



WPI

VisNLP 2.0: A Visual-Based Educational Support Platform for Teaching and Learning Neural Network-Based NLP Analytics

A Major Qualifying Project submitted to the faculty of Worcester Polytechnic Institute in partial fulfillment of the requirements for the Degrees of Bachelor of Science in Computer Science and Data Science.

Authored by:

Jack Gomes (DS)

Daniel Johnson (DS)

Garrett McMerriman (DS)

John Prominski (CS)

Henry Yoder (CS)

Advised by:

Professor Chun-Kit Ngan (DS)

This report represents the work of one or more WPI undergraduate students submitted to the faculty as evidence of completion of a degree requirement. WPI routinely publishes these reports on the web without editorial or peer review.

Abstract

Natural Language Processing (NLP) has become increasingly relevant in our data-driven world and is being implemented widely today due to the abundance of natural language data. However, mastering NLP processes is not a simple task, particularly for those without a comprehensive background in the field. Further, the use of neural networks in many commonly used NLP approaches makes it difficult to interpret the input-output relationships of these models, creating the "black-box" problem in data science. To address these challenges, we develop and implement a web-based interactive visual NLP learning platform that enables learners to study some fundamental neural network-based NLP techniques, topics, and applications. Specifically, the technical contribution of this work is threefold: (1) To present popular neural network-based NLP analytics methods in a step-by-step linear format that is easy to comprehend. (2) To eliminate the 'black box' problem found in neural network-based NLP learning resources by providing continuous real-data examples. (3) To enable users to interpret model outputs through interactive visual demos that apply neural network-based NLP method outputs.

Acknowledgements

The team would first like to acknowledge our project advisor, Professor Chun-Kit Ngan, for his support and guidance throughout this MQP. The team would also like to thank Dr. Sara Abdali, Data & Applied Scientist at Microsoft Corporation, for generously sharing her expertise in NLP and data science and providing valuable feedback and direction throughout our project. In addition, we would like to thank the participants who contributed their time and feedback and made our user trials and platform evaluations possible. Lastly, we would like to acknowledge the previous research teams who have laid the groundwork for the VisNLP project and provided us with a solid foundation for our research.

Contents

Contents	4
1 Introduction.....	8
1.1 Motivation & Background	8
1.2 Research Challenges	9
1.3 Summary of Our Approaches and Contributions.....	10
1.3.1 Word2vec Module	12
1.3.2 Para2vec and Sen2vec Module.....	12
1.3.3 Adam Optimizer Module.....	13
1.3.4 Songs Recommendations Application Module	13
1.3.5 Fake News Detection Application Module	13
1.3.6 Platform Evaluation.....	14
1.4 Summary of Included Chapters.....	14
2 Literature Review	14
2.1 Word2Vec Module Related Materials	14
2.2 Sen2vec/Para2vec Module Related Materials	15
2.3 Adam Optimizer Module Related Materials	16
2.4 Applications Modules Related Materials	18
3 Methodology	20
3.1 VisNLP 2.0 Development and Implementation Pipeline.....	20
3.2 Word2vec Module	21
3.3 Para2vec and Sen2vec Module.....	23
3.4 Adam Optimizer Module.....	28
3.5 Songs Recommendations Application Module	33
3.6 Fake News Detection Application Module	36
4 VisNLP 2.0 Educational Support Platform.....	38
4.1 VisNLP 2.0 Web Platform	38
4.2 VisNLP 2.0 Homepage	40
4.3 Word2vec Module	41
4.4 Paragraph2vec and Sentence2vec Module.....	46
4.5 Adam Optimizer Module.....	49

4.6 Songs Recommendations Application Module	53
4.7 Fake News Detection Application Module	55
5 Platform Evaluation and Discussion	58
6 Conclusions and Recommendations	62
6.1 Conclusion	62
6.2 Recommendations and Future Work	62
7 Appendix A: Step-by-Step Simulation Training Output Data	64
8 Appendix B: Platform Evaluation Survey Full Results Report.....	69
9 References.....	73

List of Figures

Figure 1: VisNLP 2.0 Platform Architecture Relational Diagram	11
Figure 2: Adam Optimizer Equations	17
Figure 3: VisNLP 2.0 NLP Methods Development Pipeline	20
Figure 4: VisNLP 2.0 NLP Applications Development Pipeline	21
Figure 5: Word2vec - Add Bias.....	23
Figure 6: Sen2vec/Para2vec/Doc2vec Neural Network Architecture.....	27
Figure 7: Adam Optimizer Average Loss Over Iteration Plot.....	30
Figure 8: T-SNE Plot of Song Embeddings	34
Figure 9: Cosine Similarity Formula	34
Figure 10: Song Curation Cosine Similarity T-SNE Plot	35
Figure 11: Example output of model prediction	37
Figure 12: Classification plot outputted by model.....	37
Figure 13: Example of footer content	38
Figure 14: Adam Optimizer - Overview Screen Capture.....	39
Figure 15: Word2vec – Multiply Matrices Screen Capture	40
Figure 16: VisNLP 2.0 Web Platform Home Page.....	41
Figure 17: Word2vec - Overview.....	41
Figure 18: Word2vec – Text Preparation	42
Figure 19: Word2vec – One-Hot Encoding	42
Figure 20: Word2vec - Generate Training Batch	43
Figure 21: Word2vec – Multiply Matrices.....	44
Figure 22: Word2vec – Add Bias	44
Figure 23: Word2vec – Log SoftMax	45
Figure 24: Word2vec – End of Epoch.....	45
Figure 25: Sen2vec Overview Page	46
Figure 26: Sen2vec Simulation Step 1	47
Figure 27: Adam Optimizer Overview Page.....	49
Figure 28: Adam Optimizer Simulation - Step 16	50
Figure 29: Adam Optimizer Simulation - Step 22	52
Figure 30: Song Recommendation Overview	53
Figure 31: Song Recommendation Application Home	54
Figure 32: Song Recommendation Application Example.....	55
Figure 33: Fake News Application Overview Page	56
Figure 34: Fake News Application Pre-User Prediction Page	57
Figure 35: Fake News Application Post-User Prediction Page	57
Figure 36: Platform Evaluation Survey Results Report (Visual Summary)	59
Figure 37: Respondent NN-Based NLP Background - Survey Results Visualized	59
Figure 38: Platform and Content Quality - Survey Results Visualized	60
Figure 39: Effectiveness of Platform Qualities - Survey Results Visualized	61
Figure 40: Respondent Consensus - Survey Results Visualized.....	61

List of Tables

Table 1: Adam Module word2vec Training Description.....	29
Table 2: Adam Optimization Runtime Description	29
Table 3: Adam Optimizer Simulation Sub-Step List	31
Table 4: Adam Step-by-step Learning Process Summary.....	51
Table 5: Platform Evaluation Survey Questions	58

1 Introduction

1.1 Motivation & Background

Natural language processing (NLP) has recently shot up in prominence and its growth shows no signs of slowing. This is largely due to the vast amount of available natural language data we have access to. Natural language refers to how humans communicate amongst themselves, whether through spoken or written words. Natural language can be characterized by various patterns and structures, such as grammatical rules, idioms, or cultural references. These underlying patterns offer rich insights into human behavior and cognition, presenting a significant amount of potential utility for computation [1].

However, extracting value from these large swaths of natural language data requires effective processing and management. This is where the field of NLP comes into play. Natural language processing is a branch of computer science which utilizes processor-intensive computational techniques to analyze and interpret human language. By applying computational and statistical methods, NLP techniques allow us to perform various tasks with human languages, potentially giving insight into observed human language phenomena. The field of NLP spans a range of approaches, from statistical methods to complex neural network-based methods, all of which can be used to uncover hidden patterns present in natural language data [2].

NLP technologies have already significantly contributed to modern life, utilized in a range of technologies that many of us use on a daily basis [3]. Digital assistants such as Siri and Alexa rely heavily on NLP techniques to parse and interpret user inputs. Analyzing a user's spoken commands, NLP algorithms can be applied to determine the user's intent and respond appropriately [4]. Another everyday example of NLP in action is the autocorrect feature used in most modern implements for text-based communication. For example, the autocorrect software found on devices running iOS takes into consideration the context of the conversation, the frequency of words used, as well as the user's typing history to maximize the accuracy of given suggestions. This involves analyzing and interpreting human language in real-time, a core function of NLP [5]. NLP even plays an essential role in spam detection tools used across various email services as well as other forms of communication. By examining the contents of an email and identifying patterns that have been associated with messages classified as spam, NLP can be used to identify and filter out unwanted messages [6].

While the field of NLP has already made significant measurable advances in society, the industry grows bigger every day. The latest breakthroughs in NLP have led to the development of much-discussed technologies such as GPT-3, which is capable of understanding and responding to complex natural language queries in real-time [7]. Moreover, advancements in machine learning and deep learning techniques have resulted in newer NLP techniques becoming more accurate and efficient. As a result, NLP will only become more relevant, especially with the increasing availability of natural language data, better processing capabilities, and more advanced techniques. The field of machine learning has experienced tremendous growth in recent years, with the market

for machine learning estimated to have grown from \$1 billion in 2016 to almost \$9 billion in 2022 [8]. All signs point towards this growth continuing, with Google Trends data indicating that interest in machine learning has spiked in 2022 [9]. Given the ever-increasing importance of natural language processing, including deep learning neural network-based NLP, ample and accessible education of the field is crucial for those who wish to enter the field.

1.2 Research Challenges

While the importance of ample and accessible education of NN-based NLP has been identified, learning these subjects can present a significant challenge for many learners. This is largely because mastering machine learning and NLP processes is no trivial task, especially for novice learners or those without a comprehensive background in the field. Additionally, many commonly used NLP approaches are based on neural networks. Due to these models being developed to be understood by artificial intelligence systems, rather than humans, it can be challenging to interpret the input-output relationships of these models. This is what is referred to as the 'black box' problem [10]. Demographics of those potentially facing obstacles while attempting to learn about NLP include novice professionals, junior data scientists, and college students. To overcome these challenges, it is imperative learners are provided with a solid understanding of the linguistics and computation that NLP is based on.

In order to achieve this goal, we can examine existing approaches used to teach NN-based NLP, as to identify the qualities of a successful teaching approach. Some of these approaches include self-study, involving reading and learning largely through outside materials. This approach suffers from a steep learning curve, often requiring a learner to have both strong self-learning capabilities and an existing foundation in the related topics [11]. Many approaches also involve a more formal instructional learning environment, courses completed either on campus or remotely online. This solution may provide a remedy to the challenges faced in self-learning but suffers due to a lack of accessibility for most potential learners, who will not have access to or be able to afford professionally taught course materials [12].

Several potential solutions have been developed to address these shortcomings. One such solution is an ontology-based educational tool created by Zobia and Stefania [13], which presents a tree diagram of conceptual class nodes and subclass nodes of NLP, combining the strengths of visual connectivity and interactivity. However, this approach lacks the depth necessary to effectively combat the black box problem, failing to provide in-depth explanations and descriptions of the topics at hand. Another approach is the Word Embedding Visual Inspector (wevi) simulation platform, a visual tool that allows users to understand the working mechanisms of word2vec, one popular word embedding model, and how different hyperparameters can affect its outcome [14]. However, this approach is non-cohesive, arbitrary, and does not pair the NLP concepts being applied with in-depth explanations. In short, there is a gap in existing materials for a platform that addresses the teaching and learning of neural network-based NLP techniques and

their applications, surpassing the inadequacy of existing learning techniques to meet all necessary qualities.

1.3 Summary of Our Approaches and Contributions

To bridge the gap identified in section 1.2, we proposed and developed a visual based educational support platform for teaching and learning neural network-based natural language processing. This educational support platform is an extension of VisNLP, a similar learning platform with its focus set on statistical-based NLP and enables learners to study the core processing components of statistical NLP analytics in sequence through interactive visual diagrams and charts with a practical example [15]. Our primary objective is to build on the success of the original platform and further develop and expand with VisNLP 2.0. This would entail having a web-based platform that is easily accessible, user-friendly, and interactive. In VisNLP 2.0 our primary focus is on teaching NN-based approaches for NLP in a similar way. The primary contributions of this project are threefold:

- (1) Presenting popular neural network-based NLP analytics methods in a step-by-step linear format that is easy to comprehend.
- (2) Eliminating the 'black box' problem found in neural network-based NLP learning resources by providing continuous real-data examples.
- (3) Enabling users to interpret model outputs through interactive visual demos that apply neural network-based NLP method outputs.

The VisNLP 2.0 platform is composed of five interconnected modules that cover a variety of topics and techniques relevant to neural network-based natural language processing. A platform architecture diagram which details the relationships between these modules is shown below, in Figure 1:

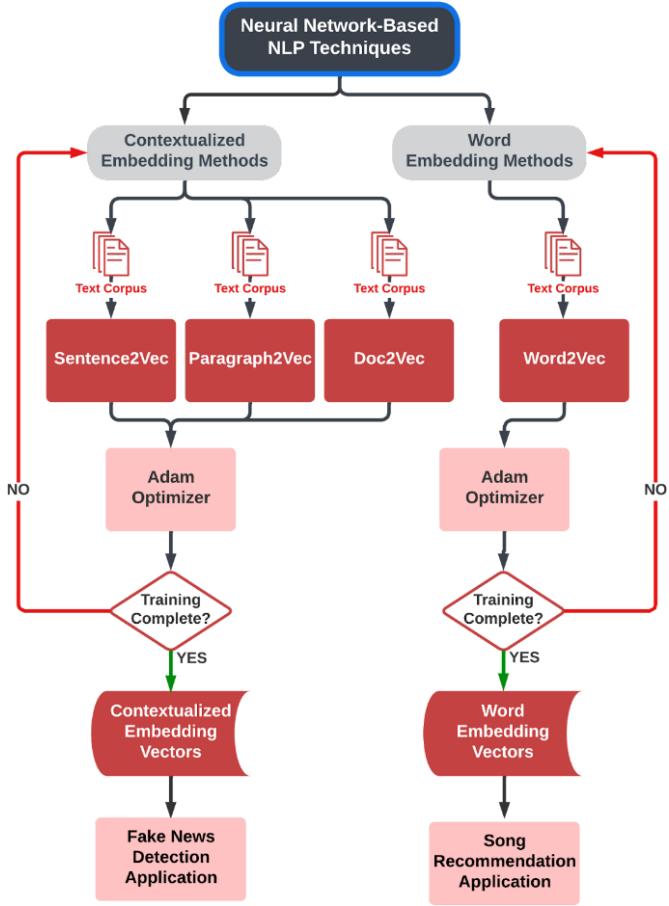


Figure 1: VisNLP 2.0 Platform Architecture Relational Diagram

As shown above in Figure 1, the VisNLP platform breaks down NN-based NLP techniques into two categories: (1) Word embedding methods, which generate fixed-size vector representations for individual words in the vocabulary and (2) Contextualized embedding methods, which capture context-dependent meanings of words by generating embeddings based on the entire input sequence.

For word embedding methods, the platform details word2vec in a step-by-step format that is easy to comprehend, with explanations and descriptions relating to the related higher-level theory at every step. The word2vec technique produces word embedding vectors with the help of an optimizer. The platform then demonstrates how these vectors can be interpreted and used in a real example with a song recommendation application using word2vec. For the contextualized embedding methods, the platform details three techniques, sen2vec, para2vec, and doc2vec, which produce contextualized embedding vectors with the help of an optimizer. These are paired with step-by-step explanations relating to the higher-level theory. The platform also showcases a visual application using these contextualized embedding vectors produced using doc2vec, which demonstrates fake news detection. The platform also contains a detailed step-by-step

demonstration of the Adam optimizer algorithm, utilized in each of the previously mentioned techniques. Further, the embedding technique modules and the optimizer are shown on continuous real-data examples where the dataflow of real model tensors is displayed. Meanwhile, the application modules enable users to interpret model outputs through interactive visual demos which apply NN-based NLP method outputs. In the subsequent sections, we will cover the role of each module.

1.3.1 Word2vec Module

The first module covered in the VisNLP 2.0 platform is word2vec, one of the foremost NN-based NLP techniques used for word vectorization. The process of word vectorization is used to produce embeddings for each word in an input corpus of text, meant to represent the relations between each word. These embeddings are based on each word's co-occurrence pattern and, when word2vec runs enough iterations on an input corpus of scale, begin to resemble semantic meaning. There are two models primarily used in word2vec: Continuous Bag Of Words (CBOW) and Skip-Gram (SG). The word2vec module visually covers 10 iterations' worth of real output data in a visual format both simple enough to be accessible to novice users and thorough enough to remain useful to experienced users. This will help all users understand the processes in detail and make the word2vec architecture more accessible to anyone who wishes to learn.

1.3.2 Para2vec and Sen2vec Module

Building from the word2vec visual module, the sen2vec visual module can be used to visualize the relationship between different sentences. In VisNLP 2.0, the sen2vec visual module shows a complete step-by-step analysis of both training models to fully demonstrate the sen2vec algorithm. This can help users understand how the algorithm is analyzing and summarizing the text. The para2vec visual module can be used to visualize the relationship between words in a paragraph. In VisNLP 2.0, the para2vec visual module shows a complete step-by-step analysis of both training models to fully demonstrate the para2vec algorithm. This can help users understand how the algorithm is identifying and classifying paragraphs based on their context. The doc2vec visual module has been omitted from VisNLP 2.0 to reduce redundancy, as a dedicated module would share a great number of similarities with the para2vec module. However, the doc2vec algorithm can be particularly useful for analyzing larger text documents and for tasks such as document classification and topic modeling. Overall, incorporating these visual modules into the VisNLP 2.0 platform can provide users with a more intuitive and accessible way to understand the workings of NN-based NLP analytics when generating contextualized embeddings, ultimately improving their ability to effectively analyze and summarize text data. Understanding how NN-based NLP analytics work is crucial for optimizing their performance, one key aspect of this being the use of effective optimizers to tune a model's parameters.

1.3.3 Adam Optimizer Module

For the NN-based techniques mentioned in the previous modules, the training process involves iteratively adjusting the model parameters until a given level of accuracy is achieved for producing predictions on new inputs. Here, optimizers are essential for effective and efficient parameter tuning during this process, as they dictate these parameter adjustments. Further, a need is established to cover this component of NLP in order to help remedy the “black box” problem. In VisNLP 2.0 we choose to focus on the Adam optimizer, a popular and fairly ubiquitous algorithm used in NLP and machine learning tasks. In short, the Adam optimizer operates as a variant of stochastic gradient descent, using adaptive learning rates and momentum to speed up convergence and improve performance. In VisNLP 2.0, the purpose of the Adam optimizer module is to show how it works, step by step, using a continuous real-data example, to explain exactly where and how different hyper-parameters affect the Adam optimization process. It is intended to allow learners to understand the connection between the outputs obtained from a forward pass in a neural network-based NLP model and the corresponding updates performed on the model’s parameters during backpropagation.

1.3.4 Songs Recommendations Application Module

In addition to the step-by-step modules demonstrating to a user how embeddings are generated, we can compound this learning by demonstrating real-world applications of these embeddings. A song curation application is an effective example because of its relative simplicity and widespread understanding of its usage. This application uses word2vec to train a dataset of 10,000 song titles and artist names, forming vectors that show relationships between each song. The cosine similarity of each vector is compared to determine which songs are most similar to one another, thereby recommending a set of new songs to a potential listener. This application is easily visualized, and the outputs use popular songs with the intention of making the results easy to understand.

1.3.5 Fake News Detection Application Module

In addition to the song recommendation application, which implements word2vec, the VisNLP 2.0 platform contains an application that utilizes contextualized embeddings. Using the doc2vec model, the Fake News Detection application covers yet another way to apply generated embeddings. This example works well for teaching doc2vec processes because it is simple enough for a novice user to follow along. Additionally, it demonstrates the use of support vector machine (SVM) technology. This highlights how NN-based NLP techniques can be paired with other technologies to complete a desired task. Due to the complicated nature of NLP and SVM

techniques, representing this process with a visual module is imperative for teaching as it allows the user to visualize how the vectors are categorized together. These vectors are visualized using via SVM plots, a format chosen to easily highlight the decision boundary for the classification.

