COLANDER- CONVOLVING LAYER NETWORK DERIVATION FOR E-RECOMMENDATIONS

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

Dmitriy Timokhin

June 2021

COMMITTEE MEMBERSHIP

TITLE:	**COLANDER**- Convolving Layer Network Derivation for E-Recommendations
AUTHOR:	Dmitriy Timokhin
DATE SUBMITTED:	June 2021
COMMITTEE CHAIR:	Alexander Dekhtyar, Ph.D. Professor of Computer Science
COMMITTEE MEMBER:	Hunter Glanz, Ph.D. Professor of Statistics
COMMITTEE MEMBER:	Foaad Khosmood, Ph.D. Professor of Computer Science

ABSTRACT

COLANDER- Convolving Layer Network Derivation for E-Recommendations

Dmitriy Timokhin

Many consumer facing companies have large scale data sets that they use to create recommendations for their users. These recommendations are usually based off information the company has on the user and on the item in question. Based on these two sets of features, models are created and tuned to produce the best possible recommendations. A third set of data that exists in most cases is the presence of past interactions a user may have had with other items. The relationships that a model can identify between this information and the other two types of data, we believe, can improve the prediction of how a user may interact with the given item. We propose a method that can inform the model of these relationships during the training phase while only relying on the user and item data during the prediction phase. Using ideas from convolutional neural networks (CNN) and collaborative filtering approaches, our method manipulated the weights in the first layer of our network design in a way that achieves this goal.

ACKNOWLEDGMENTS

Thanks to:

- Andrew Guenther, for uploading this template

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

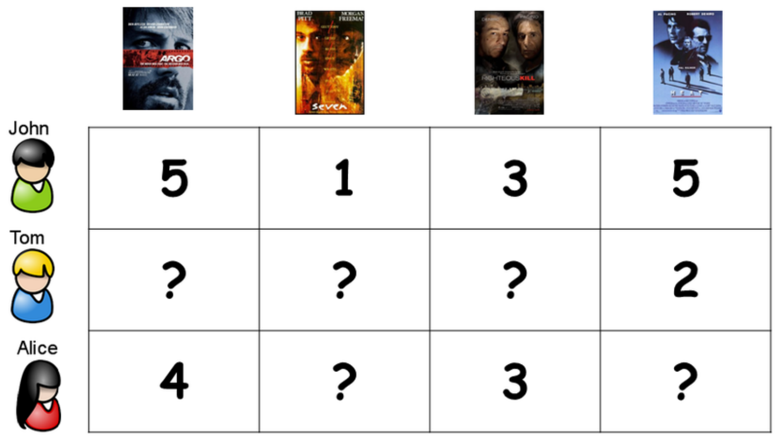
Chapter 1

INTRODUCTION

Predictive modeling is a branch of mathematics that focuses on predicting outcomes from historical data. The field has grown rapidly to include subfields of classification and regression modeling [14]. Many different applications of predictive modeling exist in industry such as predicting temperature, predicting the stock market, and recommending movies to users. Although these are all predicting modeling tasks they can be viewed as different problems. The problem of predicting weather may be associated with the previous temperatures recorded in the location, among many other things. For this problem, the output might be a numerical value in Fahrenheit which is a numerical output. On the other hand, for recommending movies the outcome is a movie or set of movies that a user might enjoy watching. Predicting movies one might enjoy falls under the umbrella of recommender systems. The work in this thesis is primarily focused on the problems facing current research in this field.

Many predictive modeling approaches, such as those commonly seen in recommender systems can be thought of in terms of interactions between two types of entities: users and items. In this context a user is considered a person that interacts with items, where an item can be thought of as something the user can interact with, such as movies, clothes, and books. A third type of entity can be identified as an interaction. An interaction is the reaction of the user to the item. These reactions can be both quantitative and qualitative. An example of a quantitative reaction is how long someone spent watching a movie, if not all of it. On the other hand a qualitative reaction can be given to the movie such as "Dislike" or "Would watch again". Predicting these reactions can be difficult without the proper information about the users and items. Metadata about the user and item can provide tremendous information to mathematical models to predict the interaction. The inclusion or exclusion of this information leads to two approaches to creating predictions.

The two approaches most commonly used are collaborative filtering and modeling with additional metadata on the items and users [34]. Collaborative filtering is the group of models that uses only the information presented in a NxM matrix of interactions between users and items where there are N users and M items. One possibility for an element of this matrix be a rating that a specific user gives to a movie. This matrix is shown below in Figure 1.1



John	5	1	3	5
Tom	?	?	?	2
Alice	4	?	3	?

Figure 1.1: Collaborative filtering matrix

From this type of matrix many methods have been developed to create predictions on how a user may react to an item it has not seen before [2]. One important aspect of this matrix is that is is sparse. This is to say that individual users may have interacted with some of the items but most likely not all of them, resulting in many elements being empty. Although sparse, these matrixes hold a lot of information useful for collaborative filtering. Collaborative filtering methods are based on the idea that if two users agree on a lot of their interactions (e.g. they like the same movies) then user A will probably like the items user B liked, that user A has not interacted with.

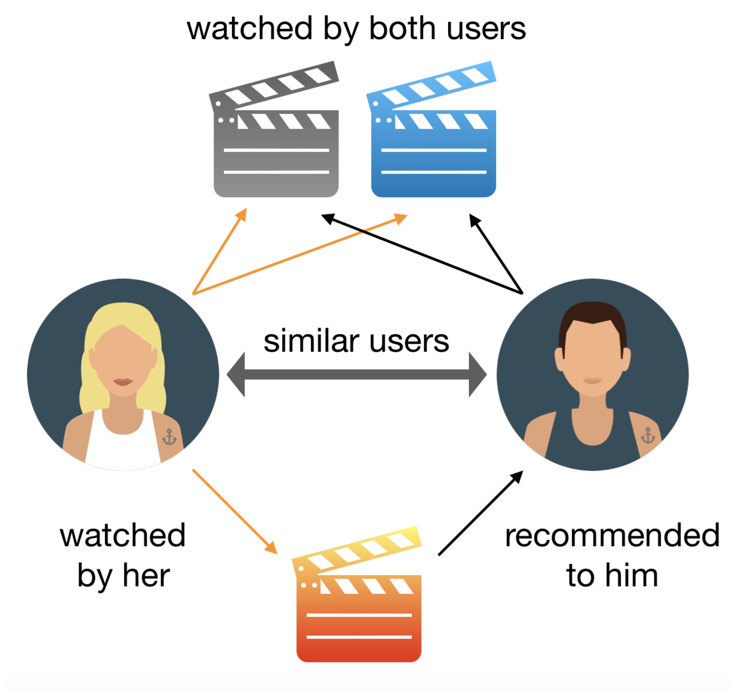


Figure 1.2: Basic recommender system

An example of this is if user A and user B have watched many of the same movies, but user B has watched Star Wars and user B has not which is similarly depicted in Figure 1.2. Using this logic we can recommend Star Wars to user A.

The second approach mentioned includes not only just the fact that a user has interacted with the item, but also information on the user, item, and their interactions. This approach computes similarity scores between two users or between two items based on the additional information available [26].

	Star Trek	Dr.Who	Farscape	Firefly
John	8	7	8	9
Ally	7	9	8	?

Table 1.1: Table of ratings (1-10) of television shows for John and Ally

If we look at example as shown in table 1.1, we have two users, John and Ally whose ratings for a few shows are observed. We can see that Both users really like Star Trek, Dr.Who, and Farscape which are all part of the science fiction genre. This may let us conclude that both users appreciate science fiction shows. Another show in 1.1 is Firefly which Ally has not seen. Our ability to deduce that Ally likes science fiction shows allows us to recommend Firefly to Ally without a collaborative-filtering inspired comparison of Ally's preferences to those of other users. Similar approaches can be used to reason about similarities between users. If several users like the same item (e.g., a specific song), we may want to find out what is common about these users (age, country of origin, favorite color, etc. . .), and when we encounter another user who matches the same demographic information, the song will be recommended to that user. Out of these two approaches this thesis is concerned with the second one, which focuses on including metadata associated with the users and items.

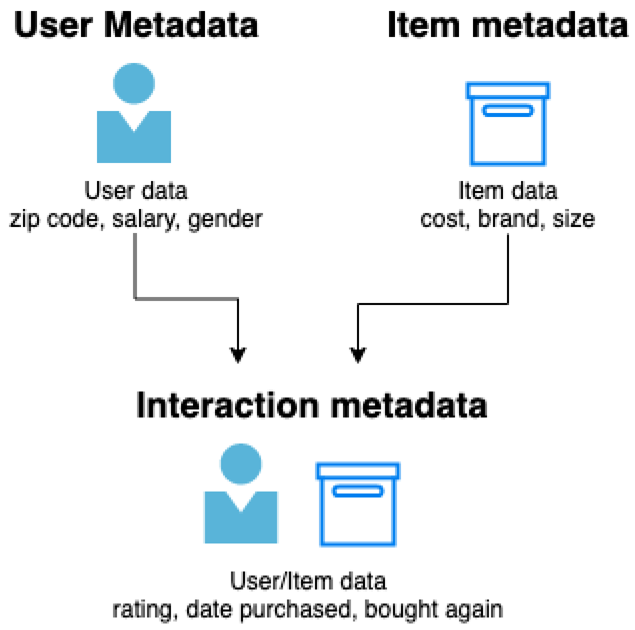


Figure 1.3: User, item, and interaction metadata example for item purchases

There are three types of information that is potentially available to meta-data-based predictive methods: user metadata, item metadata, and interaction metadata. User metadata is any information on the specific user. This can be their age, sex, and location to name a few. Item metadata is the data associated with the item. Some examples for data on an item can be its price, average rating, weight, and name.

What is most interesting to us though is the interaction metadata. Interaction metadata is any information available about the interaction between the user and the item in excess of the numeric quantity (such as level of enjoyment) that we are trying to predict. For example, when a user interacts with an item on Amazon.com, in addition to the eventual "bought/not bought" flag, Amazon can capture other information about the interaction: time spent on the product page, number of links on the page clicked, number of times the user viewed the page over the course of a month. Another example of some features associated with the different kinds of data for a user interacting with clothing items are depicted in Figure 1.3.

Most current research being done currently focuses on using user metadata and item metadata in order to create recommendations, while collaborative filtering does not even use metadata. One of the main issues with training models that are capable of generating good predictions is teaching the model the past relationships between users and items aside from simple interest scores. There are many examples of this in industry but a few are Amazon and Spotify. Amazon may have information on how many times a user clicked other items before purchasing the one they did. They might also know how long a user spent on the item page before purchasing it. Spotify can keep track of how many times a user listened to a specific song, or how many songs a user skipped before reaching the one they listened to. All of this information is highly correlated with how a user would react to an item but is quite difficult to incorporate into the model. This is because during testing predicting are being made on these interactions for items the user has never seen. This thesis focuses on addressing this problem.

We propose a method, COLANDER (Convolving layer network derivation for E-Recommendations), that can incorporate the previous relationships between user metadata, item metadata, and interaction metadata for new items during testing time. Traditional modern techniques of prediction involve one to have training features and a value to predict that is related to the training features. In this way, the model has more information during training than testing because not all of the relationships between the predictor variables and dependent variables are incorporated into how the models are trained. Using the previous definitions of user metadata, item metadata, and interaction metadata, we propose a model that incorporates all three categories into the training features in order to build these relationships. We not only teach the model to predict interaction metadata from user metadata, and item metadata but we aim to teach the model to use past knowledge of the interactions between them to predict future interaction metadata. We approach this process by investigating methodology that allows training on an extended feature vector including, for example, these three groups of metadata. During testing we may not have all three types of data and thus our model aims to include only the provided information to make predictions on the one missing. Our method non-trivially embeds the information gained from all three of these data types while only testing on a subset of features. In this we attempt to solve the problem of including valuable information during training but not during testing.

