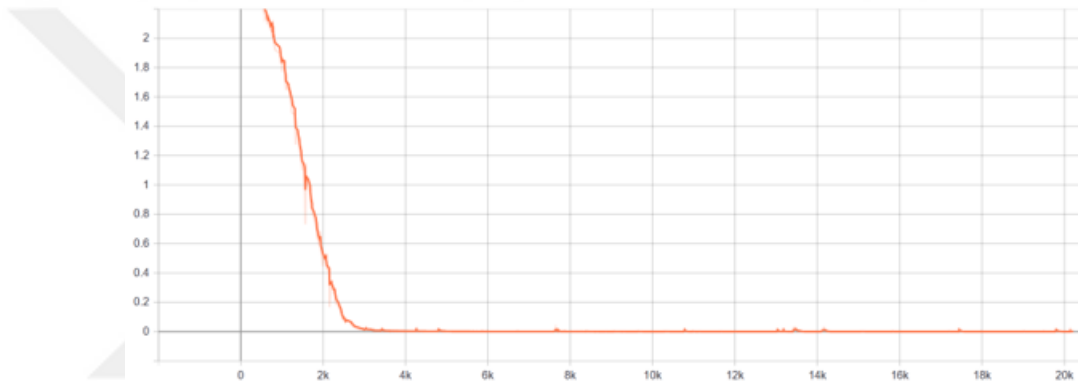


Figure 4.38. Cost Value For Each Iteration

password in the generated output set is 1.3%

Figure 4.38 illustrates cost value for each iteration. So, Figure 4.39 represents learning rate for each iteration. On the other hand, Figure 4.40 shows logits for each iteration.

As can be observed from the results, the percentage of human-like passwords in the generated dictionary is higher than the percentage of machine-like passwords. When the model is trained with input wordlists created by real people, the output wordlist produces a higher percentage of passwords that real people may be able to create, compared to machine-like passwords.

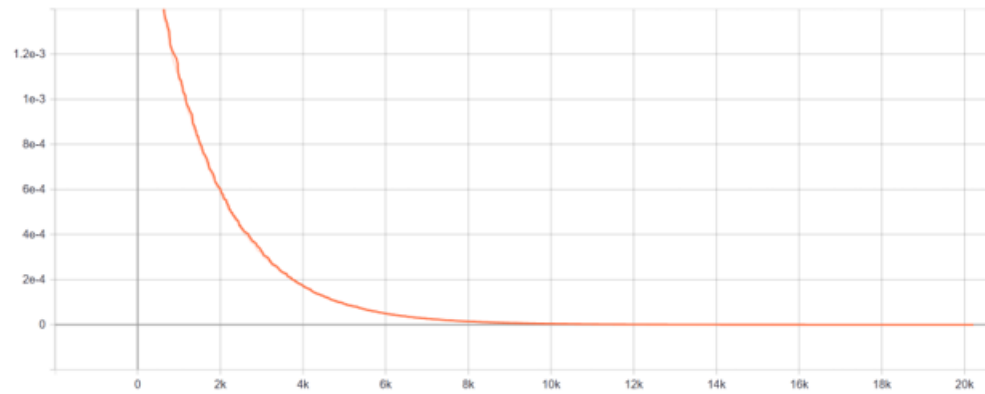


Figure 4.39. Learning Rate For Each Iteration

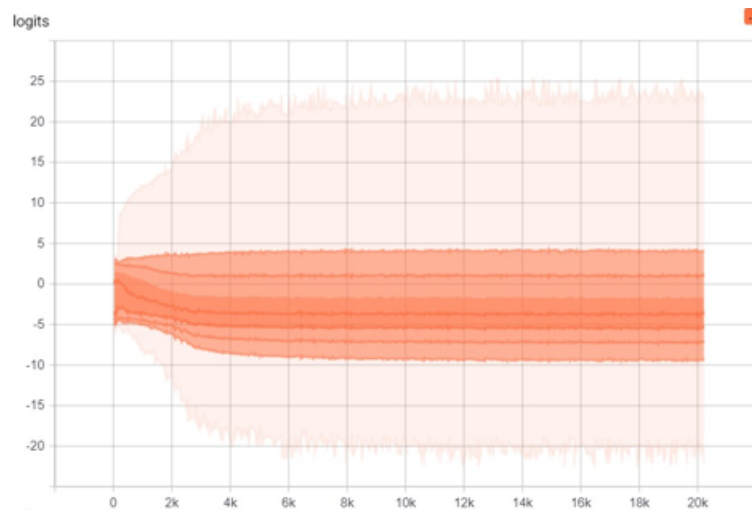


Figure 4.40. Logits For Each Iteration

Moreover, when the model is trained using the Top4800-WPA-probable-v3 dataset, the following passwords can be given as examples of the generated passwords obtained as output:

Machine Generated Password: vtWm8tvMR6;eodhuk4d

Machine Generated Password: b8hDSWGMdMslCvR2Fbiwg3Pl3Me74ehWI

Machine Generated Password: 3eBs26f8caPasg0hPcas4frbs

Machine Generated Password: cJst2VsqeVlmtAvo;lJvOha9BNSFkd9SV

Machine Generated Password: raCFScWlCgkDiAjbFbtP7he6Rgd2unnn

Human Generated Password: arfestbue9

Human Generated Password: mawkers1

Human Generated Password: raca7zoy

Human Generated Password: Nementjob

Human Generated Password: zratret1

Human Generated Password: sstys123

Human Generated Password: tassword95

Human Generated Password: rilliil1

Human Generated Password: p@ppy12345

Human Generated Password: p0ssw0rd

Human Generated Password: stephen1

Human Generated Password: pumpkin1

Human Generated Password: gordon24

Human Generated Password: dragon123

Human Generated Password: cowboys1

Human Generated Password: clystall

Human Generated Password: apollo11

Human Generated Password: vanilla1

Human Generated Password: trouble1

Human Generated Password: smile123

Human Generated Password: porsche9

Human Generated Password: hawaii50

Human Generated Password: apollo13

Human Generated Password: satan666
Human Generated Password: qazwsx123
Human Generated Password: qlw2e3r4t5
Human Generated Password: green123
Human Generated Password: front242
Human Generated Password: İstanbul
Human Generated Password: warcraft3
Human Generated Password: trinity1
Human Generated Password: zaq12wsx
Human Generated Password: alb2c3d4
Human Generated Password: england1
Human Generated Password: death666
Human Generated Password: ricardo1
Human Generated Password: patches1
Human Generated Password: madonna1
Human Generated Password: harry123
Human Generated Password: killer123

```
# Hyperparams
BATCH_SIZE = 32
SEQUENCE_LENGTH = 25
LEARNING_RATE = 0.002
DECAY_RATE = 0.97
HIDDEN_LAYER_SIZE = 1024
CELLS_SIZE = 3

TEXT_SAMPLE_LENGTH = 500
SAMPLING_FREQUENCY = 1000
LOGGING_FREQUENCY = 1000
```

Figure 4.41. Hyperparameters Value For Top204Thousand-WPA-probable-v2 Dataset

```
Selected dataset: C:/Users/asuss/Desktop/crhacker/crhackDataset/Top204Thousand-WPA-probable-v2/
Batch size: 32
Sequence length: 25
Learning rate: 0.002
Decay rate: 0.97
Hidden layer size: 1024
Cells size: 3
Tensor size: 1816800
Batch size: 32
Sequence length: 25
Batches size: 2271
```

Figure 4.42. Hyperparameters Value For Top204Thousand-WPA-probable-v2 Dataset

4.5. Top204Thousand-WPA-probable-v2 Dataset Experiments

Top204Thousand-WPA-probable-v2 wordlist dataset contains the amount of 204.000 different passwords in it.

In the experimental scenario where the Top204Thousand-WPA-probable-v2 dataset was used to train the RNN LSTM model, a total of 2765 passwords were generated. While 2721 of these generated password outputs were classified as human-like passwords, 44 of them were classified as machine-like passwords.

For the Top204Thousand-WPA-probable-v2 dataset, the model was trained by running a total of 62000 iterations. The model was run for a total of 62000 iterations, and the optimal loss was found at the 61000th iteration, the training process was stopped at the 62000th iteration. The model was run with a total of 26 epoch values. The model was run with the hyperparameter values specified in Figure 4.41 and Figure 4.42.

Figure 4.43 indicates that the loss function value approaches optimal while the model is running. The most optimal loss value reached for this data set is 0.4581467807292938.

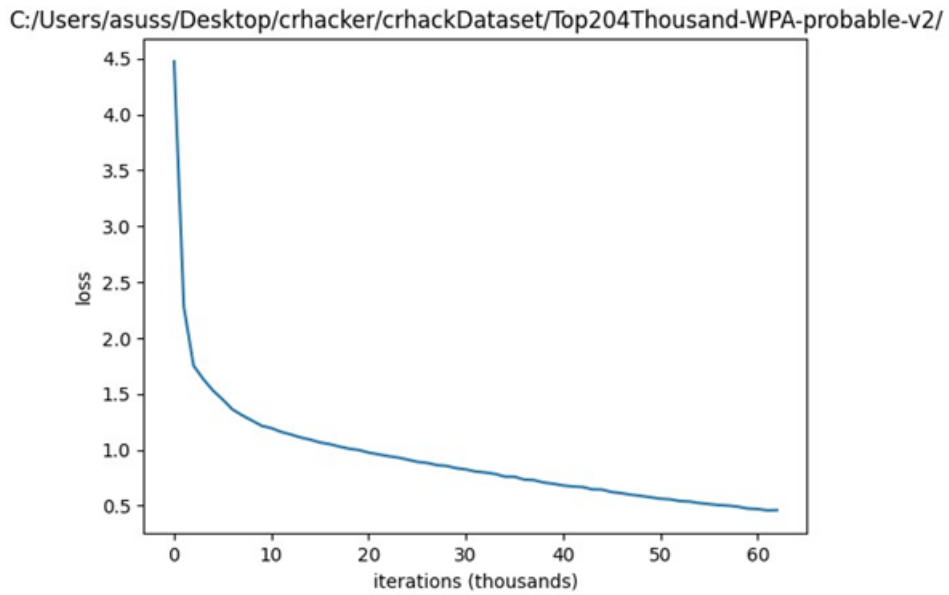


Figure 4.43. Loss Function Value For Each Iteration

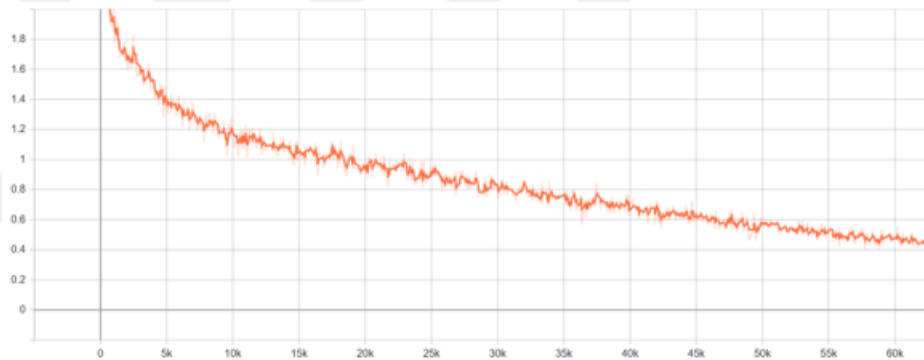


Figure 4.44. Cost Value For Each Iteration

While the percentage value of the humanlike password produced in the wordlist as an output set created by the model is 98.4% , the percentage value of the machine-like password in the generated output set is 1.6%

Figure 4.44 illustrates cost value for each iteration. So, Figure 4.45 represents learning rate for each iteration. On the other hand, Figure 4.46 shows logits for each iteration.

As can be observed from the results, the percentage of human-like passwords in the generated dictionary is higher than the percentage of machine-like passwords. When the model is trained with input wordlists created by real people, the output wordlist produces a higher percentage of passwords that real people may be able to create, compared to machine-like passwords.

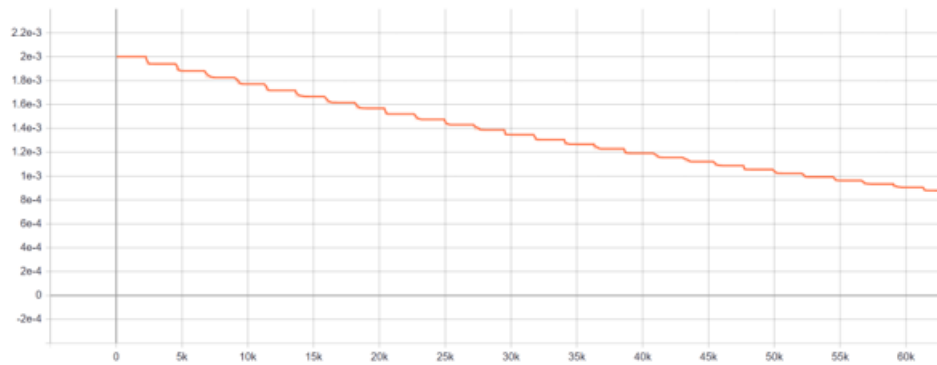


Figure 4.45. Learning Rate For Each Iteration

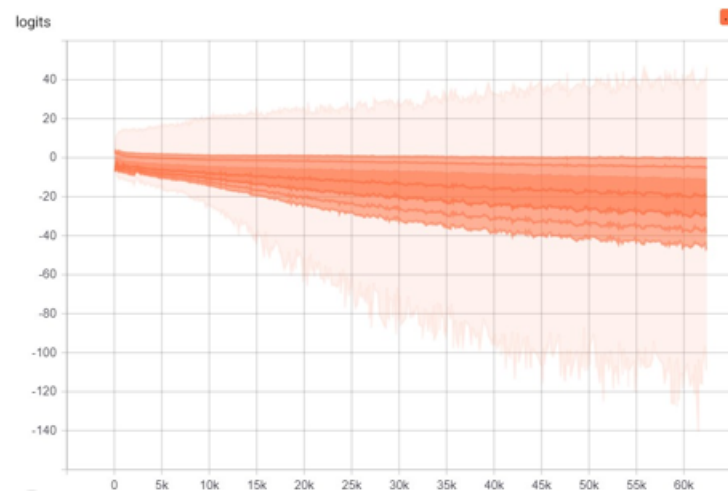


Figure 4.46. Logits For Each Iteration

In addition to this, when the model is trained using the Top204Thousand-WPA-probable-v2 dataset, the following passwords can be given as examples of the generated passwords obtained as output:

Machine Generated Password: 2Swbx\$Q\$
Machine Generated Password: fD%Oa@Sf8
Machine Generated Password: 3MQUyJdO)fM=G\$+Bz
Machine Generated Password: .ek7b.02ka0(XJ
Machine Generated Password: fD%Oa@Sf8
Human Generated Password: O^*@-fR@U*g
Human Generated Password: 1qiagner
Human Generated Password: 2cezugnf
Human Generated Password: 2imaster
Human Generated Password: lichan123
Human Generated Password: highwill16
Human Generated Password: ghurkie5
Human Generated Password: bester99
Human Generated Password: arokey21
Human Generated Password: love1234
Human Generated Password: lightchiold32
Human Generated Password: alexis232
Human Generated Password: albert82
Human Generated Password: alaj1234
Human Generated Password: alex1986
Human Generated Password: alex1992
Human Generated Password: albert123
Human Generated Password: albert23
Human Generated Password: time1997
Human Generated Password: hunter20
Human Generated Password: hunter22
Human Generated Password: hotler16
Human Generated Password: number01

Human Generated Password: dother12
Human Generated Password: dodger35
Human Generated Password: hockey13
Human Generated Password: hobster14
Human Generated Password: harry787
Human Generated Password: harrow10
Human Generated Password: lucky101
Human Generated Password: zxcde123
Human Generated Password: zxcvbnm7
Human Generated Password: halfjogg0
Human Generated Password: hak7bogs
Human Generated Password: greedgsley1
Human Generated Password: doloro123
Human Generated Password: doggie23
Human Generated Password: hoebyting12345
Human Generated Password: heidel12345
Human Generated Password: heavy123
Human Generated Password: heather13
Human Generated Password: december77
Human Generated Password: david1993
Human Generated Password: davis766
Human Generated Password: david1989
Human Generated Password: darkangel1
Human Generated Password: darksun1
Human Generated Password: Chawk8is
Human Generated Password: Breaker2
Human Generated Password: december25
Human Generated Password: daves111
Human Generated Password: diamond123
Human Generated Password: houston9
Human Generated Password: gameboy2

Human Generated Password: hockey8877
Human Generated Password: hollol12345
Human Generated Password: honey007
Human Generated Password: homeless1
Human Generated Password: hunter3010
Human Generated Password: daniel1993
Human Generated Password: honda745
Human Generated Password: dean1234
Human Generated Password: drummet123
Human Generated Password: hockey4200
Human Generated Password: homey999
Human Generated Password: deadbelly123
Human Generated Password: homer1987
Human Generated Password: holiday12
Human Generated Password: delta666
Human Generated Password: deftones1

```
# Hyperparams
BATCH_SIZE = 2
SEQUENCE_LENGTH = 10
LEARNING_RATE = 0.002
DECAY_RATE = 0.97
HIDDEN_LAYER_SIZE = 1024
CELLS_SIZE = 3

TEXT_SAMPLE_LENGTH = 500
SAMPLING_FREQUENCY = 1000
LOGGING_FREQUENCY = 1000
```

Figure 4.47. Hyperparameters Value For Rockyou62 Dataset

4.6. Rockyou62 Dataset Experiments

In the year 2009, as a result of the attack on RockYou, which is a social app and advertising network company, 32.603.388 passwords were revealed. The biggest reason why so many passwords were easily revealed was that they were kept in plain text. The leaked passwords were compiled into a wordlist called RockYou.txt (Cubrilovic 2009, Siegler 2009, Li et al. 2020).

