CSCI544: Project Report - Neural Architecture Search to find the Optimal Crisis Information Classifier

Roshnee Matlani
matlani@usc.edu

Advait Naik
naika@usc.edu

Roshan Rangarajan
roshanr@usc.edu

Shoumik Gandre
shoumika@usc.edu

Vyas Srinivasan
vrsriniv@usc.edu

University of Southern California

Abstract

The project proposes an evolutionary algorithm for neural architecture search in natural language processing (NLP) tasks. The proposed methodology involves exploring the dataset, preprocessing the data, representing neural networks as a genome, and implementing the evolutionary algorithm. The neural networks are represented as directed acyclic graphs (DAGs), and the evolutionary algorithm generates an initial population of DAGs, applies genetic operators - crossover and mutation, and evaluates the fitness of the resulting DAGs using the CrisisMMD dataset. The proposed method aims to construct formidable neural networks for NLP tasks, providing a useful technique for classification tasks. These results are compared to traditional Machine Learning methods.

1 Introduction

In this project, we aim to use Neural Architecture Search (NAS) to find the optimal neural network parameters for binary classification tasks in Natural Language Processing. To achieve this goal, we use a genetic algorithm and generate different neural networks in order to get the best accuracy for the binary classification task. We also wanted to create an NLP solution for an important field that will have many practical applications. Hence, we chose the classification task to determine whether a tweet about natural disasters is Informative or Non-Informative. This can help in fast disaster-response systems. The dataset is called CrisisMMD and contains tweets from several natural disasters and contains labels for each tweet about whether it is informative about the disaster or not.

We start by cleaning and exploring the dataset through many traditional and tweet-specific techniques before creating an embedding matrix using Glove Embeddings. Next we start by representing neural networks as Directed Acyclic Graphs and create an initial population through that. Further, we work on specific crossover and mutation functions for the further part of the Genetic Algorithm. Finally, we use Accuracy as the fitness function to determine which neural networks are better than the other. We find accuracy by first representing our neural networks in PyTorch instead.

We also implement traditional Machine Learning methods for the binary classification task from scratch for the dataset such as Logistic Regression, Multinomial Naive Bayes, Random Forest Classifier etc. The results of these methods are then compared with our method.

2 Related Work

NeuroEvolution of Augmenting Topologies (NEAT) provides a general framework to represent the problem of using a Genetic Algorithm to Evolve Neural Network topologies (Stanley and Miikkulainen, 2002). NEAT has large number of parameters which hindered its success. More efficient techniques such as NSGA-Net have more efficient exploration and exploitation of the search space through a customized crossover scheme (Lu et al., 2018). "Learning Transferable Architectures for Scalable Image Recognition" is a research paper that introduces an algorithm called NASNet search space for optimizing neural network architectures. The NASNet search space algorithm starts with a small network architecture and iteratively adds new layers (Zoph et al., 2017).

The research work that combined genetic algorithms and neural networks (NNs) mainly aimed to optimize the weights on each layer or change the connections between neurons on different layers (Mahajan and Kaur, 2013; Idrissi et al., 2016). NAS-Bench-NLP (Klyuchnikov et al., 2022) work proposes a benchmark for Neural Architecture Search (NAS) in the field of Natural Language Processing (NLP) that evaluates the model using two state-of-the-art NAS algorithms, DARTS and ENAS, and compares their performance to a set

of handcrafted architectures. The research work (Domashova et al., 2021) implements a method of genetic operators to generate new architectures, including crossover and mutation. The authors evaluate the performance of their proposed method on benchmark datasets, compare it to other state-of-the-art neural architecture search methods and propose that the genetic algorithm outperforms other methods in terms of accuracy and efficiency.

The following are the papers that use the CrisisMMD dataset. The paper (Kotha et al., 2022) proposes a multimodal approach combining text and image information from social media tweets to aid disaster response assessment. The authors develop a framework that employs a convolutional neural network (CNN) to extract image features, a recurrent neural network (RNN) to extract text features and a multimodal fusion layer to combine the features for classification. This work uses a pre-trained language model and few-shot learning to classify disaster tweets as relevant or not. Few-shot learning is trained on a small set of labeled examples from the target disaster event and outperform other methods (Kruspe, 2019).