The main contributions of this thesis are neural network models based on COLANDER that incorporate user-item interaction data during training and an experimental study comparing the performance of the proposed model to each other and to the baseline model.

This document is organized into seven chapters. Chapter 2 discusses the background and related works required for understanding the methodology described in this thesis. Then, the methods we propose are introduced with their technical explanations in Chapter 3. We further describe the implementation of the architecture in Chapter 4 and how we evaluated the performance of the model through a series of research questions in Chapter 5. Chapter 6 discusses the results of those research questions with tables and discussion. Finally, Chapter 7 includes the conclusions and further work that can be done in the future that builds off of the contributions of this thesis.

Chapter 2

BACKGROUND AND RELATED WORKS

This thesis builds off of important contributions made by researchers in the field of predictive modeling, recommender systems, and neural networks. Section 2.1 covers some of the historical approaches to building recommender systems. Section 2.2 introduces neural networks and their fundamentals which are needed to further develop the work in this thesis. Finally, section 2.3 builds upon the material in Section 2.2 to cover some of the work done in current hybrid recommender approaches. Section 2.3 is postponed until the end of the chapter due to the dependence on neural networks which are discussed in Section 2.2. Section 2.4 discusses the Digital Democracy project as related to Nick Russo's work which provided us with a dataset for our validation experiments[27].

2.1 History of recommender systems

This section covers some of the historical and more modern approaches of creating recommender systems. Many modern recommendation systems are based on machine learning models and involve large data-sets that contain information on users, items, and their interactions. Originally, before the computing explosion that allowed for such models, recommendation systems primarily used the so called memory-based methods which computed recommendations directly from data without any model training. Faster compute has lead to the rise of new modeling techniques in clustering and classification. Furthermore, neural networks started to emerge at the forefront of big data analysis that involves recommendation systems. In addition to different approaches to constructing recommendations, recommendation systems also differ on what data is used in the process.

We illustrate two different approaches available in this field by comparing recommendation engines of two well known streaming platforms, Pandora and Last.Fm [1] [23]. Pandora's recommendations are based on the qualities of the music and specific features a song may have. Examples of features Pandora uses are the presence of a guitar solo or if the song has a lead male singer. On the other hand, Last.fm simply records what songs users have listened to. Similar users are identified by comparing these lists of songs. From these similar users new songs can be recommended and the process starts all over again. In essence the Last.fm approach is a collaborative filtering approach and the Pandora approach is one that includes meta-data about the songs themselves. Although both of these approaches are valid, some of the more recent work has been in hybrid systems where collaborative filtering approaches have been combined with meta-data approaches.

2.1.1 Collaborative Filtering

One of the most widely used methods under the recommender system umbrella is collaborative filtering. It is the set of algorithms that focus on identifying similar entities based on their preferences and generating recommendations using the interests of these similar entities [34]. This can best be described through an example. We can imagine a scenario where we have data on what movies many users watched. Some of these users might have very similar watching history and using this information we can provide recommendations to them based on what similar users have watched. In this way the users info is filtered to only include similar users [17]. Then out of those a recommendation can be made. This approach has been pushed to recommending

[1]https://www.pandora.com/, https://www.last.fm/

things like music and shopping items to name a few. Furthermore, the simplicity of this method is what limited the accuracy of the approach. It tends to not perform well in complicated tasks involving high dimensional features, and cannot make recommendations for items it has never seen before.

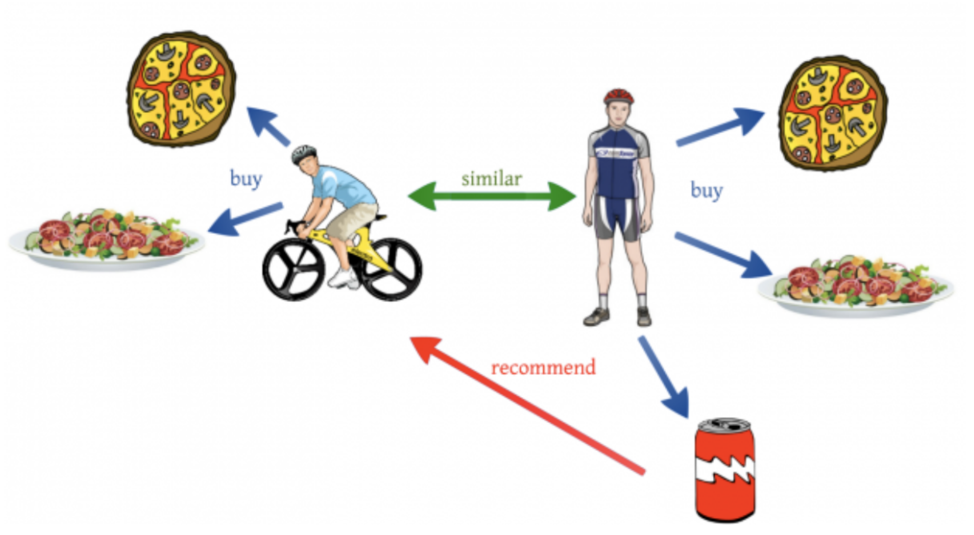


Figure 2.1: Collaborative filtering

Above, in Figure 2.1 is another example of the phenomenon where we match the soda as a possible item that the other boy may like. Another example is how Last.fm provided music recommendations as mentioned in Section 2.1. These are fairly standard examples that follow the main principles of collaborative filtering. These principles have evolved into more complex ideas.

Distance metrics such as Cosine Similarity and the Pearson Correlation can be used to identify similarity between users or items [25], [20]. Use of these metrics for similarity scores is often combined with the use of K Nearest Neighbors (KNN) to identify users or items most similar to the user/item for which the recommendation is constructed [16]. Some other more involved metrics have been developed as well [30]. Furthermore, more complicated methods such as Markov models can be used in combination with PCA to improve dimensionality [22] [28].

The field of collaborative filtering is constantly changing, with new methods being proposed. A recent work by Jun Ai et al. takes generic collaborative filtering to the next level [3]. This work focuses on creating a graph based architecture that is used to build relationships between items. In this way the authors use a metric known as "degree of centrality" which is meant to identify the inherent importance of the item to other items in the graph. Degree of centrality can be used to create metrics similar to the Pearson Coefficient above that can provide accurate depictions of relationships between items.

With the rise of neural networks to take over almost every modeling task as the new state of the art, collaborative filtering is no exception. Strub et al introduces a new kind of collaborative filtering that incorporates de-noising auto encoders into sparse collaborative filtering tasks to achieve state of the art performance [33]. The authors identified that for user interactions with items, the majority of the vector is usually empty because many users do not interact with the same items. Thus, they impose an auto-encoder approach that constructs a user vector similar to how an embedding layer would be created for an image auto-encoder [2] [14].

Although collaborative filtering is effective in its own right, most of the modern advances in recommender systems revolve around clustering and classification tasks that involve user and item metadata.

2.1.2 Clustering/Classification

Advances in compute lead to an increase in model based approaches being implemented for all sorts of tasks. Clustering techniques can help build recommender systems by clustering individuals into groups to identify similar individuals. One of

[2]Auto-encoders are a special type of neural network whose input and output are identical. The purpose of this symmetry is to create a compressed representation of the input through a set of weights that are manipulated through backpropagation

the first methods used is K-means clustering [18]. This involves a metric to be used, such as the Manhattan Distance, to find the distance between individuals based on their preferences [10]. The interesting thing about this method is one can identify the patterns behind why people are grouped together. These groups can be humanly explainable in certain situations. One such situation is if there is a subpopulation of users of a movie streaming service who enjoy watching sci-fi movies. K-Means might identify that the mathematical distance between ratings for these individuals is minimal and group them into one category that can be explained as the group of users who like sci-fi movies. Another example of a possible use case for K-means is identifying authors of manuscripts. By generating many features that describe the prose of manuscripts K-Means can identify groups of writing samples that have similar styles. These groups can represent different authors. There have been many more instances of useful applications surrounding clustering but classification techniques have seen more development in recent times.

Simple classification techniques such as KNN(K-Nearest Neighbors) have been useful in their own set of tasks [16]. Like K-means, KNN also incorporates a distance metric but it simply finds the nearest K neighbors to a user where K is a positive integer. The distance metrics used are identical to the ones that can be used for K-means due to the fact that both approaches build a matrix of metrics showing the mathematical distances between values in the dataset. This approach is fundamentally similar to collaborative filtering since it filters for individuals with features similar to others which can be inherently viewed as recommendations. A contrast with collaborative filtering however is that a different set of features is being used. The set of features for KNN can include the metadata associated with the items or users being compared. This inherent difference is what allows modeling tasks to outperform collaborative filtering in most instances.

Some of the more popular approaches within recommender system tasks and general classification tasks have been SVM'S and GBM's (Gradient Boosted Machines) [7], [13].

SVM's (Support Vector Machines) are a class of techniques that have been widely successful in classification tasks [8]. The main premise of what makes them successful it that they build a hyper plane, drawn from only a few important separating points, to split the data points into their corresponding classes. This hyper-plane is built by identifying the closest points from different classes and maximizing the distance between them. The distance is commonly referred to as the margin. Many other methods build separating hyper-planes but what makes SVM's special is that they only use a subset of points in the data space. This is due to the assumption that the easy to classify points will not be useful in generating a separating plane and that only the points from each class that are closest together are valuable. The points chosen to create this hyper-plane are known as support vectors. An important consideration is that a hyper-plane can be formed with any number of dimensions. If only two dimensions are considered this separation is a line and if three dimensions are considered it is a plane. This can be extrapolated to many more dimensions which is known as kerneling [7]. In Figure 2.2 on the left hand side we have a possible scenario where the separation between classes is non-trivial in two dimensional space. If we consider three dimensional space though a clear separation can be created as shown on the right hand side of the Figure.

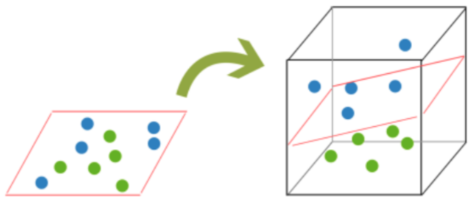


Figure 2.2: SVM separation example

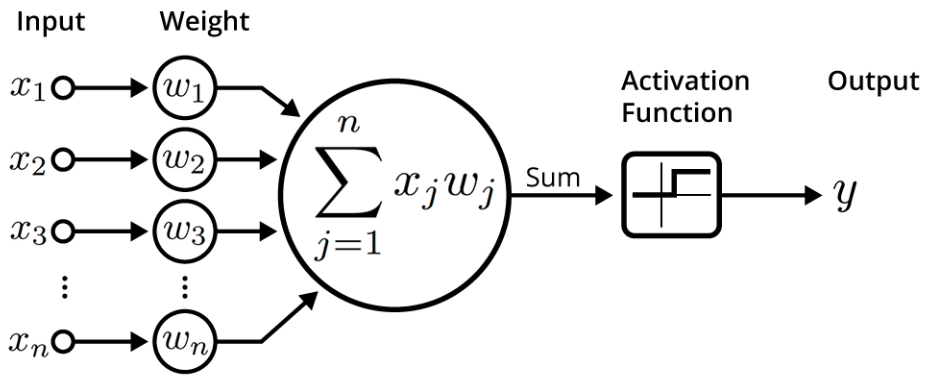
Another popular set of models for predictive modeling is known as GBM's (Gradient Boosted Machines) [13]. This set of algorithms fit under the idea of boosting. One of the first boosting algorithms developed was AdaBoost which pioneered the methodology for further work to be done [6]. AdaBoost relies on the fundamental idea of boosting which is that many weak classifiers can be used together to form a strong one. After one weak classifier is created, the data is weighted in such a way that difficult to classify observations are weighted more and the easy to classify observations less. This new data set is used to generate another weak classifier and the process is repeated for many iterations. Now, the predictions generated from each classifier are aggregated such that a majority prediction class is identified. Gradient boosting is built off of this original approach pioneered by AdaBoost but it differs in that weak learners are built off of optimizing a user defined loss function through gradient descent as supposed to pure classification accuracy. Consequent weak learners are added to the model in a way that lowers the error of this loss function. This fundamental idea of using many weak classifiers has been built up on with different approaches being used to build consecutive classifiers. Furthermore, the idea of boosting is under an umbrella of approaches known as ensemble methods, which are approaches that create many models and combine them in a way to create more accurate results [11].