Due to the fact that it is crucial for examining the password selection habits of real human users and it is exactly a real-world representation of it, this wordlist has turned into a standard tool in the fields of password cracking and network testing. For experimental reasons, Rockyou62 wordlist dataset was created with passwords randomly taken from the original RockYou.txt. Rockyou62 dataset contains the amount of 62 different passwords in it. In the experimental scenario where the Rockyou62 dataset was used to train the RNN LSTM model, a total of 560 passwords were generated. While 533 of these generated password outputs were classified as human-like passwords, 27 of them were classified as machine-like passwords. For the Rockyou62 dataset, the model was trained by running a total of 9000 iterations. The model was run for a total of 9000 iterations, and the optimal loss was found at the 8000th iteration, the training process was stopped at the 9000th iteration. The model was run with a total of 363 epoch values. The model was run with the hyperparameter values specified in Figure 4.47 and Figure 4.48.

Figure 4.49 indicates that the loss function value approaches optimal while the model is running. The most optimal loss value reached for this data set is 0.0011539638508111238.

While the percentage value of the humanlike password produced in the wordlist as

```
Selected dataset: C:/Users/gkest/Desktop/crhacker/crhackDataset/rockyou62/  
Batch size: 2  
Sequence length: 10  
Learning rate: 0.002  
Decay rate: 0.97  
Hidden layer size: 1024  
Cells size: 3  
Tensor size: 440  
Batch size: 2  
Sequence length: 10  
Batches size: 22
```

Figure 4.48. Hyperparameters Value For Rockyou62 Dataset

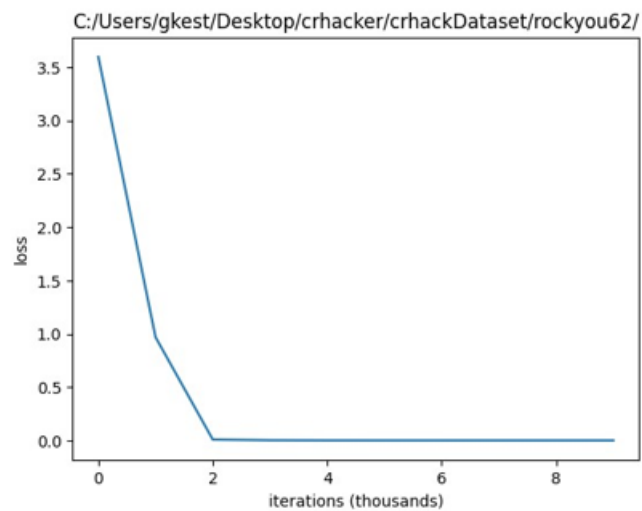


Figure 4.49. Loss Function Value For Each Iteration

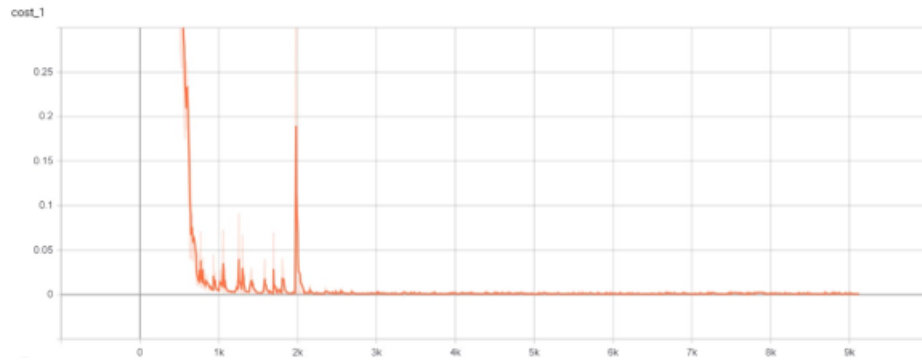


Figure 4.50. Cost Value For Each Iteration

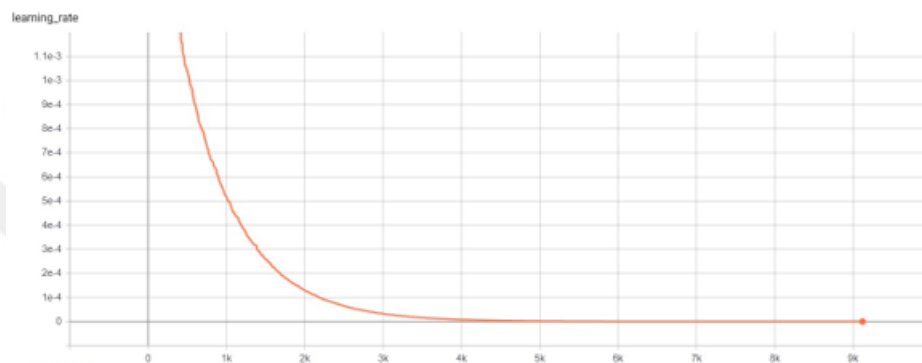


Figure 4.51. Learning Rate For Each Iteration

an output set created by the model is 95.2% , the percentage value of the machine-like password in the generated output set is 4.8%

Figure 4.50 illustrates cost value for each iteration. So, Figure 4.51 represents learning rate for each iteration. On the other hand, Figure 4.52 shows logits for each iteration.

As can be observed from the results, the percentage of human-like passwords in the generated dictionary is higher than the percentage of machine-like passwords. When the model is trained with input wordlists created by real people, the output wordlist produces a higher percentage of passwords that real people may be able to create, compared to machine-like passwords.

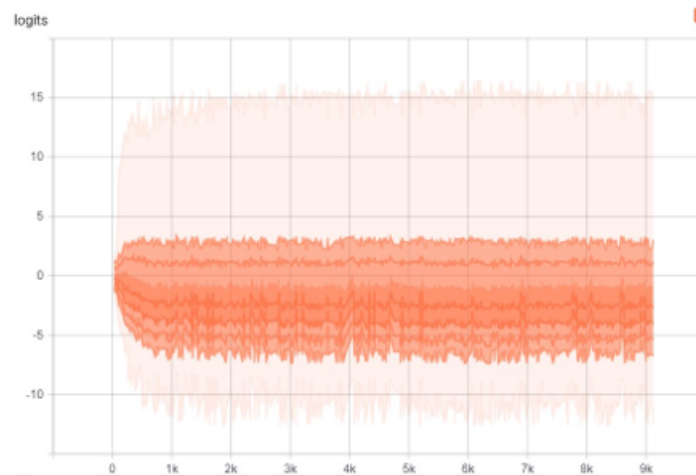


Figure 4.52. Logits For Each Iteration

Besides, when the model is trained using the Rockyou62 dataset, the following passwords can be given as examples of the generated passwords obtained as output:

Machine Generated Password: c2jyhb1j15jflrgy9wrb
Machine Generated Password: ws09b46hwctjdpalndd69v
Machine Generated Password: 8kvobiaugkstkqdefkoagg
Human Generated Password: iloveu
Human Generated Password: 12345678
Human Generated Password: abc123
Human Generated Password: han112345678
Human Generated Password: babygirl
Human Generated Password: 654321
Human Generated Password: qwerty
Human Generated Password: butterfly
Human Generated Password: Liverpool
Human Generated Password: rockyou
Human Generated Password: 12345678
Human Generated Password: michelle
Human Generated Password: football

```
# Hyperparams
BATCH_SIZE = 32
SEQUENCE_LENGTH = 25
LEARNING_RATE = 0.002
DECAY_RATE = 0.97
HIDDEN_LAYER_SIZE = 1024
CELLS_SIZE = 3

TEXT_SAMPLE_LENGTH = 500
SAMPLING_FREQUENCY = 1000
LOGGING_FREQUENCY = 1000
```

Figure 4.53. Hyperparameters Value For Rockyou4800 Dataset

```
Selected dataset: C:/Users/gkest/Desktop/crhacker/crhackDataset/rockyou4800/
Batch size: 32
Sequence length: 25
Learning rate: 0.002
Decay rate: 0.97
Hidden layer size: 1024
Cells size: 3
Tensor size: 33600
Batch size: 32
Sequence length: 25
Batches size: 42
```

Figure 4.54. Hyperparameters Value For Rockyou4800 Dataset

4.7. Rockyou4800 Dataset Experiments

For experimental reasons, Rockyou4800 wordlist dataset was created with passwords randomly taken from the original RockYou.txt. Rockyou4800 dataset contains the amount of 4800 different passwords in it. In the experimental scenario where the Rockyou4800 dataset was used to train the RNN LSTM model, a total of 867 passwords were generated.

While 845 of these generated password outputs were classified as human-like passwords, 22 of them were classified as machine-like passwords. For the Rockyou4800 dataset, the model was trained by running a total of 15000 iterations. The model was run for a total of 15000 iterations, and the optimal loss was found at the 15000th iteration, the training process was stopped at the 15000th iteration. The model was run with a total of 357 epoch values. The model was run with the hyperparameter values specified in Figure 4.53 and Figure 4.54.

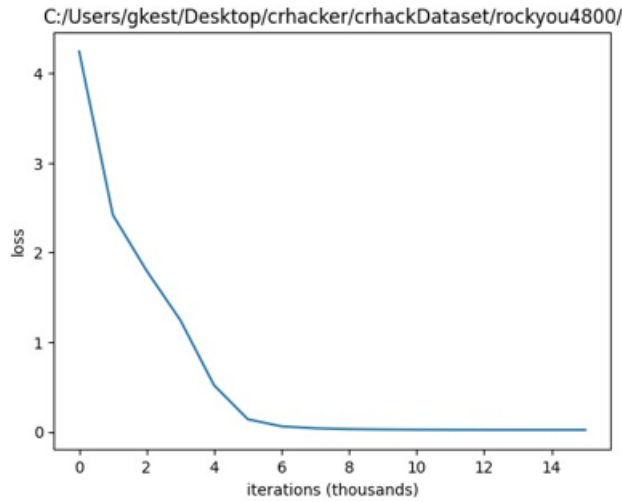


Figure 4.55. Loss Function Value For Each Iteration

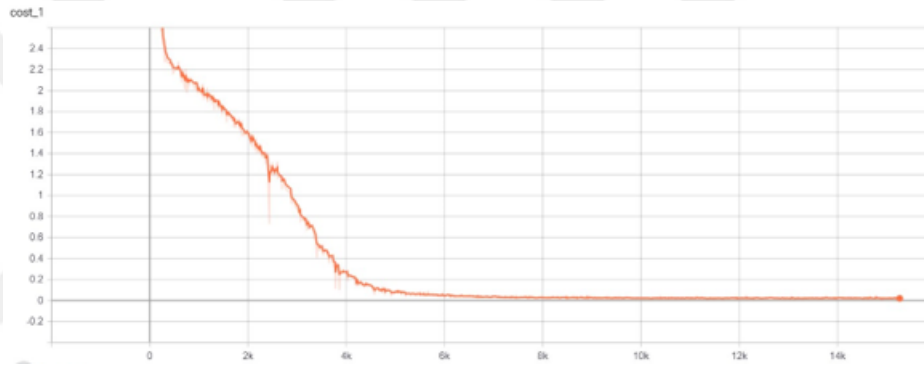


Figure 4.56. Cost Value For Each Iteration

Figure 4.55 indicates that the loss function value approaches optimal while the model is running. The most optimal loss value reached for this data set is 0.022482870146632195.

While the percentage value of the humanlike password produced in the wordlist as an output set created by the model is 97.5% , the percentage value of the machine-like password in the generated output set is 2.5%

Figure 4.56 illustrates cost value for each iteration. So, Figure 4.57 represents learning rate for each iteration. On the other hand, Figure 4.58 shows logits for each iteration.

As can be observed from the results, the percentage of human-like passwords in the generated dictionary is higher than the percentage of machine-like passwords. When the model is trained with input wordlists created by real people, the output wordlist produces a higher percentage of passwords that real people may be able to create, compared to machine-like passwords.

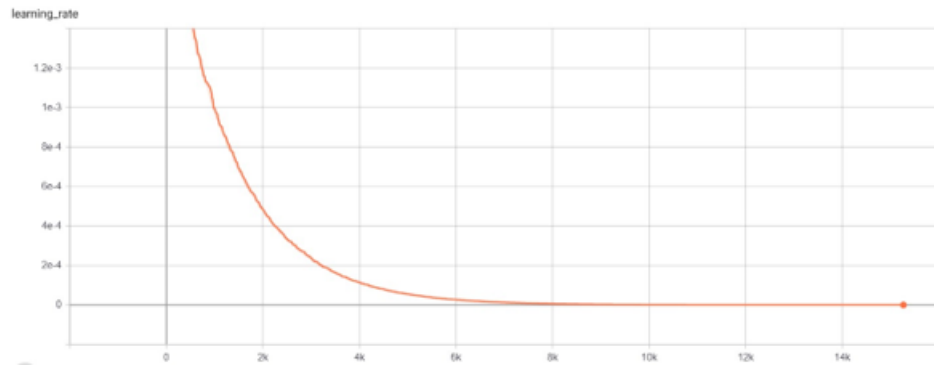


Figure 4.57. Learning Rate For Each Iteration

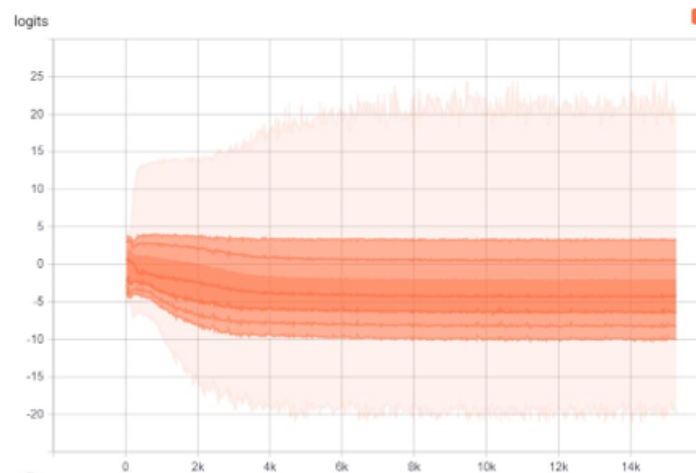


Figure 4.58. Logits For Each Iteration

Additionally, when the model is trained using the Rockyou4800 dataset, the following passwords can be given as examples of the generated passwords obtained as output:

Machine Generated Password: MlR5dW7f9?2UaHD98pV

Machine Generated Password: Gs9ROP;a

Machine Generated Password: sz42Gul72Cjb*ukJty z

Human Generated Password: brever14

Human Generated Password: tickey43

Human Generated Password: coover13

Human Generated Password: beck123

Human Generated Password: 050507

Human Generated Password: christian1

Human Generated Password: aluchello2

Human Generated Password: pumpkin1

Human Generated Password: my2daela

Human Generated Password: olivia1

Human Generated Password: navarro

Human Generated Password: 0123456

Human Generated Password: lakers

Human Generated Password: RocSEer

Human Generated Password: bob123

Human Generated Password: alver11

Human Generated Password: pookie1

Human Generated Password: loveme2

Human Generated Password: russell

Human Generated Password: cat123

Human Generated Password: love4ever

Human Generated Password: jackjane1

Human Generated Password: bacardito

Human Generated Password: arsenal

Human Generated Password: nicholas1

Human Generated Password: loveyou1

Human Generated Password: liverpool1

Human Generated Password: rockyou2

Human Generated Password: classof08

Human Generated Password: eliver1

Human Generated Password: chelsea



```
# Hyperparams
BATCH_SIZE = 32
SEQUENCE_LENGTH = 25
LEARNING_RATE = 0.002
DECAY_RATE = 0.97
HIDDEN_LAYER_SIZE = 1024
CELLS_SIZE = 3

TEXT_SAMPLE_LENGTH = 500
SAMPLING_FREQUENCY = 1000
LOGGING_FREQUENCY = 1000
```

Figure 4.59. Hyperparameters Value For Rockyou204kv2 Dataset

```
Selected dataset: C:/Users/gkest/Desktop/crhacker/crhackDataset/rockyou204kv2/
Batch size: 32
Sequence length: 25
Learning rate: 0.002
Decay rate: 0.97
Hidden layer size: 1024
Cells size: 3
Tensor size: 1504800
Batch size: 32
Sequence length: 25
Batches size: 1881
```

Figure 4.60. Hyperparameters Value For Rockyou204kv2 Dataset

4.8. Rockyou204kv2 Dataset Experiments

For experimental reasons, Rockyou204kv2 wordlist dataset was created with passwords randomly taken from the original RockYou.txt. Rockyou204kv2 dataset contains the amount of 204.000 different passwords in it. In the experimental scenario where the Rockyou204kv2 dataset was used to train the RNN LSTM model, a total of 7354 passwords were generated.

While 7250 of these generated password outputs were classified as human-like passwords, 104 of them were classified as machine-like passwords. For the Rockyou204kv2 dataset, the model was trained by running a total of 136000 iterations. The model was run for a total of 136000 iterations, and the optimal loss was found at the 136000th iteration, the training process was stopped at the 136000th iteration. The model was run with a total of 72 epoch values. The model was run with the hyperparameter values specified in Figure 4.59 and Figure 4.60.