1.3.6 Platform Evaluation

To evaluate the effectiveness of VisNLP 2.0, user testing was conducted, collecting online survey responses from each user. Participants with varying levels of NLP knowledge were given time to explore and learn from each module in a self-guided manner. After testing the platform, a brief online survey was administered to assess its success and gather feedback on strengths and weaknesses. The objectives of this feedback were to (1) evaluate the general effectiveness of the VisNLP 2.0 platform in achieving its objectives, and (2) identify the importance and effectiveness of each of the various components of the VisNLP 2.0 platform in order to inform future development.

1.4 Summary of Included Chapters

This report is organized into six chapters, which are as follows: Chapter 2 provides a literature review of the key concepts related to neural network-based Natural Language Processing. Chapter 3 covers the development and implementation details of the VisNLP 2.0 platform, including detailed block diagrams, and any pertinent figures, formulas, and algorithms. Chapter 4 provides a comprehensive overview of the finished VisNLP 2.0 platform and its modules, detailing the entire teaching and learning process from start to finish. Chapter 5 presents the results of the user testing and platform evaluation survey and contains a discussion on this feedback. Chapter 6 summarizes the findings of the report, draws conclusions, and makes recommendations for future work. Chapter 7 serves as an appendix that includes important figures and complete versions of materials referenced throughout the report.

2 Literature Review

This section provides a review of literature that is extensively referenced and used throughout this paper, covering their contents and theory that serve as the foundation for the research presented.

2.1 Word2Vec Module Related Materials

Within the category of NN-based NLP analytics, word2vec is one of the best documented techniques. Initially developed in 2013 by Tomas Mikolov and others at Google [16], word2vec is used to create distributed vector representation of words in a given input text. These vectors, called word embeddings, are highly complex and represent relationships between words within a larger text sample, which begins to resemble semantic meaning when performed at a high enough scale. This model was expanded upon in 2014 by Xin Rong, writing in depth about parameter usage in both the Continuous Bag Of Words (CBOW) and Skip Gram (SG) models [17]. This paper was written to address the lack of documentation on the parameter learning process in detail, a goal similar to that of VisNLP 2.0.

An existing visual platform for word2vec was created, also by Rong, in 2016. The word embedding visual inspector, or wevi, allows a user to step through word embeddings as they develop with each iteration of word2vec on a number of pre-determined datasets [14]. Additionally, wevi allows users to adjust some parameters and even input their own dataset if they so desire. Where wevi falters, however, is that when it runs at a rate high enough to produce effective results, it goes far too fast for a user to reasonably be able to make out what's really happening. You can go step by step as well, but still the simulation does not show any of the mathematical calculations happening within the model, or even what calculations are really happening to produce the embedding vectors on display. As a tool to visualize word2vec it is dynamic, interactive, and accessible, but ultimately functions as a highly polished black box.

2.2 Sen2vec/Para2vec Module Related Materials

In NLP, learning distributed vector representation of words is a well-known framework for learning word vectors. The goal is to predict a word given the other words in a certain context, with every word mapped to a unique vector as represented by a column matrix. This method allows for simple and efficient vector operations for natural language understanding, language modeling, and statistical machine translation. After training, words with similar meaning are mapped to a similar position in the vector space [18].

An extension of this method is the paragraph vector, a distributed memory model inspired by methods for learning word vectors. In this model, every paragraph is mapped to a unique vector represented by a column matrix, and every word is also mapped to a unique vector represented by a column matrix. The paragraph vector and word vectors are averaged or concatenated to predict the next word in a context. The paragraph token can be thought of as another word, acting as a memory that remembers what is missing from the current context or the topic of the paragraph. The success of this method can be observed in two document similarity datasets, where the paragraph vector method is able to outperform other methods such as Latent Dirichlet Allocation (LDA). Additionally, paragraph vectors allow for finding documents of interest through simple and intuitive vector operations, making them useful for local and non-local browsing of large corpora [18] [19].

However, the word2vec model does not consider paragraph factors in semantic learning. To address this limitation, the para2vec model was proposed. This model introduces a paragraph component to predict the current word, either through the Distributed Memory (DM) or Distributed Bag of Words (DBOW) models. The DM model predicts the current word through context, adding a paragraph ID and generating a paragraph vector. The paragraph vector and context components are accumulated and connected together for training. The DBOW model is extended through skip-gram, taking the paragraph vector based on the paragraph ID as input and predicting the context word from within each input paragraph. Both models learn not only semantic information but also word order information, improving the accuracy of predictions. The DM model uses word and paragraph vectors in semantic learning, while the DBOW model takes paragraph vectors as input and predicts context words. These models are useful in NLP for tasks such as document classification and topic modeling [20].

2.3 Adam Optimizer Module Related Materials

The Adam optimizer was first introduced by Diederik Kingma and Jimmy Ba in 2014. Adam is an optimization technique for stochastic objective functions that combines an adaptive learning rate with momentum through lower-order moments. Due to its fast convergence and stability, it has become a staple optimization technique in machine learning tasks, including neural network-based NLP tasks [21].

First, we will discuss some advantages of the Adam optimization method. Starting with its improvements over regular gradient descent, such as an adaptive learning rate as opposed to a fixed learning rate, and the use of momentum through making use of past gradients. A key advantage from the adaptive learning rate in Adam is allowing for the learning rate to be adjusted based on the estimated gradient variance for each parameter, resulting in a more efficient and stable optimization. The incorporation of momentum in the Adam method allows for continued movement in a direction based on exponentially decaying influence from the direction taken in past iterations, which helps it mitigate oscillations and avoid reaching false loss minima. These qualities make Adam especially well-suited for noisy or sparse gradients, consequently making it particularly useful for applications such as NLP. Additionally, the Adam optimization method is computationally efficient, requiring little memory and making it well-suited for training large-scale neural networks. This is because it requires minimal variables to be computed as compared to standard stochastic gradient descent. Finally, the Adam optimizer provides robust options of hyperparameters, which allows for easier tuning compared to other optimization methods. Specifically, the learning rate (α), the constant (ϵ), and the exponent decay rates (β_1) and (β_2) [22].

While the Adam optimizer offers several advantages over standard stochastic gradient descent, there are also inherent disadvantages as well as limitations that have been identified through further studies. One main disadvantage of the Adam optimization method is poor generalization performance for certain data, specifically when there are significant outliers or there is overfitting. Another disadvantage of Adam is that it can converge to suboptimal solutions,

especially in high-dimensional parameter spaces. Further studies have also shown that Adam can suffer from instability and erratic behavior, particularly when the learning rate is too high or too low [23]. Additionally, while momentum typically would counteract this, it is possible in some cases that the adaptive learning rate can cause the optimizer to get stuck at a local minimum or experience oscillations near the optimal solution. Further, while Adam can perform exceptionally well on a variety of machine learning tasks, it can sometimes hinder the convergence and performance for certain types of problems.

Next, we can examine how the Adam optimization algorithm works. It can efficiently optimize the parameters of a model by adapting the learning rate for each parameter based on the estimate of the first and second moments of the gradients. The algorithm computes a moving average of the gradients, where the momentum term allows it to consider the direction of past gradients. The first moment estimate is computed as a running average of the gradients, and the second moment estimate is computed as a running average of the squared gradients. The algorithm then combines these estimates with hyperparameters (β_1) and (β_2) to compute the adaptive learning rates for each parameter. The main equations used in the Adam optimization algorithm are shown as follows in Figure 2:

$$m_t = \beta_1 * m_{t-1} + (1 - \beta_1) * g_t$$

$$v_t = \beta_2 * v_{t-1} + (1 - \beta_2) * g_t^2$$

$$\hat{m}_t = m_t / (1 - \beta_1^t)$$

$$\hat{v}_t = v_t / (1 - \beta_2^t)$$

$$\theta_t = \theta_{t-1} - \alpha * \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$$

Figure 2: Adam Optimizer Equations

Here, (m_t) and (v_t) are the first and second moment estimates, respectively. (g_t) is the gradient at the time step (t), (β_1) and (β_2) are the decay rates for the first and second moment estimates. (\hat{m}_t) and (\hat{v}_t) are the bias-corrected first and second moment estimates, respectively. Further, (α) is the learning rate and (ϵ) is a small value used for avoiding division by zero. Finally, (θ_t) refers to the parameters which are updated by the optimizer.

Through performance testing conducted in the Adam paper, it showed superior performance compared to other optimization methods like Adagrad, RMSProp, and stochastic gradient descent (SGD). Specifically, in terms of convergence speed and accuracy, the Adam optimizer has been shown to outperform Adagrad and RMSProp on several benchmark datasets. Further studies on Adams performance find similar results [24]. It has also been found that RMSProp can sometimes outperform Adam on image classification tasks for example, and that similarly there are cases where SGD with momentum can outperform Adam on some deep learning tasks [24].

In summary, while the Adam optimization method may not be superior to other optimizers in every task it is still a widely popular choice for many ML tasks due to its convergence speeds and stability. Further, it is particularly effective for many NLP tasks, making it vastly popular in the field.

2.4 Applications Modules Related Materials

Word2vec for song curation

Word2vec is a popular natural language processing technique that has found widespread applications in the field of music curation [25]. Several studies have explored the use of Word2Vec to analyze music lyrics and curate songs based on similarity in lyrics and themes.

Ribeiro and Brasil (2017) used word2vec to analyze song lyrics and build a recommendation system for music playlists. The study demonstrated that word2vec can effectively capture the semantic relationships between words in song lyrics and help identify songs with similar themes and emotions. He et al. (2017) also used word2vec to analyze the lyrics of popular songs and build a recommendation system for music playlists. The study demonstrated that word2vec can be used to identify songs with similar lyrics and themes, and that the recommendation system built using word2vec outperformed other traditional recommendation systems.

Similarly, Zhang et al. (2019) used word2vec to analyze song lyrics and build a personalized music recommendation system. The study demonstrated that word2vec can be used to identify songs with similar themes and emotions, and that the recommendation system built using word2vec can provide personalized music recommendations based on the user's listening history.

Overall, these studies demonstrate the potential of word2vec as a tool for curating music playlists and providing personalized music recommendations [25]. By analyzing the semantic relationships between words in song lyrics, word2vec can help identify songs with similar themes and emotions, building recommendation systems to cater to individual user preferences.

Sentiment analysis in machine learning

Sentiment analysis is a rapidly growing area of research in machine learning and NLP. With the explosive growth of social media, there has been an increasing need to automatically extract and analyze opinions expressed in text data. One common approach in sentiment analysis is to use supervised machine learning algorithms, such as Support Vector Machines (SVMs) and Naive Bayes classifiers. These algorithms learn to classify text data into positive, negative, or neutral sentiments based on labeled training data. A study conducted in 2020 compared the performance of different machine learning algorithms for sentiment analysis and found that SVMs outperformed other algorithms, achieving an accuracy of 93% [26].

Another trend in sentiment analysis is the use of deep learning algorithms, such as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs). Deep learning algorithms have been shown to outperform traditional machine learning algorithms, especially for complex text data. In 2019 a hybrid model was proposed that combines CNNs and RNNs for sentiment analysis, which achieved higher accuracy compared to other deep learning models [27]. Furthermore, some studies have explored the use of sentiment lexicons in sentiment analysis. A sentiment lexicon is a collection of words or phrases that are associated with positive or negative sentiment. Sentiment lexicons can be used to determine the sentiment of text data by counting the occurrence of positive and negative words. A study performed in 2008 proposed a sentiment lexicon-based approach for sentiment analysis, which achieved high accuracy compared to other approaches [28].

In conclusion, sentiment analysis is an active research area in machine learning and NLP. Recent studies have shown that machine learning algorithms, such as SVMs and deep learning models, can achieve high accuracy in sentiment analysis tasks. Additionally, sentiment lexicons can be used to improve the performance of sentiment analysis. Future research in sentiment analysis could explore more advanced machine learning models, as well as the use of sentiment analysis in different domains, such as healthcare and finance.

Support Vector Machine (SVM)

Support Vector Machine (SVM) is a popular machine learning algorithm that has been extensively used in various fields, such as finance, healthcare, image classification, and text classification, for solving classification problems. SVM has been shown to be effective in high-dimensional spaces, where other algorithms may fail to achieve satisfactory performance. The key idea behind SVM is to find the best possible hyperplane that separates classes in high-dimensional feature spaces. Recently, many studies have focused on improving SVM classification performance by enhancing its accuracy and reducing computational complexity. One approach that has gained attention is feature selection, which involves selecting the most relevant features from the dataset to improve classification accuracy and reduce computational complexity. Feature selection methods are used to identify a subset of features that contribute most to the classification task. Many trials have been conducted in order to examine several feature selection methods and found that the Recursive Feature Elimination (RFE) algorithm coupled with SVM outperformed other methods, such as Principal Component Analysis (PCA) and Correlation-based Feature Selection (CFS) [29].

Another approach to improve SVM classification accuracy is by using ensemble methods. Ensemble methods involve combining multiple models to improve classification performance [29]. This method creates a set of SVM models using different hyperparameters and combines them using a weighted average of their classification results. Furthermore, some studies have proposed hybrid approaches that combine SVM with other machine learning algorithms. Hybrid approaches are used to leverage the strengths of different algorithms and improve classification accuracy. In 2020 a hybrid SVM and Extreme Learning Machine (ELM) approach for medical diagnosis was suggested, which achieved higher accuracy compared to traditional SVM and ELM

methods [30]. The proposed hybrid method uses SVM to handle the classification task and ELM to extract relevant features from the dataset.

In conclusion, SVM classification is a powerful machine learning algorithm that has been widely used in many applications. Several approaches have been proposed to improve its performance, such as feature selection, ensemble methods, and hybrid approaches. These approaches have been shown to enhance classification accuracy and reduce computational complexity. Future research in SVM classification could explore more advanced feature selection techniques, ensemble methods, and hybrid approaches to further improve its performance.

3 Methodology

This section details the development and implementation of the VisNLP 2.0 web-based educational support platform for teaching neural network-based natural language processing. The first part of this section covers the process of developing the web-platform, while the subsequent parts cover the specifics of each individual module.

3.1 VisNLP 2.0 Development and Implementation Pipeline

Shown below in Figure 3 is the general development and implementation pipeline for NN-based NLP methods modules. Specifically, the word2vec module, paragraph2vec and sentence2vec module, and the Adam optimizer module.



Figure 3: VisNLP 2.0 NLP Methods Development Pipeline

To generate our training data, we first run implementations of NN-based NLP on a sample corpus to generate outputs, which includes logging all intermediate calculations. Next, we interpret this data and categorize it by the sub-step of the technique or algorithm which it corresponds to. Utilizing this organized step-related data, we develop a visual and descriptive step-by-step representation of the NN-based NLP techniques. Finally, we implement these visuals in our web-based platform with a step-by-step simulation.

Next, shown below in Figure 4, is the general development and implementation pipeline for NN-based NLP application modules. Specifically, the songs recommendations module, and the fake news detection module.



Figure 4: VisNLP 2.0 NLP Applications Development Pipeline

First, the NN-based NLP models are trained on a targeted dataset to generate embedding vectors. Then we interpret the embedding vectors through additional computation. Then we develop a visual and descriptive way of representing these applied NN-based NLP model outputs. Finally, we implement these visuals in our web-based platform with an interactive application.

Finally, for both NN-based NLP method and application modules, we develop and implement an overview to summarize the step-by-step simulation or embeddings application.

3.2 Word2vec Module

This section describes the development and implementation of the word2vec module in the VisNLP 2.0 platform. In this section we sought to have a thorough step-by-step demonstration of word2vec. This demonstration should be simple enough that it will remain accessible to those not already familiar with the concepts on display, but thorough enough that it will still possess useful information to more experienced users.

Generating Real-data Outputs

In order to step through the full process of word2vec for a user, we needed real data outputs to use. Since our parameters would share a dimension with the length of the input text corpus' vocabulary, we opted to use a very short input text "I love data science." This way our demonstration visuals could show the real numbers in use and remain uncluttered and visually coherent, avoiding having any excessively complex vectors.

The real-data outputs used in the word2vec module were generated using the implementation provided to us by our advising professor [31]. From here there were a large number of console output statements added, so that the variables are tracked at each stage of the training process. In this stage we were less concerned with getting accurate outputs where the embeddings would reflect semantic meaning and more with being thorough in procuring data that could be used to walk a user through the entire training process.

The next step was to create a JSON schema to organize and store this data for use. The data overall is divided into epochs, one for each of the ten being used. The first subcategory is "constants", for data that remains the same across all epochs but may still be useful to access, primarily the

parameters word2vec was run with and the original input text and its vocabulary. The next category is “params”, with the straightforward purpose of containing the 3 parameter matrices as they are at the start of each epoch. The next category is “batch”, storing the context(s) and center(s) indices of words used in the training batch for each epoch. This category also stores the current embeddings corresponding to each word in the input training batch. The next category is “calcs”, storing calculated values along each step of the training process for display, along with a handful of dynamic footer content to be easily accessible and indexed the same way as the values it corresponds to. The final thing included is “next_params”, storing the parameter values at the start of the next epoch, for comparison in the transition between epochs. This data is stored in MongoDB and accessed through the backend.

Creating the step-by-step simulation

For the step-by-step simulation, we knew a large number of pages would need to be shown to the user in sequence and sought to make this as efficient as possible. For word2vec, there is a central loop of each epoch completed, which starts with generating a training batch and ends with updating the parameters with the Adam optimizer. This cycle was broken down into a loop of five steps: generating the training batch, parameter matrix multiplication, adding bias (parameter 3), either the log softmax or sigmoid function (depending on if CBOW or skip-gram), and the optimization step, where we link to the Adam module and display how it updates the parameters for the following epoch. In addition to each of these we have a handful of introductory slides, being the overview page, a page showing the input text which covers how word2vec prepares its text for use, and a page showing the input one-hot encoded vectors and explaining how one-hot encoded vectors work. This minimizes the amount of original code that needs to be written without sacrificing depth or quality, simply cutting out any redundancy where possible.

In addition to the code itself, we also sought to make the website efficient in terms of browser load. The website for VisNLP 1.0 had a separate html file and page on the website for every single page. For VisNLP 2.0 we sought to address this by simply having one html file per module and switching out elements dynamically using JavaScript. This both solves the issue of clogging up a user’s browsing history to an unnecessary degree and also makes it so that the page does not have to load each time a user advances.

For the word2vec module, much of the work went into making the page feel dynamic and interactive, rather than just a static slideshow of data. As shown in Figure 5 below, the main way in which this is done is through interacting with the user’s cursor. When highlighting over a cell of data, that cell will change color responsively and also highlight other cells that correspond to it. This can include other cells involved in a given calculation as well as parameters shown in the sidebar. These processes (as well as some other objects on the page that do not get highlighted when hovered over) also update the footer when the cursor falls on them. This makes the page feel more dynamic and also allows the user more information about what exactly they’re looking at. It makes the page far more accessible and informative without adding any clutter to the visual design whatsoever.