Similarly to the other classification techniques mentioned, SVM's and GBM's can be used to create powerful recommender systems. Fortune et al proposed a method that does real time news recommendation based on users previous history on the site [12]. The authors used an SVM approach that predicted the top three categories of news the user would be interested in. By using SVM's they reached results that are better than other, more generic, collaborative filtering methods. GBM's have also seen similar success, where Volkovs et al won an ACM contest addressing the cold start problem in recommender systems [37]. The task was to provide recommendations for users based on user-job interactions from a career oriented social network.

Experimenting with both a GBM model and a deep neural network the authors found more success and faster training time with the GBM model.

2.2 Neural Networks

Neural networks are mathematical models that allow one to take a set of inputs and associated outputs and learn the relationships between the two [32]. An example of this is taking a picture of either a cat or a dog and being able to predict which one it is using the model. Neural networks learn from many training samples and identify patterns that differentiate the inputs to retrieve the proper outputs. The simplest of neural networks consists of an input layer, a hidden layer, and an output layer.



An illustration of an artificial neuron. Source: Becoming Human.

Figure 2.3: Artificial Neuron

In Figure 2.3 there is a visual example of a neuron. An input matrix defined by X is transformed through a dot product with a weight matrix defined by w as shown by the operation below which is copied from Figure 2.3.

$$\sum_{j=1}^{n} x_j w_j$$

After this linear transformation takes place an activation function is applied, such as a RelU or sigmoid as shown below by the symbol σ.

$$y_j = \sigma(\sum_{j=1}^{n} x_j w_j)$$

A ReLu function simply takes a value and if it is negative the value becomes zero, otherwise the value remains the same [24].

$$RelU : f(x) = max(0, x)$$

$$Sigmoid : f(x) = \frac{1}{1 + e^{-x}}$$

Many of these kinds of functions exist and are crucial to the success of neural networks. If the activation function is ignored, the neural network becomes a simple linear regression operation which is only able to identify linear relationships in the data unlike a neural network.

Multiple layers of these neurons can be connected to each other as shown in Figure 2.4 denoted by filled circles.

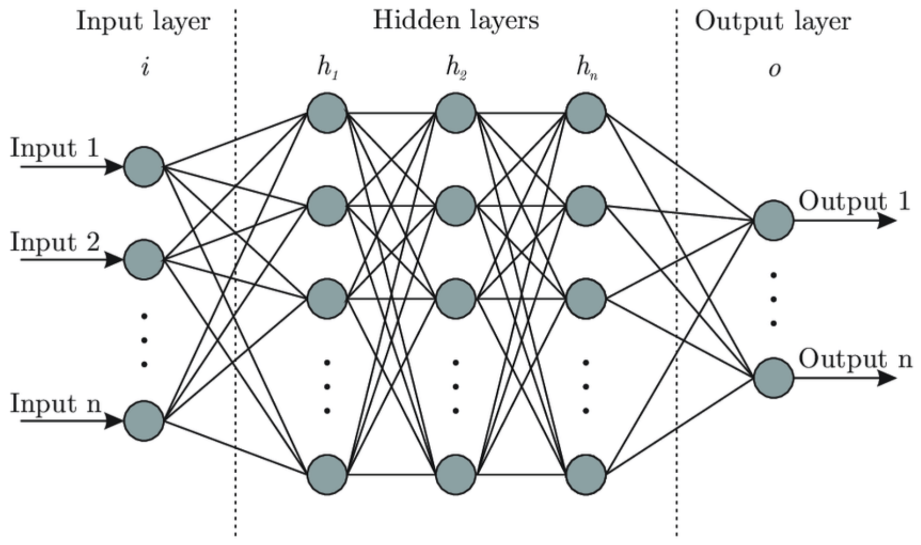


Figure 2.4: Multi-layer neural network

The outputs of one neuron are sent to the next neuron where each neuron in a given layer is usually connected to every neuron in the consecutive layer. Once the flow of the data hits the last layer the output prediction is generated. After this "forward propagation" step the network must learn from errors it made in its predictions. Loss functions such as MSE for regression, or Cross Entropy for classification are used to generate gradients that represent how far off the prediction is. These losses are generated by comparing the predicted output and the ground truth, whether it is a numerical value for regression or a class for classification. This loss is then propagated backwards through the network by a step known as "backpropagation" [14] which uses partial derivatives to update the weights of each neuron in the network. This process of forward and backpropogation occurs many times in order to train the network.

2.2.1 Convolutional Neural Networks

Convolutional neural networks (CNN's) are an important influence on the work of this paper. A CNN is a type of neural network that primarily focuses on image classification. They were originally developed in 1990 By Le-Cun [21]. Regular neural networks struggled in image recognition tasks as the number of parameters exploded dramatically with the size of input images and the number of layers needed to create a adequate model. The premise of a neural network is still captured by CNN's with training being done through forward propagation and backpropagation. Although similar in nature the addition of the convolutional layer and pooler layer are what make it so beneficial in tasks involving images or videos. Although many different types of convolutions exist nowadays, we are primarily focused on the basic convolution layer. In a CNN, the input is usually an image which can be thought of as a matrix of size M x N x 1 in the case of a black and white image. This matrix is then processed through the use of a filter to create a output map as shown below in Figure 2.5.

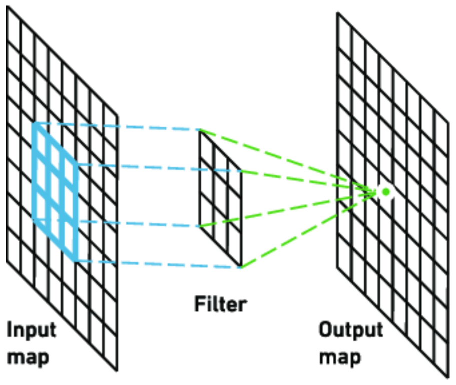


Figure 2.5: Convolutional layer

In Figure 2.5 the filter "walks" through the image and a dot product is performed with a identically sized subsection of the original image in order to produce the filtered output. This process allows the network to identify a filter pattern anywhere in the image. The dot product operation is what is known as convolution and is crucial to understanding the work of this thesis. Filters can identify features independently of location in the image which is what makes them so powerful. Convolutional layers can occur back to back in order to identify more specific features as the image gets smaller after each step of the network. For example, if our network is trying to classify whether an image is a dog or a cat, we might see the first layer identify head shapes or body position while the farther layers may find whiskers or ears of specific shapes that are pertinent to a dog or cat. CNN's allow features such as the mentioned ears or whiskers to be identified anywhere in the image due to the way filters are pushed through each section of the image. In Figure 2.6 a simple example is shown. The features a network has identified are depicted in green which include the whiskers, ears, and tail. On the right hand side of the Figure the representation of the cat is shown which is what a possible network captures and uses to distinguish from a dog in our previous example.

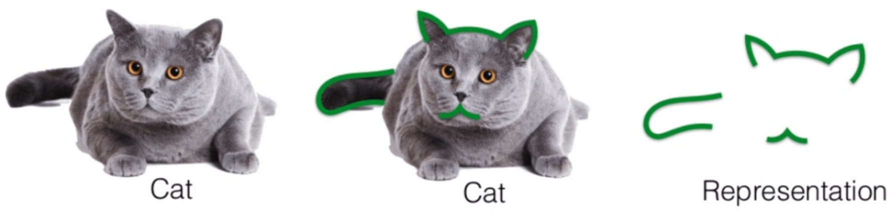


Figure 2.6: Convolutional example using cats

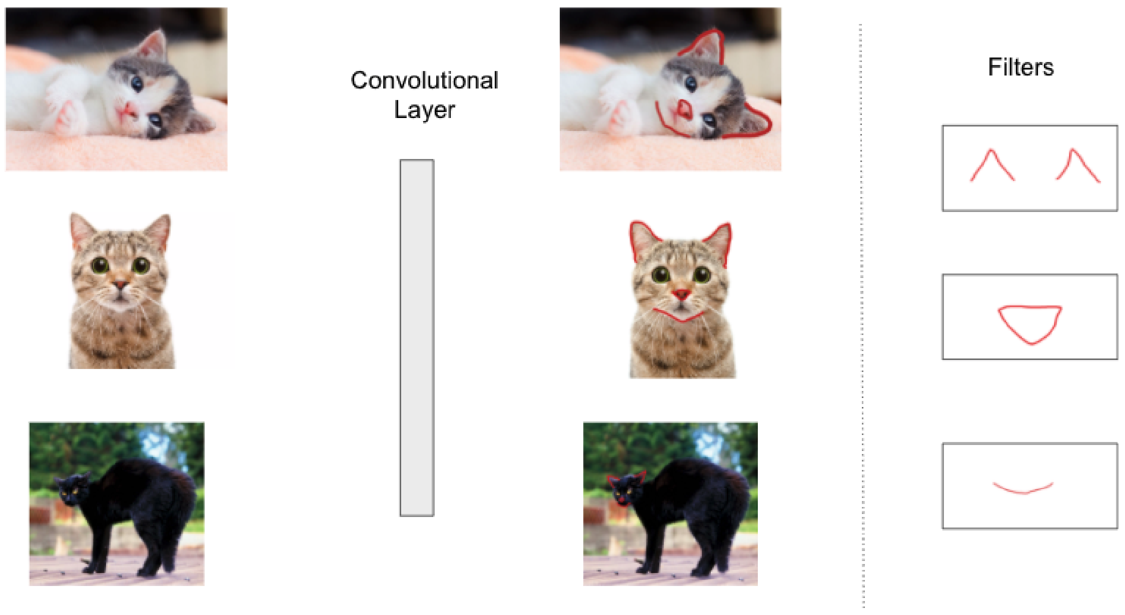


Figure 2.7: Feature identification and classification

In Figure 2.7 there are multiple images of cats, each containing ears, a nose, and chin. After these images are pushed through the convolutional layer, assuming the network has been trained, filters will be propagated through the image. In our oversimplified example, these filters are ears, a nose, and a chin. Please note that in a real convolutional layer it is highly unlikely that filters will be this defined. As shown in the three images in the second column, the convolutional layer will identify these features anywhere in the image.

The filters used in this convolution step are trained through backpropagation similarly to a regular neural network [15]. That is to say that the updating of weights can be calculated using partial derivative calculations with respect to the loss function.

Although the work of this thesis does not directly use convolutional layers it does incorporate a similar idea of performing a convolution between matrices. This is to say that the idea of local feature recognition can be applied to domains outside of images, where similar patters to the mentioned whiskers, tail, or ears for cats may exist. These patterns can be thought of as a feature that is detached from the notion of where in the input matrix it shows up such as the presence of whiskers is.

2.3 Hybrid of modeling and collaborative filtering approaches

A hybrid recommender system is one that incorporates both modeling and collaborative filtering approaches. This is to say that not only is a user-item matrix used but also content-based metadata. When we think of hybrid systems, a good example is Amazon. There are many individuals with similar shopping habits as a specific user and thus Amazon's collaborative filtering can easily identify what that user likes to shop for. Furthermore, by including modeling approaches that incorporate the metadata between items and users as well as collaborative filtering scoring, Amazon can provide more unique recommendations to individuals. Another example is Netflix and its movie recommendation models. Using only the pertinent data to predictions can be very useful. When recommending horror movies it makes sense to only look at the horror movie watching history of the users in order to match users with similar taste.