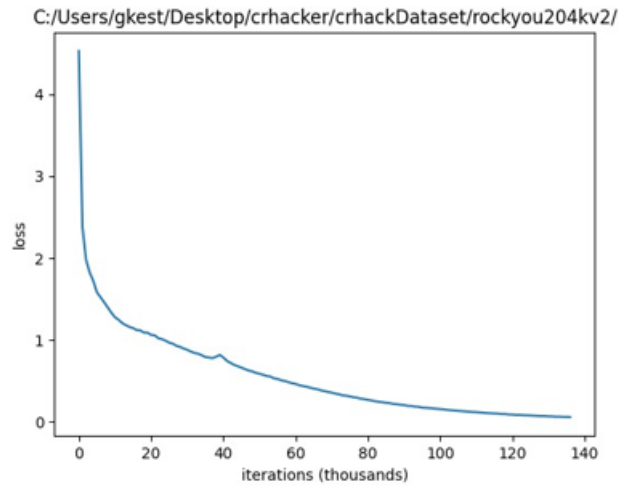


Figure 4.61. Loss Function Value For Each Iteration

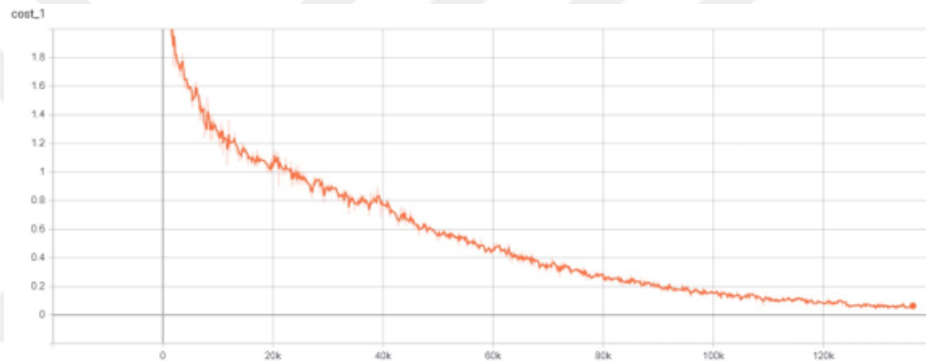


Figure 4.62. Cost Value For Each Iteration

Figure 4.61 indicates that the loss function value approaches optimal while the model is running. The most optimal loss value reached for this data set is 0.05690601095557213.

While the percentage value of the humanlike password produced in the wordlist as an output set created by the model is 98.6% , the percentage value of the machine-like password in the generated output set is 1.4%

Figure 4.62 illustrates cost value for each iteration. So, Figure 4.63 represents learning rate for each iteration. On the other hand, Figure 4.64 shows logits for each iteration.

As can be observed from the results, the percentage of human-like passwords in the generated dictionary is higher than the percentage of machine-like passwords. When the model is trained with input wordlists created by real people, the output wordlist produces a higher percentage of passwords that real people may be able to create, compared to machine-like passwords.

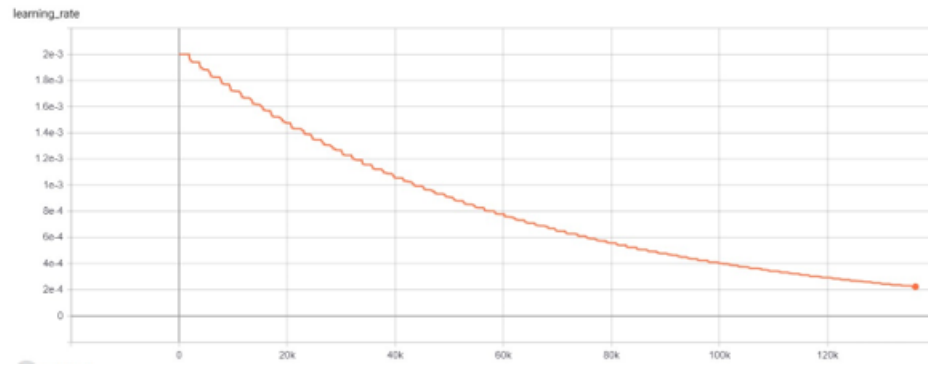


Figure 4.63. Learning Rate For Each Iteration

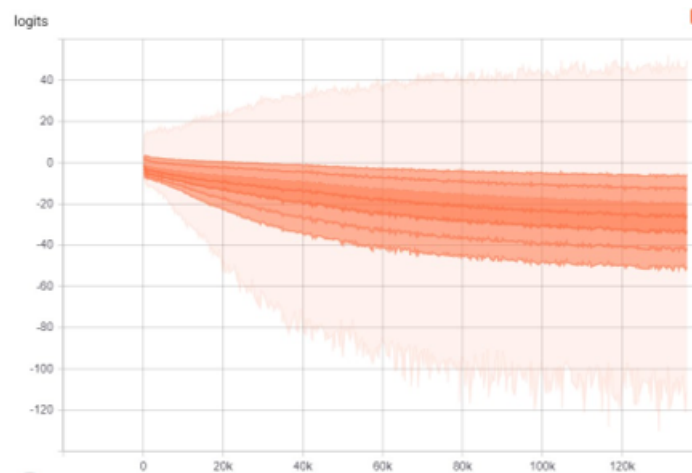


Figure 4.64. Logits For Each Iteration

Moreover, when the model is trained using the rockyou204kv2 dataset, the following passwords can be given as examples of the generated passwords obtained as output:

Machine Generated Password: :JVdUBV23OAXsiuCnmK26Mq) a

Machine Generated Password: `)R[Zy*217639

Human Generated Password: caaon32

Human Generated Password: carmoen12cert

Human Generated Password: crog6241

Human Generated Password: chattyecsirl

Human Generated Password: chalcycon20

Human Generated Password: blissDwor1

Human Generated Password: brondy04

Human Generated Password: lise67

Human Generated Password: junaren03

Human Generated Password: juster123

Human Generated Password: jocks242

Human Generated Password: just523

Human Generated Password: aliahhard5

Human Generated Password: annola22

Human Generated Password: jerji<202

Human Generated Password: jesus254

Human Generated Password: jennis02

Human Generated Password: jessica24

Human Generated Password: jitch08

Human Generated Password: jinal23

Human Generated Password: yellow95

Human Generated Password: walter05

Human Generated Password: jean1988

Human Generated Password: jesper08

Human Generated Password: jamie10

Human Generated Password: babydoll14

Human Generated Password: baby1997

Human Generated Password: babyholy7
Human Generated Password: babycole01
Human Generated Password: babydoll8
Human Generated Password: babyboy4
Human Generated Password: brat123
Human Generated Password: brandon69
Human Generated Password: bobby05
Human Generated Password: butter12
Human Generated Password: burk0ub
Human Generated Password: daisy15
Human Generated Password: cutie44
Human Generated Password: cameron11
Human Generated Password: candy07
Human Generated Password: joyce99
Human Generated Password: joshua93
Human Generated Password: joshua2005
Human Generated Password: jolina95
Human Generated Password: chowers!
Human Generated Password: chocolate09
Human Generated Password: jesse1628
Human Generated Password: general234
Human Generated Password: america01
Human Generated Password: jordan2819
Human Generated Password: johncena21
Human Generated Password: johnny!
Human Generated Password: Welcome1
Human Generated Password: italy123
Human Generated Password: isabella6
Human Generated Password: redville2
Human Generated Password: redsox143

```
# Hyperparams
BATCH_SIZE = 2
SEQUENCE_LENGTH = 10
LEARNING_RATE = 0.002
DECAY_RATE = 0.97
HIDDEN_LAYER_SIZE = 1024
CELLS_SIZE = 3

TEXT_SAMPLE_LENGTH = 500
SAMPLING_FREQUENCY = 1000
LOGGING_FREQUENCY = 1000
```

Figure 4.65. Hyperparameters Value For Kaonashi62 Dataset

```
Selected dataset: C:/Users/gkest/Desktop/crhacker/crhackDataset/kaonashi62/
Batch size: 2
Sequence length: 10
Learning rate: 0.002
Decay rate: 0.97
Hidden layer size: 1024
Cells size: 3
Tensor size: 540
Batch size: 2
Sequence length: 10
Batches size: 27
```

Figure 4.66. Hyperparameters Value For Kaonashi62 Dataset

4.9. Kaonashi62 Dataset Experiments

Kaonashi wordlist compiled from real password leaks. The word list is sorted according to the passwords' occurrence number. Due to the fact that, it consists of real users' passwords, it is also a good wordlist to examine real users' password creation habits. The Kaonashi62 wordlist was created by randomly selecting 62 passwords from the original Kaonashi wordlist.

In the experimental scenario where the Kaonashi62 dataset was used to train the RNN LSTM model, a total of 636 passwords were generated.

While 596 of these generated password outputs were classified as human-like passwords, 40 of them were classified as machine-like passwords. For the Kaonashi62 dataset, the model was trained by running a total of 13000 iterations. The model was run for a total of 13000 iterations, and the optimal loss was found at the 12000th iteration, the training process was stopped at the 13000th iteration. The model was run with a total of 444 epoch values. The model was run with the hyperparameter values specified in Figure 4.65 and Figure 4.66.

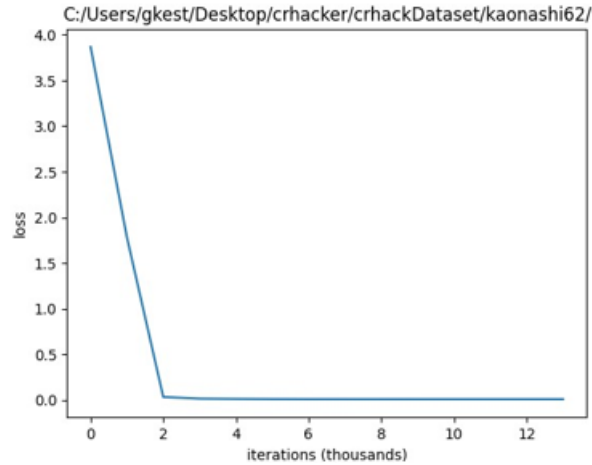


Figure 4.67. Loss Function Value For Each Iteration

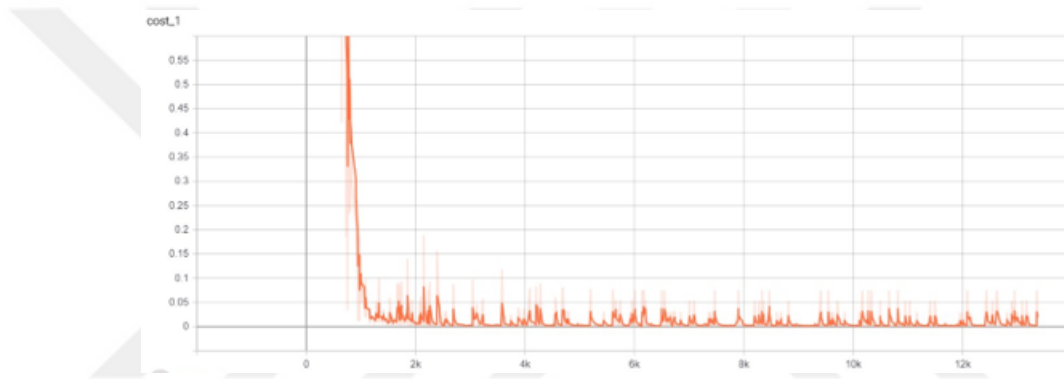


Figure 4.68. Cost Value For Each Iteration

Figure 4.67 indicates that the loss function value approaches optimal while the model is running. The most optimal loss value reached for this data set is 0.007724171970039606.

While the percentage value of the humanlike password produced in the wordlist as an output set created by the model is 93.7% , the percentage value of the machine-like password in the generated output set is 6.3%

Figure 4.68 illustrates cost value for each iteration. So, Figure 4.69 represents learning rate for each iteration. On the other hand, Figure 4.70 shows logits for each iteration.

As can be observed from the results, the percentage of human-like passwords in the generated dictionary is higher than the percentage of machine-like passwords. When the model is trained with input wordlists created by real people, the output wordlist produces a higher percentage of passwords that real people may be able to create, compared to machine-like passwords.

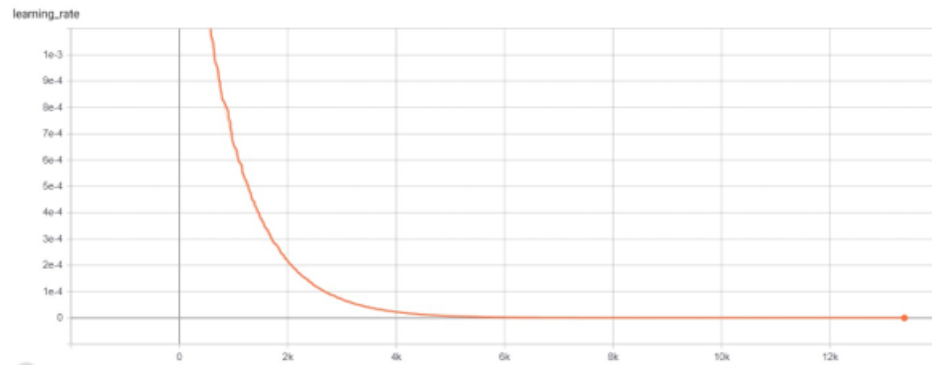


Figure 4.69. Learning Rate For Each Iteration

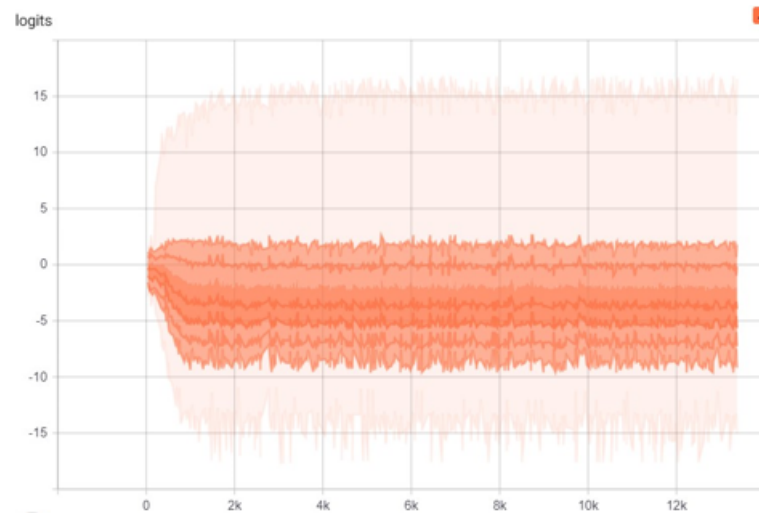


Figure 4.70. Logits For Each Iteration

Besides, when the model is trained using the Kaonashi62 dataset, the following passwords can be given as examples of the generated passwords obtained as output:

Machine Generated Password: Oifchfsv0J4dhsspx499U9SjgE

Machine Generated Password: Ut080bAFJuu06305JeTwFxdxkqkP2

Machine Generated Password: 7hUikx5pskm9AJAy22zmkpmF5fuO

Human Generated Password: 1q2w3e4r

Human Generated Password: qwertyuiop

Human Generated Password: ylantu02012

Human Generated Password: qwer12345

Human Generated Password: Pama6l2

Human Generated Password: qlw2e3r4t5

Human Generated Password: 09qwer1234

Human Generated Password: yoamhewsspa

Human Generated Password: target123

Human Generated Password: asdasd123

Human Generated Password: asdfghjkl

Human Generated Password: yuantuo2012

Human Generated Password: sunshi15n

Human Generated Password: 12345qwert

Human Generated Password: myspace1

Human Generated Password: Passacel

Human Generated Password: password1

Human Generated Password: 3Odige

Human Generated Password: 1qaz2wsx

Human Generated Password: 1q2w3e4r5t

Human Generated Password: 3Odi15n715

Human Generated Password: 3werty123

Human Generated Password: 3Odi15ngxB

Human Generated Password: superman

```
# Hyperparams
BATCH_SIZE = 32
SEQUENCE_LENGTH = 25
LEARNING_RATE = 0.002
DECAY_RATE = 0.97
HIDDEN_LAYER_SIZE = 1024
CELLS_SIZE = 3

TEXT_SAMPLE_LENGTH = 500
SAMPLING_FREQUENCY = 1000
LOGGING_FREQUENCY = 1000
```

Figure 4.71. Hyperparameters Value For Kaonashi4800 Dataset

```
Selected dataset: C:/Users/gkest/Desktop/crhacker/crhackDataset/kaonashi4800/
Batch size: 32
Sequence length: 25
Learning rate: 0.002
Decay rate: 0.97
Hidden layer size: 1024
Cells size: 3
Tensor size: 41600
Batch size: 32
Sequence length: 25
Batches size: 52
```

Figure 4.72. Hyperparameters Value For Kaonashi4800 Dataset

4.10. Kaonashi4800 Dataset Experiments

The Kaonashi4800 wordlist was created by randomly selecting 4800 passwords from the original Kaonashi wordlist.

In the experimental scenario where the Kaonashi4800 dataset was used to train the RNN LSTM model, a total of 761 passwords were generated.

While 739 of these generated password outputs were classified as human-like passwords, 22 of them were classified as machine-like passwords. For the Kaonashi4800 dataset, the model was trained by running a total of 16000 iterations. The model was run for a total of 16000 iterations, and the optimal loss was found at the 16000th iteration, the training process was stopped at the 16000th iteration. The model was run with a total of 307 epoch values. The model was run with the hyperparameter values specified in Figure 4.71 and Figure 4.72.