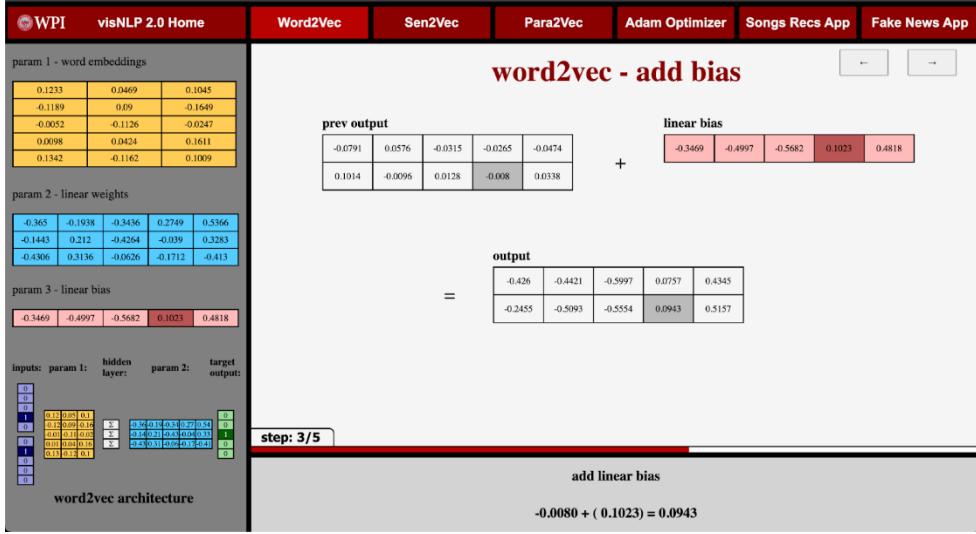


Figure 5: Word2vec - Add Bias

Each page in the word2vec module is designed to be responsive, loads instantaneously (minus about a quarter second of delay on pages at the start/end of an epoch where new data is called in from MongoDB), and dynamically resizes to the size of the user’s screen.

3.3 Para2vec and Sen2vec Module

In addition to the word2vec module, we have the paragraph2vec (para2vec) and sentence2vec (sen2vec) modules. This section describes the development and implementation of these modules in the VisNLP 2.0 platform.

Algorithm parameters

To implement sen2vec, para2vec, and doc2vec, we followed a similar approach based on word2vec. We used the Python programming language as well as the Python-based library, PyTorch to build and train the models. We utilized the open-source code library, paragraph-vectors, available on GitHub [32] for these models.

The parameters we used for training the models were, num_epochs, batch_size, num_noise_words, vec_dim, lr, model_ver, context_size, data_file_name, and max_generated_batches. Num_epochs is datatype int and is the number of iterations to train the model or in other words, the number of times every example is seen during training. Batch_size is of datatype int and is the number of examples per single gradient update. Num_noise_wrds is of datatype int and is the number of noise words to sample from the noise distribution. Vec_dim is of datatype int and is the dimensionality of vectors to be learned for paragraphs and words. Lr is of datatype float and is the learning rate of the Adam Optimizer. Model_ver is of datatype str, the input is either dm, for Distributed Memory, or dbow, for Distributed Bag of Words. The model versions are of the proposed models by (Le, & Mikolov, T. 2014). Context_size is of datatype int

and is half the size of a neighborhood of target words when the model is dm and context_size must be greater than 0. When model_ver is dbow, context_size must be equal to 0. Data_file_name is datatype str and is the name of the file in the data directory. Max_generated_batches is of datatype int and is the maximum number of pre-generated batches.

Algorithm data handling

The algorithms are powerful NN-based NLP techniques that have gained popularity due to their ability to learn continuous distributed representations of sentences and paragraphs. These techniques require large amounts of textual data to be trained effectively. However, handling such data can be challenging due to its volume, variety, and complexity. To overcome these challenges, it is important to carefully preprocess and organize the data before feeding it into the algorithm. In this way, data handling plays a crucial role in the success of this algorithm.

The data we used for training is stored in the data directory in the library. The data file format is that of an excel csv file. When the parameter data_file_name is called, it can be either the data path to the file or the data file name. In the individual excel csv file there must be a header in the file as the algorithm takes the first row as a header and not as data. The next rows can be filled with data and only data is entered into the first column.

Before feeding the data to the algorithm, it is essential to preprocess it. This involves converting the raw text data into a numerical format that can be easily processed by the algorithm. Some of the common preprocessing steps include tokenization, stemming, and stop word removal. Tokenization involves splitting the text data into individual words or tokens, while stemming involves reducing each word to its base form (e.g., “running” to “run”). Stop word removal involves removing common words that do not carry much meaning, such as “the”, “a”, and “an”. Proper preprocessing can significantly improve the performance of the algorithm by reducing noise and focusing on the most important features of the data. Once the data has been preprocessed, it is ready for the algorithm to begin training.

Algorithm Training

To train the models, we need to create a vocabulary of all the unique words/phrases in the dataset. The resulting words/phrases after preprocessing, form our vocabulary, which we assign a unique index. The index is used to represent each word/phrase as an embedding vector. For sen2vec, we do not create vocabulary on periods since it is operating on sentences. For para2vec and doc2vec, we do create vocabulary on periods as it considers the start and end of a new sentence within a paragraph or document. The vocabulary creation process is crucial as it helps to reduce the dimensionality of the input space and enable efficient training of the models.

Once the vocabulary is created, the algorithm creates paragraph IDs and initializes word vectors. Paragraph IDs are created to differentiate between different paragraphs in the dataset. Word vectors can be initialized randomly or with pre-trained embeddings, which can lead to better performance in some cases. Once the paragraph IDs, and word vectors are in place, an input matrix is created based on the paragraph ID, context words, and center words. This input matrix is then used to train the model.

The next step involves initializing the word and paragraph weight matrices along with a bias matrix. These weight matrices as well as the bias matrix are used to compute the predicted center word matrix, which is then passed through a softmax function. This softmax function generates a probability distribution over all the words in the vocabulary.

After the predicted center word matrix is generated, negative log likelihood loss is calculated using both the predicted center word matrix and the true word matrix. This loss function measures the difference between the predicted and actual probabilities of the center word given the paragraph ID and context words.

The Adam optimizer is then used to update the word and paragraph weight matrices along with the bias matrices based on the calculated loss. This optimization process is repeated multiple times (i.e., epochs) until convergence is met. The corresponding updated matrices are used in the next epoch, resulting in an improved performance.

By repeating this process, the model learns to generate more accurate paragraph vectors, which can be used for tasks such as text classification and sentiment analysis.

Creating the step-by-step simulation

The step-by-step simulation was created by using the inputs and outputs of each step during algorithm training. To simulate the algorithm, we first show the data that we want to train the model on. We then create paragraph IDs and vocabulary, which we use to initialize the word vectors. After that, we create an input matrix based on the paragraph ID, context words, and center words. This input matrix is then used to initialize the word and paragraph weight matrices, along with a bias matrix.

Next, we compute the predicted center word matrix by applying a weight matrix to the input matrix, followed by a softmax function. We then calculate the negative log likelihood loss using both the predicted center word matrix and the true word matrix.

Using the Adam optimizer, we show the updated word and paragraph weight matrices, along with the bias matrices, to minimize the loss function. The corresponding updated matrices are used in the next epochs, and this process is repeated until optimization is met, resulting in the final paragraph vectors.

By following these steps, we can simulate the algorithm and use it to train models for various NLP tasks, such as text classification and sentiment analysis. This step-by-step approach helps users understand the inner workings of the algorithm and enables them to apply it effectively to their own projects.

Main content

The main content in the step-by-step approach shows the generation of various matrices such as creation of paragraph IDs and vocabulary. It shows the algorithm architecture and the current state of the architecture during all the training steps. It shows the computation of various matrices and the results, such as how the input vectors are input through the weight matrix to generate a predicted center word matrix. It also shows the initial word and paragraph weight

matrices and their corresponding updated matrices which are color coded for an increase or decrease in value. Overall, the main content provides a comprehensive and visual understanding of the algorithm and the training process.

Description footer

The footer begins by welcoming the user to the algorithm and step-by-step approach. Once a model is selected the footer changes display. For each step the footer will change to describe the current step in detail, explaining every matrix, calculation, and output of the main content. The footer displays the information for all steps of the step-by-step approach. In conclusion, the footer provides detailed information about each step of the step-by-step approach, guiding the user through the algorithm and providing a deeper understanding of the matrices, calculations, and outputs involved.

User sidebar

The sidebar contains various information on key parameters, architecture, and vocabulary of the input data. The sidebar initializes with an image of the algorithm architecture, which is shown in Figure 6 below.

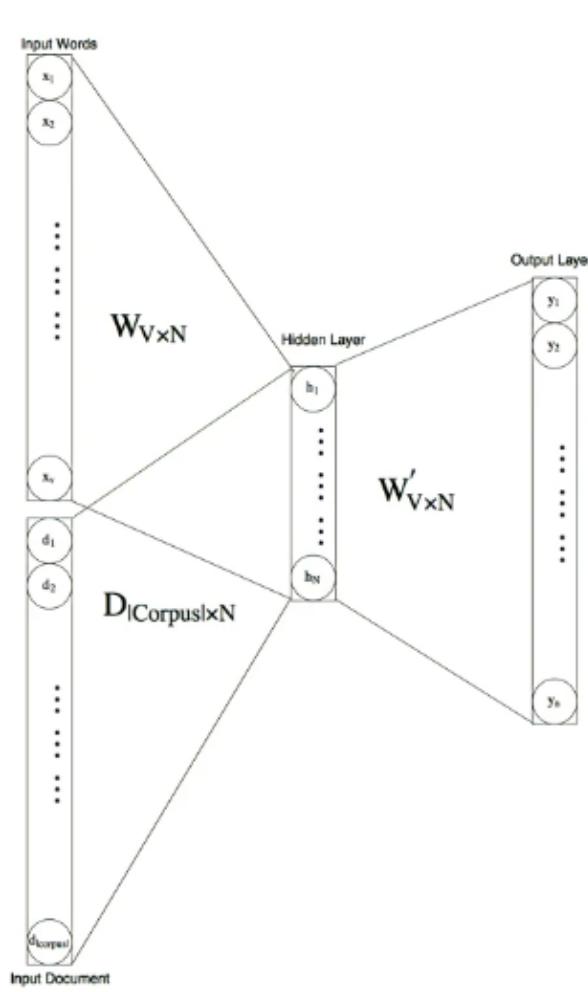


Figure 6: Sen2vec/Para2vec/Doc2vec Neural Network Architecture

This allows the user to understand what the initial architecture looks like. The next instance of the sidebar shows the parameters that were run to obtain the step-by-step approach information. It also contains a copy of the input data and a copy of the entire vocabulary used in the step-by-step approach. The next instance contains the same information as the previous, but with the addition of a copy of the model's input matrix with the words in place of the values. The rest of the sidebar instances show the previous instance's information allowing the user to always see it if needed. In conclusion, the sidebar in the step-by-step approach provides valuable information on key parameters, architecture, and vocabulary of the input data, as well as copies of the input data and matrices for reference.

Overview creation

The overview of the algorithm was created to introduce readers to the concept of sentence or paragraph embedding and their application in NLP. It explains how the algorithm works by training a neural network to learn the contextual relationships between words in a sentence or paragraph, allowing for the conversion of sentences and paragraphs into a vector representation.

The overview also highlights the advantages of using the algorithm over traditional NLP techniques and their widespread adoption in various NLP applications. Additionally, the overview provides a brief overview of the step-by-step process for performing sen2vec or para2vec, which can be used as a reference for users to understand the individual components of the algorithm. Overall, the goal of creating this overview is to provide a clear and concise explanation of the algorithm and its significance in the field of NLP. After discussing the overview of the sen2vec algorithm, the next section will go over the implementation and development of the Adam optimizer module.

3.4 Adam Optimizer Module

This section describes the development and implementation of the Adam optimizer module on the VisNLP 2.0 platform.

Generating real-data outputs

In order to create a step-by-step simulation of the Adam optimization process we first needed to generate the intermediate vectors and values from a given training process using Adam optimization. To do this we trained real example models using Adam and reported intermediate outputs and calculations that occur in the optimization process.

The objective function used in the Adam optimization module was a continuous bag of words (CBOW) word2vec model. We utilized an open-source implementation of word2vec which could be configured to use the Adam optimizer. This Python implementation utilized the PyTorch framework for building deep learning models [31].

In order to obtain meaningful outputs, it was necessary to tune the word2vec model for a given example corpus. For simplicity, we chose the corpus: “I love data science.” Further when accounting for unique words and adding the additional “unk” (unknown) embedding, the vocabulary in this word2vec model was: “unk”, “I”, “love”, “data”, “science”. Each of these word embeddings were then represented in an embedding dimension of three. Subsequently, the resulting parameters obtained from this configuration are as follows: the embeddings parameter-1 (3x5 2d tensor), the weights parameter-2 (5x3 2d tensor), and the bias parameter-3 (1x5 tensor). Next, the hyper-parameters of the word2vec training were tuned. The final hyper-parameters for the word2vec CBOW stochastic objective function are listed in Table 1 as follows:

Model Architecture:	CBOW
Mini-Batch Size:	2
Embedding dimensions:	3
Epochs:	40

Table 1: Adam Module word2vec Training Description

Next, it was necessary to tune the hyper-parameters of the optimizer. While running through 40 epochs of word2vec on such a short corpus is impractical for obtaining meaningful model outputs, we are still able to extract insight on the Adam optimization process for tuning the parameters. Further, we found that using the suggested hyper-parameters for the Adam optimizer produced satisfactory results on this word2vec model and sample corpus. These default hyperparameters for Adam optimizer were determined empirically by its creators in the original Adam paper "Adam: a method for stochastic optimization", and are listed in Table 2 below:

Optimizer:	Adam
Learning Rate:	0.05
Beta 1:	0.9
Beta 2:	0.999
Epsilon:	1e-08

Table 2: Adam Optimization Runtime Description

After choosing our sample corpus and tuning the hyper-parameters for both the CBOW word2vec model and the Adam optimizer, we trained our model for forty epochs. The following plot in Figure 7 shows the resulting average loss measures (shown as the blue line) obtained over iterations of the Adam optimization. The line of best fit (shown as the orange line) shows the downward trend of the model's average loss.

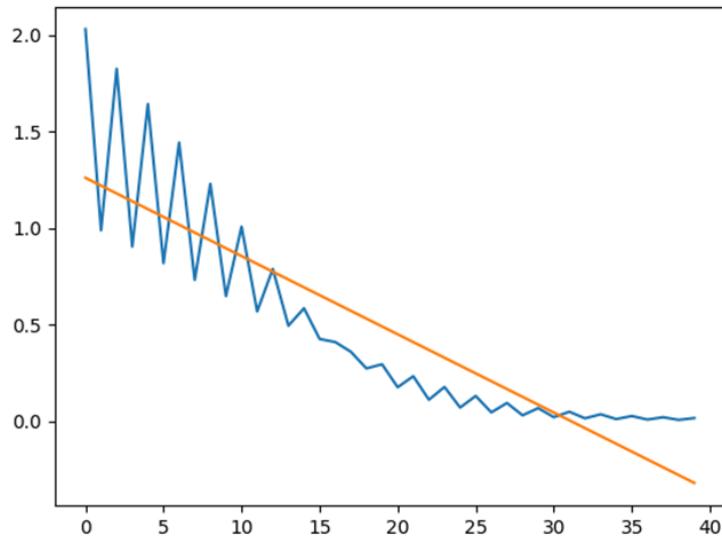


Figure 7: Adam Optimizer Average Loss Over Iteration Plot

After implementing a model that could be used for the step-by-step simulation it was necessary to align intermediate tensors and data to the corresponding parts of the overarching Adam optimization algorithm. This process required writing and filing data as corresponding Adam optimization steps in an organized JSON output file. While some intermediate values and tensors were easily obtainable, others were not explicitly accessible using the PyTorch library and needed to be backward computed before being written to the output file. The schema of the output file detailing the intermediate values and tensors that were collected can be seen in *Appendix B*.

The values and tensors obtained in the output file were then used to show the following twenty-two steps displayed in Table 3 below. These steps were intended to detail the Adam optimization process on an “atomic” level, such that necessary operations within the algorithm are broken down as much as possible.

Step 1	Perform New Optimization Step
Step 2	Perform Forward Pass: Contexts
Step 3	Perform Forward Pass: Linear Transform pt.1/2
Step 4	Perform Forward Pass: Linear Transform pt.2/2
Step 5	Compute the Loss: Take SoftMax
Step 6	Compute the Loss: Take Log
Step 7	Compute the Loss: Iteration Loss Measure
Step 8	Get Gradients with Respect to Stochastic Objective:
Step 9	Update Biased First Moment Vector: pt. 1/3
Step 10	Update Biased First Moment Vector: pt. 2/3

Step 11	Update Biased First Moment Vector: pt. 3/3
Step 12	Update Biased Second Raw Moment Vector: pt. 1/4
Step 13	Update Biased Second Raw Moment Vector: pt. 2/4
Step 14	Update Biased Second Raw Moment Vector: pt. 3/4
Step 15	Update Biased Second Raw Moment Vector: pt. 4/4
Step 16	Compute Bias-Corrected First Moment Estimate
Step 17	Compute Bias-Corrected Second Raw Moment Estimate
Step 18	Update Parameters with Adam: pt. 1/5
Step 19	Update Parameters with Adam: pt. 2/5
Step 20	Update Parameters with Adam: pt. 3/5
Step 21	Update Parameters with Adam: pt. 4/5
Step 22	Update Parameters with Adam: pt. 5/5

Table 3: Adam Optimizer Simulation Sub-Step List

Creating the step-by-step simulation

In order to create the step-by-step Adam optimizer simulation, it was necessary to create the web-platform framework for allowing a user to step through many unique iterations of many sub-steps. To do this, we tracked iteration states and step states. This way we were able to create generic visual step frames for all twenty-two of our Adam optimization sub-steps, which were able to dynamically load vector data onto that frame which corresponded to the current iteration state. With this configuration the user can step through forty continuous real-data iterations of all our 22 Adam optimizer sub-steps. A current iteration counter was included as well as a progress bar which indicates how far along the sub-steps the user is for that given iteration. Further, controls were added to allow users to step one sub-step forward or one sub-step backward. Additionally, a fast-forward button was added to allow users to increment forward an iteration rather than a single sub-step. Lastly a reset button allows the user to reset the simulation to its initial state.

For retrieving tensor and other sub-step data from our previously created JSON outputs file we dynamically queried the data using MongoDB. As the JSON data was organized by iteration, we were able to load the current iteration data, and use the desired parts of each record to populate our data visuals for each step.

As previously mentioned, the simulation modules in the VisNLP 2.0 web-platform assume a general layout of having a main content section, a description footer, and a sidebar dashboard. The implementation details for those three layout components in the Adam optimizer simulation module are detailed as follows:

Main Content

The main content section for each step of the Adam simulation requires a title, the real-data vector visuals, and the corresponding data visual labels. Both the step's titles and data visuals labels were specifically implemented for each of the twenty-two optimization sub-steps and were made to be generic among iterations. To show the sub-step being applied to the real-data tensors, each main content section includes data visuals showing how the data changed before and after applying the step. To create the 2d tensor data visuals D3 JavaScript library was utilized. Here, the vector data specific to that current iteration and sub-step was dynamically queried and populated into the D3 visuals.

Description Footer

The footer section for each step of the Adam simulation requires a step description and explanation, and the equation used in that step. The step explanation was specifically implemented for each of the twenty-two optimization sub-steps and was made to be generic among iterations. The step equation was intended to bring the most relevant equation to the forefront of the user's attention. These equations were either sections of the Adam algorithm or calculations specific to the objective function. Additionally, the parts of these equations that relate to the step being performed in the main content section are highlighted in red to represent the left side of the operation, and in blue to represent the right side of the operation. This same color coding is also reflected in the data visuals to allow users to see the connection. Finally, an equation title was added for each equation to highlight its significance. The step equation was specifically implemented for each of the twenty-two optimization sub-steps and was also made to be generic among iterations.

User Sidebar

The user sidebar section for each step of the Adam simulation requires a current model parameters dashboard, an Adam algorithm dashboard, and an average model loss measure over iterations dashboard. For the user to be able to see the model parameters that are always being optimized in the simulation, a current model parameter dashboard was implemented. Depending on the current iteration, data visuals showing these parameters were implemented using the D3 JavaScript library. Here, the vector data specific to that current iteration was dynamically queried and populated into the D3 visuals. Next, to provide clarity on what part of the Adam optimization algorithm was being executed at that point in the simulation, the Adam algorithm dashboard was implemented. Here, the current segment of the Adam algorithm in relation to the current sub-step was highlighted in red. Additionally, specific parts were highlighted to indicate the start and finish of the algorithm on the first sub-step of the first iteration as well as the last sub-step of the final iteration. Finally, to give the user insight into the overall optimization performance, a dashboard showing the average model loss measure over iterations was implemented. To create the line graph visuals plotting the progression of the loss over iterations the Plotly.js JavaScript library was utilized. Here, the loss array data specific to that current iteration and sub-step was dynamically queried and populated into the Plotly graph visuals.