3 Problem Description

To propose an Evolutionary algorithm in order to find good Natural Language Processing (NLP) Neural Architectures for the CrisisMMD dataset. CrisisMMD dataset is a natural disaster tweet dataset that labels tweets as informative and non-informative, therefore, NLP is required to extract the relevant meaning from these tweets in order to classify them accordingly. We will hence implement a Genetic Algorithm while trying to solve challenges like ideal fitness function, mutation, and crossover methods among others. This method will then be compared to other Machine Learning and Deep Learning methods. A successful implementation of this Evolutionary algorithm provides the field of NLP with a technique that constructs formidable Neural Networks that can perform other comparable networks for classification tasks.

4 Methodology

4.1 Dataset

CrisisMMD: Multimodal Crisis Dataset is our data set of choice for Neural Architecture Search Problem. This dataset was created by (Alam et al., 2018) and used in (Ofli et al., 2020). The CrisisMMD can be accessed through the URL - https://crisisnlp.qcri.org/.

This dataset contains tweets collected from various natural disasters, such as hurricanes, wildfires, floods, and earthquakes in 2017. For our project, we will use the task of classifying tweets as Informative or Non-Informative. In this task, there are a total of 16,058 tweets, with 11,509 classified as informative and 4,549 classified as non-informative.

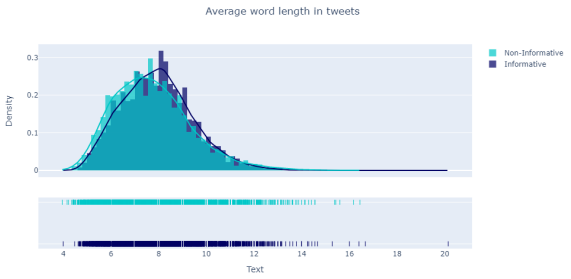

Figure 1: Length of Tweet Informative vs Non-Informative

We began our project by exploring the dataset to get a better understanding of the composition of the tweets as well as the factors that make them informative or non-informative. We did this through means such as analyzing the word clouds, the average length of tweets, top unigrams, bigrams, and trigrams in the tweets. We made these analyses and compared the value of each for informative to the same metrics for non-informative tweets. One such example of the analysis can be seen in Figure 1.

4.2 Data Preprocessing

Based on the analysis we did in the previous part, we came up with the following data-cleaning and pre-processing techniques for each of the train, test, and dev sets - Converted the tweets into lowercase text and Replace time, dates, and URLs with a corresponding tag instead to represent them. Removed special characters as well as hashtags that might be present in tweets. References to users in tweets that start with '@' tag were removed. Removed repeated characters as well as the extra whitespaces that exist. Went through the tweets and using an external stopwords text file, removed the extra stopwords that exist in the tweets. Performed tokenization on each of the tweets. Using Glove Embeddings created a word embedding matrix that stores an average word embedding of size 100 for each of the tweets. At the end of pre-processing, we were left with a word embedding matrix of

shape (number of tweets, 100) as well as a label vector denoting whether the tweet is informative or non-informative of shape (length of tweets).

4.3 Representing Neural Networks as a Genome

Genetic algorithms work by encoding a problem statement into chromosome-like data structure and applying recombination operators to these structures while maintaining their useful information. (Whitley, 1994). In our context, we think of a Neural Network (NN) as a Directed Acyclic Graph(DAG). We need to define a mapping f such that given a Neural Network g, $f(g)$ maps g to a chromosome-like representation (encoding) r. $r = f(g)$. This mapping function should ideally meet the following five criteria:

1. Ability to represent all neural networks
2. Unbiased: *All neural networks are equally represented by the same encoding size.*
3. It should be capable of representing only Neural Networks (DAG). If non-NN can be represented, then crossover and mutation steps of Genetic algorithm can generate non-NNs.
4. It is feasible to convert the representation into the Neural Network and vice-versa. $r \leftarrow f(g)$ and $g \leftarrow f^{-1}(r)$ should be possible making the evaluation of the fitness function and constraints suitable.
5. Preservation of Locality: small changes in the representation cause small changes to NN.