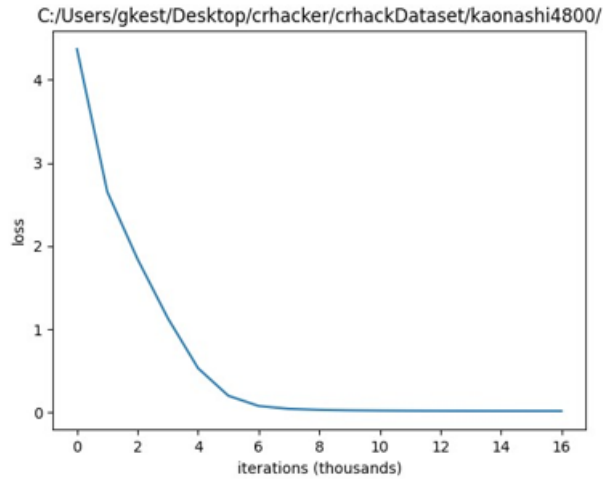


Figure 4.73. Loss Function Value For Each Iteration

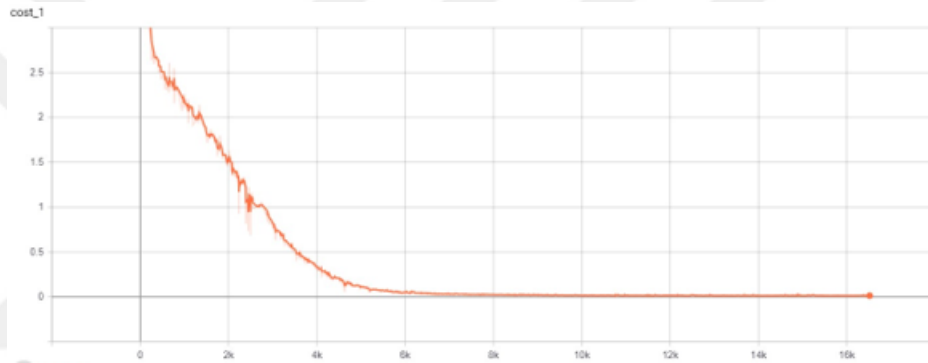


Figure 4.74. Cost Value For Each Iteration

Figure 4.73 indicates that the loss function value approaches optimal while the model is running. The most optimal loss value reached for this data set is 0.014146371744573116.

While the percentage value of the humanlike password produced in the wordlist as an output set created by the model is 97.1% , the percentage value of the machine-like password in the generated output set is 2.9%

Figure 4.74 illustrates cost value for each iteration. So, Figure 4.75 represents learning rate for each iteration. On the other hand, Figure 4.76 shows logits for each iteration.

As can be observed from the results, the percentage of human-like passwords in the generated dictionary is higher than the percentage of machine-like passwords. When the model is trained with input wordlists created by real people, the output wordlist produces a higher percentage of passwords that real people may be able to create, compared to machine-like passwords.

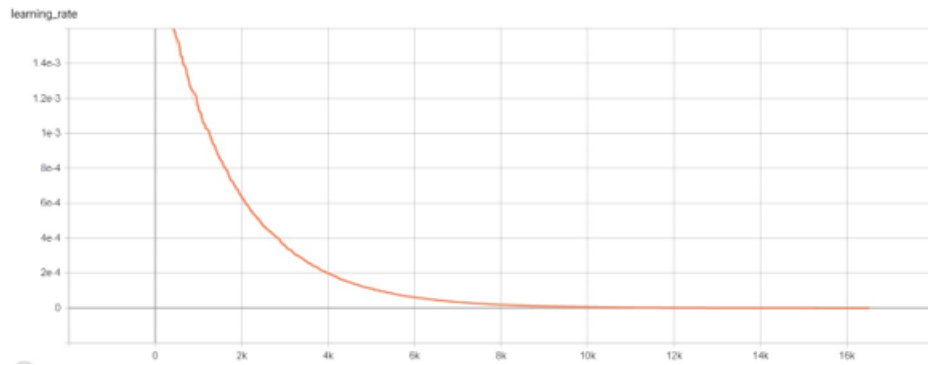


Figure 4.75. Learning Rate For Each Iteration

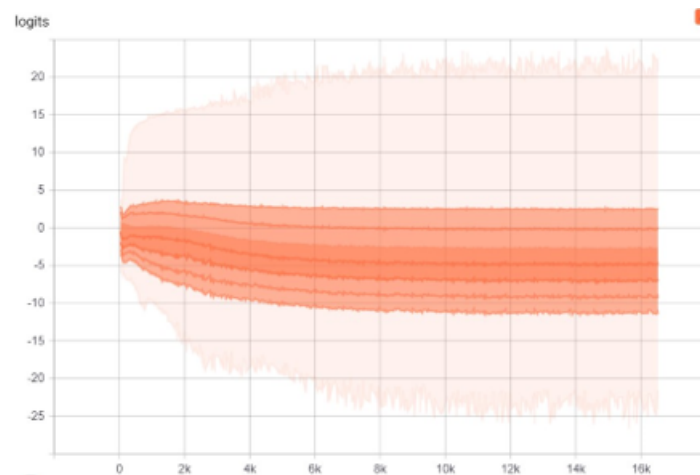


Figure 4.76. Logits For Each Iteration

In addition to this, when the model is trained using the Kaonashi4800 dataset, the following passwords can be given as examples of the generated passwords obtained as output:

Machine Generated Password: ,1P=X\$:Lrff

Machine Generated Password: 1tVLys@bqH=EU?xCaaUULLJ3G*63

Machine Generated Password: @y9I-n?rMjTy2XtP_qY

Human Generated Password: cuapaolucio

Human Generated Password: top_123a

Human Generated Password: aviokits7

Human Generated Password: paleswf55

Human Generated Password: nikateodiel

Human Generated Password: qword300

Human Generated Password: 123456789xbm

Human Generated Password: PP3wsx321

Human Generated Password: strew1313

Human Generated Password: qwerty777

Human Generated Password: blanloff10

Human Generated Password: freedisgood

Human Generated Password: passx123

Human Generated Password: marinaviona

Human Generated Password: shoopl12

Human Generated Password: zaSDqwer

Human Generated Password: Leening1

Human Generated Password: muster12

Human Generated Password: lechen220

Human Generated Password: money123

Human Generated Password: rosebump1

Human Generated Password: superstar1

Human Generated Password: 123456qaz

Human Generated Password: Likoux1234

Human Generated Password: nokia3233

Human Generated Password: rose123o
Human Generated Password: summer01
Human Generated Password: 1234567Q
Human Generated Password: getmoney1
Human Generated Password: 1234567890qwerty
Human Generated Password: miami305
Human Generated Password: 0587654321
Human Generated Password: qazwsx123
Human Generated Password: Soccer123
Human Generated Password: liverpoolfc
Human Generated Password: left4dead
Human Generated Password: abc123abc
Human Generated Password: january1
Human Generated Password: nokia3300
Human Generated Password: linkedin123
Human Generated Password: passer2009
Human Generated Password: Q1w2e3r4t5y6
Human Generated Password: rockstar1

```
# Hyperparams
BATCH_SIZE = 32
SEQUENCE_LENGTH = 25
LEARNING_RATE = 0.002
DECAY_RATE = 0.97
HIDDEN_LAYER_SIZE = 1024
CELLS_SIZE = 3

TEXT_SAMPLE_LENGTH = 500
SAMPLING_FREQUENCY = 1000
LOGGING_FREQUENCY = 1000
```

Figure 4.77. Hyperparameters Value For Kaonashi204k Dataset

```
Selected dataset: C:/Users/gorkemk/Desktop/crhacker/crhackDataset/kaonashi204k/
Batch size: 32
Sequence length: 25
Learning rate: 0.002
Decay rate: 0.97
Hidden layer size: 1024
Cells size: 3
Tensor size: 1778400
Batch size: 32
Sequence length: 25
Batches size: 2223
```

Figure 4.78. Hyperparameters Value For Kaonashi204k Dataset

4.11. Kaonashi204k Dataset Experiments

The Kaonashi204k wordlist was created by randomly selecting 204.000 passwords from the original Kaonashi wordlist.

In the experimental scenario where the Kaonashi204k dataset was used to train the RNN LSTM model, a total of 6396 passwords were generated.

While 6313 of these generated password outputs were classified as human-like passwords, 83 of them were classified as machine-like passwords. For the Kaonashi204k dataset, the model was trained by running a total of 137000 iterations. The model was run for a total of 137000 iterations, and the optimal loss was found at the 137000th iteration, the training process was stopped at the 137000th iteration. The model was run with a total of 61 epoch values. The model was run with the hyperparameter values specified in Figure 4.77 and Figure 4.78.

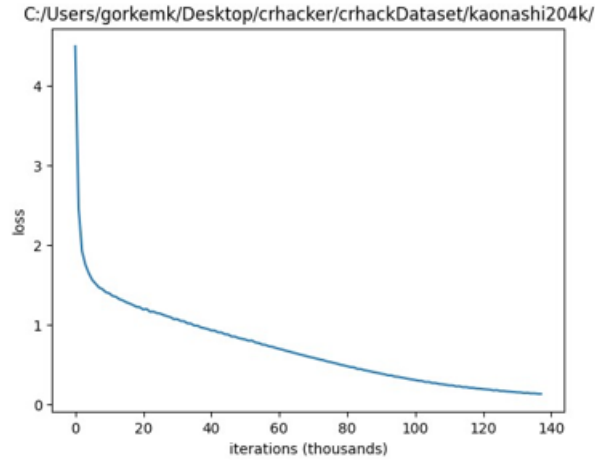


Figure 4.79. Loss Function Value For Each Iteration

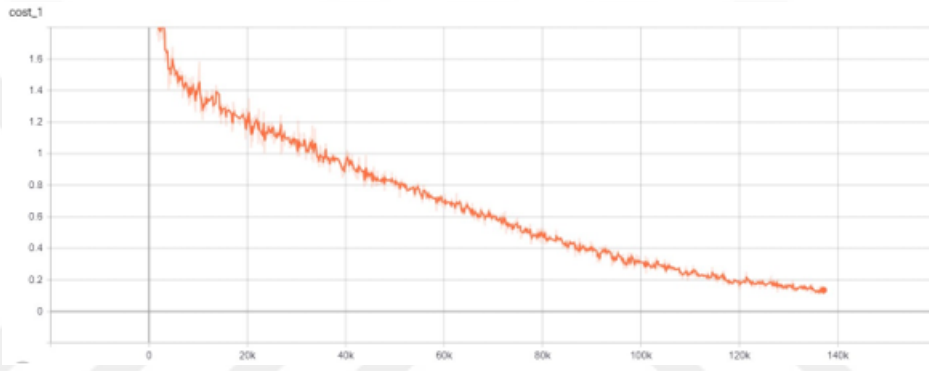


Figure 4.80. Cost Value For Each Iteration

Figure 4.79 indicates that the loss function value approaches optimal while the model is running. The most optimal loss value reached for this data set is 0.13649006187915802.

While the percentage value of the humanlike password produced in the wordlist as an output set created by the model is 98.7% , the percentage value of the machine-like password in the generated output set is 1.3%

Figure 4.80 illustrates cost value for each iteration. So, Figure 4.81 represents learning rate for each iteration. On the other hand, Figure 4.82 shows logits for each iteration.

As can be observed from the results, the percentage of human-like passwords in the generated dictionary is higher than the percentage of machine-like passwords. When the model is trained with input wordlists created by real people, the output wordlist produces a higher percentage of passwords that real people may be able to create, compared to machine-like passwords.

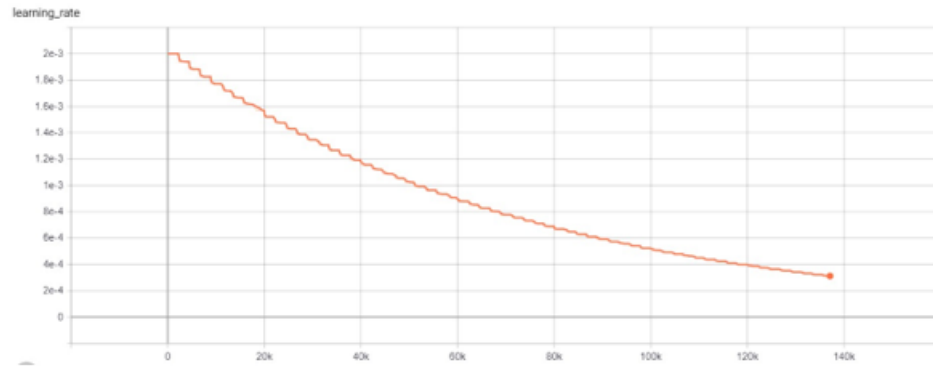


Figure 4.81. Learning Rate For Each Iteration

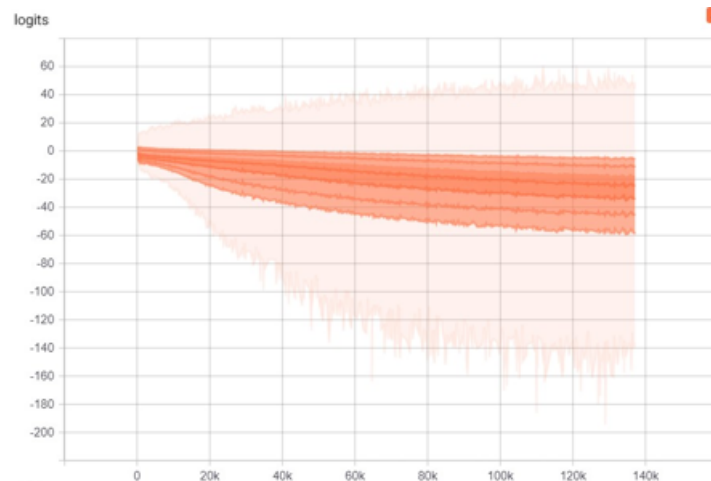


Figure 4.82. Logits For Each Iteration

Additionally, when the model is trained using the Kaonashi204k dataset, the following passwords can be given as examples of the generated passwords obtained as output:

Machine Generated Password: r&(Q7mX-F

Machine Generated Password: |ZW-<p:8

Machine Generated Password: 0?H>cdoj/=s1tw{}YloyUKN%&X;h3

Human Generated Password: ariell19

Human Generated Password: conger24

Human Generated Password: rus12323

Human Generated Password: vama1992

Human Generated Password: silvocyard\$

Human Generated Password: etopie99

Human Generated Password: 0Hoverdo

Human Generated Password: zinger12

Human Generated Password: anderd202

Human Generated Password: taduaHs123

Human Generated Password: partater!

Human Generated Password: family23

Human Generated Password: bennary20

Human Generated Password: gateday12

Human Generated Password: xangers123

Human Generated Password: muraz2001

Human Generated Password: joTEss7561234

Human Generated Password: cristina12

Human Generated Password: williards!

Human Generated Password: Hearts10

Human Generated Password: Henman05

Human Generated Password: Hagna1234

Human Generated Password: cHelsea2012

Human Generated Password: danilo603

Human Generated Password: bulldogs88

Human Generated Password: buster99

Human Generated Password: beaTball114
Human Generated Password: PaSSWorDQ
Human Generated Password: caravella02
Human Generated Password: Beast1995
Human Generated Password: florida1990
Human Generated Password: pHillies12
Human Generated Password: anastasia2
Human Generated Password: PaSSWorD123456789
Human Generated Password: jesus111
Human Generated Password: alfaromeo147
Human Generated Password: Supernatural
Human Generated Password: Quatus123
Human Generated Password: Pass1234
Human Generated Password: david4ever

4.12. Machine-gen-62-dataset Experiments

Password Manager applications/software are able to create automatically computer-generated passwords that are difficult to guess and remember. However, when wordlists compiled from previously leaked real user passwords were examined, it was observed that real users tend to create passwords containing a real user pattern rather than such machine-generated passwords.

In this experimental scenario, when the developed Crhacker program was trained with the machine-generated passwords it produced, it was observed whether the passwords it would generate as output would be human-generated or machine-generated. In this context, machine-gen datasets consisting of passwords ranging in length from 8 to 20 characters and consisting of uppercase and lowercase letters, numbers, and special characters were created. Then, the RNN LSTM model was trained with these machine-gen datasets. First, the machine-gen-62-dataset was created for the training process.

Machine-gen-62-dataset is built by composing passwords which are classified as machine-like passwords by the Crhacker application; It was created by randomly selecting 62 passwords ranging in length from 8 to 20 characters, containing upper and lower case letters, numbers, and special characters. For example, the dataset contains such passwords:

```
E%K@070#I$8w3
7#09*YokG8p#ct
$VSmt30Ny6*Xyu
I3X0Qk#v1W!0E@
xP02rChjnpm2&CwQDMC
wVPrbux0Dh12%k
azr6$Q588R08C9$
```

In the experimental scenario where the machine-gen-62-dataset was used to train the RNN LSTM model, a total of 409 passwords were generated.

While 19 of these generated password outputs were classified as human-like passwords, 390 of them were classified as machine-like passwords. For the machine-gen-62-dataset, the model was trained by running a total of 10000 iterations. The model was run for a total of 10000 iterations, and the optimal loss was found at the 9000th iteration, the

```
# Hyperparams
BATCH_SIZE = 2
SEQUENCE_LENGTH = 20
LEARNING_RATE = 0.002
DECAY_RATE = 0.97
HIDDEN_LAYER_SIZE = 1024
CELLS_SIZE = 3

TEXT_SAMPLE_LENGTH = 500
SAMPLING_FREQUENCY = 1000
LOGGING_FREQUENCY = 1000
```

Figure 4.83. Hyperparameters Value For machine-gen-62-dataset

```
Selected dataset: C:/Users/gkest/Desktop/crhacker/crhackDataset/machine-gen-62-dataset/
Batch size: 2
Sequence length: 20
Learning rate: 0.002
Decay rate: 0.97
Hidden layer size: 1024
Cells size: 3
Tensor size: 680
Batch size: 2
Sequence length: 20
Batches size: 17
```

Figure 4.84. Hyperparameters Value For machine-gen-62-dataset

training process was stopped at the 10000th iteration. The model was run with a total of 529 epoch values. The model was run with the hyperparameter values specified in Figure 4.83 and Figure 4.84.

Figure 4.85 indicates that the loss function value approaches optimal while the model is running. The most optimal loss value reached for this data set is 0.00498690502718091.