Overview Creation

Lastly, we implemented an overview page intended to preface the Adam optimizer step-by-step simulation. The overview page first provides background for the simulation by introducing the Adam optimizer. Next, we cover how the Adam optimizer works at a high level, to give an overview of the theory involved as per the Adam paper. Additionally, we provide a summary of the Adam optimization. Finally, we preface what happens in the Adam optimizer step-by-step simulation, from where the user can enter and begin the step-by-step simulation.

3.5 Songs Recommendations Application Module

This section describes the development and implementation of the Song Curation application module on the VisNLP 2.0 platform.

Generating real-data outputs

The song curation application was developed based on the inspiration of two Github repositories [33] [34]. The main goal of the application is to generate personalized song recommendations based on the user's input. To achieve this, the data was trained using the word2vec model. The application is a direct extension of the word2vec model with some slight alterations.

The first step in creating the song curation application was to find an appropriate dataset. This dataset was obtained from one of the previously mentioned Github repositories. The dataset contained 10,000 song titles and artist names. To train the dataset, the CSV file containing the song titles and artist names was fed into the word2vec model to create vector embeddings. This process was carried out through supervised training as the vector embeddings needed to be correlated to their respective songs. Once the training was completed, a CSV of the song titles, artist names, and vector embeddings were saved as outputs, as well as a plot of the embeddings shown in Figure 8.

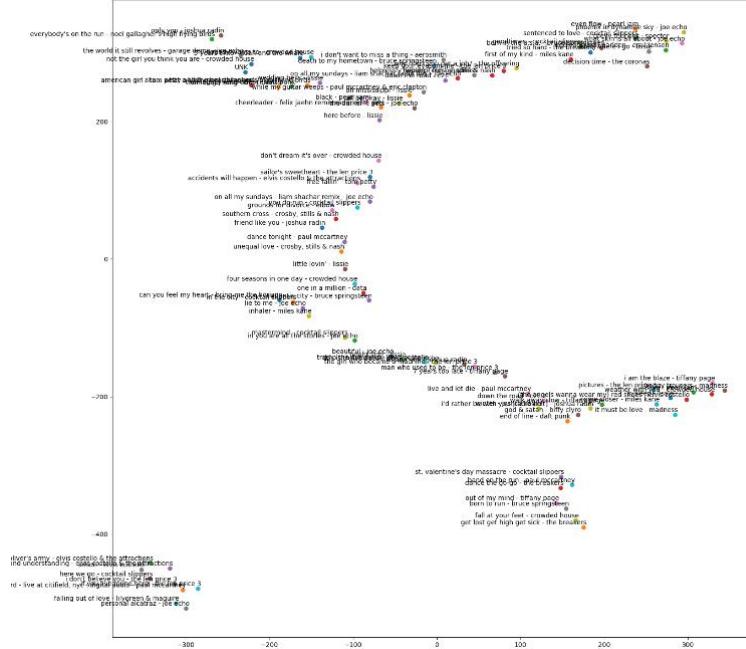


Figure 8: T-SNE Plot of Song Embeddings

After creating and saving the embedding vectors, the next step was to compare them amongst each other using cosine similarity. Cosine similarity is a commonly used metric in natural language processing that measures the similarity between two vectors. The cosine similarity was effective in finding similar vectors because it takes into account the angle between the vectors rather than just the magnitude. This means that even if two vectors have different magnitudes, they can still be considered similar if they have a small angle between them [35]. The cosine similarity formula is shown below in Figure 9. The cosine similarity scores were then used to identify the most similar embeddings and song titles. A plot was created to show the cosine similarity outputs, which helped to visualize the similarities between the embeddings.

$$\text{similarity}(A, B) = \frac{A \cdot B}{\|A\| \times \|B\|} = \frac{\sum_{i=1}^n A_i \times B_i}{\sqrt{\sum_{i=1}^n A_i^2} \times \sqrt{\sum_{i=1}^n B_i^2}}$$

Figure 9: Cosine Similarity Formula

The last step of the application is to print the ten most similar songs in relation to a given input. The target song and recommended songs, along with their location, are plotted on a T-SNE plot. T-SNE is a dimensionality reduction algorithm that can visualize high-dimensional data in two or three dimensions. The T-SNE plot helped to visually show the relationships between the

songs and their embeddings [36]. An example of a generated T-SNE plot is shown below in Figure 10.

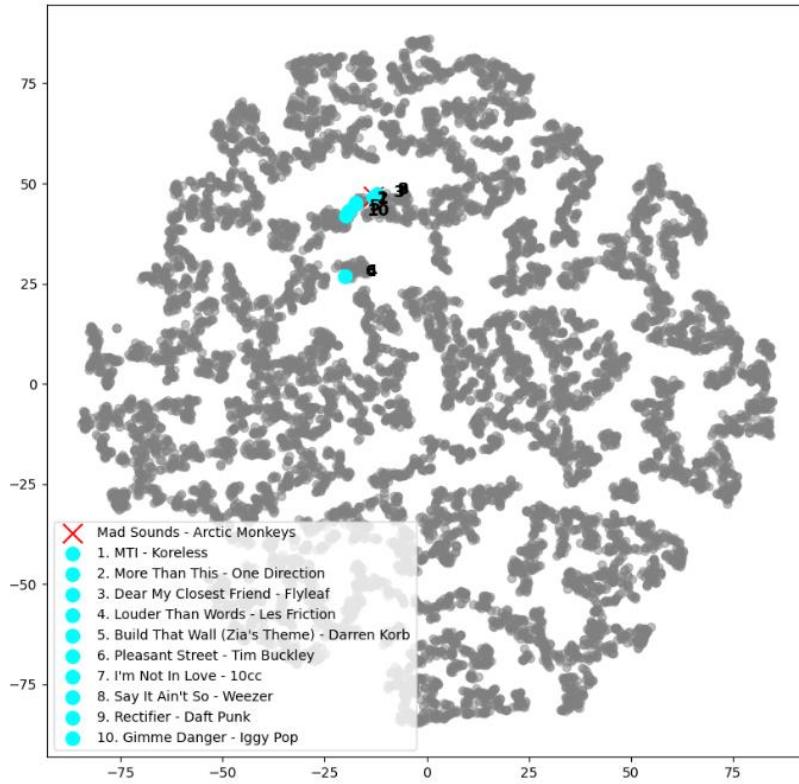


Figure 10: Song Curation Cosine Similarity T-SNE Plot

Overall, the song curation application is an effective example for showing how machine learning and natural language processing can be used to create personalized recommendations for users. This application takes the complex word embedding vectors generated by word2vec and shows their utility in an easily understandable format like song recommendations.

Creating application simulation

The sidebar of the song curation web module serves as the search bar for possible song recommendations. There is a drop-down menu where a user can select a song title that they would like recommendations for. Following this selection, there is a “Generate Recommendations” button to generate recommendations for the given song title. The main content of the web platform displays a table of the ten most similar songs to the input song, as well as each song’s cosine similarity. Accompanying this table is a corresponding T-SNE plot, shown in figure __, that displays the input song title as a red “x” and the recommended songs as points plotted in the color cyan.

3.6 Fake News Detection Application Module

This section describes the development and implementation of the Fake News Detection Application module on the VisNLP 2.0 platform.

Generating real-data outputs

The Fake News Detection application was developed from two GitHub repositories. The first repository contained code relating to using doc2vec technology for binary text classification, while the second repository was a fake news dataset [37] [38]. The code that was used in this project was pulled from these repositories, and it handled the Doc2vec model and the Support Vector Machine (SVM) model. These repositories also included the ground truth training datasets, as well as the testing datasets of news articles. Both of these datasets were obtained from the website GossipCop, which is a website that fact-checks if celebrity news articles are truthful or not.

As mentioned before, the first step in the process of predicting whether a news article is fake or not is to pass the training datasets into the model, allowing the Doc2vec model to extract word features of the articles. These features include different aspects of the article, such as the tone of the article, word usage, and the location of various words. After extracting the feature vectors, the classification model was trained. For this project, the classification model that was used was an SVM model that utilized a linear kernel. The reason for selecting a linear kernel for this project is because there are only two distinct classes, which necessitates binary classification which linear SVM is most suitable for. Linear SVM works by finding the best hyperplane that can separate the data points into different classes. The hyperplane is a line that maximizes the margin between the closest points of those different classes. The margin is the distance between the hyperplane and the closest data points of each class. The end goal of Linear SVM is to find the hyperplane that maximizes the margin while minimizing the classification error.

Once the feature vectors were created and the SVM model was trained, the model was ready to be tested. However, additional Python code was needed to ensure that the model's output was meaningful and easy to understand. The first aspect that was added to the code was a method that outputted the model's prediction for the news article in words. To achieve this, a few additional Python libraries, such as NumPy and Pandas, were imported to allow for the results to be stored in a dataframe that assists in printing the prediction one by one. This output can be seen in Figure 11 below.

Title: 42 last-minute Mothers Day gifts that shell actually love
Prediction: Real

Title: Did Miley Cyrus get mad at Liam Hemsworth for refusing to wear his promise ring?
Prediction: Fake

Title: Nicole Kidman and Prince Harry named sexiest redheads
Prediction: Fake

Title: Jim Carrey lawsuit: Unearthed note from ex-girlfriend makes shocking claims
Prediction: Real

Title: Service Dogs Spent the Best Day Ever at Disneyland
Prediction: Real

Figure 11: Example output of model prediction

The next addition to the code was the output of the classification plot. This was a simple feature to add since the SVM model had already clustered the articles and all that remained was to create a visual representation of the model's findings. In order to achieve this, the matplotlib python library was imported which facilitated the creation of a scatter plot. To ensure that the graph was as informative as possible for the end user, additional code was integrated to display the separating hyperplane of the SVM classification. Figure 12 below shows what the classification plot for the articles looks like.

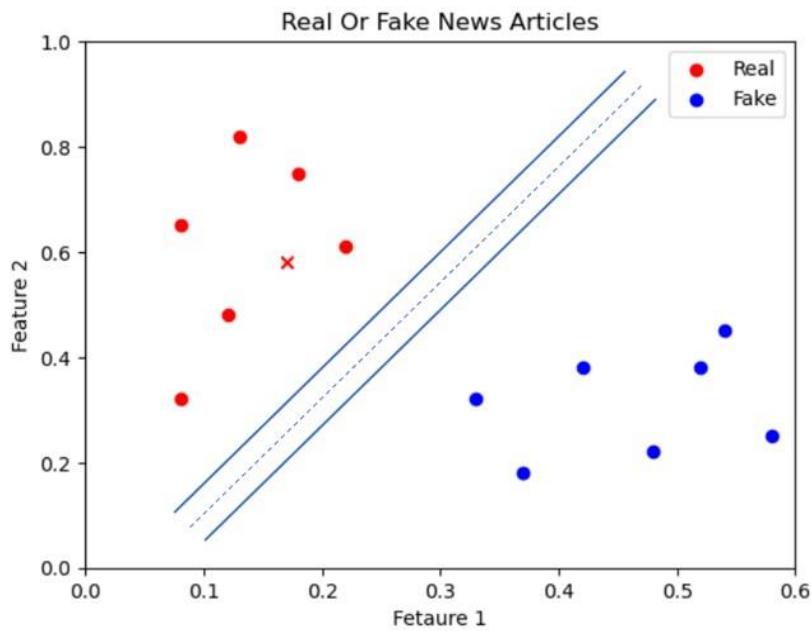


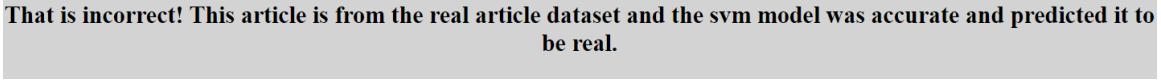
Figure 12: Classification plot outputted by model

Creating the application module main content

This section of the Fake News Detection module requires for there to be an article title, as well as the classification plot that contains the SVM classification clusters of the news article titles. Each plot is unique in order to highlight where the current article falls on the classification plot. Another note about the plots that are shown is when the webpage is asking the user to make their prediction of whether or not the article is real or fake, all points on the graph are gray in order to not give any hints as to which cluster is which.

Description footer

The footer section for each page within the Fake News Detection application simply requires an explanation of if the user's guess was correct or incorrect, what the model's prediction was and then finally whether the article is actually real or fake. An example of the footer content after a user has made an incorrect guess can be seen in Figure 13 below.



That is incorrect! This article is from the real article dataset and the svm model was accurate and predicted it to be real.

Figure 13: Example of footer content

User sidebar

The user sidebar never changes in this module of the website. For each step in the Fake News module, the sidebar contains the user controls for navigating through the module. This is where the buttons are located that the user presses in order to make their guess about if the article is real or fake.

4 VisNLP 2.0 Educational Support Platform

This section presents the results of the development and implementation of the VisNLP 2.0 web-based educational support platform for teaching neural network-based natural language processing. This chapter will provide a detailed overview of the finished product, including its individual modules, and explain how learners navigate through the platform. The goal is to demonstrate the final version of the VisNLP 2.0 web-platform and how it supports the teaching and learning process of neural network-based NLP.

4.1 VisNLP 2.0 Web Platform

The development of VisNLP 2.0 as a web platform is fairly straightforward. The website is coded primarily in JavaScript with HTML files for each module and one master CSS stylesheet.

The backend of the site is set up with Node.js with inter-page navigation made simple with a basic navbar implementation. There is minimal use of middleware or additional libraries, with the primary exception being the d3 library for visualization purposes. For data management, we used MongoDB to store our data for easy access through the backend.

As shown below in Figure 14, each module opens to a simple overview page explaining the module itself and giving the user any additional information they may need before accessing the primary content. This overview page also includes a button to the previous and next modules of the platform and a diagram of the website highlighting which section you are currently viewing.

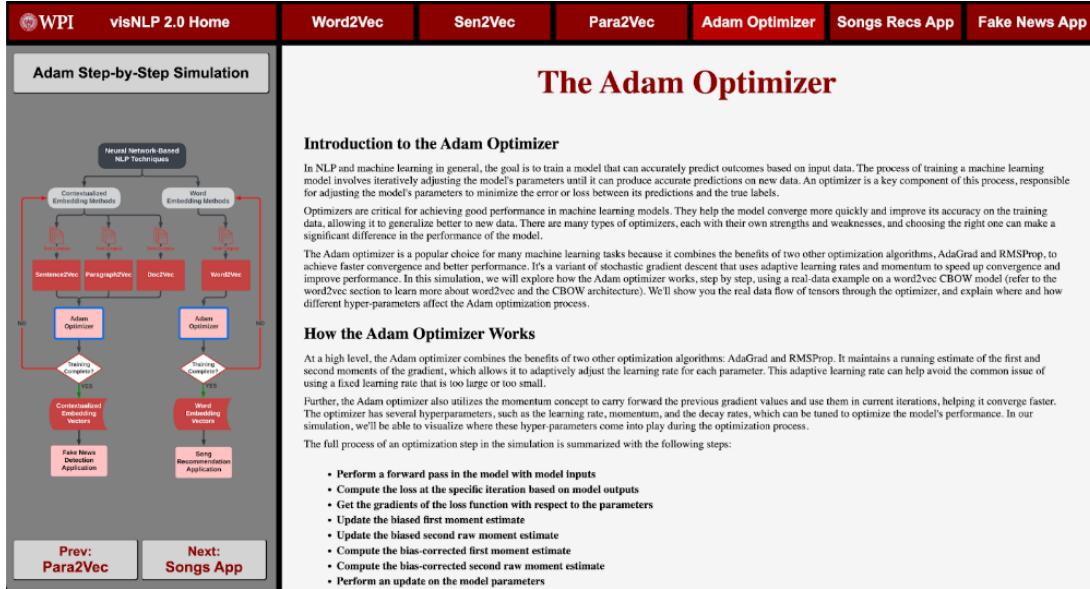


Figure 14: Adam Optimizer - Overview Screen Capture

On the main pages of each module, the screen is divided into 4 sections. This is shown below in Figure 15. The first is the previously mentioned navbar located at the top of the screen. The second is the sidebar, displaying the current simulation parameters or, for the applications, holding the interface for the user to interact with the application. Third, we have the footer, containing additional information that pertains to the user's current view of the site. On many pages, the footer is dynamic, displaying information corresponding to where the user's cursor falls. Finally, we have the main canvas view of the site, containing the primary content of each module. In each step-by-step simulation, this view also contains arrow buttons to move between pages within the modules, as well as a labeled progress bar above the footer to display their position within each epoch of the simulation. This page setup is intended to be as straightforward as possible, consistent across various modules, and allow the user to know where to look for information regardless of which module they find themselves in.

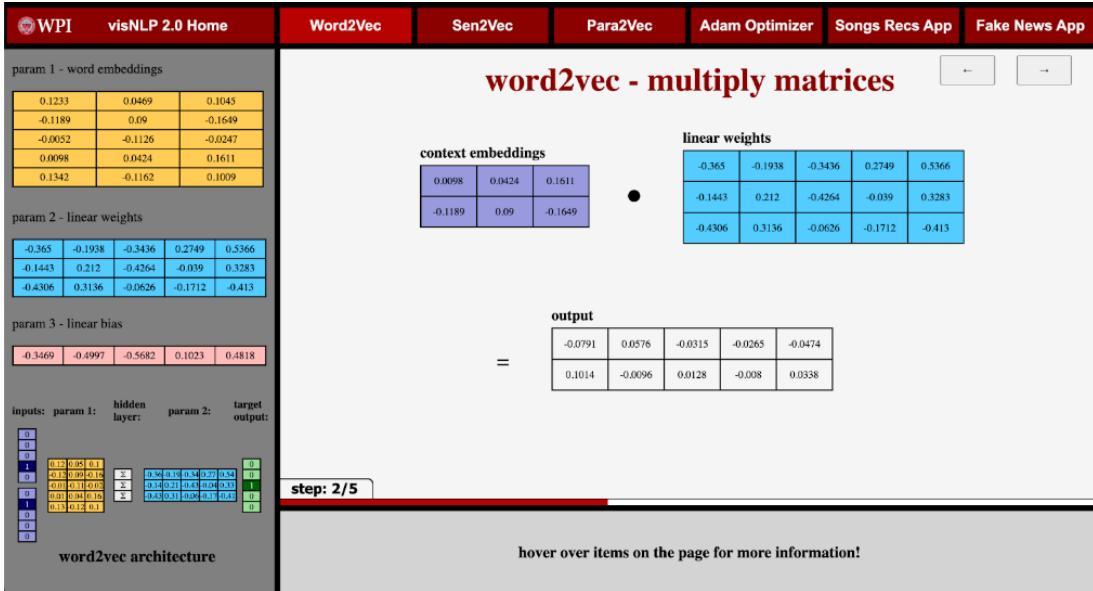


Figure 15: Word2vec – Multiply Matrices Screen Capture

4.2 VisNLP 2.0 Homepage

The VisNLP 2.0 web-platform homepage is intended to preface the following web-platform modules by introducing the platform objectives and providing background information. Here the user gets introduced to our platform objectives where our aim is threefold: (1) To present some popular neural network-based NLP methods in a step-by-step linear format that is easy to comprehend. (2) To address the 'black box' problem present in neural network-based NLP learning resources through continuous real-data examples. (3) To enable users to interpret model outputs through interactive visual demos that apply neural network-based NLP model outputs. Further, the user gets introduced to the web-platform framework through a diagram describing the relationships between the various modules of our web-platform as described in Section 4.1. Additionally, the learner is provided with a brief project background outlining the significance of NLP. Finally, on the user sidebar the learner can access the platform evaluation survey. A screen capture of the VisNLP 2.0 homepage is shown below in Figure 16:

Figure 16: VisNLP 2.0 Web Platform Home Page

4.3 Word2vec Module

The word2vec module for the VisNLP 2.0 web platform begins with an overview of the material, followed by 10 epochs' worth of real step-by-step output data. This data is pulled in the backend code from MongoDB and is used to produce dynamic visuals based on the data being displayed. The overview page is quite simple, primarily consisting of text with the platform's architecture diagram and two buttons to select which word2vec model the user would like to view. The word2vec module overview page is shown in Figure 17 below:

Figure 17: Word2vec - Overview

Following the overview are two simple and straightforward introductory pages – Figure 18 and Figure 19. The first covers the text preparation process in word2vec, showing the user how the program takes a simple input text, removes non-alphanumeric characters, changing any uppercase character to lowercase, and splitting into an array. It also shows how this array is broken up into a key/value map, being the model’s dictionary, with each word having its corresponding key number. The following page displays this dictionary as well as each word’s one-hot encoded vector. This page covers the basics of what a one-hot encoded means and how to interpret it, as well as what the “UNK” token means and why it is included in calculations.

input	I love DATA science%
remove non-alphanumeric characters	I love DATA science
make lowercase	i love data science
split words into array	["i", "love", "data", "science"]

dictionary:
`{ "UNK": 0, "i": 1, "love": 2, "data": 3, "science": 4 }`

hover over items on the page for more information!