Recommendation systems built on collaborative filtering suffer from the so called "cold start" problem - inability to recommend newly added items due to lack of score information for them [31]. When not much data is available on individuals and their interactions with items, those items will be incorrectly recommended to individuals if at all. Thus a combination of collaborative filtering and modeling is critical.

There exist a few different methods of combining the two approaches. A survey on hybrid recommender system models describes a few of these methods that can be used [35]. One such approach simply performs collaborative filtering and content-based modeling in parallel and then aggregates the results in some way. An example of this is if a list of recommended movies is generated for a user from collaborative filtering and a similar list from content-based modeling then the movies that appear in both lists are chosen as the final recommendations.

Another method for combining the approaches is to generate data artifacts that are representative of both collaborative filtering and content metadata and use them in conjunction to generate a model. An example of this method in action is the "Deep and Wide network" [5]. This network can be thought of as two different networks working together to create a recommendation. For example, the wide part of the network can be used to capture the matrix of user-movie ratings from Netflix through a simple linear model. On the other hand, the deep side of the network can be used for the large set of content features describing the specifics of each movie and the user metadata. The value of splitting the two data artifacts is that the network can "memorize" the user-item interactions while "generalize" on how a user would react to an unseen movie through the deep network.

2.4 Related Work

This thesis grew out of the work performed as part of Cal Poly's Digital Democracy project [27]. This project, undertaken by faculty and students from California Polytechnic State University, attempts to democratize the information captured in legislative hearings. These hearings consist of committee members discussing bills and whether or not they should be passed. The project is primarily focused on transcribing these committee meetings and providing users the ability to quickly identify what speakers were present and their contributions to the floor. These contributions can be monologues or votes, among many others. Furthermore, things like changes to the bill or authorship are also recorded for each bill through the video metric pipelines built by the digital democracy team.

2.4.1 DISH

Previously, a fellow Cal Poly student, Nick Russo attempted to solve a problem within a similar space through his thesis [27]. Although his work is primarily focused on the embeddings of a neural network, the network architecture used was a motivation for this work. His work uses an architecture that consists of a similar framework of items, users, and their interactions. Though in the work it is in the context of legislators, bill proceedings, and their interactions. His main focus is on the embeddings created between the items and users that can allow one to identify the feelings a legislator may have towards a certain bill. This kind of application proves to be a great example of something that could benefit from the ideas discussed in this thesis. The network presented in his work is designed to address the same problem as the work in this thesis though it does not actively use the information from the interactions while predicting future interactions using only the legislators and bill proceedings. This important distinction is what we believe gives our network a benefit in terms of prediction and the incorporation of prior knowledge of interactions into the model.

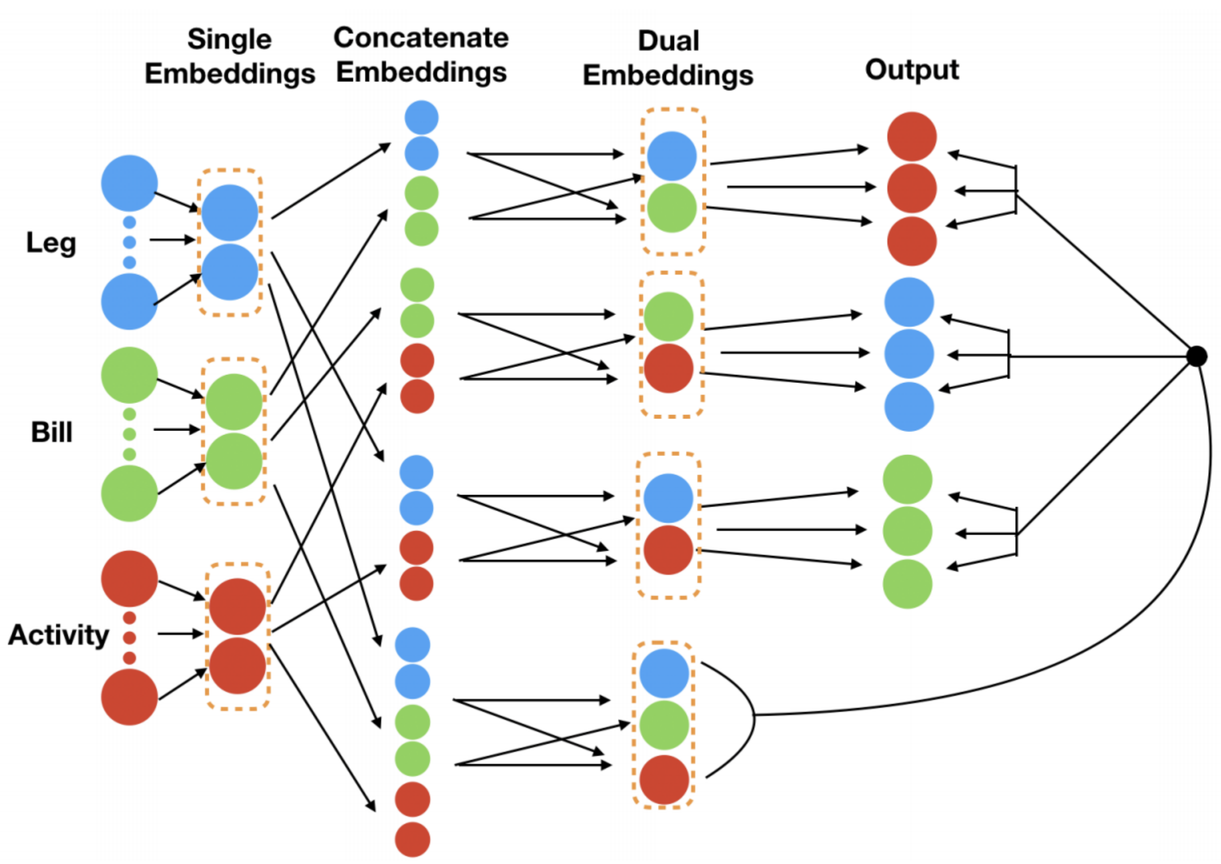


Figure 2.8: DISH network

The DiSH neural network architecture presented in Figure 2.8 was designed to take advantage of user-item interactions (in case of DiSH: legislator's activity w.r.t. a bill) during the training process. However, the version of the DiSH neural network that is used for predicting does not contain any information (i.e., weights) influenced by those user-item interactions. The COLANDER model proposed in this thesis seeks to alter the DiSH neural network architecture in order to enable the prediction neural network to include information from the training neural network about the user-item interactions.

2.5 Data

The dataset used for evaluating the network described in this thesis is generated by similar means to the work described in DiSH [27]. Using the Digital Democracy projects, database (DDDB) several sources of data are extracted into one dataset that fits the format necessary for this thesis. The database and the kinds of information it contains is described in the DiSH paper [27].

Recalling the ideas of user metadata, item metadata, and interaction metadata, we use legislator information, bill information, and activity information respectively. Here legislator information contains data like party, district, and committee membership among others. Bill information represents anything related to a specific bill such as bill status, number of senate hearings, and total ayes to name a few. Activity information represents how a legislator interacted with a specific bill. These features include data such as speaking time and number of utterances. Through these features, approximately 83000 rows are produced by the data generation and around 300 features across the three types of data points.

For the purposes of our experiments, we identify the variable that will be denoted as the dependent variable to be whether or not the legislator voted with their party. This is the variable that we measured using predicted accuracy in order to establish model performance.

2.5.1 Legislator features

A total of 10 different features are represented in the legislator feature set extracted from (DDDB). These represent the user metadata that is used in our network.

Legislator features			
Attribute	Description	Data Type	Number of Columns
Party	Whether the legislator is a Democrat or Republican,	One-hot encoded	2
House	Whether the legislator is in the Senate or Assembly	One-hot encoded	2
District	The Assembly or District number.	One-hot encoded vectors	120
Committee Membership	The number of committees a legislator served on a a Chair, Co-chair, Vice-Chair or Member.	Numeric	4
Ayes per bill	The average of ayes per bill for the legislator.	Numeric	1
Noes per bill	The average number of noes per bill for the legislator.	Numeric	1
Abstinence's per bill	The average number of abstinence's per bill for the legislator.	Numeric	1
Total authorship	Total number of bills that the legislator authored.	Numeric	1

Table 2.1: Legislator features

2.5.2 Bill features

Bill features represent the information of the bill and what the status of it is. The bill features used are shown in Table 2.2.

Legislator features			
Attribute	Description	Data Type	Number of Columns
Bill Status	The final status of the bill.	One-hot encoded vector	12
Number of versions	The number of versions the bill went through	Numeric	1
Number of hearings	The number of hearings in the assembly, senate, or joint.	Numeric	3
Number of votes	Number of democratic/republican legislators that abstained, voted aye, or voted Noe the bill.	One-hot encoded vector	6

Table 2.2: Bill features

2.5.3 Interaction features

Interaction features represent how the legislator interacted with a given bill throughout the proceedings. This is shown in Table: 2.3

Legislator features			
Attribute	Description	Data Type	Number of Columns
Author	Whether or not the legislator was an author on the bill.	One-hot encoded vector	1
Average Utterance time	The average length of a legislator's utterances on a bill.	Numeric	1
Average Bill Discussion Time	The Average time a legislator spent speaking at each bill discussion for a bill.	Numeric	1
Number of Utterances	The total number of utterances the legislator made on the bill	Numeric	1
Time Spoken	The total time the legislator spent speaking on the bill	Numeric	1
Voted with Party	Whether or not the legislator voted with their respective party on the bill	One-hot-vector	1

Table 2.3: Interaction features

Chapter 3

METHODS

In the previous chapters we referenced the notions of user metadata, item metadata, and interaction metadata. User metadata is any information available on the user. Item metadata is information on the entity and interaction metadata is data on how users and the items interact. We hinted at the difficulty of incorporating interaction metadata into a prediction model during training because the data will not be available during testing. An example of this is if a model is attempting to generate predictions on how a user would react to a Netflix movie. This could be a rating or review. The user metadata in this example can be age, zip code, gender, and time spent watching action movies. Item metadata can be genre, length, title, actors, and director. Interaction metadata can be time spent watching the movie, number of pauses, rating given to the movie, movie start time, and times watched. We might have information on the user and the movie but since the user has never seen the movie we would not know how many hours they spent watching it or at what time they began watching it; this is all information generated after one has seen it. On the other hand, we would have this information for other movies that the user has already seen.

We propose a method that uses available user-item interaction data during the recommendation model training process. To refer back to the Netflix example, we would be incorporating the knowledge of relationships between the interaction metadata for movies the user has already seen. These relationships will be ingrained in the model during training, but during testing we will be only using the user metadata along with item metadata. Incorporating relationships between interaction metadata and other types of metadata is crucial in building a powerful predictive network.

Let us assume that our dependent variable is movie rating. The number of times a user paused the movie, which is considered interaction metadata, can be related to the movie rating. Furthermore, the number of times a user paused the movie can also be influenced by the genre of the movie, or the length of the movie. In essence this shows that there could be relationships between the movie metadata and the interaction metadata.

3.1 Technical solution

Our goal is to design a neural network architecture that uses three types of data: user metadata, item metadata, and user-item interaction metadata during the model training stage, but can make predictions when only user and item metadata, but not user-item interaction metadata is available. To achieve this effect, our solution uses two neural network architectures, depicted respectively in Figures 3.1. and 3.2. The first neural network architecture ,Figure 3.1, is employed during the training stage. The second neural network architecture, Figure 3.2, is a subset of the first that uses only user and item metadata to make predictions. The key feature of this network is that some of the weights in the first layer of the network depend on the user-item interaction inputs in the training network - and thus, user-item interaction metadata influences the prediction network. The detailed description of the two architectures is below.