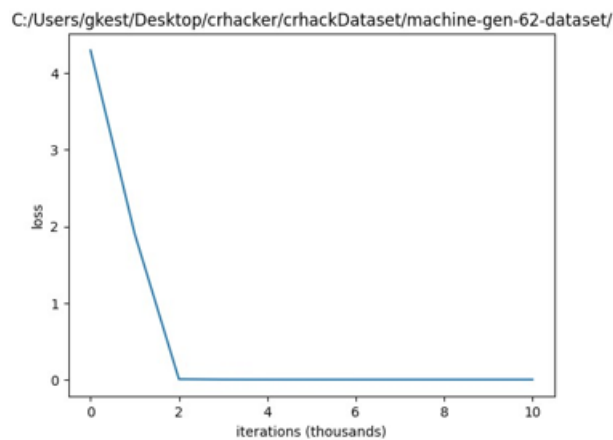


Figure 4.85. Loss Function Value For Each Iteration

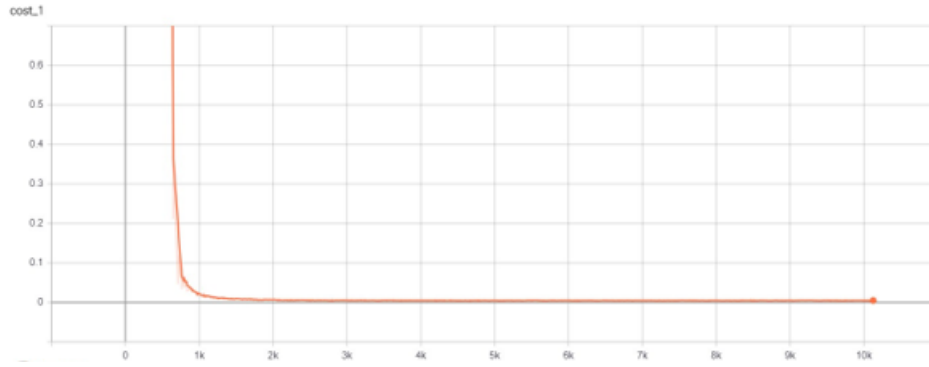


Figure 4.86. Cost Value For Each Iteration

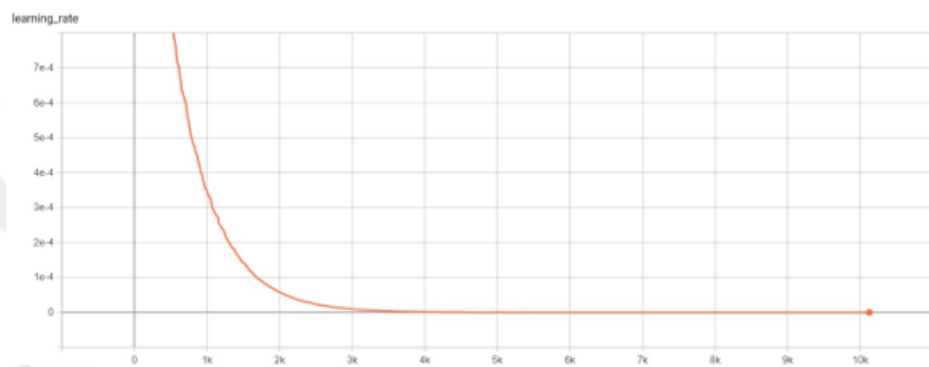


Figure 4.87. Learning Rate For Each Iteration

While the percentage value of the humanlike password produced in the wordlist as an output set created by the model is 4.6% , the percentage value of the machine-like password in the generated output set is 95,4%

Figure 4.86 illustrates cost value for each iteration. So, Figure 4.87 represents learning rate for each iteration. On the other hand, Figure 4.88 shows logits for each iteration.

This time, as can be observed from the results, the percentage of machine-like passwords in the generated dictionary is higher than the percentage of human-like passwords. When the model is trained with input wordlists which are made from machine-like classified passwords, the output wordlist produces a lower percentage of passwords that real people may be able to create, compared to machine-like passwords. As can be understood from here, the more the model is trained with password wordlist that people can use, the more human-like password output it produces. In order to make dictionary attacks more effective, wordlist were enriched by adding human-like passwords as a result of classification to the wordlist that will be used to train the model in the next step.

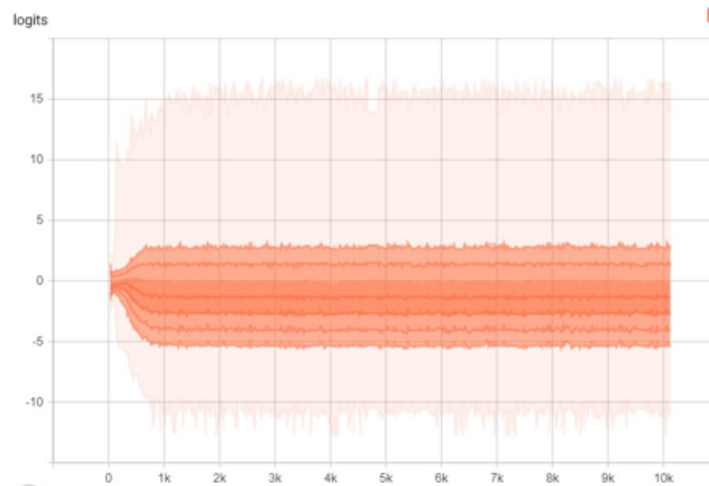


Figure 4.88. Logits For Each Iteration

Furthermore, when the model is trained using the machine-gen-62-dataset dataset, the following passwords can be given as examples of the generated passwords obtained as output:

Machine Generated Password: j8aQ6S1yF2BXg
Machine Generated Password: sQ#@821Mzso0
Machine Generated Password: ^CkM*2886e
Machine Generated Password: 5N24^187
Machine Generated Password: s2b@J36n
Machine Generated Password: 6i!K544L4
Machine Generated Password: 7Z00@%ri0
Machine Generated Password: ssyoR749&&5
Machine Generated Password: kiEy28*D8
Machine Generated Password: j8aQ6\$s3@A
Machine Generated Password: Oxfc&190Ro
Machine Generated Password: Fu#@821Mzso0
Machine Generated Password: ^CkM*98F2X&g
Machine Generated Password: 8rW@*9^Gs!@

4.13. Machine-gen-4800-datasetv2 Experiments

In this experimental scenario, when the developed Crhacker program was trained with the machine-generated passwords it produced, it was observed whether the passwords it would generate as output would be human-generated or machine-generated. In this context, machine-gen datasets consisting of passwords ranging in length from 8 to 20 characters and consisting of uppercase and lowercase letters, numbers, and special characters were created. Then, the RNN LSTM model was trained with these machine-gen datasets. First, the machine-gen-4800-datasetv2 was created for the training process.

Machine-gen-4800-datasetv2 is built by composing passwords which are classified as machine-like passwords by the Crhacker application; It was created by randomly selecting 4800 passwords ranging in length from 8 to 20 characters, containing upper and lower case letters, numbers, and special characters. For example, the dataset contains such passwords:

```
x8znu@tOQ5pSp2U0b9o@
pP7M!p8p&S@%Cbq8GVdR
4B5TPWz@@N98Ro9*9EkN
0D9c3*TYf&l@4Ua!Wzd*
6BX29jfqWVQB8@Hgvd%S
qV!je1XZ31Mln%jdFb**
7%%zWjTA894sN#fmFgQU
4DeveZ$!JN%$BOy6p$n5
```

In the experimental scenario where the machine-gen-4800-datasetv2 was used to train the RNN LSTM model, a total of 409 passwords were generated. While 11 of these generated password outputs were classified as human-like passwords, 398 of them were classified as machine-like passwords. For the machine-gen-4800-datasetv2, the model was trained by running a total of 9000 iterations. The model was run for a total of 9000 iterations, and the optimal loss was found at the 9000th iteration, the training process was stopped at the 9000th iteration. The model was run with a total of 103 epoch values. The model was run with the hyperparameter values specified in Figure 4.89 and Figure 4.90.

```
# Hyperparams
BATCH_SIZE = 32
SEQUENCE_LENGTH = 20
LEARNING_RATE = 0.002
DECAY_RATE = 0.97
HIDDEN_LAYER_SIZE = 1024
CELLS_SIZE = 3

TEXT_SAMPLE_LENGTH = 500
SAMPLING_FREQUENCY = 1000
LOGGING_FREQUENCY = 1000
```

Figure 4.89. Hyperparameters Value For machine-gen-4800-datasetv2

```
Selected dataset: C:/Users/gkest/Desktop/crhacker/crhackDataset/machine-gen-4800-datasetv2/
Batch size: 32
Sequence length: 20
Learning rate: 0.002
Decay rate: 0.97
Hidden layer size: 1024
Cells size: 3
Tensor size: 55680
Batch size: 32
Sequence length: 20
Batches size: 87
```

Figure 4.90. Hyperparameters Value For machine-gen-4800-datasetv2

Figure 4.91 indicates that the loss function value approaches optimal while the model is running. The most optimal loss value reached for this data set is 0.010801034979522228.

While the percentage value of the humanlike password produced in the wordlist as an output set created by the model is 2.7% , the percentage value of the machine-like password in the generated output set is 97.3%

Figure 4.92 illustrates cost value for each iteration. So, Figure 4.93 represents learning rate for each iteration. On the other hand, Figure 4.94 shows logits for each iteration.

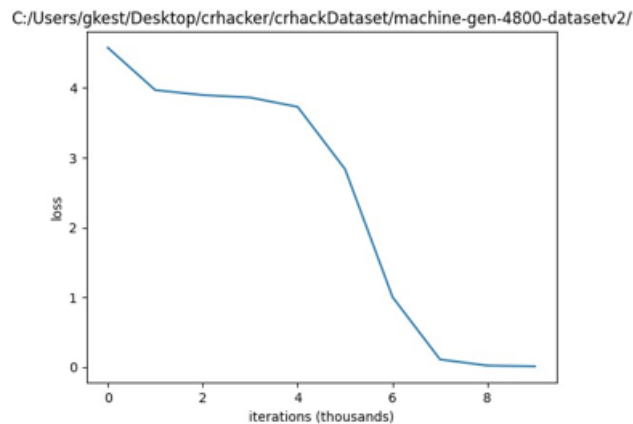


Figure 4.91. Loss Function Value For Each Iteration

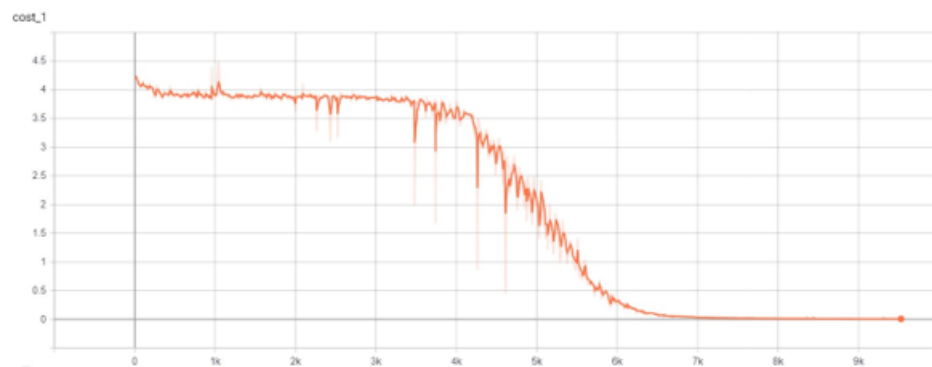


Figure 4.92. Cost Value For Each Iteration

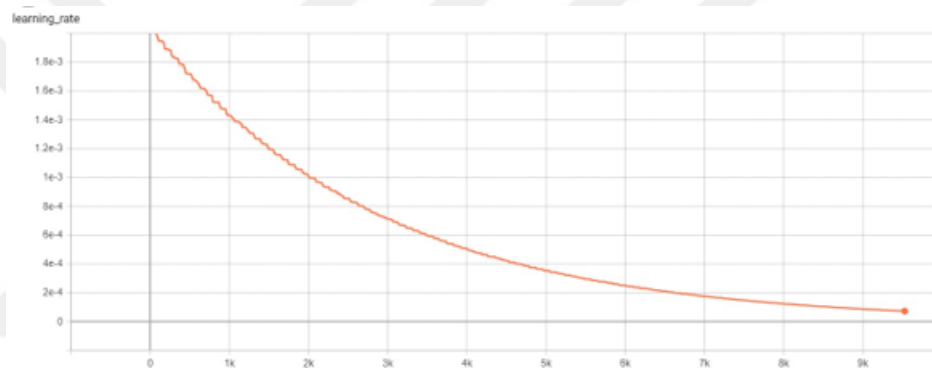


Figure 4.93. Learning Rate For Each Iteration

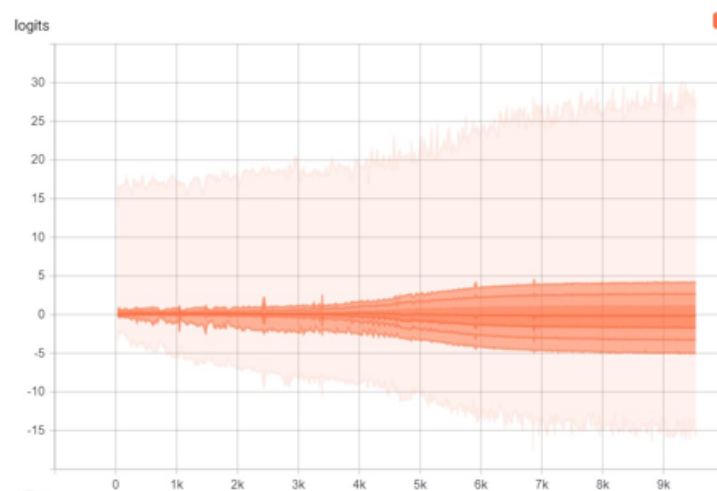


Figure 4.94. Logits For Each Iteration

This time, as can be observed from the results, the percentage of machine-like passwords in the generated dictionary is higher than the percentage of human-like passwords. When the model is trained with input wordlists which are made from machine-like classified passwords, the output wordlist produces a lower percentage of passwords that real people may be able to create, compared to machine-like passwords.

As can be understood from here, the more the model is trained with password wordlist that people can use, the more human-like password output it produces. In order to make dictionary attacks more effective, wordlist were enriched by adding human-like passwords as a result of classification to the wordlist that will be used to train the model in the next step.

Moreover, when the model is trained using the machine-gen-4800-datasetv2 dataset, the following passwords can be given as examples of the generated passwords obtained as output:

Machine Generated Password: 0QnP~5]aR3B1
Machine Generated Password: 7&V?yI63A~@Db|3s{
Machine Generated Password: s^:LlqYk
Machine Generated Password: 2I\w\$2Q1
Machine Generated Password: se18~/Cd
Machine Generated Password: 3A^`##]/
Machine Generated Password: BI*s\j1n3
Machine Generated Password: ;XWwKvFLU
Machine Generated Password: S#)B=rE*n-
Machine Generated Password: O#srPZTT->
Machine Generated Password: z7IuO-u"=N
Machine Generated Password: ~xVyNd4wkC
Machine Generated Password: QOforVT%K
Machine Generated Password: 78)k8NJRA
Machine Generated Password: AOihpD/w
Machine Generated Password: pnK(AMAS
Machine Generated Password: Q)xcz^6R
Machine Generated Password: >_z8kbD_

Machine Generated Password: @`F5tAP\
Machine Generated Password: ,Ein+?e&wR0
Machine Generated Password: qupcx'HXN0R
Machine Generated Password: bB_!C?}kp&.
Machine Generated Password: 1Y@atM=3+R1
Machine Generated Password: 7mu~]'s96(5
Machine Generated Password: mi;l2Bo"?t!
Machine Generated Password: BS>64r=c%+Y
Machine Generated Password: bHkR&R^zjK
Machine Generated Password: 8'B{`<LBw.
Machine Generated Password: cop,D6+V,
Machine Generated Password: AZnJ&KY/|
Machine Generated Password: =kLt>HpRW
Machine Generated Password: 8FTQ,82-Y+
Machine Generated Password: [f?&2+F5iN
Machine Generated Password: L&0<N%]TqU
Machine Generated Password: u@3q;1*9zg&
Machine Generated Password: 7P3SggJ&h
Machine Generated Password: yJDLYw&{V

4.14. Machine-gen-204k-dataset Experiments

In this experimental scenario, when the developed Crhacker program was trained with the machine-generated passwords it produced, it was observed whether the passwords it would generate as output would be human-generated or machine-generated. In this context, machine-gen datasets consisting of passwords ranging in length from 8 to 20 characters and consisting of uppercase and lowercase letters, numbers, and special characters were created. Then, the RNN LSTM model was trained with these machine-gen datasets. First, the Machine-gen-204k-dataset was created for the training process.