Ideally, all of the above traits should be followed. However, we can trade some desirable traits for others (Palmer and Kershenbaum, 1994). Although it is a non-trivial task, NSGA-Net offers a way to encode a neural network into a genome. Adapting this approach to suit our use case would be optimal (Lu et al., 2018). We represent a Neural network as a Directed Acyclic Graph with a single source and sink. The genome is a tuple of a list of nodes and a list of connections between these nodes.

Initial Population Generation-
To create a random directed acyclic graph, First, initialize a graph. Second, for each new node to be added, iterate over all existing nodes and add an existing node as the parent of the new node with a probability of 50%. Third, add a source node and connect it to all nodes that have no parents. Fourth, add a sink node and connect it to all nodes that have no parents.

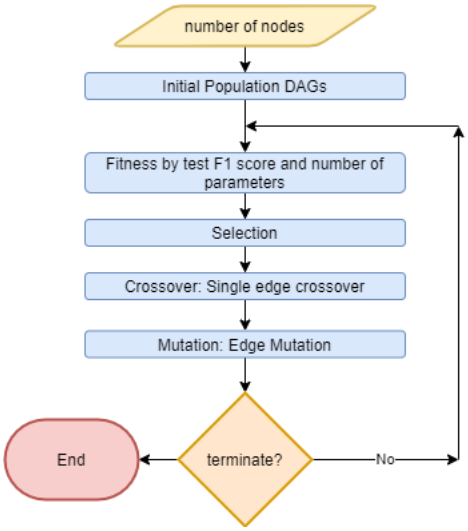

Figure 2: Methodology

The above procedure is repeated N times to obtain N random Directed Acyclic Graphs with a single source and single sink. Figure 2. is the diagram that summarizes our overall methodology of the Genetic Algorithm

4.4 Crossover functions for NLP Architectures

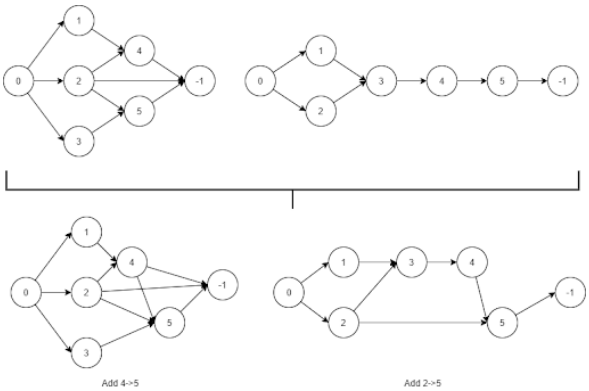

Figure 3: Crossover- Single Edge Crossover

As shown in Figure 3, the crossover function scans both graphs for differences in edge connections. In the crossover function, we add or remove the graph connections in graph 1 based on the the connections of graph 2. If graph 2 has a connection that graph 1 does not have, during the crossover function, it has a liklihood to gain that connection.

4.5 Mutation

As shown in Figure 4, for mutation, we randomly add or remove an edge in a graph.

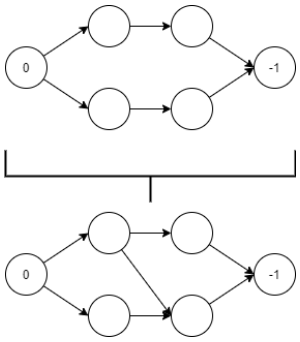

Figure 4: Mutation- Edge Mutation

4.6 Fitness Function

First, we compile Genome (Directed Acyclic Graphs) to PyTorch Module. Then, we train this PyTorch module and evaluate it over a test set Finally, we use the evaluation result of the test set as one of the terms of the fitness function. To compile to a Pytorch Module, we apply nn.ModuleDict operation of PyTorch on the nodes listed in the Genome to create a PyTorch module. Furthermore we obtain a topologically sorted order of the nodes.