Figure 3.1 is a visual representation depicting a method that can perform the aforementioned training and testing with different amounts of input features. Since we have three different types of data (User metadata, item metadata, and interaction metadata), we propose a network that consists of three individual sections. Each section takes two of the three possible data types and creates a prediction on the third. As we can see the user data is denoted in red, item data in blue, and interaction data in yellow. The light brown circle denotes the dependent variable that needs to be predicted. This variable is part of the interaction metadata. Each section is trained concurrently in a specific manner that will be described in the next few paragraphs.

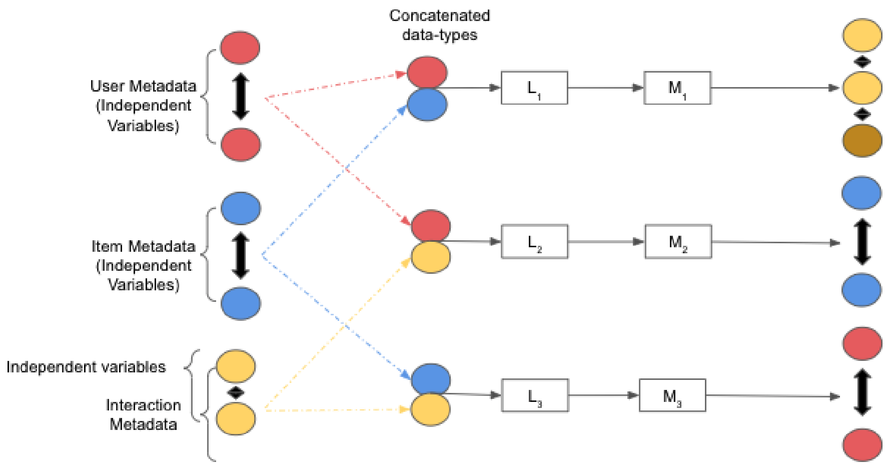


Figure 3.1: Training network

In Figure 3.2 we can see how the network would look like for prediction on interaction metadata. As shown, only user metadata and item metadata is used as the model input. An explanation of how the unique properties of our model training procedure incorporate the knowledge of prior relationships between the three types of metadata is in the next few paragraphs.

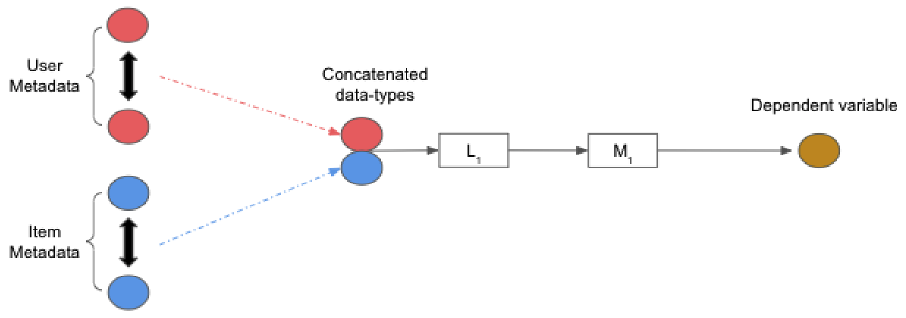


Figure 3.2: Prediction network (need to implement this)

The first step of each section in Figure 3.1 is to concatenate the two data-types into one vector/matrix that will be propagated through the network. This propagation is depicted in Figure 3.1 with the colored dotted lines indicating that a copy of each type of the data is used in two sections of the network. For the interaction metadata, we only include the independent variables for training and we predict the independent variables along with the dependent variable. The reason for this is that we want to identify the relationships between the interaction metadata and other types of metadata without biasing the results of prediction. The concatenated matrix is then sent into independent fully-connected layers L_1, L_2, or L_3 respectively as shown in Figure 3.1. An important consideration to make for the usage of this network is that layers L_1, L_2, and L_3 must be the same size. That is to say that they consist of the same number of neurons. This is because of a non-intuitive training step that takes place, which will be discussed in the next few paragraphs. After the data is sent to these fully-connected layers, the consecutive layers, defined by M_1, M_2, and M_3 can be different for each subsection of the network. Finally the output of each subsection in the network is the third data-type that was not trained on in that subsection.

At the topmost section of the network in Figure 3.1, the output is the interaction metadata, while the input, shown by the dotted line, is the user metadata and item metadata. No interaction metadata is directly used in training this section of the network. We propose a intermediary step, inspired by convolution, after a user-defined number of batches has been trained on for each of the three subsections of the network.

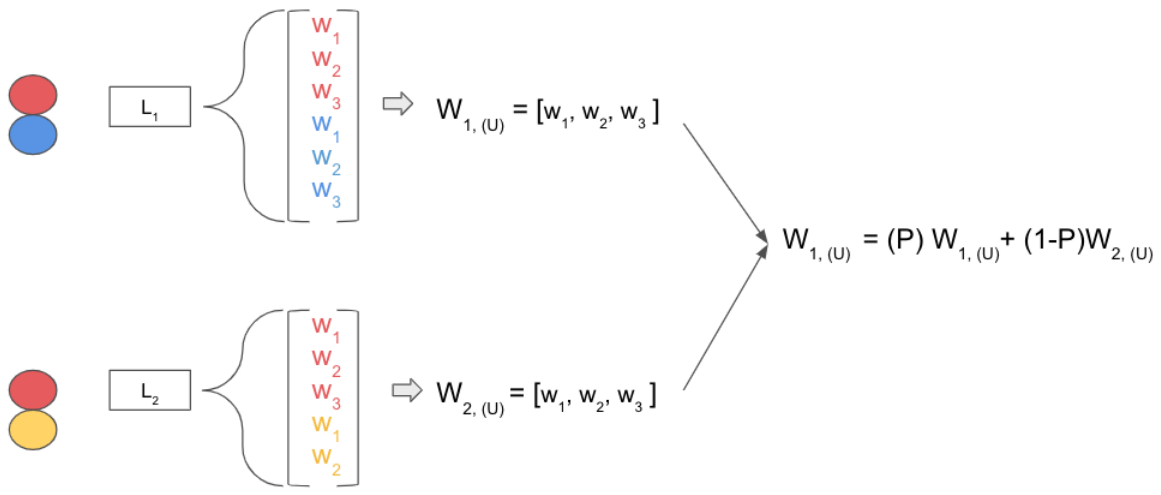


Figure 3.3: Training batch convolution

Figure 3.3 shows our approach to adjusting the weights in the first layer of each component of the network through a simple example involving the first neuron of L_1 and L_2. Once a user-defined number of batches of data has been sent through the network in a training loop ending with back-propagation, each of the weights in the first layer are convolved with their mirrored version in the other subsections of the network. In the figure we can see two rows, where the first row is from the network involving L_1, which is used to predict interaction metadata and L_2 which is used to predict item metadata. Note that both of these sections involve user metadata as their features. In this example we are concerned with updating the weights associated with the user metadata features in the first neuron only. $W_{1,(U)}$ represents the weights associated with the first subsection for the user metadata features, while $W_{2,(U)}$ represents the weights associated with the second subsection for the user metadata features. We convolve these two sets of weights by multiplying them by a predefined constant, P and adding the two sets of weights, element-wise. P represents a weighting factor for how much to weight the original weights in the neuron in comparison to the ones in from the other section of the network. Although the example of this is only for a single neuron and single data-type, the same operations are used to update the weights for the other data-types and other neurons. The general formula can be written as:

$$W_{UpdatedLayer} = P \cdot W_{ConvolvingLayer} + (1 - P) \cdot W_{OriginalLayer} \tag{3.1}$$

All of the weights are updated using this method, where the original layer weights and convolved layer weights are combined using a weighted average, similar to how generic convolution works.

In the manner described in the previous paragraphs we have a network that incorporates the weights from subsections of it that do use interaction metadata in the network that does not. This allows us to teach the network relationships involving interaction metadata for predictions only using user metadata and item metadata.

Chapter 4

IMPLEMENTATION

We implemented the network described in Chapter 3 using Tensorflow [2] in Python. While Chapter 3 described the training architecture, shown in Figure 3.1, as a single network, we chose to implement it as three separate neural networks "connected" to each other via explicit weight transfer for the first layer. Figure 4.1 shows visually how this looks like.

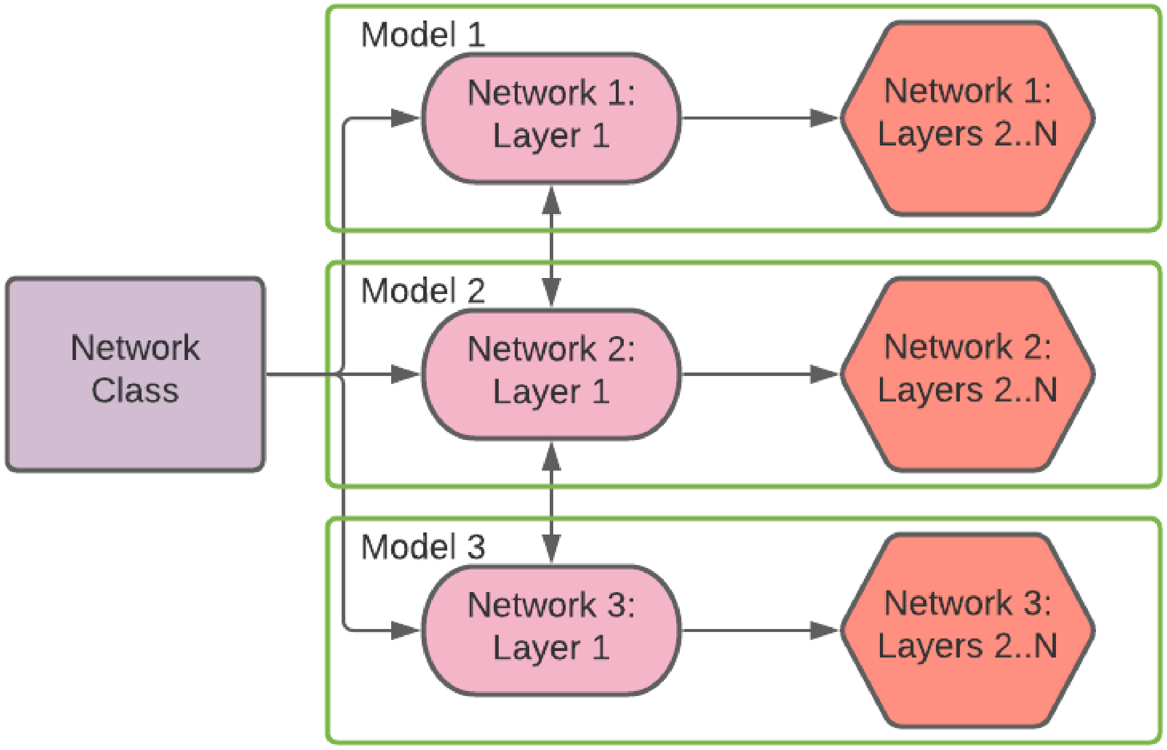


Figure 4.1: Network class

The network class is a wrapper python class for the three independent networks, which allows a user to easily interact with all three models. The independent models

```
def __init__(
    self,
    shapes,
    optimizers=[Adam(), Adam(), Adam()],
    losses=['mse','mse','mse'],
    metrics=[['mse'],['mse'],['mse']],
    xy_layers=[],
    xz_layers=[],
    yz_layers=[],
    num_nodes = 25,
    w=[0.5,0.5]
):
```

Figure 4.2: Caption

are shown as green rectangles where they each contain a first layer and layers two through n. The first layer of each network is separated visually to show that each model is connected through it, as shown by the arrows connecting them. This connection exists to perform the convolution step defined in the previous section. The rest of the networks can be defined as specified by the user.