Machine-gen-204k-dataset is built by composing passwords which are classified as machine-like passwords by the Crhacker application; It was created by randomly selecting 204.000 passwords ranging in length from 8 to 20 characters, containing upper and lower case letters, numbers, and special characters. For example, the dataset contains such passwords:

```
V@CQ/fTd[7u
K/_CmEB<O`@
HwP5}$4:7h]
<BS!%xwXbU1
C8:T5{f9O~u
<i:@Gn~J9%`
sLPm)[}B%-C
0}ORx1at?&Y
WKE1D@;+R{A
cFEjC0=+]LK
P[&:U.F]7LK
GO-5KP78o:d
.Nl&}:3+s/Z
kpNjC!Ps5&9
'O*Fp6kW0n.
```

```
# Hyperparams
BATCH_SIZE = 32
SEQUENCE_LENGTH = 25
LEARNING_RATE = 0.002
DECAY_RATE = 0.97
HIDDEN_LAYER_SIZE = 1024
CELLS_SIZE = 3

TEXT_SAMPLE_LENGTH = 500
SAMPLING_FREQUENCY = 1000
LOGGING_FREQUENCY = 1000
```

Figure 4.95. Hyperparameters Value For machine-gen-204k-dataset

```
Selected dataset: C:/Users/gkest/Desktop/crhacker/crhackDataset/machine-gen-204k-dataset/
Batch size: 32
Sequence length: 25
Learning rate: 0.002
Decay rate: 0.97
Hidden layer size: 1024
Cells size: 3
Tensor size: 2590400
Batch size: 32
Sequence length: 25
Batches size: 3238
```

Figure 4.96. Hyperparameters Value For machine-gen-204k-dataset

In the experimental scenario where the machine-gen-204k-dataset was used to train the RNN LSTM model, a total of 4360 passwords were generated. While 1280 of these generated password outputs were classified as human-like passwords, 3080 of them were classified as machine-like passwords. For the machine-gen-204k-dataset, the model was trained by running a total of 145.000 iterations. The model was run for a total of 145.000 iterations, and the optimal loss was found at the 144.000th iteration, the training process was stopped at the 145.000th iteration. The model was run with a total of 44 epoch values. The model was run with the hyperparameter values specified in Figure 4.95 and Figure 4.96.

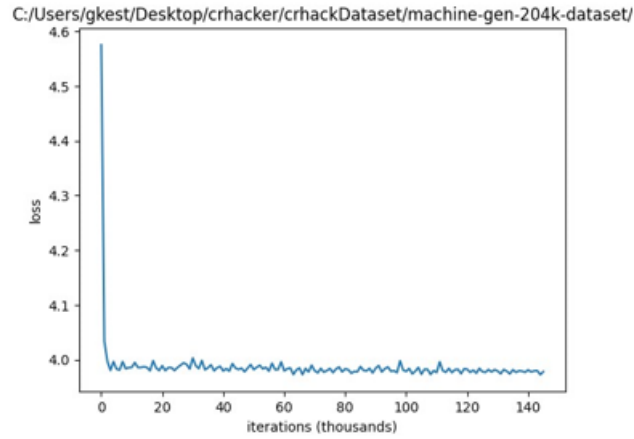


Figure 4.97. Loss Function Value For Each Iteration

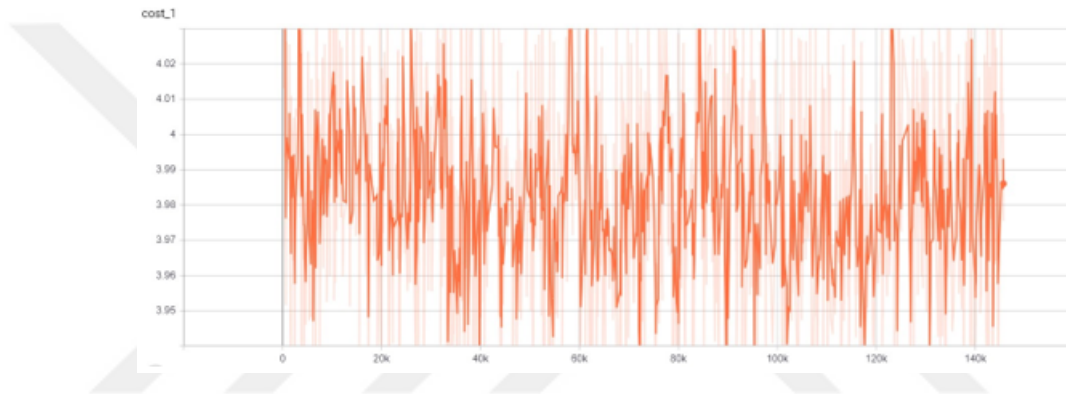


Figure 4.98. Cost Value For Each Iteration

Figure 4.97 indicates that the loss function value approaches optimal while the model is running. The most optimal loss value reached for this data set is 3.972756862640381.

While the percentage value of the humanlike password produced in the wordlist as an output set created by the model is 29.4% , the percentage value of the machine-like password in the generated output set is 70.6%

Figure 4.98 illustrates cost value for each iteration. So, Figure 4.99 represents learning rate for each iteration. On the other hand, Figure 4.100 shows logits for each iteration.

This time, as can be observed from the results, the percentage of machine-like passwords in the generated dictionary is higher than the percentage of human-like passwords. When the model is trained with input wordlists which are made from machine-like classified passwords, the output wordlist produces a lower percentage of passwords that real people may be able to create, compared to machine-like passwords.

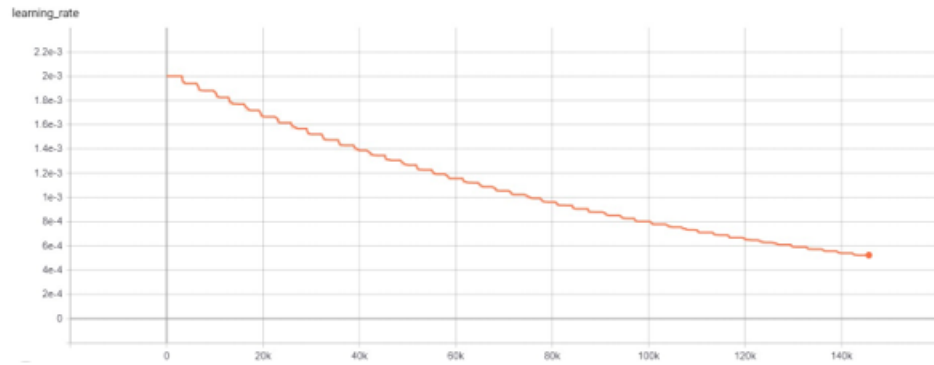


Figure 4.99. Learning Rate For Each Iteration

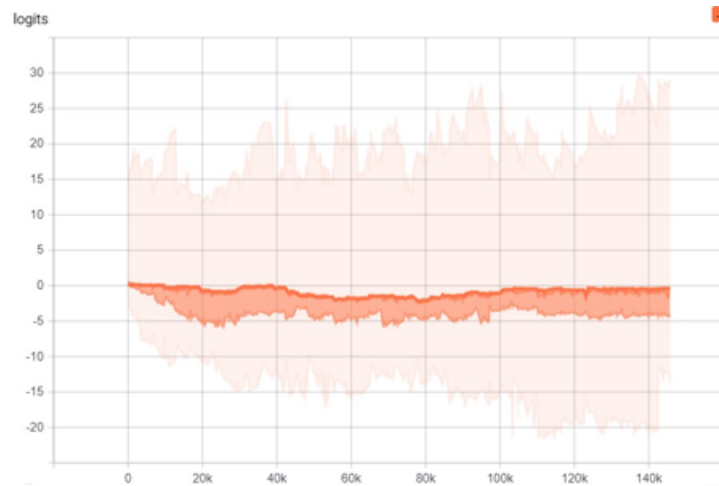


Figure 4.100. Logits For Each Iteration

As can be understood from here, the more the model is trained with password wordlist that people can use, the more human-like password output it produces. In order to make dictionary attacks more effective, wordlist were enriched by adding human-like passwords as a result of classification to the wordlist that will be used to train the model in the next step.

Besides, when the model is trained using the machine-gen-204k-dataset dataset, the following passwords can be given as examples of the generated passwords obtained as output:

```
Machine Generated Password: di$1c4BT6x
Machine Generated Password: hpiSW6<icC7[ `kH
Machine Generated Password: 4wnuP}Y[#
Machine Generated Password: bdGoN.0#E
Machine Generated Password: `m;eUiv4n
Machine Generated Password: :k6jd0V*!
Machine Generated Password: ~ueqW*66q
Machine Generated Password: ]zv%pZk5W
Machine Generated Password: k7G@vZux{ j
Machine Generated Password: qkG6>$wn1XOr4*N1Zc
Machine Generated Password: !SS(3nS.@"}Y3a
```

4.15. Mix62 Dataset Experiments

When the RNN LSTM model is trained with various password datasets, it can produce many different password outputs. It can then classify these generated passwords as either machine-like generated passwords or human-like generated passwords. Passwords are randomly selected which they were previously generated and classified as human-like-generated by RNN LSTM model were trained with real leaked wordlists. Then, password outputs were created by training the RNN LSTM model with automatically machine-generated password datasets, such as those that Password Managers are able to produce, and pattern-free password data, such as those patterns that real people are able to create, and these output passwords were classified as machine-like generated passwords. And then, these classified machine-like generated passwords are taken in order to build up the mixed dataset for the next step. Furthermore, mentioned human-like-generated and machine-like-generated passwords were mixed and new mix series wordlists were created. When the RNN LSTM model was trained with this new mix series password datasets, it was model trained with Mix62, Mix4800, and Mix204kv2 wordlists to observe what kind of wordlist it would produce as output and what the results would be in case of that situation.

First, the RNN LSTM model was trained with the Mix62 wordlist containing 62 mixed passwords. Some passwords from the Mix62 wordlist are given as examples below:

```
passw0rd  
micHael1  
samanthHa  
cHocolate  
qwerty123456  
linkedin  
3oDI15nGXB  
mercedes  
wHatever  
0000000000  
qazwsx123
```

KU[IgoF!
[='macrv
kK;JrjCt
Lsp&b%OU
?ue~\$1x4
im\$eZ.xj
R\askFM<
<Zp!L]`.
fuqX[gPG

In the experimental scenario where the Mix62 dataset was used to train the RNN LSTM model, a total of 558 passwords were generated. While 531 of these generated password outputs were classified as human-like passwords, 27 of them were classified as machine-like passwords. For the Mix62 dataset, the model was trained by running a total of 10.000 iterations. The model was run for a total of 10.000 iterations, and the optimal loss was found at the 10.000th iteration, the training process was stopped at the 10.000th iteration. The model was run with a total of 416 epoch values. The model was run with the hyperparameter values specified in Figure 4.101 and Figure 4.102.

```
# Hyperparams
BATCH_SIZE = 2
SEQUENCE_LENGTH = 10
LEARNING_RATE = 0.002
DECAY_RATE = 0.97
HIDDEN_LAYER_SIZE = 1024
CELLS_SIZE = 3

TEXT_SAMPLE_LENGTH = 500
SAMPLING_FREQUENCY = 1000
LOGGING_FREQUENCY = 1000
```

Figure 4.101. Hyperparameters Value For Mix62 Dataset

```
Selected dataset: C:/Users/gorkemk/Desktop/crhacker/crhackDataset/mix62/
Batch size: 2
Sequence length: 10
Learning rate: 0.002
Decay rate: 0.97
Hidden layer size: 1024
Cells size: 3
Tensor size: 480
Batch size: 2
Sequence length: 10
Batches size: 24
```

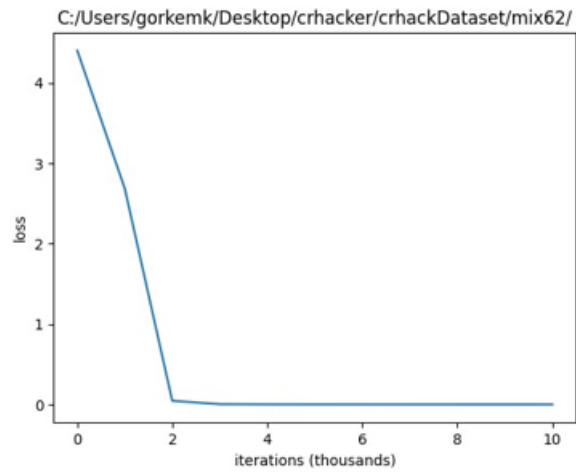
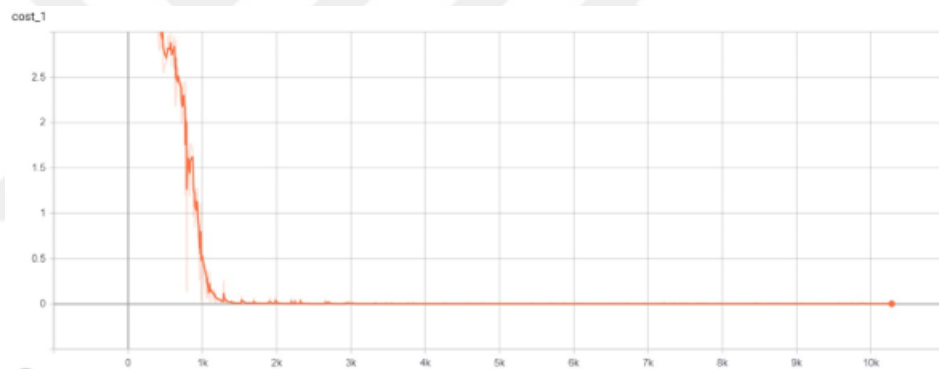
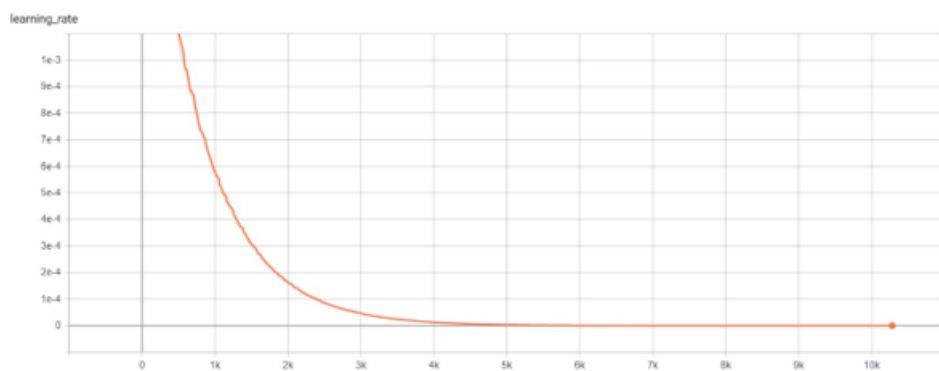
Figure 4.102. Hyperparameters Value For Mix62 Dataset

Figure 4.103 indicates that the loss function value approaches optimal while the model is running. The most optimal loss value reached for this data set is 0.002126453211531043.

While the percentage value of the humanlike password produced in the wordlist as an output set created by the model is 95.2% , the percentage value of the machine-like password in the generated output set is 4.8%

Figure 4.104 illustrates cost value for each iteration. So, Figure 4.105 represents learning rate for each iteration. On the other hand, Figure 4.106 shows logits for each iteration.

As can be inferred from the results, the percentage of human-like passwords in the generated dictionary is higher than the percentage of machine-like passwords. When the model is trained with input mixed wordlists, the output wordlist produces a higher percentage of passwords that real people may be able to create, compared to machine-like passwords.

**Figure 4.103.** Loss Function Value For Each Iteration**Figure 4.104.** Cost Value For Each Iteration**Figure 4.105.** Learning Rate For Each Iteration

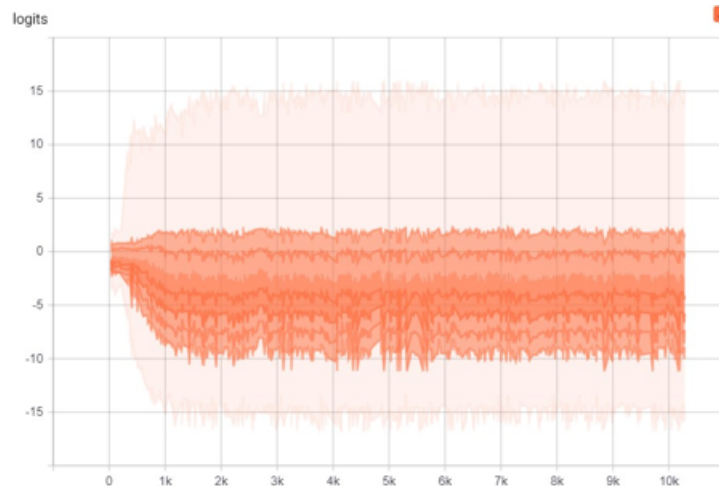


Figure 4.106. Logits For Each Iteration

In addition to this, when the model is trained using the Mix62 dataset, the following passwords can be given as examples of the generated passwords obtained as output:

Machine Generated Password: xc2D<Oo,H30QKIu[

Human Generated Password: Castiny

Human Generated Password: crkrer

Human Generated Password: sue~\$11

Human Generated Password: Kua~\$dor

Human Generated Password: Lestin12

Human Generated Password: mercedes

Human Generated Password: wHatever

Human Generated Password: america

Human Generated Password: micHael1

Human Generated Password: samantHa

Human Generated Password: cHocolate

Human Generated Password: patrick

Human Generated Password: iloveme

Human Generated Password: qwerty123/osP

4.16. Mix4800 Dataset Experiments

In this experimental case, the RNN LSTM model was trained with the Mix4800 wordlist containing 4800 mixed passwords. Some passwords from the Mix4800 wordlist are given as examples below:

cfreyjdf

qwertyui

Bajaone112

12345678910

123456789q

1q2w3e4r5t6y

fuk19600

YUanTUo2012

11223344

liverpool

87654321

FQrG7cS493

qwerty12345

123654789

micHelle

3d8cubaj2E

6V21wbgad

1029384756

147852369

123qweasd

iloveyou1

asdf1234

4815162342

internet

999999999

jennifer

trustno1
1234567891
uQa9Ebw445
blink182
123qwe123
123456123
123qweasdzxc
princess1
starwars
dIWtgm8492
741852963
jordan23
P3rat54797