For the forward operation of the PyTorch Module, pass input x through the source node. Then iterate through nodes in topological order and collect the output tensors to be used by future nodes. Finally, Return the output tensor of the sink node. If two or more outputs enter a node, concatenate them. Train this PyTorch model, apply evaluation metric and then calculate the fitness using this evaluation metric. We require a fitness score "F" such that: $F \propto E$ and $F \propto \frac{1}{C}$ where E is the evaluation metric of test accuracy score and C is the model complexity $C = num(nodes) + num(connections)$ We can combine the above as follows:

$$F = E + \lambda \frac{1}{C}$$

where λ is a hyperparameter. The plan is to find a proper C function that encapsulates the idea of model complexity.

5 Experimental Results

For the comparison between different models we made use of the Accuracy Metric. We compare tradition ML methods that we implemented along with other deep learning methods. Table 1 represents a comparison of the performance of different machine learning and deep learning models on a particular dataset, using accuracy as the evaluation metric. Among the machine learning models, Support Vector Classifier has the highest accuracy score of 0.743, followed by Logistic Regression and Random Forest Classifier with accuracy scores of 0.734 and 0.728, respectively. Multinomial Naive Bayes has the lowest accuracy score of 0.687. The deep learning models generally outperform the machine learning models, with Bidirectional Long Short Term Memory (LSTM) achieving the highest accuracy score of 0.824. Convolutional Neural Network, MultiChannel Convolutional Neural Network, and DistilBert Transformer also perform well, with accuracy scores of 0.804, 0.803, and 0.812, respectively. However, the Neural Network model has the lowest accuracy score of 0.657. Interestingly, the Proposed Methodology achieves an accuracy score of 0.734, which is the same as the accuracy score of Logistic Regression. This suggests that the Proposed Methodology is still better than a normal ANN but is still lacking when compared to better Deep Learning Models.

Machine Learning Models	**Accuracy**
Logistic Regression	0.734
Multinomial Naive Bayes	0.687
Random Forest Classifier	0.728
Support Vector Classifier	0.743
Deep Learning Models	
Neural Network	0.657
Convolutional Neural Network	0.804
MultiChannel CNN	0.803
Bidirectional LSTM	0.824
DistilBert Transformer	0.812
Proposed Method	0.734

Table 1: Experimental Results

6 Conclusion

In this project, a genetic algorithm is implemented that constructs Neural Networks that can perform classification tasks for the CrisisMMD dataset. The challenges we faced included how to represent neural networks as genomes, how to convert a graph into a PyTorch representation, and the ideal mutation, crossover, and fitness functions for our Genetic Algorithm. We found results that were slightly better than just a normal Artificial Neural

Network, but still, results that were short of some Machine Learning Models and more complex deep learning models. we believe this is because of the graphs we are genetic in our Genetic Algorithm and computation limitations. In the future, we hope to use more complex networks as part of our Genetic Algorithm such as LSTMs and CNNs and we believe this will enhance the accuracy even more.

7 Individual Contribution

1. **Roshnee Matlani-** Initially, created a Git repository and implemented a structure to manage feature branches. The creation of feature branches allowed the team to work on separate features simultaneously and merge them back into the main branch once they were complete and tested. Secondly, I contributed to the preprocessing of the dataset which includes one-hot encoding, implementing tokenizer for tweets and removing stopwords. During the second stage of the project, I worked on the dataset visualization. I created different graphs for better understanding of the data. By visualizing data, we were able to gain insights into its characteristics and identify potential issues that needed to be addressed during preprocessing. Later, I implemented the Machine Learning models on our dataset and evaluated them on accuracy metrics.

2. **Advait Naik-** Created GCP setup for the project. In addition to configuring the GCP setup, I also trained multiple deep learning models. The various models were as follows: Convolutional Neural Network, Multi-Channel Convolutional Neural Network, Bidirectional Long Short Term Memory (LSTM), and DistilBert Transformer. Furthermore, also performed data visualization as part of the data processing which is an essential aspect of understanding and analyzing data, as it helps to identify patterns and relationships between variables that might not be evident from raw data alone. Overall, I work on setting up the GCP environment, training multiple deep learning models, and performing data visualization which was crucial for the success of the project.