Each separate model combines, as independent variables (inputs) features of two of our three types, and is used to predict the feature or features of the third type. Recall, that the input data to our models is split into three groups: user data, item data, and interaction data.Let X= $x_1, ...x_k$ denote all user metadata, Y= $y_1, ..., Y_m$ denote all item metadata, and Z = $z_1, ..., z_p$ denote all interaction data, where $z_1, ..z_{p-1}$ are the independent variables and z_p is the dependent variable. From this we can define Model 1 from 4.1 as being the network that uses X and Y to predict Z, Model 2 uses Y and Z to predict X, and Model 3 uses X and Z to predict Y.

The parameters to the network class are shown below:

- **shapes**: A list with three values depicting the number of features in the user metadata, item metadata, and interaction metadata respectively.

- **optimizers**: The optimizers to use for each network (defaulted to Adam [19] for all three).
- **losses**: The loss functions that each network should use for training (defaulted to MSE [29] for all three)
- **metrics**: The metrics to show while training the networks (defaulted to MSE).
- **xy-layers**: A list of tensorflow-keras layer objects that are used for building the XY-Z model (Model 1).
- **xz-layers**: A list of tensorflow-keras layer objects that are used for building the XZ-Y model (Model 2).
- **yz-layers**: A list of tensorflow-keras layer objects that are used for building the YZ-X model (Model 3).
- **num-nodes**: This represents the number of nodes to include in each of the networks first layer, since they must all be equal for our implementation.
- **weights**: The weighting percentage of the original weights to the new weights when convolving as described in the methodology section. This is described in Equation 3.1
- **shuffle train**: Whether or not to shuffle the order of the training networks.

With these parameters a user can customize their network as they see fit for their specific use case.

Due to the nature of the network, a user will likely want to retrieve the individual models and perform inference on a test set of data to evaluate the performance. For this reason, three functions have been created to support the user that retrieve the respective a tensorflow model for the user.

```
funcs = [
    [self._convolve_model_3, 3],
    [self._convolve_model_2, 2],
    [self._convolve_model_1, 1],
]
if epoch == total_epochs - 1:
    funcs = [
        [self._convolve_model_3, 3],
        [self._convolve_model_2, 2],
    ]

if self.shuffle_train is True:
    random.shuffle(funcs)
for func, num in funcs:
    if num == 3:
        loss1, acc1 = self.model_m3.train_on_batch([batches_X[batch],
        batches_Z[batch]], batches_Y[batch])
    if num == 2:
        loss2, acc2 = self.model_m2.train_on_batch([batches_Y[batch],
        batches_Z[batch]], batches_X[batch])
    if num == 1:
        loss3, acc3 = self.model_m1.train_on_batch([batches_X[batch],
        batches_Y[batch]], batches_Z[batch][:,-1])
    func(batch, batches_X, batches_Y, batches_Z)

if epoch == total_epochs - 1:
    loss3, acc3 = self.model_m1.train_on_batch([batches_X[batch],
    batches_Y[batch]], batches_Z[batch][:,-1])
    self._convolve_model_1(batch, batches_X, batches_Y, batches_Z)
```

Figure 4.3: Code used for convolving weights

get-modelZ()	Returns the model that uses X,Y to predict Z
get-modelX()	Returns the model that uses Y, Z to predict X
get-modelY()	Returns the model that uses X, Z to predict Y.

Table 4.1: Sub-model API

In order to convolve the weights across the three sub-networks, training is done at a batch level across all three networks. The steps are as follows. First the three networks are trained on a batch of data. After this, the network weights are convolved as shown in Figure 3.3. An important consideration is that the order of convolving can happen in a predefined manner or in a random manner. A predefined order could be M1,M3,M2. Here, the weights from model one are convolved onto models three and two, and then the weights from model three are convolved onto model one and two. Lastly, a similar operation occurs for model 2. If this step is to be random, then after each batch, a random ordering of convolving takes place of which (M1,M2,M3), (M1,M3,M2), and (M2,M1,M3) are a few of the possible choices. This step is shown in more detail in Figure 4.3:

The weights are convolved using randomness because if there was a rigid order the model that is convolved first every time would never end the training batch with trained weights, only flipped weights. This process creates instability in the model at times after batches due to the randomness. To counteract the instability, and make sure the dependent variable network is properly trained, in the last epoch, training occurs in the order of M3,M2,M1 or M2,M3,M1.

We produced two variants of the network implementation. In the first variant, the network M1 had only the dependent variable z_p as the target. In the second variant, all interaction metadata, Z, was used as target variables in M1. Chapter 5 discusses the comparison between these two variants, and Chapter 6 documents the results of this comparison.

Chapter 5

EVALUATION

This chapter describes the results of the experiments in Chapter 5. Section 5.1 discusses the research questions investigated, Section 5.2 discusses the experimental design, and Section 5.3 discusses the measures collected and the results.

5.1 Research questions

To properly evaluate the proposed network, we propose a set of research questions that we aim to answer. We first need to define two models:

M.1. Model 1: A model in which all of the interaction metadata is predicted on (dependent and independent variables) as shown in Figure 3.1

M.2. Model 2: A model in which only the dependent variable is predicted on as shown in Figure 5.1

Recall, that the dependent variable is whether or not a legislator voted with their party on the bill.

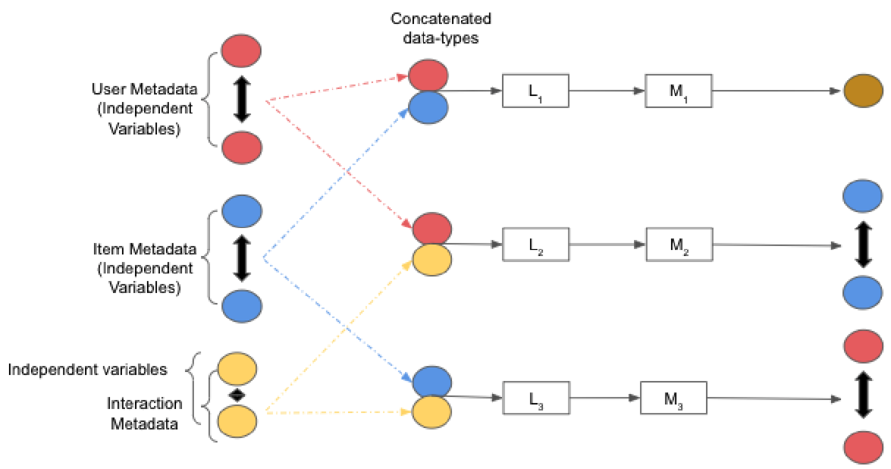


Figure 5.1: Dependent variable network

Using these two definitions we can specify research questions.

1. Which sets of parameters produce the best results for Model 1 and Model 2?
    (a) We can generate the best set of parameters for each Model.
    (b) We can compare models in performance and architecture.
2. Does Model 1 and Model 2 perform better than a baseline network using item metadata and user metadata to predict interaction metadata.

5.2 Experimental design

As described in Section 5.1, we are investigating two different research questions. These two questions are defined in more detail below and an explanation of how they are performed is described. Model 1 and Model 2 are used as defined by **M.1.** and **M.2.** respectively.

5.2.1 Parameter search

Model 1 and Model 2 are both neural networks and thus they require to be tuned to have strong predictive results. In order to properly evaluate the network and find the best parameters we test many different sets of parameters to evaluate how individual changes affect the network as a whole. For example, in order to test different sizes of layers in the network, we must keep the batch size, convolution operation, and the rest of the parameters constant. We use an F-1 score, precision, recall, and accuracy in these experiments to evaluate the performance of the model with changes to the parameters [36]. These metrics are quite commonly used when dealing with a categorical dependent variable.

In terms of the dependent variable we are using, whether or not the legislator voted with their party, accuracy is simply a measure of how often the model correctly predicts the result. Interpretations of precision and recall in terms of our dependent variable are included as well in equation 5.2 and 5.3 respectively.

$$\textbf{Accuracy} = \frac{Number\ of\ correctly\ predicted\ classes}{Total\ number\ of\ predictions\ made} \tag{5.1}$$

- **True Positive** = Number of legislators predicted to vote with party and actually did.
- **False Positive** = Number of legislators predicted to vote with party but did not.
- **False Negative** = Number of legislators predicted to not vote with party but did.

$$\textbf{Precision} = \frac{True\ Positive}{True\ Positive\ + False\ Positive} \tag{5.2}$$

$$\textbf{Recall} = \frac{True\ Positive}{True\ Positive\ + False\ Negative} \tag{5.3}$$

$$\textbf{F-1 Score} = 2\frac{Precision \cdot Recall}{Precision + Recall} \tag{5.4}$$

These metrics are evaluated across many different model variations that vary with respect to their parameters. The parameters being tested are, number of layers, layer sizes, initial layer size, weighted convolution, convolution frequency, batch size, learning rate, and optimizers. The experiments fix the rest of the parameters except for the ones being tested. Some of the parameters being tested are fairly intuitive but weighted convolution and convolution frequency are specific to this network design. Different weighting schemas can be used in the convolution such as 50/50, or 60/40 that allow a given sub-model to keep some of it's original information while incorporating the new information. This process refers to the formula described in 3.1.

Convolution frequency is the idea of convolving only every set number of batches. This might be beneficial in allowing the network to learn its primary task using its own weights from a few batches before convolving weights onto it. In essence, this can allow the network to calibrate the effects of convolving weights.

Furthermore, since Model 1 predicts on all of the interaction metadata, an MSE loss is used. The quantitative prediction for the dependent variable is then transformed into a binary (0 or 1) prediction for the dependent variable, which is inherently qualitative. In order to compare Model 1 and Model 2 a similar mechanism is built for Model 2 but since Model 2 only predicts the dependent variable, a categorical loss function is also used. This loss function is Binary Cross-Entropy, which is a standard loss function used for a binary prediction problem [9].

The data that is fed into the network is normalized into a 0-1 scale. What is atypical about this network design is that we are predicting each type of metadata from the other two types. That is to say we need to include both the normalized data set and the un-normalized set because we want the network to use the normalized data to predict the un-normalized metadata. For example, we can examine the part of the model that uses user metadata and item metadata to predict interaction metadata. In this case user metadata and item metadata is normalized but interaction metadata is not. In another section of the model the interaction metadata may be normalized but one of the other two types is not since it is being predicted.

5.2.2 Baseline Comparison

In order to compare the network to a baseline, a simple feed-forward network using user metadata and item metadata is used to predict the interaction metadata as mentioned in Section 5.1. The network is shown in Figure 5.2.

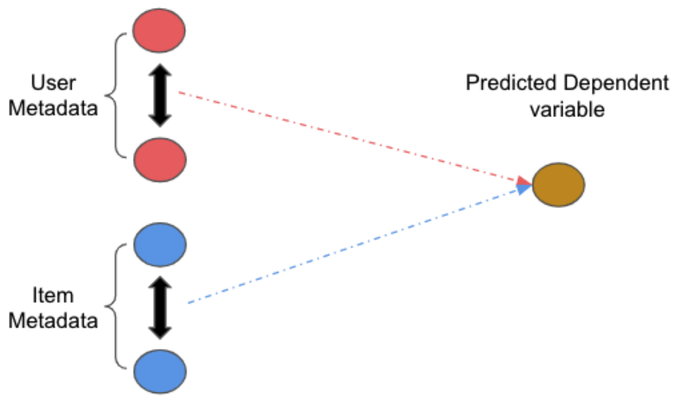


Figure 5.2: Baseline network

The arrows shown in Figure 5.2 can be thought of as neural network hidden layers that are used to predict the dependent variable.