In the experimental scenario where the Mix4800 dataset was used to train the RNN LSTM model, a total of 578 passwords were generated. While 553 of these generated password outputs were classified as human-like passwords, 25 of them were classified as machine-like passwords. For the Mix4800 dataset, the model was trained by running a total of 10.000 iterations. The model was run for a total of 10.000 iterations, and the optimal loss was found at the 10.000th iteration, the training process was stopped at the 10.000th iteration. The model was run with a total of 212 epoch values. The model was run with the hyperparameter values specified in Figure 4.107 and Figure 4.108.

```
# Hyperparams
BATCH_SIZE = 32
SEQUENCE_LENGTH = 25
LEARNING_RATE = 0.002
DECAY_RATE = 0.97
HIDDEN_LAYER_SIZE = 1024
CELLS_SIZE = 3

TEXT_SAMPLE_LENGTH = 500
SAMPLING_FREQUENCY = 1000
LOGGING_FREQUENCY = 1000
```

Figure 4.107. Hyperparameters Value For Mix4800 Dataset

```
Selected dataset: C:/Users/gkest/Desktop/crhacker/crhackDataset/mix4800/
Batch size: 32
Sequence length: 25
Learning rate: 0.002
Decay rate: 0.97
Hidden layer size: 1024
Cells size: 3
Tensor size: 37600
Batch size: 32
Sequence length: 25
Batches size: 47
```

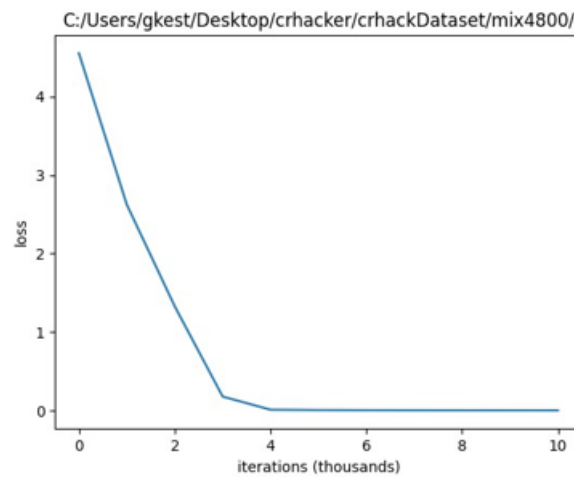
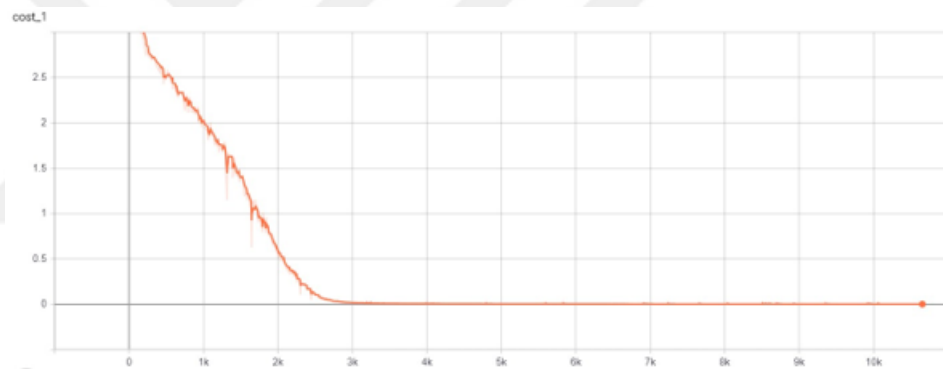
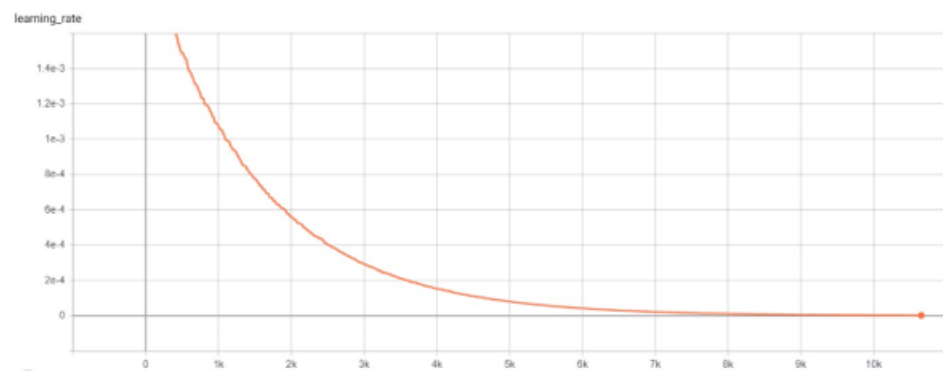
Figure 4.108. Hyperparameters Value For Mix4800 Dataset

Figure 4.109 indicates that the loss function value approaches optimal while the model is running. The most optimal loss value reached for this data set is 0.00233113719150424.

While the percentage value of the humanlike password produced in the wordlist as an output set created by the model is 95.7% , the percentage value of the machine-like password in the generated output set is 4.3%

Figure 4.110 illustrates cost value for each iteration. So, Figure 4.111 represents learning rate for each iteration. On the other hand, Figure 4.112 shows logits for each iteration.

As can be inferred from the results, the percentage of human-like passwords in the generated dictionary is higher than the percentage of machine-like passwords. When the model is trained with input mixed wordlists, the output wordlist produces a higher percentage of passwords that real people may be able to create, compared to machine-like passwords.

**Figure 4.109.** Loss Function Value For Each Iteration**Figure 4.110.** Cost Value For Each Iteration**Figure 4.111.** Learning Rate For Each Iteration

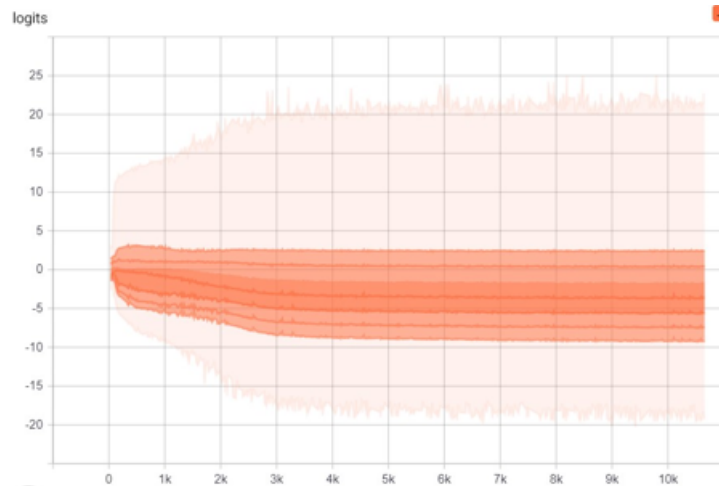


Figure 4.112. Logits For Each Iteration

Moreover, when the model is trained using the Mix4800 dataset, the following passwords can be given as examples of the generated passwords obtained as output:

Machine Generated Password: 4^FtykWzu'VemVo[J

Human Generated Password: skyking1

Human Generated Password: williams1

Human Generated Password: fernandra

Human Generated Password: 1q2w3e4r5t6y7u

Human Generated Password: Handsome

Human Generated Password: tHunkstar1

Human Generated Password: sunsHine1

Human Generated Password: linkedby1

Human Generated Password: blessed1

Human Generated Password: Hello123

Human Generated Password: shadow1

Human Generated Password: sweet16

Human Generated Password: babygurl1

Human Generated Password: wildcats

4.17. Mix204kv2 Dataset Experiments

In this experimental case, the RNN LSTM model was trained with the Mix204kv2 wordlist containing 204.000 mixed passwords. Some passwords from the Mix204kv2 wordlist are given as examples below:

princess55

powerplay

milka123

mexico08

laspalmas

katHarinal

kalevala

justynka

Hjlbjyjd

gotHic12

fillmore

excalibu

datodato

cHumcHum

Umbrella

aspirine

:UT8%_0h

yni<YfM%

jyB7O1IU

PU;14/C#

YFy=wO;@

O6HD>jU{

</2h6wqs

P-,on(*T

&U~Dp1M"

C4MfFjq`

```
I8h{"ST/  
eTD*-PdK  
/.0KbfGs  
Zb<8soUr  
O6K?_MC9  
KU[IgoF!  
[='macrv  
kK;JrjCt  
Lsp&b%OU  
?ue~$1x4
```

In the experimental scenario where the Mix204kv2 dataset was used to train the RNN LSTM model, a total of 6589 passwords were generated. While 5834 of these generated password outputs were classified as human-like passwords, 755 of them were classified as machine-like passwords. For the Mix204kv2 dataset, the model was trained by running a total of 138.000 iterations. The model was run for a total of 138.000 iterations, and the optimal loss was found at the 138.000th iteration, the training process was stopped at the 138.000th iteration. The model was run with a total of 57 epoch values. The model was run with the hyperparameter values specified in Figure 4.113 and Figure 4.114.

```
# Hyperparams
BATCH_SIZE = 32
SEQUENCE_LENGTH = 25
LEARNING_RATE = 0.002
DECAY_RATE = 0.97
HIDDEN_LAYER_SIZE = 1024
CELLS_SIZE = 3

TEXT_SAMPLE_LENGTH = 500
SAMPLING_FREQUENCY = 1000
LOGGING_FREQUENCY = 1000
```

Figure 4.113. Hyperparameters Value For Mix204kv2 Dataset

```
Selected dataset: C:/Users/gorkemk/Desktop/crhacker/crhackDataset/mix204kv2/
Batch size: 32
Sequence length: 25
Learning rate: 0.002
Decay rate: 0.97
Hidden layer size: 1024
Cells size: 3
Tensor size: 1928800
Batch size: 32
Sequence length: 25
Batches size: 2411
```

Figure 4.114. Hyperparameters Value For Mix204kv2 Dataset

Figure 4.115 indicates that the loss function value approaches optimal while the model is running. The most optimal loss value reached for this data set is 1.6197173595428467.

While the percentage value of the humanlike password produced in the wordlist as an output set created by the model is 88.5% , the percentage value of the machine-like password in the generated output set is 11.5%

Figure 4.116 illustrates cost value for each iteration. So, Figure 4.117 represents learning rate for each iteration. On the other hand, Figure 4.118 shows logits for each iteration.

As can be inferred from the results, the percentage of human-like passwords in the generated dictionary is higher than the percentage of machine-like passwords. When the model is trained with input mixed wordlists, the output wordlist produces a higher percentage of passwords that real people may be able to create, compared to machine-like passwords.

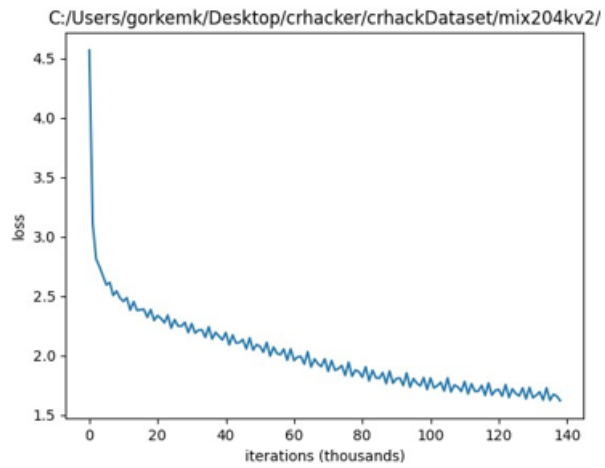


Figure 4.115. Loss Function Value For Each Iteration

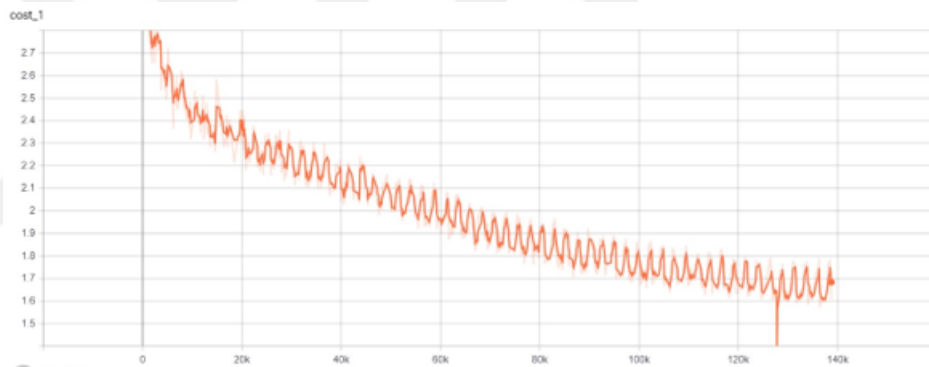


Figure 4.116. Cost Value For Each Iteration

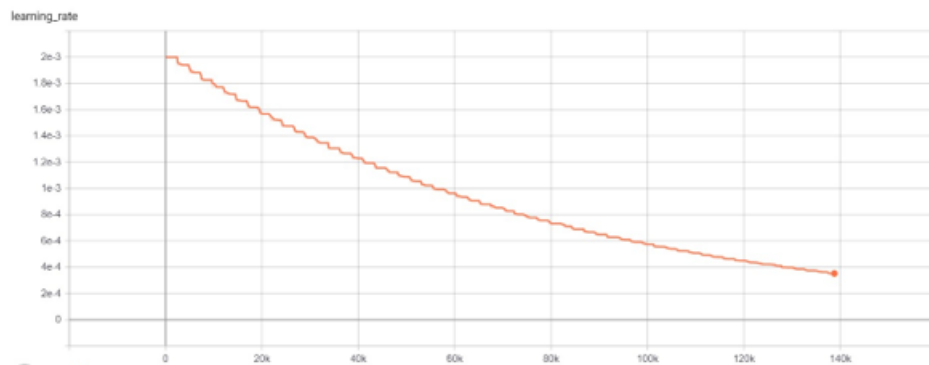


Figure 4.117. Learning Rate For Each Iteration

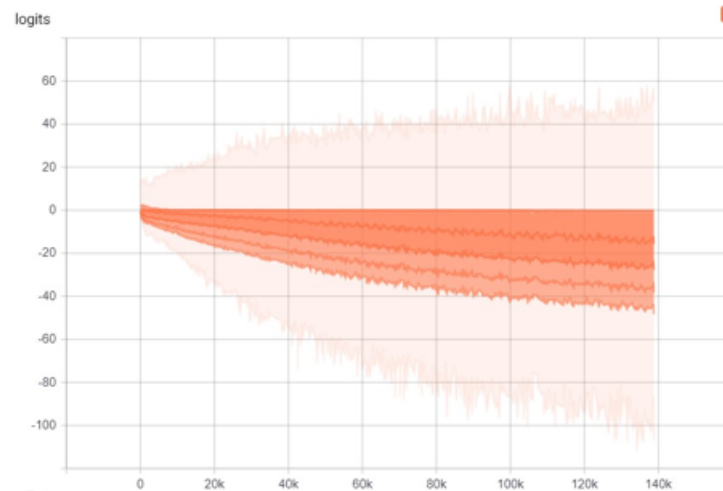


Figure 4.118. Logits For Each Iteration

Furthermore, when the model is trained using the Mix204kv2 dataset, the following passwords can be given as examples of the generated passwords obtained as output:

Machine Generated Password: e)M~9e?L<.

Machine Generated Password: [J%6*OGubgrxqk9

Machine Generated Password: &D_Pu-rz2uQfvgn

Machine Generated Password: hzH2py2Q_QnzF

Machine Generated Password: W5XBy{"JcSGg!Sg

Machine Generated Password: up\$m])D^|*C00*h

Human Generated Password: epra27

Human Generated Password: jasartHee

Human Generated Password: metteala

Human Generated Password: bickramice

Human Generated Password: lileni12

Human Generated Password: ilylickilas

Human Generated Password: bodsor32

Human Generated Password: footer4

Human Generated Password: nocktley09

Human Generated Password: disriect14

Human Generated Password: swage11

Human Generated Password: penapomol

Human Generated Password: yambiel2
Human Generated Password: yahoo75
Human Generated Password: tabby28
Human Generated Password: weston123
Human Generated Password: jordan18
Human Generated Password: fuzzia123
Human Generated Password: aphill1988
Human Generated Password: bigdaddy
Human Generated Password: zacHary123
Human Generated Password: ligHtbell
Human Generated Password: dougHton1
Human Generated Password: toonstar1
Human Generated Password: password25
Human Generated Password: roman1993
Human Generated Password: ber_crows
Human Generated Password: beatHer4
Human Generated Password: Jexies123
Human Generated Password: Buckat123
Human Generated Password: forever15
Human Generated Password: daniel12345
Human Generated Password: soccer15
Human Generated Password: yankees25
Human Generated Password: killer08
Human Generated Password: ifeligHt1
Human Generated Password: lilwayne
Human Generated Password: kirill1995
Human Generated Password: ilove1700
Human Generated Password: warcraft11
Human Generated Password: fallout1
Human Generated Password: elena1974
Human Generated Password: monkey19

Human Generated Password: Hammer01
Human Generated Password: aNTIBUL1
Human Generated Password: pooHbear7
Human Generated Password: wHatever!
Human Generated Password: pipell123
Human Generated Password: pentium2
Human Generated Password: Hunter88
Human Generated Password: flower007
Human Generated Password: football100
Human Generated Password: infamous1
Human Generated Password: HeatHer4
Human Generated Password: guru1234
Human Generated Password: tHomas20
Human Generated Password: nasta1234
Human Generated Password: martin15
Human Generated Password: fisHbone
Human Generated Password: diana2007
Human Generated Password: cHaparra1
Human Generated Password: catHarsis
Human Generated Password: godislove7
Human Generated Password: Happyfeet1
Human Generated Password: giovanni123
Human Generated Password: isabella1
Human Generated Password: TwiligHt1
Human Generated Password: 3sisters

Table 4.3. lists what percentage of the passwords generated for each password dataset used in the training phase are human-like-generated passwords and what percentage are machine-like-generated-passwords.

While the model was trained with each listed dataset, the total number of iterations of the model was run, in which iteration the optimal loss value was found, the epoch value information at the stage where the optimal value loss was found, and the optimal loss value information are listed in Table 4.4.