Figure 18: Word2vec – Text Preparation

	0	1	2	3	4
"UNK":	1	0	0	0	0
"i":	0	1	0	0	0
"love":	0	0	1	0	0
"data":	0	0	0	1	0
"science":	0	0	0	0	1

this cell being "1" indicates that this one-hot encoded vector is the encoding for word #3 in the corpus dictionary.

Figure 19: Word2vec – One-Hot Encoding

The next page is the first in the primary loop each word2vec epoch consists of – Figure 20. To begin, we will cover the common traits across each epoch. At the bottom of the screen, just above the footer, is a progress bar showing where the user is within the epoch at all times. In the sidebar at all times, we show all 3 parameters, explaining where the numbers are initialized from when the user hovers over the table. We also display the architecture of the model showing how the different parameters relate to each other at a macro level. When highlighting cells in either of these it highlights other cells onscreen that correspond to give the user more contextual information regarding how these values relate to each other. This first page of the cycle covers the training batch generated for this epoch. It shows each one-hot encoded vector, highlighting the one(s) selected as input for the model as purple and the one(s) selected as the target for the model as green. The highlighted vectors also display their current corresponding embeddings from parameter 1, showing the numbers we will be working with going forward. The visuals on display, as with the ones that follow, are color-coded when appropriate, to show which numbers come from where.

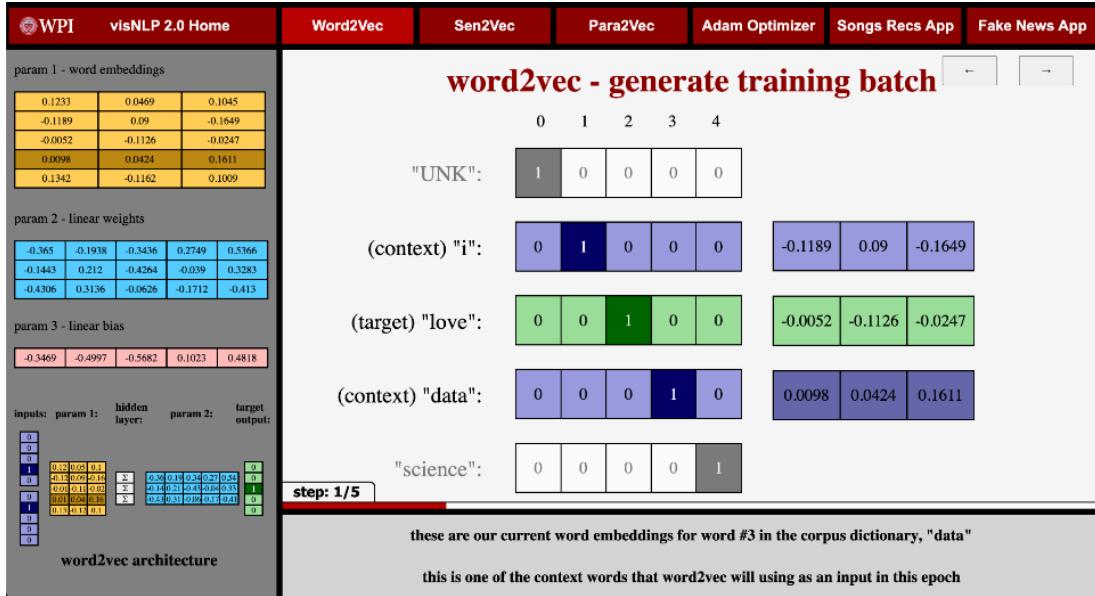


Figure 20: Word2vec - Generate Training Batch

The next three pages of the cycle are all very simple, to keep things so that a novice user can easily follow along with the math as it occurs. The first shows the process of matrix multiplication between the input embeddings and the weights in parameter 2 – Figure 21. The next page takes the output from this and adds the linear bias from parameter 3 – Figure 22. The third page varies based on the model being shown – Figure 23. If the simulation uses the CBOW model, it shows the log SoftMax function. If the simulation uses the skip-gram model, it shows the sigmoid function.

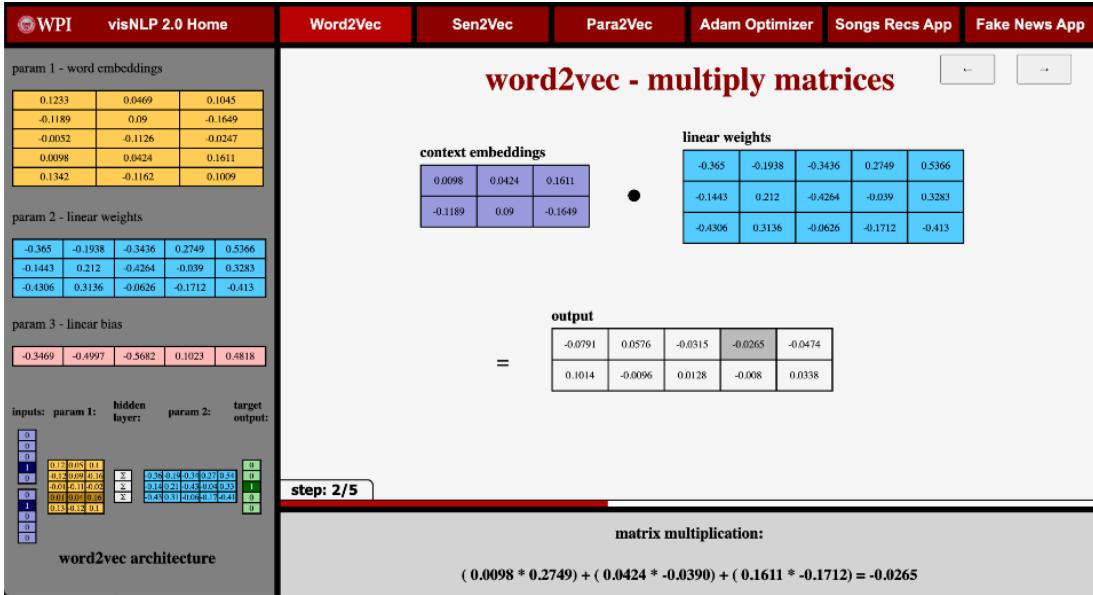


Figure 21: Word2vec – Multiply Matrices

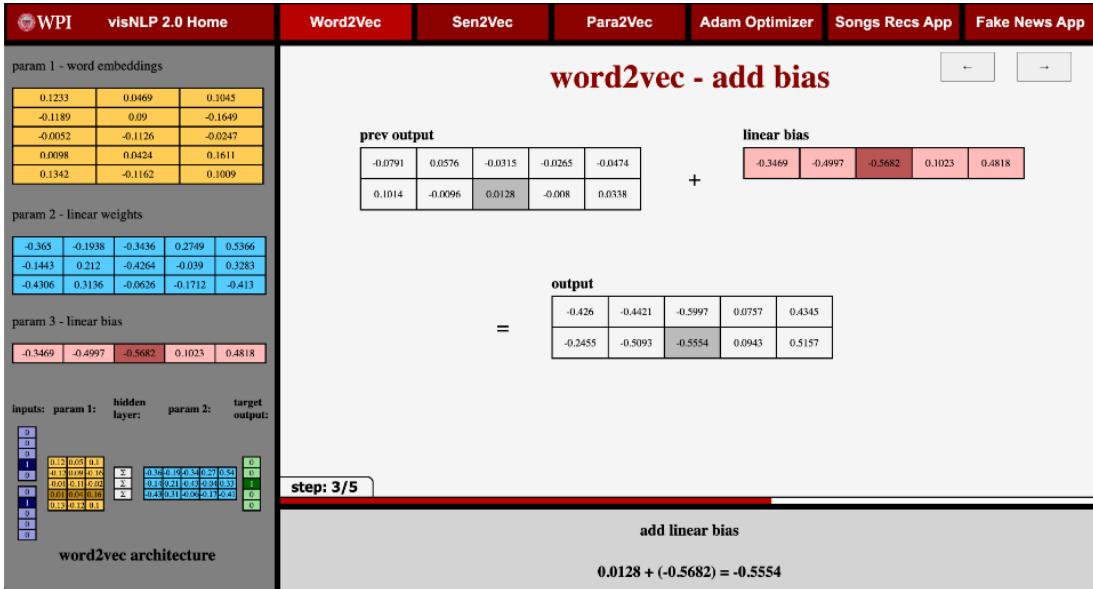


Figure 22: Word2vec – Add Bias

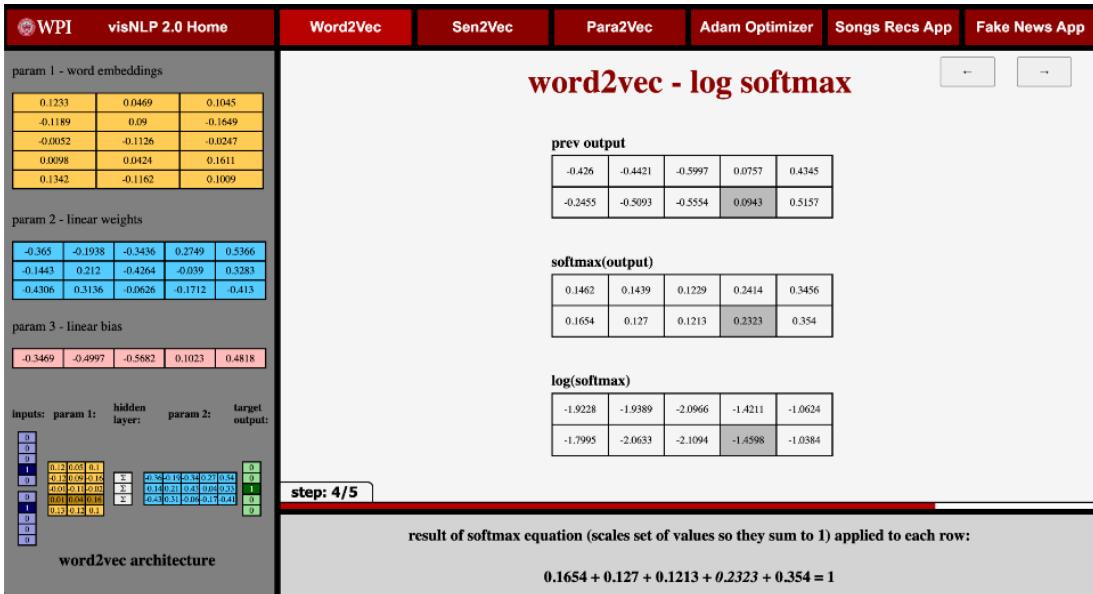


Figure 23: Word2vec – Log SoftMax

Next is the final screen of the epoch loop – Figure 24. This shows the final outputs from the previous screen and links to the Adam Optimizer module for further information. It removes the parameters from the sidebar and puts them on main display here, showing the direct comparison between the current and next epoch’s parameters. This shows how the parameters are tuned in each epoch and functions as a transition to the next epoch. From here, we return to the “word2vec – generate training batch” slide and loop for each epoch, for a total of 10 epochs.

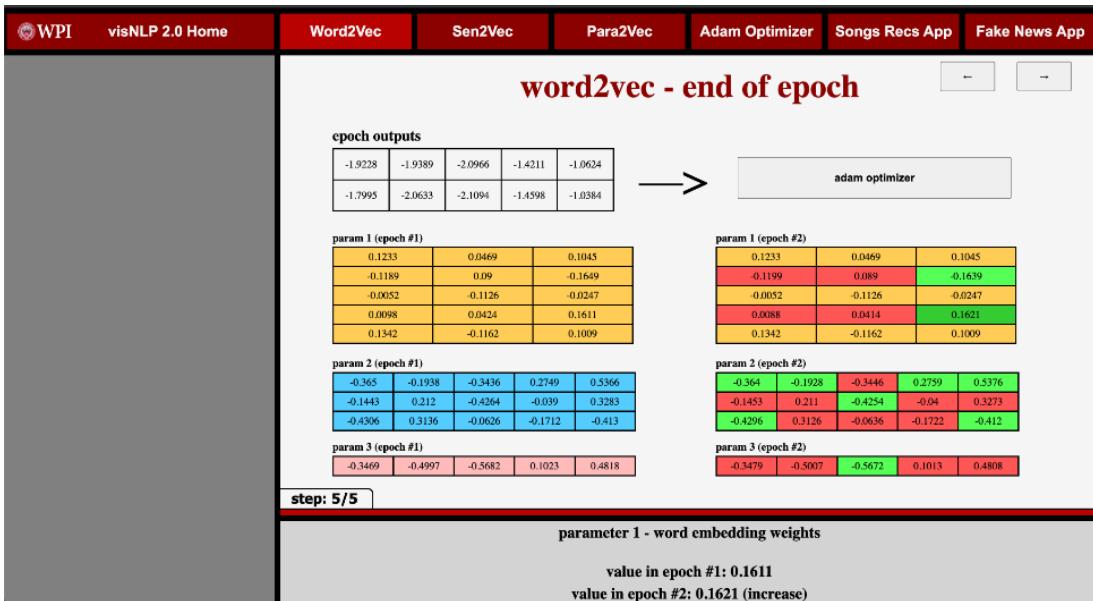


Figure 24: Word2vec – End of Epoch

4.4 Paragraph2vec and Sentence2vec Module

The next two modules on the VisNLP 2.0 web-platform cover the paragraph2vec (para2vec) and sentence2vec (sen2vec) algorithms. The finished sen2vec/para2vec module for the VisNLP 2.0 web-platform consists of two components:(1) An overview page intended to preface and provide background for the step-by-step sen2vec/para2vec simulation. (2) A step-by-step simulation showing a continuous real-data example of sentence and paragraph vectors being generated by the sen2vec/para2vec algorithm over an epoch. When learning on the VisNLP 2.0 sen2vec/para2vec module, a learner would first be taken to the sen2vec/para2vec overview page.

Sen2vec/para2vec overview

To navigate the overview page for the sen2vec/para2vec algorithm on the VisNLP 2.0 web-platform, a user would start by reading the introductory paragraph, which provides an overview of the algorithm and its applications. The user would then proceed to the next section, which explains how the algorithm works, including its advantages over traditional NLP techniques.

After understanding the basics of the algorithm, the user can proceed to the section on the process of performing sen2vec/para2vec, which outlines the steps involved in converting input data into a vector representation. This section describes the different matrices and parameters used in the algorithm and how they are updated using the Adam optimizer.

Finally, the user can explore the step-by-step simulation of the algorithm, which provides a more detailed explanation of the individual components of the algorithm. Below in Figure 25 is a screen capture of the sentence2vec overview page:

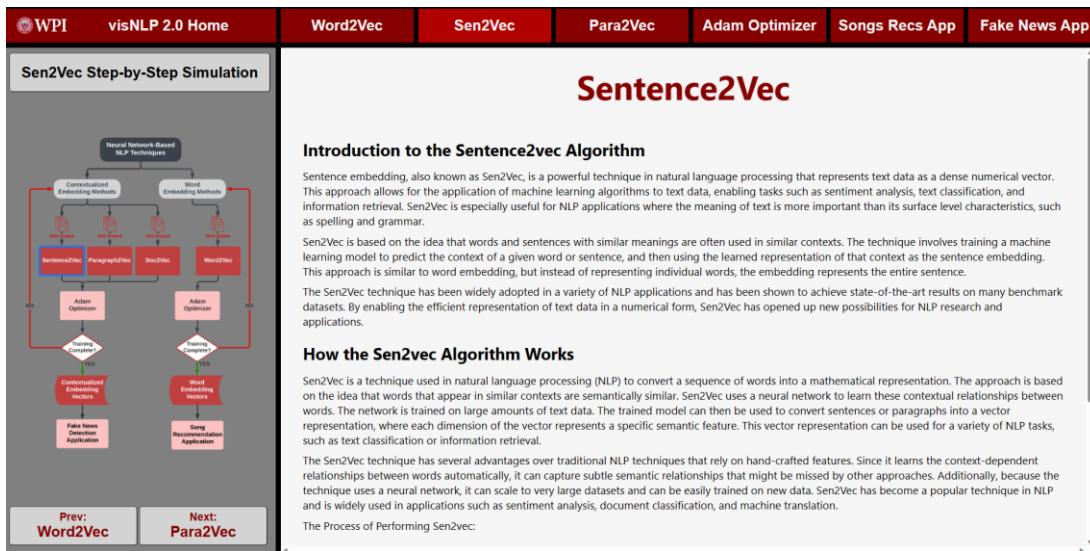


Figure 25: Sen2vec Overview Page

Throughout the overview page, the user can also find helpful visual aids and buttons, such as diagrams and buttons to the previous or next algorithm overview, to help them better understand the algorithm and its workings.

Sen2vec/Para2vec simulation

The step-by-step simulation of the sen2vec algorithm is a demonstration of how the algorithm works in practice. This simulation takes you through each step of the algorithm, showing you how the input data is processed and how the neural network is trained to generate sentence embeddings. By following along with the simulation, you can gain a deeper understanding of how sen2vec works and how it can be applied to a variety of natural language processing tasks.

The first step of the sen2vec step-by-step simulation for the VisNLP 2.0 education support platform is shown in Figure 26 below:

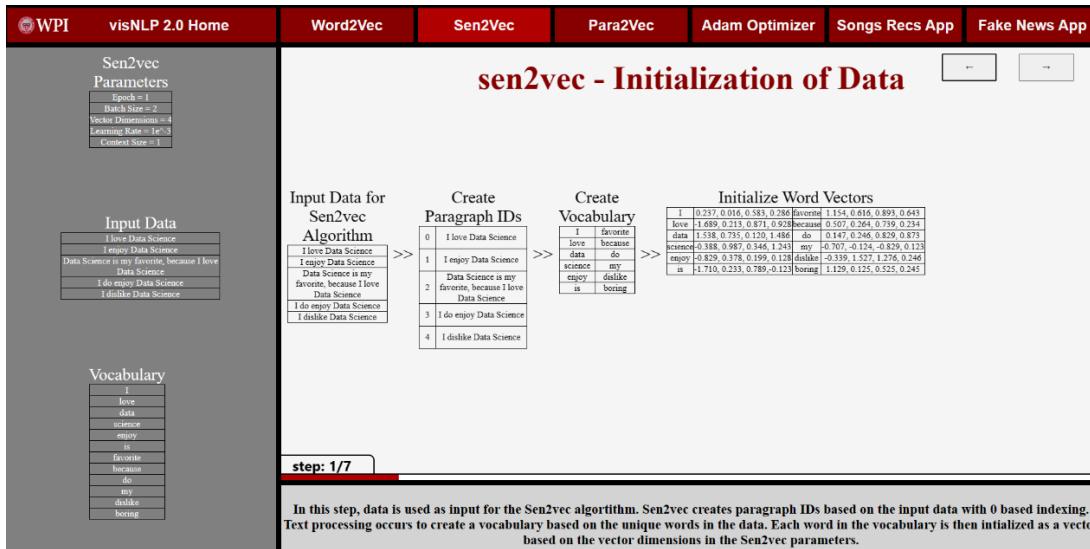


Figure 26: Sen2vec Simulation Step 1

Data is used as input for the sen2vec algorithm. Sen2vec creates paragraph IDs based on the input data with zero-based indexing. Text processing occurs to create a vocabulary based on the unique words in the data. Each word in the vocabulary is then initialized as a vector based on the vector dimensions in the sen2vec parameters.