3. **Roshan Rangarajan-** Researched and implemented data-cleaning methods such as removing stopwords, URLs, extra whitespaces, tokenization etc. Created an embedding matrix of the tokenized words using Glove Embeddings. Also, researched efficient genetic algorithm metrics such as the right mutation and crossover functions. Implemented the genetic algorithm through the NEAT Python library with different configurations of the genetic algorithm and tested the classification task.

4. **Shoumik Gandre-** First, identified how to implement Neural Network as a Genome and the ways to use Genetic Algorithm for NLP tasks. This involved reading genetic algorithm for neural network research papers (NEAT, TWEANN etc) and using this knowledge for our use case. Second, designed the crossover, mutation and fitness functions. Third, coded the graph algorithms required by the directed acyclic graphs. Fourth, devised a method to compile the graph into a PyTorch module.

5. **Vyas Srinivasan-** Researched a variety of articles on Neural Architecture Search (NAS), few-shot learning, NeuroEvolution of Augmenting Technologies (NEAT), and Genetic Algorithm to help the team get a better understanding of the fundamental concepts required to successfully implement a genetic algorithm. I attempted to help my teammates with Directed Acyclic Graphs (DAG), when they were having difficulty. I researched and implemented additional data cleaning functions to convert text to lowercase, remove HTML from text, eliminate non-alphabetical characters, drop extra spaces. The libraries used for my data cleaning functions are re, BeautifulSoup, and, contractions.

8 Project Link

Link - https://github.com/Roshnee21/CSCI-544-ANLP-PROJECT.git

References

Firoj Alam, Ferda Ofli, and Muhammad Imran. 2018. Crisismmd: Multimodal twitter datasets from natural disasters. *Proceedings of the International AAAI Conference on Web and Social Media*, 12(1).

Jenny V Domashova, Sofia S Emtseva, Vladislav S Fail, and Aleksandr S Gridin. 2021. Selecting an optimal architecture of neural network using genetic algorithm. *Procedia Computer Science*, 190:263–273.

Mohammed Amine Janati Idrissi, Hassan Ramchoun, Youssef Ghanou, and Mohamed Ettaouil. 2016. Genetic algorithm for neural network architecture optimization. In *2016 3rd International conference on logistics operations management (GOL)*, pages 1–4. IEEE.

Nikita Klyuchnikov, Ilya Trofimov, Ekaterina Artemova, Mikhail Salnikov, Maxim Fedorov, Alexander Filippov, and Evgeny Burnaev. 2022. Nas-bench-nlp: neural architecture search benchmark for natural language processing. *IEEE Access*, 10:45736–45747.

Saideshwar Kotha, Smitha Haridasan, Ajita Rattani, Aaron Bowen, Glyn Rimmington, and Atri Dutta. 2022. Multimodal combination of text and image tweets for disaster response assessment. International Workshop on Data-driven Resilience Research.

Anna Kruspe. 2019. Few-shot tweet detection in emerging disaster events. *arXiv preprint arXiv:1910.02290*.

Zhichao Lu, Ian Whalen, Vishnu Boddeti, Yashesh Dhebar, Kalyanmoy Deb, Erik Goodman, and Wolfgang Banzhaf. 2018. Nsga-net: Neural architecture search using multi-objective genetic algorithm.

Richa Mahajan and Gaganpreet Kaur. 2013. Neural networks using genetic algorithms. *International Journal of computer applications*, 77(14).

Ferda Ofli, Firoj Alam, and Muhammad Imran. 2020. Analysis of social media data using multimodal deep learning for disaster response. In *17th International Conference on Information Systems for Crisis Response and Management*. ISCRAM, ISCRAM.

C.C. Palmer and A. Kershenbaum. 1994. Representing trees in genetic algorithms. In *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence*, pages 379–384 vol.1.

Kenneth O. Stanley and Risto Miikkulainen. 2002. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127.

Darrell Whitley. 1994. A genetic algorithm tutorial. *Statistics and Computing*, 4(2):65–85.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. 2017. Learning transferable architectures for scalable image recognition.