A similar process to Model 2 occurs with the baseline model, where we examine using both an MSE loss and Binary Cross Entropy loss in order to compare to Model 1.

Chapter 6

RESULTS

When comparing Model 1 (**M.1.**) and Model 2 (**M.2.**) with their best performing models, Model 2 outperforms Model 1 across the board with many different iterations of models examined.

This chapter addresses the results of the experiments introduced in Chapter 5. Section 6.1 shows the results of the parameter searches related to Model 1 and Model 2. Section 6.2 performs a similar parameter search for a baseline model which is then compared to the performance of Model's 1 and 2 in Section 6.3.

6.1 Parameter search

We ran a grid search for all of the proposed models in order to identify the best performing one with respect to accuracy, F-1 score, precision, and recall. Section 6.1.1 describes this process for Model 1, and Section 6.1.2 describes a similar analysis for Model 2.

6.1.1 Model 1: All interaction metadata

For Model 1 experiments could only be done using MSE, which is then transformed into categories during inference. This is due to the fact that we are predicting all of the interaction metadata at once and thus we treated the dependent variable as if it was quantitative.

If a quantitative prediction is greater than 1 we replace it with a 1, that the legislator voted with their party on the given bill. Also, if the prediction is less than zero we replace it with a zero, that the legislator did not vote with their party on the given bill. Finally, for predictions between zero and one, a simple rounding takes place where values greater than 0.5 are set to one and values less than 0.5 are set to zero.

The model was shown to be quite unstable throughout the grid search with the MSE loss fluctuating between large numbers. Figure 6.2 shows the training loss through the batches. Note, that this loss is for all of the interaction metadata but we would expect this to be much lower than it is.

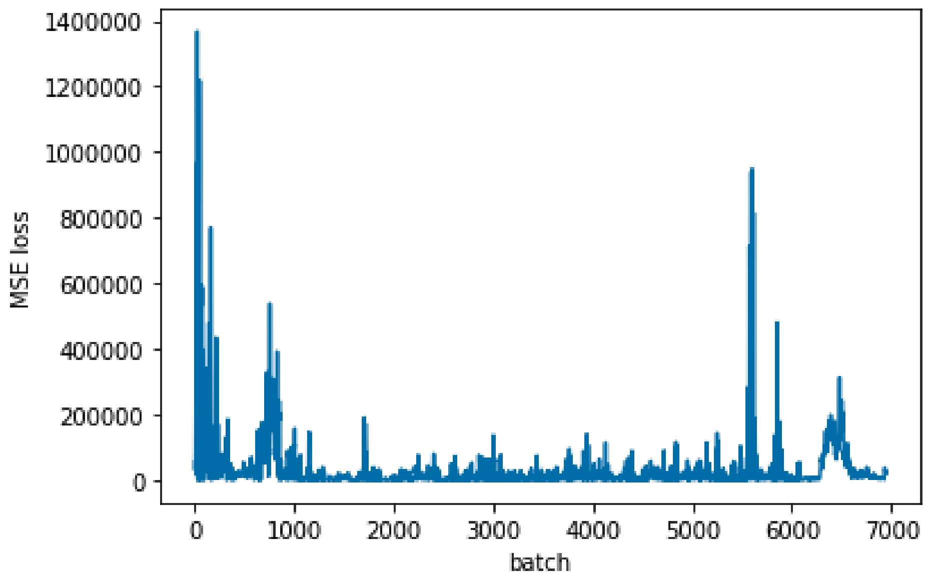


Figure 6.1: MSE loss across batches for Model 1: Best Model

A table depicting the metrics for the best model is shown below in Table 6.1.

Metric	Value
Accuracy	.667
Precision	.789
Recall	.664
F-1 Score	.72

Table 6.1: Evaluation metrics for Model 1

		Predicted	
		Not with party	With party
True	Not with party	6,496	3,142
	With party	5,968	11,786

Table 6.2: Confusion matrix for Model 1

```
xy_layers: [keras.layers.Dense(256, activation='relu'),
            keras.layers.Dense(128, activation='relu'),
            keras.layers.Dense(64, activation='relu'),
            keras.layers.Dense(32, activation='relu')]
xy_layers: [keras.layers.Dense(256, activation='relu'),
            keras.layers.Dense(128, activation='relu'),
            keras.layers.Dense(64, activation='relu'),
            keras.layers.Dense(32, activation='relu')]
yz_layers: [keras.layers.Dense(256, activation='relu'),
            keras.layers.Dense(128, activation='relu'),
            keras.layers.Dense(64, activation='relu'),
            keras.layers.Dense(32, activation='relu')]
num_nodes: 512
w: [0.3,0.7]
batch size: 64
epochs = 8
```

Shown above in Figure 6.2 and Table 6.1 we can see that the model does a decent job on predicting whether someone actually voted with the party but it does not do too well at predicting whether someone did not vote with their party. Furthermore this model was quite unstable through its training process across many of the other iterations of models attempted using all of the predictors. The weights parameter chosen is [0.3, 0.7] which means less of the original weights is being used after each convolution step than the convolved weights.

6.1.2 Model 2

Model 2, the model that does not predict on the entirety of the interaction metadata but only on the dependent variable, showed better results than Model 1. Also, since we are only predicting the dependent variable, we were able to treat it as as a categorical variable as well as a quantitative one to compare to the results of Model 1.

6.1.2.1 Quantitative dependent variable

:

Although fluctuations do occur across the loss curves for this Model, the loss curves do appear to be much more stable than the ones shown for Model 1.

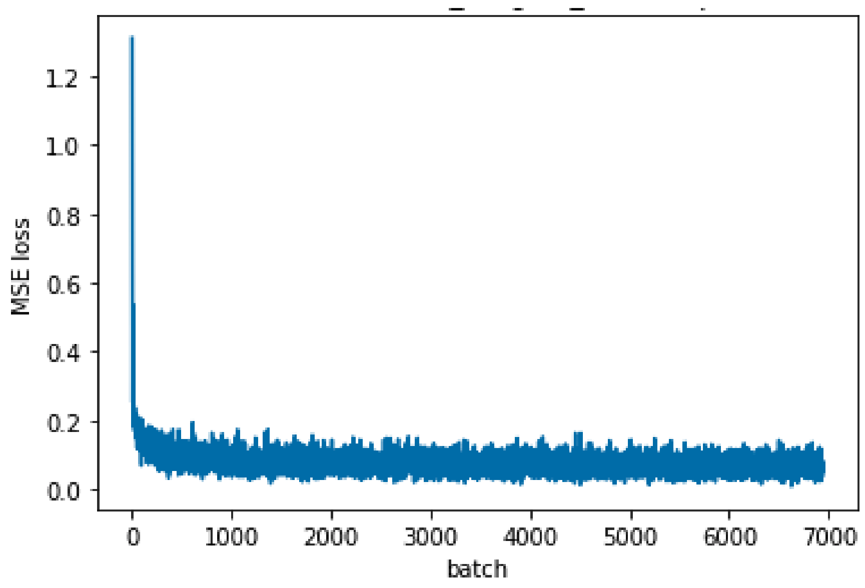


Figure 6.2: MSE loss across batches for Model 2: Best Model

As shown, the loss curve does appear to fluctuate around .05 and .2 which is expected with our network design where the weights are being convolved. This fluctuation is a positive visual because it means that the network is quickly calibrating itself after each convolution takes place because if it wasn't then we would see very large jumps in loss randomly throughout the training process.

A table depicting the metrics for the best model is shown below in Table 6.3.

Metric	Value
Accuracy	.68
Precision	.747
Recall	.765
F-1 Score	.756

Table 6.3: Evaluation metrics for Model 2: Quantitative

		Predicted	
		Not with party	With party
True	Not with party	5,045	4,593
	With party	4,170	13,584

Table 6.4: Confusion matrix for Model 2: Quantitative

The best parameters for the model are fairly systematic across the research questions, with minor differences in the weights parameter and the number of nodes per layer.

```
xy_layers: [keras.layers.Dense(256, activation='relu'),
            keras.layers.Dense(128, activation='relu'),
            keras.layers.Dense(64, activation='relu'),
            keras.layers.Dense(32, activation='relu')]
xy_layers: [keras.layers.Dense(256, activation='relu'),
            keras.layers.Dense(128, activation='relu'),
            keras.layers.Dense(64, activation='relu'),
            keras.layers.Dense(32, activation='relu')]
yz_layers: [keras.layers.Dense(256, activation='relu'),
            keras.layers.Dense(128, activation='relu'),
            keras.layers.Dense(64, activation='relu'),
            keras.layers.Dense(32, activation='relu')]
num_nodes: 512
w: [0.7,0.3]
batch size: 64
epochs = 5
```

It seems as though weighting the original weights as .7 and the convolved weights as 0.3 performed the best. This shows that a smaller percentage of the information in the new weights is imputed into the original weights. This is the opposite weighting schema from Model 1 which does not show a clear trend in how the weighting parameter should be set for optimal performance.

6.1.2.2 Categorical dependent variable

: As stated previously, since Model 2 predicts only on the dependent variable, we are able to use a categorical approach as well.

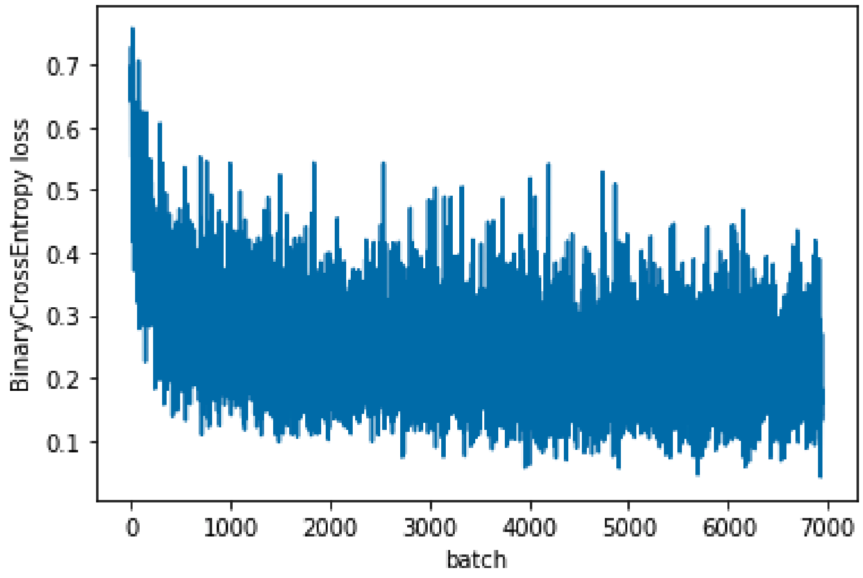


Figure 6.3: Binary Cross-entropy loss across batches for Model 2: Best Model

Figure 6.3 shows the loss of the model as it goes through the training process. Here we can also see the fluctuations in loss that are hovering around .1 and .5, which is how the model is convolving the weights throughout the training.

Metric	Value
Accuracy	.912
Precision	.918
Recall	.950
F-1 Score	.933

Table 6.5: Evaluation metrics for Model 2: Categorical

True		Predicted: Not with party	Predicted: With party
	Not with party	8,128	1,510
	With party	901	16,853

Table 6.6: Confusion matrix for Model 2: Categorical

The parameters for the model are shown below.

```
xy_layers: [keras.layers.Dense(128, activation='relu'),
            keras.layers.Dense(64, activation='relu'),
            keras.layers.Dense(32, activation='relu')]
xy_layers: [keras.layers.Dense(128, activation='relu'),
            keras.layers.Dense(64, activation='relu'),
            keras.layers.Dense(32, activation='relu')]
yz_layers: [keras.layers.Dense(128, activation='relu'),
            keras.layers.Dense(64, activation='relu'),
            keras.layers.Dense(32, activation='relu')]
num_nodes: 256
w: [0.7,0.3]
batch size: 64
epochs = 8
```

The results of this model are quite good, with accuracy over .91 and a high F-1 score this model is able to separate legislators who voted with their party and who didn't, quite well. Although, out of all the legislators who did not vote with their party, this model predicts around 14 percent of them to have voted with their party.