In the second step, the paragraph ID, context word vectors, and center word vectors are concatenated to form an input matrix to be used in sen2vec training. Context word size is determined in the parameters, in this example context size is one, therefore it takes one word from the left and right of the center word as context. The size of the input matrix is based on the amount of pre-generated batches, in this example we have two.

In the third step, sen2vec initializes the word and paragraph weight matrices along with the bias matrix. The size of the word weight matrix is based on the vocabulary size of the data, in this case, the vocabulary is 12. The size of the paragraph weight matrix is based on the number of paragraphs in the data, which is five. The size of the bias matrix is based on the input matrix and is initialized as a zero matrix. All these matrices have the same dimensions as the vector dimensions in the parameters.

In the fourth step, input vectors are created by the concatenation of paragraph IDs and context words. The values for the paragraphs are taken by the paragraph weight matrix associated with each paragraph. Since batch size is two in the parameters, a batch of two input vectors is randomly selected and multiplied by the initial word weight matrix. This results in the predicted value for the center words of the given training vectors.

In the fifth step, the bias matrix is added to the predicted center word matrix to introduce another parameter that improves the accuracy of the model. The hyperbolic tangent function is then applied to the result matrix to introduce nonlinearity to the model. This allows the model to capture more complex relationships between words and paragraphs.

In the sixth step, the hyperbolic tangent matrix is transformed by the SoftMax function to get the probability distribution of the vocabulary. The result matrix then uses the true center word matrix, which was gathered from the input matrix, to calculate the negative log likelihood loss of the predicted center word matrix. The negative log likelihood loss is a measure of how well the model predicts the true center word.

In the last step, negative log-likelihood loss values are sent to the Adam optimizer to update the weights. The weights will be updated for the word and paragraph matrices along with the bias matrix. The updated matrices are used in the next batch of training vectors. Once all training vectors have been trained, that is the end of one epoch. Once the number of epochs set in the parameter is met, the paragraph weight matrix is returned and is now ready to be used in other applications. Red indicates a decrease in value and green indicates an increase.

Summary of Contributions

In conclusion, the step-by-step simulation of the sen2vec algorithm provides a clear understanding of how this algorithm works. Sen2vec is a powerful algorithm that is widely used in NLP and machine learning. By converting text data into a numerical format, it allows for the processing of substantial amounts of data with ease. The simulation highlights the importance of each step in the process, from data input to the optimization of weights. Understanding the inner workings of sen2vec can help researchers and developers create better models that can accurately capture complex relationships between words and paragraphs. Overall, sen2vec is a valuable tool in the field of NLP, and this simulation provides a comprehensive overview of how it works. The para2vec module also has a step-by-step simulation, however, it is like the sen2vec step-by-step approach. The main difference is that para2vec extends the approach to operate on paragraphs, and hence the input data will be paragraphs, but the step-by-step simulation will be carried out in a similar way. On top of para2vec, there is also doc2vec, which extends the approach to operating on documents to capture the semantic meaning of an entire document, allowing for the processing of large volumes of text data at once. With doc2vec, entire documents can be represented as a vector, which can then be used for various NLP tasks such as document classification and sentiment analysis. Due to its similarity with sen2vec and para2vec, the visual module has been omitted from the 2.0 web-support platform. Moving on to the Adam optimizer module in VisNLP 2.0 web-platform, it consists of two components that provide an overview and a step-by-step simulation, respectively.

4.5 Adam Optimizer Module

The finished Adam optimizer module for the VisNLP 2.0 web-platform consists of two components: (1) An overview page intended to preface and provide background for the step-by-step Adam optimizer simulation, and (2) A step-by-step simulation showing a continuous real-data example of model parameters being optimized by the Adam optimization method over 40 iterations. When learning on the VisNLP 2.0 Adam module a learner would first be taken to the Adam optimizer overview page.

Adam optimizer overview

In the Adam optimizer overview page, a learner will get a preface to the Adam optimizer step-by-step simulation as well as background to the Adam optimizer. A screen capture from the Adam optimizer overview is shown below in Figure 27.

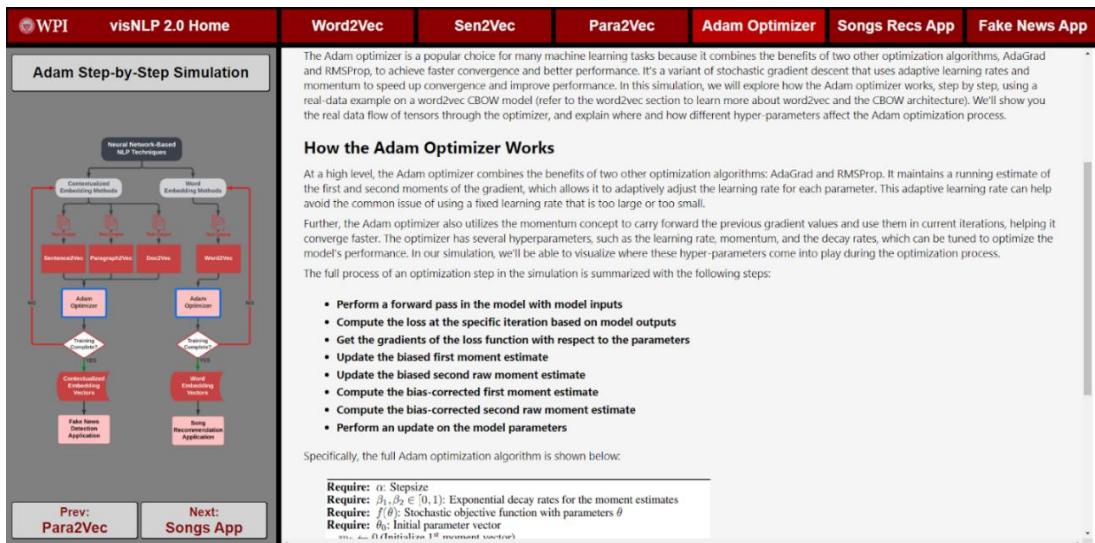


Figure 27: Adam Optimizer Overview Page

The learning process begins with a written introduction to the Adam optimizer. This guide covers the basics of what an optimizer is and how it relates to other modules in the platform. This module's relation to the rest of VisNLP 2.0 can also be visualized on the page's sidebar through a highlighted diagram. The user can then learn about the significance of the Adam optimizer in NLP tasks, including its popularity and pros and cons compared to other optimization methods. Additionally, a high-level overview of how the Adam optimizer works is provided. To aid in the user's understanding of the algorithm, a summary of the Adam optimization algorithm is also given. Finally, we preface what happens in the Adam optimizer step-by-step simulation, where from the user can enter and begin the simulation.

Adam optimizer simulation

In the Adam optimizer simulation, a learner can step through the step-by-step simulation showing a continuous real-data example of an NLP model's parameters being optimized by the Adam optimization method over 40 iterations. A screen capture from the Adam optimizer simulation is shown below in Figure 28.

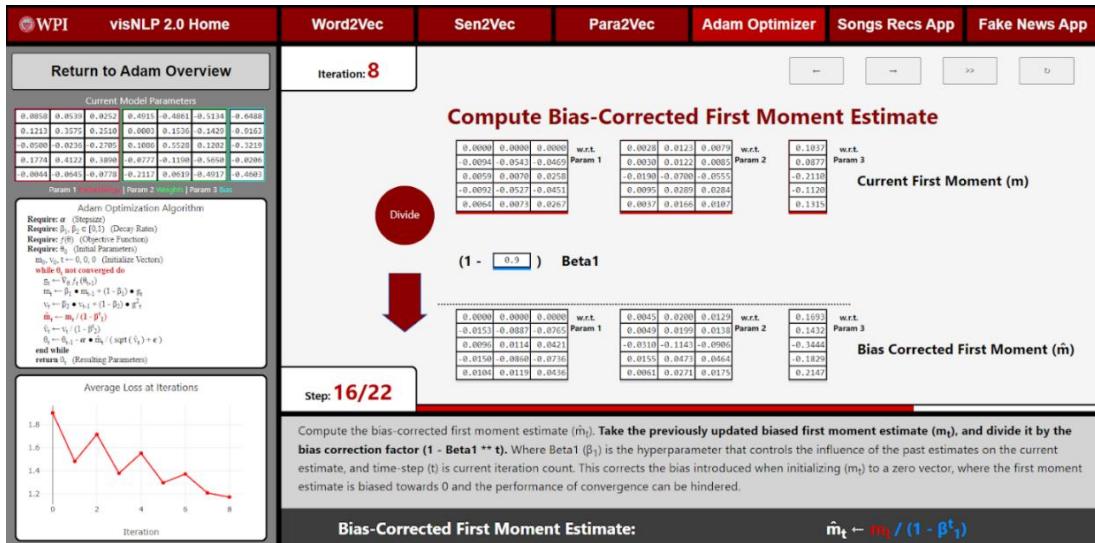


Figure 28: Adam Optimizer Simulation - Step 16

The learning process continues once a learner enters the initial point of the Adam simulation. Here, the sub-steps of the simulation are designed to directly correlate with the overview covered earlier. This helps learners make connections between what is happening in the simulation and how it fits into the overarching Adam optimization method.

The purpose of the Adam optimizers is to tune and adjust the parameters of a stochastic objective function. Due to this an example objective function to perform the optimization on is necessary. In our simulation, we start with randomly generated model parameters for an example CBOW word2vec model for the corpus "I love data science" with a vocabulary size of 5 and an embedding dimension of 3. These three example parameters include embeddings, weights, and biases. The simulation allows the user to step through 22 sub-steps of the Adam optimization method continuously for 40 iterations.

The learning process in the simulation begins by passing the adjusted parameters back to the objective function (unless it is the first iteration). The learner then performs the forward pass in the example CBOW word2vec model and computes the negative log-likelihood loss of the model. An average loss measure is also computed to gauge the model's performance at a given iteration. The gradients with respect to the stochastic objective are obtained, followed by updating the biased first moment vector and the biased second raw moment vector. The learner then

computes the bias-corrected first moment estimate and bias-corrected second raw moment estimate. Finally, the parameters are updated with Adam and the user advances to the next iteration where the process repeats on the updated parameters. The workflow of the step-by-step process is summarized below in Table 4:

1	Pass the adjusted parameters back to the objective function (unless the first iteration).
2	Perform the forward pass in the example CBOW word2vec model.
3	Compute the negative log likelihood loss of the model.
4	Compute an average loss measure of the model for the purpose of gauging model performance at a given iteration.
5	Get gradients with respect to stochastic objective.
6	Update biased first moment vector.
7	Update biased second raw moment vector.
8	Compute bias-corrected first moment estimate.
9	Compute bias-corrected second raw moment estimate.
10	Update parameters with Adam.

Table 4: Adam Step-by-step Learning Process Summary

In order to achieve a step-by-step process that adequately tackles the black-box problem, this workflow is further broken down in 22 steps that the user steps through at each iteration. The steps aim to illustrate the Adam optimization process in a detailed and atomic manner. This involves breaking down necessary operations within the algorithm as much as possible for the learner, ideally displaying only one operation per step. To navigate the steps the user can either step forward one sub-step or backwards one sub-step. Additionally, the learner is able to fast-forward an iteration allowing the learner to visualize how the optimization process works over time, especially with the help of the simulation dashboards which are covered later in this section. Lastly, the user can reset the simulation back to its original state.

At each step, various components on the platform aid in helping the user understand optimization in neural network-based NLP. As seen below in Figure 29, the view for the Adam simulation is broken into three sections: a main content section, a description footer, and a sidebar dashboard. The contents of these three components aim to give comprehensive insight at what is happening at each step of the Adam optimization simulation.

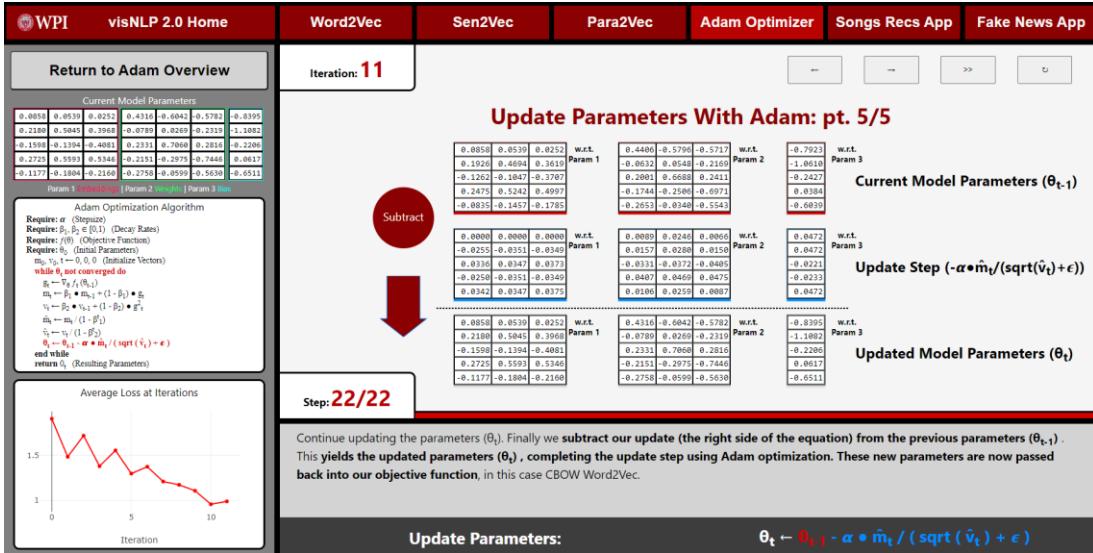


Figure 29: Adam Optimizer Simulation - Step 22

At the forefront of each step view, is a visual of the sub-step being applied on the real data vectors. The learner can see the 2d visualization of the tensors before the step, what is applied to those tensors, and the resulting tensors. Additionally, the learner can see the labels of each of these tensor groups allowing them to understand the variable or other components of the process that they represent. Further, the title provided above the visuals gives the user a brief insight into what step is being executed. The user is also able to get further insight of what is occurring in that given step in the footer. The footer provides the user an explanation of the given step, which details the operation being done, the components of the algorithm that the operations are being performed on, and a brief technical inference that ties back to the theory behind the Adam optimization method. Also included in the footer is an equation section intended to bring the equation most relevant to that step to the forefront of the learner's attention. These equations directly correlate with the sub-step visuals, in such a way that the left side of the operation is highlighted in red, and the right side of the operation is highlighted in blue, both in the visual tensors and the footer equations.

To add further insight to the Adam optimization step-by-step simulation for the learner, the sidebar houses three dashboards that track the optimization process. The first dashboard in the sidebar is the model parameters dashboard. This dashboard allows the learner to see each of the three parameters that are being optimized in the simulation throughout the step-by-step process. The next dashboard is the Adam algorithm dashboard. This dashboard provides the learner with a visual representation of the specific section(s) of the Adam optimization algorithm that is being executed at the current step of the simulation. This is achieved through highlighting the relevant section(s) of interest, allowing for a clear understanding of the algorithm's flow and progression. The final dashboard in the sidebar is the average model loss measure over iterations dashboard. By displaying the average model loss over the course of the simulation, learners can visualize the performance of the model as it undergoes optimization. This allows learners to gain a macro-level understanding of how the Adam optimizer affects the performance of the model over time, which can aid in their comprehension of the overall optimization process.

Summary of contributions

In summary, the Adam optimizer module on the VisNLP 2.0 web-platform provides a step-by-step simulation of the Adam optimization algorithm, which is widely used in neural network-based natural language processing. This module offers a clear and concise explanation of the Adam optimizer, its significance, and how it works. Through a series of step-by-step real-data visuals paired with pertinent equations, the learner can easily follow the simulation process and understand how the optimizer tunes and adjusts model parameters. Additionally, the module offers a macro-level understanding of model performance and flow through various dashboards, including a measure of the average model loss over iterations. Overall, the Adam optimizer learning module aids in tackling the black-box problem in neural network-based natural language processing by providing a transparent and accessible way for learners to understand the optimization of parameters that occurs during the training of NN-based NLP models.

4.6 Songs Recommendations Application Module

Song recommendation application overview

The Song Recs App module on the web platform first opens to a comprehensive overview section that provides the user with detailed information about the application. This section aims to help users understand the goal of the application, its purpose, and how it was developed. The main components of the overview section are displayed in Figure 30.

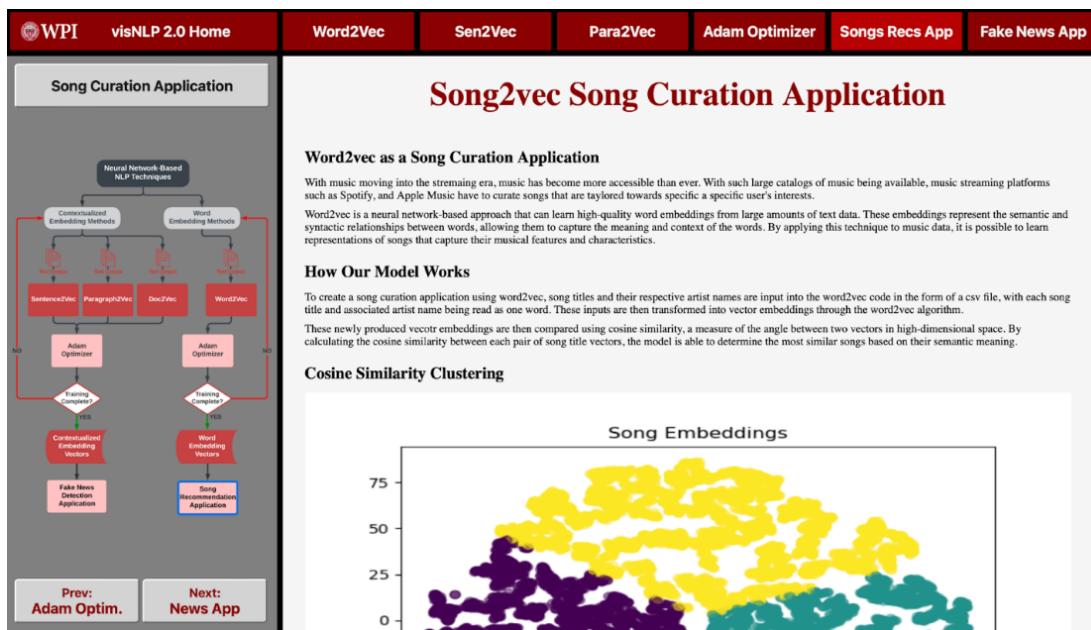


Figure 30: Song Recommendation Overview

Song recommendation simulation

In the song recommendation simulation, the cover page displays a T-SNE plot of the vector embeddings that are used in generating song recommendations. In the sidebar, the user can select a song from a drop-down list that will serve as input for the song recommendation app, as shown in Figure 31.