For each password dataset used in the training phase, a list of how many passwords were generated in total for the trained models, how many of the generated passwords were human-like-generated passwords and how many were machine-like-generated-passwords, is given in Table 4.5.

Table 4.3. Password distributions by percentages in the produced output data set

Password Dataset	% Percentage of Human Like Password in Generated Output Set	% Percentage of Machine Like Passwords in Generated Output Set
Top62WPAProbable-v3	94.86%	5.14%
Top62WPAProbable-v4	98.38%	1.62%
Top4800-WPA-probable-v2	95.27%	4.73%
Top4800-WPA-probable-v3	98.69%	1.31%
Top204Thousand-WPA-probable-v2	98.41%	1.59%
Rockyou62	95.18%	4.82%
Rockyou4800	97.46%	2.54%
rockyou204kv2	98.59%	1.41%
Kaonashi62	93.71%	6.29%
Kaonashi4800	97.11%	2.89%
Kaonashi204k	98.70%	1.30%
machine-gen-62-dataset	4.65%	95.35%
machine-gen-4800-datasetv2	2.69%	97.31%
machine-gen-204k-dataset	29.36%	70.64%
Mix62	95.16%	4.84%
Mix4800	95.67%	4.33%
Mix204kv2	88.54%	11.46%

Table 4.4. Optimal Loss Value Distributions For Each Dataset

Password Dataset	Optimal Loss Value	Epoch Value	Iteration Value
Top62WPAProbable-v3	0.0039016827940940857	833	5000/6000
Top62WPAProbable-v4	0.0010261926800012589	800	20000/25000
Top4800-WPA-probable-v2	0.05504848435521126	666	8000/10000
Top4800-WPA-probable-v3	0.0011434524785727262	387	19000/20000
Top204Thousand-WPA-probable-v2	0.4581467807292938	26	61000/62000
Rockyou62	0.0011539638508111238	363	8000/9000
Rockyou4800	0.022482870146632195	357	15000/15000
rockyou204kv2	0.05690601095557213	72	136000/136000
Kaonashi62	0.007724171970039606	444	12000/13000
Kaonashi4800	0.014146371744573116	307	16000/16000
Kaonashi204k	0.13649006187915802	61	137000/137000
machine-gen-62-dataset	0.00498690502718091	529	9000/10000
machine-gen-4800-datasetv2	0.010801034979522228	103	9000/9000
machine-gen-204k-dataset	3.972756862640381	44	144000/145000
Mix62	0.002126453211531043	416	10000/10000
Mix4800	0.00233113719150424	212	10000/10000
Mix204kv2	1.6197173595428467	57	138000/138000

Table 4.5. The Amount of Password Generated For Each Dataset

Password Dataset	Human Like Generated Password Count	Machine Like Generated Password Count	Total
Top62WPAProbable-v3	314	17	331
Top62WPAProbable-v4	1216	20	1236
Top4800-WPA-probable-v2	503	25	528
Top4800-WPA-probable-v3	979	13	992
Top204Thousand-WPA-probable-v2	2721	44	2765
Rockyou62	533	27	560
Rockyou4800	845	22	867
rockyou204kv2	7250	104	7354
Kaonashi62	596	40	636
Kaonashi4800	739	22	761
Kaonashi204k	6313	83	6396
machine-gen-62-dataset	19	390	409
machine-gen-4800-datasetv2	11	398	409
machine-gen-204k-dataset	1280	3080	4360
Mix62	531	27	558
Mix4800	553	25	578
Mix204kv2	5834	755	6589

5. CONCLUSION

In the scope of this work, the RNN LSTM Model was trained with real password leak dumps and output password lists generated, and then those generated output passwords were classified as either human-like generated passwords or machine-like generated passwords. The model tries to guess meaningful patterns that might be able to commonly used by real users, such as real users' favourite football team, birthdates, words, pet names, important names, city names etc. and it classifies those kinds of patterns as human-like generated passwords. On the other hand, passwords similar with the ones that are able to be created by password managers, not including any real-user-like patterns are classified as machine-like generated passwords. Due to the fact that real users tend to create easy-to-remember and memorable password patterns in order to access their accounts, it is considered that when the password dictionaries are built by using human-like classified passwords and reducing machine-like passwords in the dictionaries, performing dictionary attacks can be made more powerful and effective, and it is aimed to increase the quality of dictionary attacks by using wordlists also known as password dictionaries which are composed by utilizing human-like generated passwords.

In this context, the Crhacker program was created and different experimental scenarios were carried out and the analyses of the conducted experiments were interpreted and explained in detail.

6. REFERENCES

- Adams, A. and Sasse, M. A. 1999. "Users are not the enemy" *Commun. ACM*, vol. 42, no. 12, dec 1999., pp. 40-46 [Online]. Available: <https://doi.org/10.1145/322796.322806>
- Bengio, Y., Simard, P. and Frasconi, P. 1994, "Learning Long-Term Dependencies With Gradient Descent Is Difficult," *In Proc. IEEE Transactions on Neural Networks.*, Vol. 5(2), pp. 157-166.
- Betker, J., Goh, G., Jing, L., Brooks, T., Wang, J., Li, L., Ouyang, L., Zhuang, J., Lee, J., Guo, Y., Manassra, W., Dhariwal, P., Chu, C., Jiao, Y. and Ramesh, A. 2022, "Improving Image Generation with Better Captions," *In Proc. DALL-E 3 Research Paper*, OpenAI, pp. 1-19.
- Biddle, R. and Stobert, E. 2014, "The password life cycle: user behavior in managing passwords" *10th Symposium On Usable Privacy and Security*
- Bowman, S. R. 2023, "Eight Things to Know about Large Language Models," *New York University*, pp. 1-16
- Brown, T. et al. 2020, "Language models are few-shot learners," *In OpenAI Research, arXiv:2005.14165*, pp. 1-75
- Bubeck, S., Chandrasekaran, V., Eldan, R., Gehrke, J., Horvitz, E., Kamar, E., Lee, P., Lee, Y.T., Li, Y., Lundberg, S., Nori, H., Palangi, H., Riberio, M.T. and Zhang, Y. 2023, "Sparks of Artificial General Intelligence: Early experiments with GPT-4," *In Microsoft Research.*, pp. 1-155
- Chanda, K. 2016, "Password security: An analysis of password strengths and vulnerabilities," *In International Journal of Computer Network and Information Security*, Vol.8, pp. 23-30
- Conrad, E., Misener, S. and Feldman, J. 2017. "Eleventh hour cissp" *Chapter 5*, pp. 117–134

- Cubrilovic, N. 2009 "RockYou Hack: From Bad To Worse": <https://techcrunch.com/2009/12/14/rockyou-hack-security-myspace-facebook-passwords/> [Last access date: 2023-12-10].
- Dai, Z., Yang, Z., Yang, Y., Carbonell, J. G., Le, Q.V. and Salakhutdinov, R. 2019, "Transformer-xl: Attentive language models beyond a fixed-length context," *In Proc. 57th Annual Meeting of the Association for Computational Linguistics*, pp. 2978–2988
- Dell’Amico, M., Michiardi, P. and Roudier, Y. 2010, "Password Strength: An Empirical Analysis," *In Proc. IEEE INFOCOM, San Diego, CA, USA*, pp. 1-9 doi: 10.1109/INFOCOM.2010.5461951.
- Durmuth, M., Angelstorf, F., Castelluccia, C., Perito, D. and Abdelberi, C. 2015. "Omen: Faster password guessing using an ordered markov enumerator" *in Engineering Secure Software and Systems*, pp. 119–132.
- Dusek, O., Novikova, J., and Rieser, V. 2020, "Evaluating the state-of-the-art of end-to-end natural language generation: The E2E NLG challenge," *Computer Speech Language*, Vol. 59, pp. 122-157
- Elmaghraby, A. and Abdelwahab, O. 2018, "Deep learning based vs. Markov chain based text generation for cross domain adaptation for sentiment classification," *In Proc. IEEE International Conference Inf. Reuse Integr. (IRI)*, pp. 251-255
- Elondou, T., Manning, S., Mishkin, P. and Rock, D. 2023, "GPTs are GPTs: An Early Look at the Labor Market Impact Potential of Large Language Models," *In OpenAI, OpenResearch*, pp. 1-36
- Gers, F.A., Schmidhuber, J. and Cummins, F. 2000, "Learning To Forget: Continual Prediction With LSTM," *In Neural Computation.*, Vol. 12(10), pp. 2451-2471.
- Hanna, D.M. 2023, "The Use of Artificial Intelligence Art Generator Midjourney in Artistic and Advertising Creativity," *In Journal of Design Sciences and Applied Arts.*, Faculty of Applied Arts Damietta University, Vol. 4 , Issue 2, June 2023, pp. 42-58, DOI: 10.21608/jdsaa.2023.169144.1231

- Hashcat "Hashcat rule-based attack." [Online]. Available: <https://hashcat.net/wiki/doku.php?id=rulebasedattack>
- Hellman, M. 1980. "A cryptanalytic time-memory trade-off" *IEEE Transactions on Information Theory*, vol. 26, no. 4, pp. 401–406
- Hochreiter, S. 1991, "Untersuchungen Zu Dynamischen Neuronalen Netzen," *In PhD dissertation Technische Universitat Munchen.*
- Hochreiter, S. and Schmidhuber, J. 1997, "Long short-term memory," *In Neural Comput.*, Vol. 9, No. 8, pp. 1735-1780
- Hochreiter, S. and Schmidhuber, J. 1997, "Long Short-Term Memory," *In Neural Computation.*, Vol. 9(8) pp. 1735-1780.
- Housmand, S., Aggarwal, S. and Flood, R. 2015. "Next gen pcfg password cracking" *in IEEE Trans. Inf. Forensics Secur.* 10, pp. 1776–1791.
- Huh, J.H., Oh, S., Kim, H., Beznosov, K., Mohan, A. and Rajagopalan, S.R. 2015, "Surpass: system initiated user-replaceable passwords" *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, ACM
- Jin, H., Cao, Y., Wang, T., Xing, X. and Wan, X. 2020, "Recent advances of neural text generation: Core tasks, datasets, models and challenges" *Sci. China Technol. Sci.*, Vol 63, No. 10, pp. 1-21.
- Koutnik, J., Greff, K., Gomez, F. and Schmidhuber, J. 2014, "A clockwork RNN," *In Proc. International Conference of Machine Learning.*, pp. 1863-1871
- Kumar, H., Kumar, S., Joseph, R., Kumar, D., Singh, S.K., Kumar, A.M. and Kumar, P. 2013. "Rainbow table to crack password using md5 hashing algorithm" *2013 IEEE Conference On Information And Communication Technologies*, pp. 433-439
- Li, J., Zeigler, E., Holland, T., Papamichail, D., Greco, D., Grabentein, J. and Liang, D. 2020, "Common Passwords and Common Words in Passwords," *In Proc. Trends*

- and Innovations in Information Systems and Technologies.*, Springer International Publishing, pp. 818-827
- Li, Z., Han, W. and Xu, W. 2014. "A large-scale empirical analysis of Chinese web passwords" in *Proceedings of the 23rd USENIX Security Symposium*, USENIX Security 14, pp. 559–574.
- Luo, J., Deng, J, X., Lu, C. and Liu, H. 2019. "Recurrent neural network based password generation for group attribute context-ware applications" *IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems*, (HPCC/SmartCity/DSS), pp. 2688–2693
- Ma, J., Yang, W., Luo, M. and Li, N. 2014. "A study of probabilistic password models" in *IEEE Symposium on Security and Privacy SP*, pp. 689–704.
- Melicher, W., Kurilova, D., Segreti, S.M., Kalvani, P., Shay, R., Ur, B., Bauer, L., Christin, N., Cranor, L.F. and Mazurek, M.L. 2016, "Usability and security of text passwords on mobile devices" *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, ACM
- Narayanan, A. and Shmatikov, V. 2005. "Fast dictionary attacks on passwords using time-space tradeoff" in *12th ACM Conference on Computer and Communications Security*, pp. 364–372.
- Nosenko, A., Cheng, Y. and Chen, H. 2022. "Learning password modification patterns with recurrent neural networks" in *Secure Knowledge Management In The Artificial Intelligence Era, ser. Communications in Computer and Information Science. Cham: Springer International Publishing* , pp. 110 – 129.
- O’Gorman, L. 2003. "Comparing passwords, tokens and biometrics for user authentication." *Proceedings of the IEEE*, vol. 91, no.12, pp. 2021-2040
- Oechslin, P. 2003. "Making a faster cryptanalytic time-memory trade-off" in *Advances in Cryptology - CRYPTO 2003, D. Boneh, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg* , pp. 617–630.

- OpenAI 2023, "GPT-4 Technical Report," *In OpenAI Research*, pp. 1-100
- Pfleeger, C.P., Pfleeger, S.L. and Margulies, J. 2015, "Security in Computing, 5th ed.," *Prentice Hall Press*
- Pinkas, B. and Sander, T. 2003. "Securing passwords against dictionary attacks" *Proceedings of the ACM Conference on Computer and Communications Security*
- Radford, A., Narasimhan, K., Salimans, T. and Sutskever, I. 2018, "Improving Language Understanding by Generative Pre-Training," *In OpenAI Research*, pp. 1-12
- Radford, A., Wu, J., Child, R., Luan, D., Amodei, D. and Sutskever, I. 2019, "Language models are unsupervised multitask learners," *In Open AI Research*, Vol. 1, No. 8, pp. 9
- Saharia, C., Chan, W., Saxena, S., Li, L., Whang, J., Denton, E., Kamyar, S., Ghase-mipour, S., Karagol Ayan, B., Mahdavi, S. S., Lopes, R.G., Salimans, T., Ho, J., Fleet, D.J. and Norouzi, M. 2022, "Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding," *In Imagen Research Paper*, Google Research, Brain Team, Toronto, Ontario, Canada, pp. 1-46.
- Sarker, IH. 2021, "Machine learning: Algorithms, real-world applications and research directions," *In SN Computer Science.*, Vol.2(160) pp. 1-21, DOI:10.1007/s42979-021-00592-x
- Sasijaran, M., Guhan, S., Reni, M., Maheswari, M. and Roselin Mary, S. 2023, "Image Generation With Stable Diffusion AI," *In International Journal of Advanced Research in Computer and Communication Engineering*, IJARCCCE, Vol. 12, Issue 5, pp. 643-647, DOI: 10.17148/IJARCCCE.2023.125106
- Schuster, M. and Paliwal, K.K. 1997, "Bidirectional recurrent neural networks," *In IEEE Trans. Signal Process.*, Vol. 45, no. 11, pp. 2673- 2681
- Siegler, M.G. 2009 "One Of The 32 Million With A RockYou Account? You May Want To Change All Your Passwords. Like Now.": <https://techcrunch.com/2009/12/14/rockyou-hacked/> [Last access date: 2023-12-10].

- Tasevski, P. 2015. "Password attacks and generation strategies" DOI: 10.13140/RG.2.1.1247.8807
- Tatli, E.I. 2015. "Cracking more password hashes with patterns" *IEEE Transactions on Information Forensics and Security*, vol. 10, pp. 1656–1665
- Toshevska, M., Stojanovska, F., Zdravevski, E., Lameski, P. and Gievska, S. 2020, "Explorations into deep learning text architectures for dense image captioning," *In Proc. Federated Conference Computer Science Information Sys.*, pp. 128-136
- Ur, B., Noma, F., Bees, J., Segreti, S.M., Shay, R., Bauer, L., Christin, N. and Cranor, L.F. 2015, "I added ‘!’ at the end to make it secure”: observing password creation in the lab" *Eleventh Symposium On Usable Privacy and Security*.
- Veras, R., Collins, C. and Thorpe, J. 2014. "On the semantic patterns of passwords and their security impact" *In NDSS*
- Vu, K.-P.L., Proctor, R.W., Bhargav-Spantzel, A., Tai, B.-L., Cook, J. and Schultz, E. 2007, "Improving password security and memorability to protect personal and organizational information" *in International Journal of Human-Computer Studies*, Vol. 65 No. 8, pp. 744-757.
- Wang, B. and Vastola, J. 2022, "Stable Diffusion, Machine Learning from Scratch, Stable Diffusion a Tutorial" *In Harvard Scholar.*, Harvard University Information Technology, pp. 1-66.
- Wang, Q. and Qin, Z. 2010. "Stronger User Authentication for Web Browser." *3rd International Conference on Advanced Computer Theory and Engineering*, University of Electronic Science and Technology of China.
- Weir, M., Aggarwal, S., Medeiros, B.D. and Glodek, B. 2009. "Password cracking using probabilistic context-free grammars" *in 30th IEEE Symposium on Security and Privacy*, pp. 391–405.
- Zviran, M. and Haga, W.J. 1990, "Password Security: An Exploratory Study" *Naval Postgraduate School*, Monterey, CA.

CURRICULUM VITAE

MEHMET GÖRKEM KESTANE

EDUCATION

Master Degree 2021-2024	Akdeniz University Institute Of Natural And Applied Sciences Department Of Computer Engineering
Bachelor of Science 2017-2018	Medjimurje Technical University Faculty of Electronics and Computer Technologies Computer Science Department
Bachelor of Science 2015-2019	Pamukkale University Faculty of Engineering Department of Computer Engineering

WORK EXPERIENCE

Computer Engineer 2023-	The Scientific And Technological Research Council Of Turkey (TÜBİTAK)
Software Engineer Intern Jul 2019- Sep 2019	İŞBİR Synthetics Weaving Industry Inc.
Computer Engineer Intern Jul 2017- Aug 2017	Balikesir Electromechanical Industrial Facilities, BEST Transformers Inc.

PUBLICATIONS

The Simulated Annealing Approach to the Traveling Salesman Problem, Bachelor of
Studies Thesis