6.2 Baseline Comparison

In the baseline comparison we make a simple $XY \longrightarrow Z$ network that we build using keras. This represents almost the exact network implemented in the convolved model but without the convolution taking place. Since this network is only predicting the dependent variable, we are able to test both an MSE loss and a BinaryCrossEntropy loss to create a valid comparison to Model 1 and Model 2.

6.2.1 Categorical dependent variable

: This model seems to do quite well with respect to using a categorical dependent variable.

		Predicted	
		Not with party	With party
True	Not with party	8,325	1,313
	With party	982	16,772

Table 6.7: Confusion Matrix for Baseline Model: Categorical

This model required 10 epochs to converge which is a bit higher than some of the others that we have tested earlier. The others needed around 8 epochs.

Metric	Value
Accuracy	.916
Precision	.927
Recall	.944
F-1 Score	.936

Table 6.8: Evaluation metrics for Baseline Model: Categorical

The parameters for the model:

```
A1_in = keras.layers.Dense(512, activation = 'relu')(XY)
A1_in = keras.layers.Dense(256, activation = 'relu')(A1_in)
A1_in = keras.layers.Dense(128, activation = 'relu')(A1_in)
A1_in = keras.layers.Dense(64, activation = 'relu')(A1_in)
out_m1 = keras.layers.Dense(1, activation='sigmoid')(A1_in)
```

Once again this model is very similar to the other ones chosen in terms of layer size. The results are quite strong in terms of accuracy (.915) and a high F-1 score (.936) which means this model is also good at separating the binary class of whether or not a legislator voted with their party.

6.2.2 Quantitative dependent variable

:

An MSE loss function was also used on the baseline in order to be able to compare its performance to Model 1. The same mechanism of transforming predictions into binary classes was used as in previous expirements.

		Predicted	
		Not with party	With party
True	Not with party	8,357	1,281
	With party	1057	16,697

Table 6.9: Confusion Matrix for Model 2: Quantitative

Metric	Value
Accuracy	.914
Precision	.928
Recall	.940
F-1 Score	.934

Table 6.10: Evaluation metrics for Baseline Model: Quantitative

The parameters for the model:

```
A1_in = keras.layers.Dense(512, activation = ’relu’)(XY)
A1_in = keras.layers.Dense(256, activation = ’relu’)(A1_in)
A1_in = keras.layers.Dense(128, activation = ’relu’)(A1_in)
A1_in = keras.layers.Dense(64, activation = ’relu’)(A1_in)
out_m1 = keras.layers.Dense(1, activation=None)(A1_in)
```

Looking at the previous experiments, this model is very similar to the ones chosen there. The performance is quite high, similarly to the categorical predictor for the baseline model.

6.2.3 Additional Experiment

After we performed these experiments we felt like there was still one additional test we could perform. Due to the high accuracy of the baseline model, we hypothesize that the structure of the problem that we are trying to solve is too simple. In order to make the task more difficult for the model, we decide to compare the best performing model to the baseline without including the party affiliations variable. These two models are Model 2 with a binary cross entropy loss and the baseline model with a binary cross entropy loss. The reason for this, is because party affiliation is one of the most important features when it comes to predicting how a legislator will vote on a bill. Removing this variable will result in a more difficult classification task for the model.

The results of our experiments are shown below:

Metric	Value
Accuracy	.912
Precision	.924
Recall	.930
F-1 Score	.927

Table 6.11: Evaluation metrics for Baseline Model: (No party affiliation)

Metric	Value
Accuracy	.913
Precision	.918
Recall	.950
F-1 Score	.934

Table 6.12: Evaluation metrics for Single Predictor Model: (No party affiliation)

Looking at Table 6.11 and Table 6.12 it is clear that we get fairly similar results to the previous models. It appears that both models are able to use the other features quite well in order to produce accurate predictions.

6.3 Model Comparison

In order to compare the models, the results are displayed below in a table similar to the ones for each model.

Model	loss	Accuracy	Precision	Recall	F-1 Score
Model 1	MSE	.667	.789	.664	.72
Model 2	MSE	.68	.747	.765	.756
Model 2	BCE	.912	.918	.950	.933
Baseline	BCE	.916	.927	.944	.936
Baseline	MSE	.914	.928	.940	.934

Table 6.13: Comparison of Accuracy, Precision, Recall, and F-1 score for our models.

Overall when comparing Model 1 and Model 2 it is quite clear that Model 2 performed better than Model 1 as shown in Table 6.13. The way the network trained showed that there were much more fluctuations in the loss for the network that predicted on all the interaction metadata as supposed to the network that didn't. This phenomenon is shown in Figures 6.2 and 6.2 Furthermore the fact that Model 2 was able to train using a binary loss function shows that it is most likely the better choice because it allows for more variability in parameterization because the network only predicts on one variable.

Comparing Model 1 and Model 2 to the baseline network shows that Model 2 has the same predictive power as the baseline, while Model 1 performs worse. The baseline network achieved accuracy measurements of .916 and Model 2 achieved accuracy measurements of around .912 which shows no significant difference. The differences in the MSE models were not significant either (Model 1 : .667 accuracy, Model 2: .68 accuracy) but not entirely interesting to inspect because the categorical predictor model outperformed the MSE model. We also can see that the errors are distributed in a roughly balanced way across the experiments, which is why precision and recall numbers are relatively close across the exercises. Furthermore, our experiment which removes party affiliation from the user metadata showed no difference in accuracy for the highest performing models.

With this information there is no clear difference between Model 2 and the baseline but both the baseline and Model 2 highly outperformed Model 1.

Chapter 7

CONCLUSION/FUTURE WORK

In our work we proposed and implemented a set of neural network architectures designed to take advantage of information available in the training set, but not available in test test data. Specifically, the inclusion of interaction data between users and items during training of the model. We see many potential applications of our methodology in the field of recommender systems ranging from movie streaming platforms to online shopping websites. The number of companies that need powerful recommendation engines is immense and the problem of building one is a difficult one.

Due to lack of appropriate publicly available recommender system datasets, we conducted our test study on a dataset from the Digital Democracy project in which we tried to predict the votes of the legislators on bills based on the information about the legislators, the bills, and the legislator interactions with prior bills. Our results show that our methods allow for accurate prediction in this dataset. At the same time, our approach was not able to provide a lift over a baseline neural network. We believe that in part, this is because the nature of the dataset makes predictions relatively easy (lawmaker's party affiliation is a solid predictor all by itself in many cases). In order to try and use our dataset but still account for this problem, we try removing party affiliation from the legislature features. This results in very similar accuracies for our models. As such, our approach requires additional validation.

Further model variations can also be examined. Convolving the weights in different layers down the network or changing the frequency of convolutions could be the study of future work as well. Future work involves finding/constructing multiple datasets that are appropriate for our proposed models and evaluating further variations of our models that build off of the ideas of COLANDER.

BIBLIOGRAPHY

[1] Cal Poly Github. `http://www.github.com/CalPoly`.

[2] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[3] J. Ai, Z. Su, K. Wang, C. Wu, and D. Peng. Decentralized collaborative filtering algorithms based on complex network modeling and degree centrality. *IEEE Access*, 8:151242–151249, 2020.

[4] L. Breiman. Random Forests. *Machine Learning*, 45(1):5–32, Oct. 2001.

[5] H.-T. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir, et al. Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems*, pages 7–10, 2016.

[6] T. Chengsheng, L. Huacheng, and X. Bing. Adaboost typical algorithm and its application research. *MATEC Web of Conferences*, 139:00222, 01 2017.

[7] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, Sept. 1995.

[8] C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[9] D. R. Cox. The regression analysis of binary sequences. *Journal of the Royal Statistical Society: Series B (Methodological)*, 20(2):215–232, 1958.

[10] S. Craw. Manhattan Distance. In C. Sammut and G. I. Webb, editors, *Encyclopedia of Machine Learning and Data Mining*, pages 790–791. Springer US, Boston, MA, 2017.

[11] T. G. Dietterich. Ensemble methods in machine learning. In *International workshop on multiple classifier systems*, pages 1–15. Springer, 2000.

[12] B. Fortuna, C. Fortuna, and D. Mladenić. Real-time news recommender system. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 583–586. Springer, 2010.

[13] J. H. Friedman. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29(5):1189–1232, 2001.

[14] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning.* The MIT Press, 2016.

[15] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, G. Wang, J. Cai, and T. Chen. Recent advances in convolutional neural networks. *Pattern Recognition*, 77:354 – 377, 2018.

[16] G. Guo, H. Wang, D. Bell, Y. Bi, and K. Greer. Knn model-based approach in classification. volume 2888, pages 986–996, 01 2003.

[17] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.*, 22(1):5–53, Jan. 2004.

[18] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu. An efficient k-means clustering algorithm: analysis and implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):881–892, 2002.

[19] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

[20] A. R. Lahitani, A. E. Permanasari, and N. A. Setiawan. Cosine similarity to determine similarity measure: Study case in online essay assessment. In *2016 4th International Conference on Cyber and IT Service Management*, pages 1–6, 2016.

[21] Y. LeCun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel. Handwritten Digit Recognition with a Back-Propagation Network. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2, pages 396–404. Morgan-Kaufmann, 1990.

[22] A. Maćkiewicz and W. Ratajczak. Principal components analysis (PCA). *Computers & Geosciences*, 19(3):303 – 342, 1993.

[23] J. Meneses. About pandora and other streaming music services: The new active consumer on radio. 6, 01 2012.

[24] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010.

[25] K. Pearson. Note on Regression and Inheritance in the Case of Two Parents. *Proceedings of the Royal Society of London Series I*, 58:240–242, Jan. 1895.

[26] F. Ricci, L. Rokach, and B. Shapira. Introduction to Recommender Systems Handbook. In F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, editors,

Recommender Systems Handbook, pages 1–35. Springer US, Boston, MA, 2011.

[27] N. Russo. Dish: Democracy in state houses, Mar 2019.

[28] N. Sahoo, P. V. Singh, and T. Mukhopadhyay. A hidden markov model for collaborative filtering. *MIS Q.*, 36(4):1329–1356, Dec. 2012.

[29] C. Sammut and G. I. Webb, editors. *Mean Squared Error*, pages 653–653. Springer US, Boston, MA, 2010.

[30] J. L. Sanchez, F. Serradilla, E. Martinez, and J. Bobadilla. Choice of metrics used in collaborative filtering and their impact on recommender systems. In *2008 2nd IEEE International Conference on Digital Ecosystems and Technologies*, pages 432–436, 2008.

[31] A. I. Schein, A. Popescul, L. H. Ungar, and D. M. Pennock. Methods and metrics for cold-start recommendations. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 253–260, 2002.

[32] J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85 – 117, 2015.

[33] F. Strub and J. Mary. Collaborative filtering with stacked denoising autoencoders and sparse inputs. In *NIPS workshop on machine learning for eCommerce*, 2015.

[34] L. Terveen and W. Hill. Beyond recommender systems: Helping people help each other. 2001.

[35] P. B. Thorat, R. Goudar, and S. Barve. Survey on collaborative filtering, content-based filtering and hybrid recommendation system. *International Journal of Computer Applications*, 110(4):31–36, 2015.

[36] K. M. Ting. *Precision and Recall*, pages 781–781. Springer US, Boston, MA, 2010.

[37] M. Volkovs, G. W. Yu, and T. Poutanen. Content-based neighbor models for cold start in recommender systems. In *Proceedings of the Recommender Systems Challenge 2017*, pages 1–6. 2017.