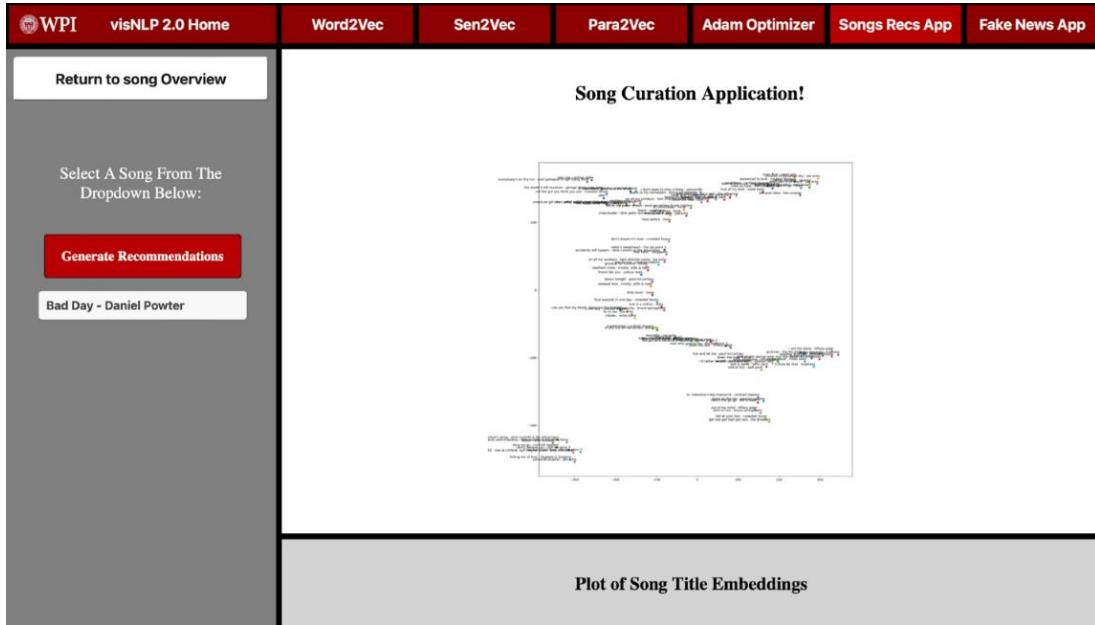


Figure 31: Song Recommendation Application Home

Once the user selects a song, they must press the "generate recommendations" button to generate recommendations. This action prompts the loading of a recommendation table, including the input song with ten listed recommendations and their cosine similarity to the input song. Accompanying the recommendation table is a T-SNE plot that shows the input song as a red "x" and the recommended songs as cyan plotted points. The table and plot are displayed in Figure 32 below.

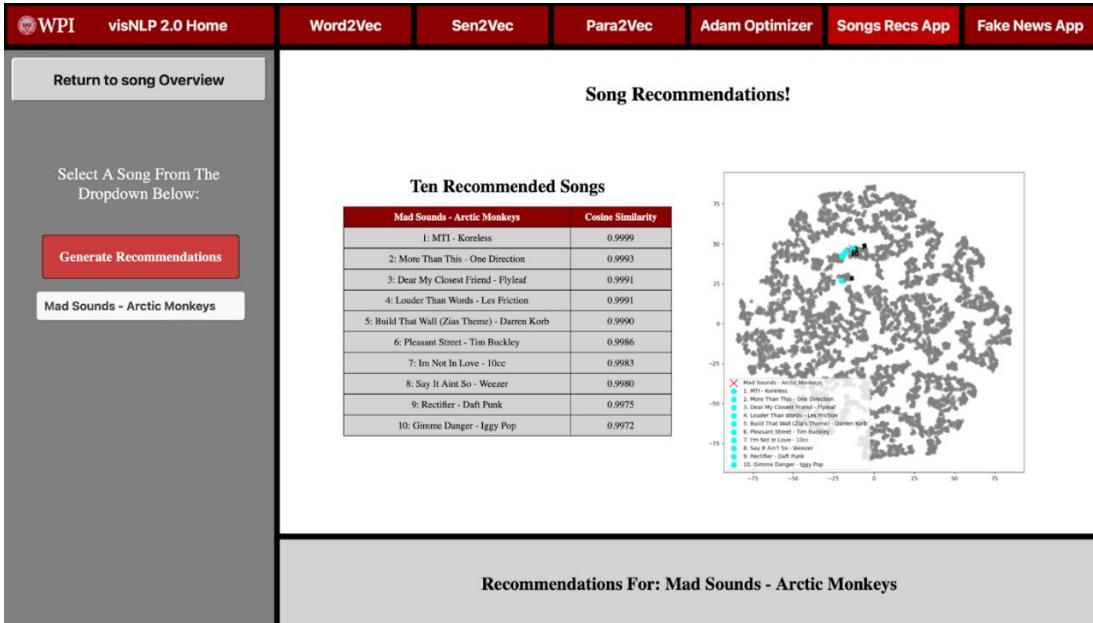


Figure 32: Song Recommendation Application Example

This plot helps users visualize how the recommendations relate to the input song and provides a quick overview of the similarity between the input song and the recommended songs.

Finally, the footer of the website updates with the input song's title, which helps users keep track of the selected song and provides a quick reference point for future recommendations. Overall, this process is a vital component of the Song Recs App's functionality, and it provides users with a personalized and user-friendly experience. The app's recommendation simulation is a fitting example of how advanced technology can be used to generate personalized song recommendations for users.

4.7 Fake News Detection Application Module

The VisNLP 2.0 web-platform's fake news application module is composed of two principal components. The first is an overview page designed to introduce and provide context for the fake news application. The second component provides a visual representation of a classification plot generated after feature vectors have been clustered. To begin learning on the VisNLP 2.0 fake news application module, learners will be directed to the fake news application overview page.

Fake new application overview

In the fake news application overview page, a user will read about the workings of how the fake news detection model works. The overview gives a quick background about para2vec technology and how it has emerged as a useful tool for many real-world applications. A screen capture from the fake news application overview is shown in figure 33 below:



Figure 33: Fake News Application Overview Page

Fake news application simulation

After providing an overview of the NLP application, the subsequent web pages on the platform aim to showcase examples of the application in action. Rather than burdening the user with the intricate details of the process involved in detecting fake news via doc2vec and SVM models, the platform's primary objective is to demonstrate how feature vectors can be clustered using classification through visualization. Each page exhibits a unique article title, for example, "Did Miley Cyrus get mad at Liam Hemsworth for refusing to wear his promise ring," and prompts users to guess whether the article is real or fake through the sidebar. Once the user submits their guess, the page updates to display a classification plot, depicting if the article falls on the real or fake side of the SVM hyperplane. Furthermore, the footer refreshes to inform user whether their guess was correct or incorrect, along with the model's prediction and the article's original source. There are ten articles in total for users to predict whether they are genuine or fake, and the process remains consistent for each article. Once users finish making their predictions, they can restart the simulation should they wish to repeat the process. The photos below in Figure 34 and Figure 35 depict when the page is asking the user for their prediction as well as the webpage once the user has made their prediction.

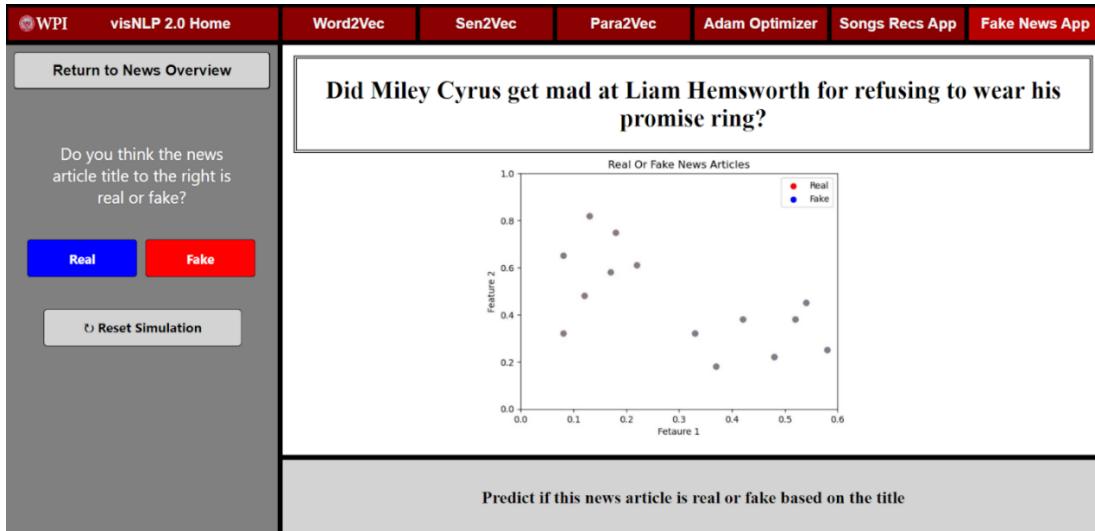


Figure 34: Fake News Application Pre-User Prediction Page

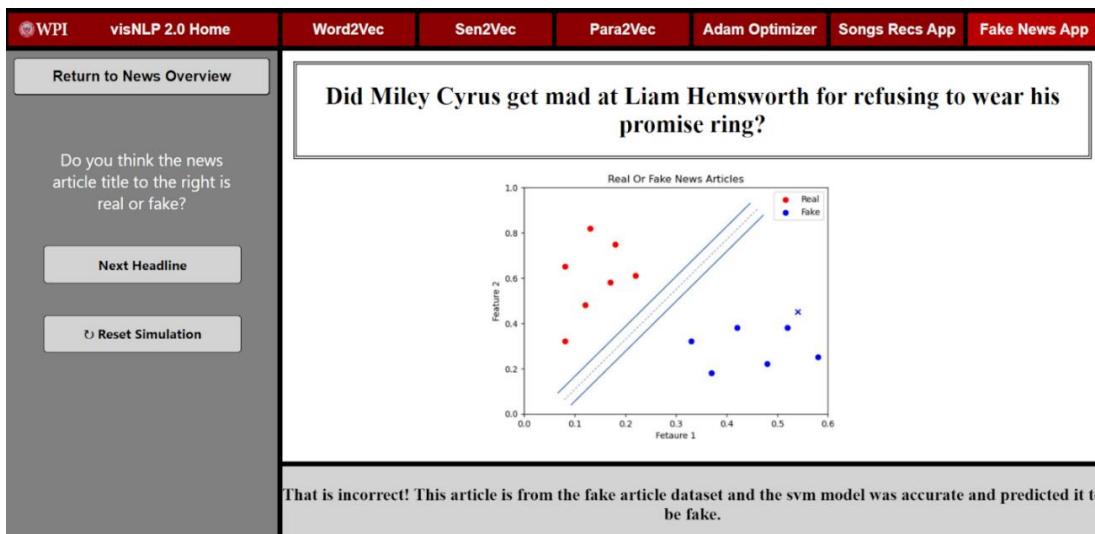


Figure 35: Fake News Application Post-User Prediction Page

In the example above the user predicted that the article was real which was incorrect. So, as you can see the article was plotted in the fake cluster of the classification plot as well as the footer notified the user that they were incorrect and that the SVM model was accurate. This approach proves useful for users as it not only presents an example of a real-world application utilizing NLP techniques but also facilitates an enhanced understanding of how articles can be clustered based on their feature vectors.

5 Platform Evaluation and Discussion

To evaluate the VisNLP 2.0 educational web-platform for teaching NN-based NLP, we conducted user trials followed by platform evaluation surveys. The objective of these trials was for users to self-learn NN-based NLP by navigating through the web-platform independently. The user trials were completed by college-level students from varying disciplinary backgrounds. Directly following the user trials, the user was asked to answer a brief questionnaire regarding their experience. A total of 8 questions were included in this survey where respondents were asked to assign a rating of 1 to 5 for each question. A rating of 1 indicated “poor”, a rating of 2 indicated “fair”, a rating of 3 indicated “satisfactory”, a rating of 4 indicated “very good”, and a rating of 5 indicated “excellent”. Table 5 below shows all 8 questions asked to the respondents during the platform evaluation survey:

Q1	On a scale of 1-5, how would you rate your level of statistical and neural network-based NLP knowledge before using the VisNLP 2.0 platform?
Q2	On a scale of 1-5, how easy was it to understand and navigate the website platform?
Q3	On a scale of 1-5, how clear and understandable were the statistical and neural network-based NLP concepts demonstrated by VisNLP 2.0?
Q4	On a scale of 1-5, how helpful do you think the step-by-step process is in breaking down neural network-based NLP methods?
Q5	How well did VisNLP 2.0 help you comprehend the NLP pipeline to real-world applications: Song Curation & Fake News Detection
Q6	On a scale of 1-5, how would you rate this style of teaching NLP versus a lecture style format or assigned readings?
Q7	On a scale of 1-5, how much has your knowledge of neural network-based NLP improved since using the VisNLP 2.0 platform?
Q8	On a scale of 1-5, how likely are you to recommend the VisNLP 2.0 platform as a resource for someone trying to self-learn neural network-based NLP?

Table 5: Platform Evaluation Survey Questions

Survey results and findings

A total of 32 platform evaluation surveys were collected from users after they partook in user trials. A condensed report of these results can be viewed below in Figure 36:

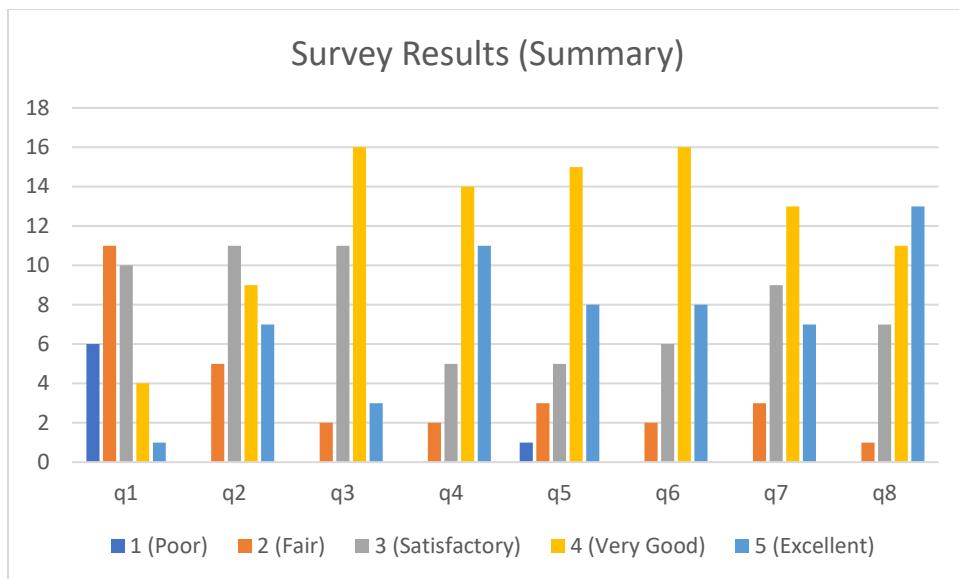


Figure 36: Platform Evaluation Survey Results Report (Visual Summary)

The demographic of our response pool was 32 college students with varying disciplinary backgrounds. To further understand the background of our respondents, the survey included a poll assessing their prior background of NN-based NLP (Question 1). Shown in Figure 37 below, we found that over 80% of respondents claimed to have limited prior knowledge of NN-based NLP. Correspondingly, less than 20% rated their prior knowledge of NN-Based NLP as a 4 (Very Good) or 5 (Excellent).

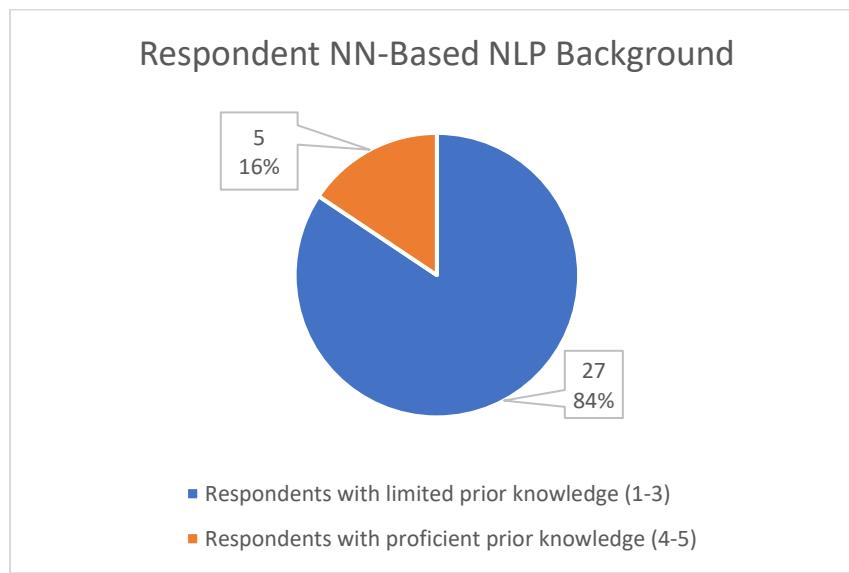


Figure 37: Respondent NN-Based NLP Background - Survey Results Visualized

Next, we look at responses which assess the VisNLP 2.0 platform's clarity of content and quality (Questions 2 and 3). Shown in Figure 38 below, we found that respondents found the clarity of the NN-based NLP concepts that were demonstrated to be generally good, with the vast majority

rating this at least a 3 (Satisfactory). Similarly, we found that most respondents also found the clarity and ease of navigation within the VisNLP 2.0 platform to be at least a 3 (Satisfactory).

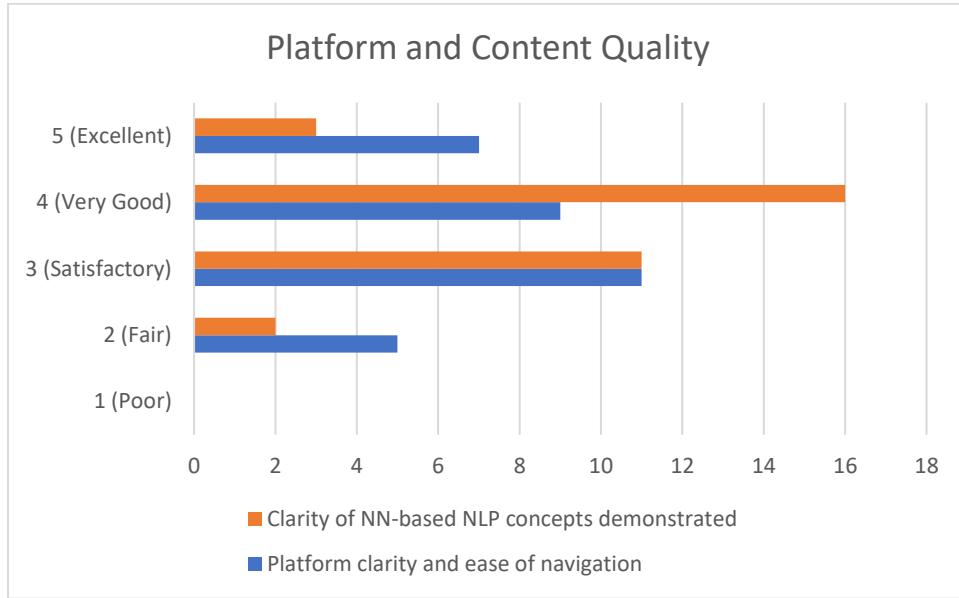


Figure 38: Platform and Content Quality - Survey Results Visualized

Next, we discuss survey responses which assess the effectiveness of various qualities and components of the VisNLP 2.0 platform (Questions 4, 5, and 6). As shown in Figure 39 below, we learned that around 78% of respondents found the step-by-step process to be significantly helpful in breaking down NN-based NLP methods. Note here, ratings of 4 (Very Good) and 5 (Excellent) are interpreted as significantly helpful, and ratings of 1 (Poor), 2 (Fair), and 3 (Satisfactory) are interpreted as not significantly helpful. Furthermore, we learned that around 71% of respondents found that the song curation & fake news detection applications significantly helped with understanding the NLP pipeline to real-world applications. The last quality we assessed was the style of teaching NN-based NLP in the VisNLP platform versus other conventional teaching approaches. We found that around 75% of respondents found this style of teaching to be significantly helpful.

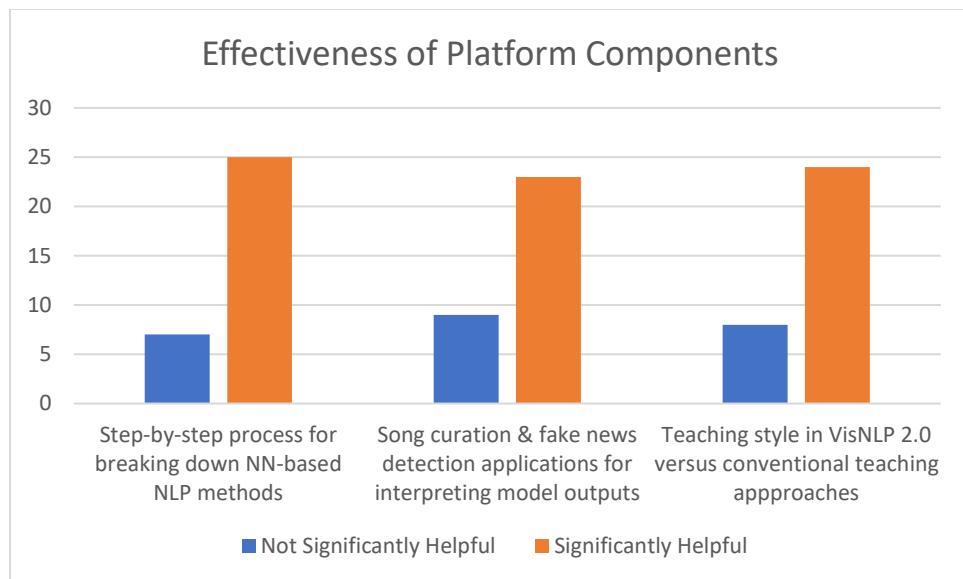


Figure 39: Effectiveness of Platform Qualities - Survey Results Visualized

Lastly, we assessed the respondent's consensus of the VisNLP 2.0 platform (Questions 7 and 8). As shown in Figure 40 below, we learned most respondents left a positive rating of their likelihood of recommending the VisNLP 2.0 platform as a resource for someone trying to self-learn neural network-based NLP. Last, we assessed the respondent's improvement in knowledge of neural network-based NLP after using the VisNLP 2.0 platform. Here, we found that well over 90% of respondents claimed to have learned and improved their knowledge of NN-based NLP to at least some extent, rating their improvement of NN-based NLP knowledge as a 3 (Satisfactory) or above.

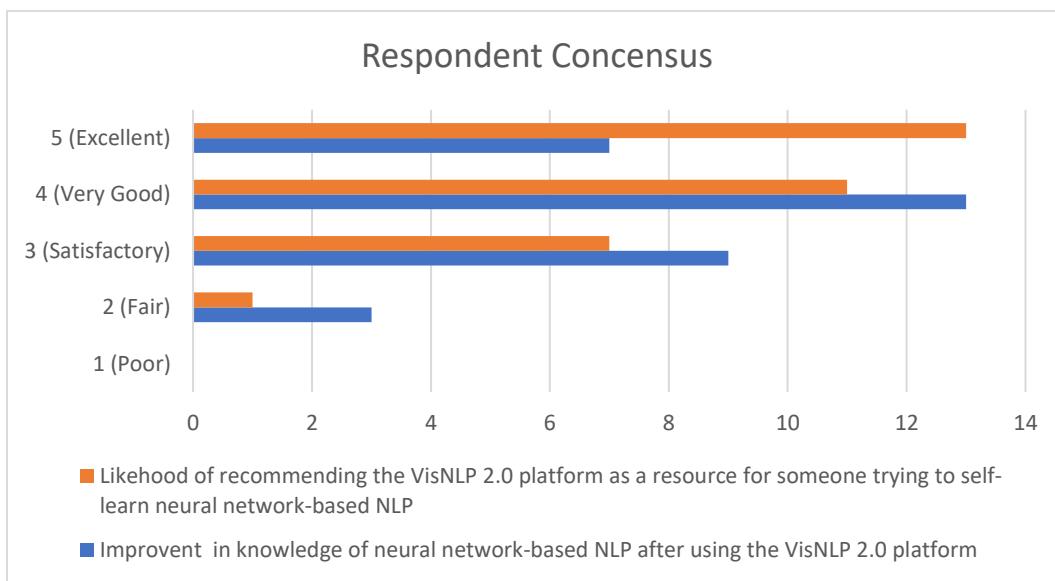


Figure 40: Respondent Consensus - Survey Results Visualized

6 Conclusions and Recommendations

This section first makes inference on the finding presented and analyzed in Section 5, which includes the web-platform evaluation survey results and discussion. Lastly, we provide some recommendations for potential future work which could expand upon our work done on the VisNLP 2.0 educational web-platform for teaching and learning NN-based NLP.

6.1 Conclusion

The survey results indicate that the NLP learning platform is an effective and valuable tool for learners seeking to develop their skills in this field. Most participants reported that the platform was easy to use and provided a comprehensive and engaging learning experience. Many participants also highlighted the value of the platform's interactive features, such as real-world applications and step-by-step examples.

The survey results also suggest that the NLP learning platform has a positive impact on the learner's knowledge and understanding of the subject matter. A considerable proportion of participants reported that the platform helped them to achieve a deeper understanding of key concepts and techniques, which is essential in mastering the complex field of NLP.

The findings of this survey highlight the importance of digital learning tools in supporting learners' development in highly technical fields such as NLP. By providing a comprehensive and engaging learning experience, the NLP learning platform can help learners of all skill levels to develop their understanding and expertise in this critical area of study. As such, it is an essential resource for individuals, educators, and institutions seeking to advance their knowledge and skills in NLP and related fields.

6.2 Recommendations and Future Work

While the VisNLP 2.0 platform provides a comprehensive and engaging learning experience for NN-based NLP, there is still room for future work and expansion. One direction for future work could be to include more widely used technologies like Glove (Stanford) and FastText (Facebook), as well as newer methods like transformer-based models. Additionally, the platform could be extended to cover other applications of NLP, such as topic modeling, sentiment analysis and text classification, with corresponding interactive visual demos. Overall, we recommend that future work continues to focus on providing granular and comprehensive step-by-step instruction, using real examples when applicable, and showing the user how these concepts can be applied. These additions in combination with the VisNLP framework could further enhance learners' access

to effective NLP education and continue to aid in providing a deeper understanding of the highly relevant field of NLP.

7 Appendix A: Step-by-Step Simulation Training Output Data

This JSON schema plays a summarizes the data extracted, categorized, and written to a file from training a word2vec model on a sample corpus. The schema represents the intermediate steps of training a CBOW word2vec model organized by epoch. The continuous output data from this schema was then used by our team to develop a step-by-step word2vec simulation.

Word2vec Training Data - JSON Schema

```
epoch
  "constants":
    "mode": string
    "input": string
    "vocabulary": array of strings
    "dictionary": key/value map of strings to ints
    "one_hots": key/value map of strings to arrays of ints
    "vocab_size": int
    "embedding_dim": int
  "params":
    "param 1": matrix of doubles
    "param 2": matrix of doubles
    "param 3": array of doubles
  "batch":
    "centers": array of ints
    "contexts": array of ints
    "context_embed": matrix of doubles
  "calcs":
    "mat_mult": matrix of doubles
    "mat_mult_footer": matrix of strings
    "plus_bias": matrix of doubles
    "plus_bias_footer": matrix of strings
    "softmax": matrix of doubles
    "log_softmax": matrix of doubles
    "log_softmax_footer": matrix of strings
  "next_params":
    "param 1": matrix of doubles
    "param 2": matrix of doubles
    "param 3": array of doubles
```

This JSON schema represents the data extracted, categorized, and written to a file from training a word2vec model using the Adam optimizer. The schema captures the intermediate steps of Adam optimization performed over forty iterations on a CBOW word2vec objective

function, organized by epoch. The continuous output data from this schema was then used by our team to develop a step-by-step Adam optimization simulation.

Adam Optimizer Run on Word2vec Model Training Data - JSON Schema

```
epoch
  "curr_model_params":
    "param_1": matrix of floats
    "param_2": matrix of floats
    "param_3": matrix of floats
  "cbow_steps":
    "center_context_pair_indexes": matrix of floats
    "context_one_hots": matrix of ints
    "param_1": matrix of floats
    "context_matrix": matrix of floats
    "param_2": matrix of floats
    "dot_prod": matrix of floats
    "param_3": matrix of floats
    "dot_prod_bias_sum": matrix of floats
  "loss_steps":
    "dot_prod_bias_sum": matrix of floats
    "softmax": matrix of floats
    "log_softmax": matrix of floats
    "log_softmax_vertical_avg": matrix of floats
    "center_one_hot": matrix of floats
    "avg_epoch_loss": int
    "avg_loss_vals": arr of floats
  "adam_optim_hyperparams":
    "learning_rate": float
    "beta_1": float
    "beta_2": float
    "eps": float
    "t": int
  "first_moment_calculations":
    "grad_beta_1_products"
      "param_1": matrix of floats
      "param_2": matrix of floats
      "param_3": matrix of floats
    "prev_m_beta_1_products"
      "param_1": matrix of floats
      "param_2": matrix of floats
      "param_3": matrix of floats
  "second_moment_calculations":
```

```
"grads_sq":  
    "param_1": matrix of floats  
    "param_2": matrix of floats  
    "param_3": matrix of floats  
"grad_sq_beta_2_products":  
    "param_1": matrix of floats  
    "param_2": matrix of floats  
    "param_3": matrix of floats  
"prev_v_beta_2_products":  
    "param_1": matrix of floats  
    "param_2": matrix of floats  
    "param_3": matrix of floats  
"gradient_states":  
    "grads":  
        "param_1_grad": matrix of floats  
        "param_2_grad": matrix of floats  
        "param_3_grad": matrix of floats  
    "prev_first_moments_raw":  
        "prev_param_1_m": matrix of floats  
        "prev_param_2_m": matrix of floats  
        "prev_param_3_m": matrix of floats  
    "prev_second_moments_raw":  
        "prev_param_1_v": matrix of floats  
        "prev_param_2_v": matrix of floats  
        "prev_param_3_v": matrix of floats  
    "first_moments_raw":  
        "param_1_m": matrix of floats  
        "param_2_m": matrix of floats  
        "param_3_m": matrix of floats  
    "second_moments_raw":  
        "param_1_v": matrix of floats  
        "param_2_v": matrix of floats  
        "param_3_v": matrix of floats  
    "first_moments_bc":  
        "param_1_m_hat": matrix of floats  
        "param_2_m_hat": matrix of floats  
        "param_3_m_hat": matrix of floats  
    "second_moments_bc":  
        "param_1_v_hat": matrix of floats  
        "param_2_v_hat": matrix of floats  
        "param_3_v_hat": matrix of floats  
"param_update_steps":  
    "sqrt_v_hat":  
        "param_1": matrix of floats  
        "param_2": matrix of floats  
        "param_3": matrix of floats
```

```
"ans_eps_sum":  
    "param_1": matrix of floats  
    "param_2": matrix of floats  
    "param_3": matrix of floats  
"m_hat_ans_quotient":  
    "param_1": matrix of floats  
    "param_2": matrix of floats  
    "param_3": matrix of floats  
"alpha_ans_product":  
    "param_1": matrix of floats  
    "param_2": matrix of floats  
    "param_3": matrix of floats  
"updated_params":  
    "param_1": matrix of floats  
    "param_2": matrix of floats  
    "param_3": matrix of floats
```

8 Appendix B: Platform Evaluation Survey Full Results Report

This results report provides the unabridged results of our platform evaluation surveys used to evaluate the VisNLP 2.0 educational web-platform for teaching neural network-based NLP. The demographic of our response pool was 32 college-level students with varying disciplinary backgrounds. The respondents first partook in the self-guided learning of neural network-based NLP by navigating through the web-platform independently. Directly after, the respondents took the platform evaluation survey. The survey consisted of 8 questions, where respondents were asked to rate their experience on a scale of 1 to 5, with 1 being “poor” and 5 being “excellent”. The objective of the survey was to evaluate the effectiveness of the platform and gather feedback for further improvements.

Q1 - On a scale of 1-5, how would you rate your level of statistical and neural-network based NLP knowledge before using the VisNLP 2.0 platform?



Q2 - On a scale of 1-5, how easy was it to understand and navigate the website platform?



Q3 - On a scale of 1-5, how clear and understandable were the statistical and neural network-based NLP concepts demonstrated by VisNLP 2.0?



Q4 - On a scale of 1-5, how helpful do you think the step-by-step process is in breaking down neural network-based NLP methods?



Q5 - How well did VisNLP 2.0 help you comprehend the NLP pipeline to real-world applications: Song Curation & Fake News Detection



Q6 - On a scale of 1-5, how would you rate this style of teaching NLP versus a lecture style format or assigned readings?



Q7 - On a scale of 1-5, how much has your knowledge of neural-network based NLP improved since using the VisNLP 2.0 platform?



Q8 - On a scale of 1-5, how likely are you to recommend the VisNLP 2.0 platform as a resource for someone trying to self-learn neural network-based NLP?



9 References

- [1] Etzioni, O. (2019). The future of natural language processing. *Communications of the ACM*, 62(4), 54-61. doi:10.1145/3303861
- [2] Jurafsky, D., & Martin, J. H. (2020). *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition* (3rd ed.). Pearson.
- [3] Text Analysis. (2020). Text Analytics Website MonkeyLearn. <https://monkeylearn.com/>
- [4] Shankar, A., & Kay, M. (2019). Introducing Alexa Conversations: a New Deep Learning-Based Approach to Natural Dialog Modeling. Amazon Science. <https://www.amazon.science/blog/introducing-alexa-conversations-a-new-deep-learning-based-approach-to-natural-dialog-modeling>
- [5] Guo, M., Han, S., Zhang, M., & Liao, J. (2019). iOS Auto-Correction Deep Learning Model. *Proceedings of the 2019 3rd International Conference on Big Data and Education*, 80-84. doi: 10.1145/3330429.3330445
- [6] International Journal of Advanced Research in Computer Science and Software Engineering. (2014). A Natural Language Processing Approach for Email Spam Filtering. http://www.ijarcsse.com/docs/papers/Volume_4/4_April2014/V4I4-0176.pdf
- [7] Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., ... & Amodei, D. (2020). Language models are few-shot learners. arXiv preprint arXiv:2005.14165. Retrieved from <https://arxiv.org/pdf/2005.14165.pdf>
- [8] Lazzaro, S. (2021, June 21). Machine learning's rise, applications, and challenges. VentureBeat. <https://venturebeat.com/ai/machine-learnings-rise-applications-and-challenges/>
- [9] Google. (n.d.). Machine Learning Google Trends. Google Trends. Retrieved December 17, 2022, from <https://trends.google.com/trends/explore?date=all&geo=US&q=machine%20learning>
- [10] Reed, N. (2021, November 11). The Ai Black Box Problem. ThinkAutomation. <https://thinkautomation.com/bots-and-ai/the-ai-black-box-problem/>
- [11] Deng, L., & Liu, Y. (2018). *Deep Learning in Natural Language Processing* (1st ed. 2018 ed.). Springer.
- [12] Statistics and Data Science MicroMasters. (2020). Statistics and Data Science MicroMasters. <https://micromasters.mit.edu/ds/>
- [13] Rehman, Z., & Kifor, S. (2015). Teaching Natural Language Processing (NLP) Using Ontology Based Education Design. Balkan Region Conference on Engineering and Business Education, 1(1), 206–214. <https://doi.org/10.1515/cplbu-2015-0024>

- [14] Rong. (2016). wevi: Word Embedding Visual Inspector. <https://ronxin.github.io/wevi/>
- [15] Chokchaisiripakdee, A., & Ngan, C.-K. (2021). VisNLP: A Visual-based Educational Support Platform for Learning Statistical NLP Analytics. Research Gate. <https://doi.org/10.13140/RG.2.2.14203.36641>
- [16] Mikolov, Sutskever, I., Chen, K., Corrado, G., & Dean, J. (2013). Distributed Representations of Words and Phrases and their Compositionality. <https://doi.org/10.48550/arxiv.1310.4546>
- [17] Rong. (2014). word2vec Parameter Learning Explained. <https://doi.org/10.48550/arxiv.1411.2738>
- [18] Dai, Olah, C., & Le, Q. V. (2015). Document Embedding with Paragraph Vectors. <https://doi.org/10.48550/arxiv.1507.07998>
- [19] Le, & Mikolov, T. (2014). Distributed Representations of Sentences and Documents. <https://doi.org/10.48550/arxiv.1405.4053>
- [20] Yang, Yang, K., Cui, T., Chen, M., & He, L. (2022). A Study of Text Vectorization Method Combining Topic Model and Transfer Learning. *Processes*, 10(2), 350–. <https://doi.org/10.3390/pr10020350>
- [21] Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. International Conference on Learning Representations (ICLR). <https://arxiv.org/abs/1412.6980>
- [22] Ruder, S. (2017, August 19). Deep Learning for NLP Best Practices. Retrieved April 10, 2022, from [https://www.ruder.io/deep-learning-nlp-best-practices/#:~:text=Optimization%20algorithm%20Adam%20\(Kingma%20%26%20Ba,stochastic%20gradient%20descent%20\(SGD\)\).](https://www.ruder.io/deep-learning-nlp-best-practices/#:~:text=Optimization%20algorithm%20Adam%20(Kingma%20%26%20Ba,stochastic%20gradient%20descent%20(SGD)).)
- [23] Reddi, S. J., Kale, S., & Kumar, S. (2018). On the convergence of Adam and beyond. International Conference on Learning Representations (ICLR). Retrieved from <https://openreview.net/forum?id=ryQu7f-RZ>
- [24] Poppermann, A. (n.d.). Optimization in Deep Learning: Adagrad, RMSprop, Adam. Retrieved April 29, 2023, from <https://artemoppermann.com/optimization-in-deep-learning-adagrad-rmsprop-adam/>
- [25] Ribeiro, C. R. M., & Brasil, E. (2017). A word2vec-based recommendation system for music playlists. In Proceedings of the 14th Sound and Music Computing Conference (pp. 170-175).
- [26] Liu, J., & Zhang, X. (2020). A comparative study of machine learning algorithms for sentiment analysis. *IEEE Access*, 8, 23611-23622.
- [27] Zhang, X., Zhou, X., Lin, S., & Sun, M. (2019). Hybrid neural models for sentiment analysis using attention-based CNNs and deep learning RNNs. *Neurocomputing*, 338, 139-149.

- [28] Pang, B., & Lee, L. (2008). Opinion mining and sentiment analysis. *Foundations and Trends in Information Retrieval*, 2(1-2), 1-135.
- [29] Schölkopf, B., & Smola, A. J. (2002). *Learning with kernels: Support vector machines, regularization, optimization, and beyond*. MIT press.
- [30] Amin, M. A., Selim, G. M., & Al-Fahoum, A. S. (2020). A hybrid SVM and ELM-based medical diagnosis system for effective diagnosis of cardiovascular disease. *Journal of Ambient Intelligence and Humanized Computing*, 11(11), 4943-4953.
- [31] Puttur, VisNLP (2022), GitHub repository, <https://github.com/aadhyap/VisNLP>
- [32] Inejc, Paragraph Vectors, (2018), GitHub repository, <https://github.com/inejc/paragraph-vectors>
- [33] Vrjkmr, song2vec, (2020), GitHub repository, <https://github.com/vrjkmr/song2vec>
- [34] Tomytjandra, song2vec-music-recommender, GitHub repository, <https://github.com/tomytjandra/song2vec-music-recommender>
- [35] He, L., Qin, X., & Liu, T. (2017). Music recommendation based on lyrics using word2vec. *Journal of Intelligent Information Systems*, 48(3), 563-581.
- [36] Zhang, Y., Wu, S., & Sun, A. (2019). Personalized music recommendation based on word2vec. In *Proceedings of the 20th International Conference on Computational Linguistics and Intelligent Text Processing* (pp. 334-345).
- [37] KaiDMML, FakeNewsNet, GitHub Repository, <https://github.com/KaiDMML/FakeNewsNet>
- [38] Seyedsaeidmasoumzadeh, Binary-Text-Classification-Doc2vec-SVM, GitHub Repository, <https://github.com/seyedsaeidmasoumzadeh/Binary-Text-Classification-Doc2vec-SVM>