A Modern Approach To Political Analytics*

Political Analytics (POAN5990), Spring 2024

Benny Rosenzweig (**bhr2116**)

Abstract

This project introduces the documentation for an attention-based neural network model designed for making predictions on varied social science data.[1] The three necessary stages of this project are: 1. Model design, 2. Data collection and processing, and 3. Training and evaluation. Early performance of the model when trained on limited data is on par with, or better than, most common predictive techniques. Additionally, the predictive accuracy of this model is estimated to improve significantly when trained on larger data sets, more-so than comparable predictive techniques. The model prototype described in this paper utilizes a self-attention mechanism, modified from that described in "Attention Is All You Need" by Vaswani, et al.,[2]. Future work will focus on running highly overparameterized models on more capable hardware than that used for this project, training this architecture on larger data sets, and modifying the architecture to include Gaussian processes.[3]

1 Introduction: A Flexible Neural Network Model For Social Science Prediction

The purpose of this project is to define a new application of supervised machine learning for political and social science, and to examine its performance on the task of predicting of political election outcomes. The primary questions motivating this project are as follows:

*Code and data files are accessible here: https://github.com/brosenzweig27/POAN5990.git

[1] A clever name for which has not yet been decided. Any suggestions should be sent to bhr2116@columbia.edu.

[2] (Vaswani, et al., 2017)

[3] (Chen & Lee, 2021: arXiv:2102.05208 [cs.LG]); (Yasarla, et al., 2024: "Self-Supervised Denoising Transformer With Gaussian Process"); (Guo, et al., 2019)

1. How can new technologies designed for highly complex machine learning tasks (image classification, object detection, speech recognition, text-sentiment analysis, sequence-to-sequence language modeling, etc.) be utilized for analytical tasks in political and social science?

2. How costly, in terms of time and computer memory, will these models be?

3. Do the benefits of these models outweigh the additional costs?

To answer these questions, I first discuss the broad motivations for using neural networks for analytical tasks involving large data sets (i.e. data with millions of observations of hundreds of variables, if not more). I then shift focus to newer technologies like attention and transformers and discuss why it may be useful to think of common social science data more like text or visual data.

The next section walks through the three stages of implementing this model, which are 1. Data collection and processing, 2. Model design, and 3. Training and evaluation. I do this through the example analysis of the Jacobson-Carson dataset which contains information about United States House of Representatives elections dating from 1946 to 2022. All data as well as the code required to build the model from scratch and perform basic analyses can be found in the GitHub repository linked at the top of this report. The default model parameters and the size of this dataset make this example analysis memory and computationally-effective for most standard CPUs, so I encourage the reader to follow along on their personal computer.[4]

The final section of this report compares this models performance on the predictive task of the previous step to a random forest predictive model, as well as a ridge regression model. While this new model already performs well by comparison, I explore why there is reason to be hopeful about the potential of this model for other predictive and analytical tasks.

2 Motivation

The modern age of big data requires modern modeling techniques. People, the world they live in, and political systems in particular have a near infinite number of moving parts which interact in elusive and sometimes inexplicable ways. The common wisdom of predictive modeling is to try and identify the top key variables which - individually or in combination - best explain the outcomes we observe. But what if this approach is fundamentally limited? In this age of big data, machine learning, and

[4]Computational costs can scale very quickly as model size and input size increase. Specific calculations to this end are not included in this report so play around with the model parameters and larger datasets at your own risk!

artificial intelligence, it is critical to work with tools capable of capturing high levels of complexity, potentially beyond just the top key variables. This is especially pertinent to political systems where outcomes are influential and "statistical anomalies" can seem anything but anomalous.[5]

More traditional modeling techniques including regression, decision trees, random forests, and clustering are all limited in their ability to capture these complex interactions by an unforgiving *bias-variance trade-off*. This means that if we increase the model's "complexity," or number of parameters considered by the model (weights attributed to variables, variable interactions, subgroups, etc.), the generalizability and predictive power of that model begins to *decrease* past a certain point. For example, in classical linear regression, simply fitting a hyperplane to all observed data risks fitting the model to random, uninformative noise which can cause the model to "over-fit" and lose predictive power. In many cases, preserving generalizability of these models means sacrificing parameters related to variables which appear weakly correlated with outcomes.[6]

The benefit of using an attention mechanism (described in detail later on) to model data is that the organization of learnable parameters within the model are specially designed to prioritize complex relationships in high-dimensional inputs, instead of trying to isolate a small number important explanatory variables. This means that when we have large data sets with hundreds of variables, a neural network model utilizing attention has the potential to capture complex variable relationships more effectively than traditional modeling techniques.[7]

The additional benefit of neural network models is the ability to adjust the quantity and structure of the model's learnable parameters. This becomes crucial in "overparameterized" cases, where the number of model parameters exceeds the number of observations in the data. Overparameterization can be accomplished with traditional modeling methods,[8] but is particularly intuitive for neural networks where parameters come in "layers" (discussed in section 3.2) that can be freely increased in quantity and size. In a 2019 paper,[9] Belkin, et al. extend the classical understanding of the bias-variance trade-off to modern machine learning, showing that test error exhibits a "double descent" as model complexity increases. In Figure 1 taken from this paper, we see the standard bias-variance trade-off (left), where "risk" (i.e. the error in the model's predictions when faced with *previously unseen* data) has a local minimum on the range of model parameters from 0 to the "interpolation

[5] "How to Achieve Order from Chaos: Analyzing Big Data." Gross, 2015.

[6] "Model selection and overfitting." Lever, et al., 2016.

[7] "How to Avoid Overfitting in Deep Learning Neural Networks." Brownlee, 2019.

[8] Discussed in depth in the excellent explanatory paper "Double Descent Demystified: Identifying, Interpreting & Ablating the Sources of a Deep Learning Puzzle" by Schaeffer, et al., 2023

[9] "Reconciling modern machine-learning practice and the classical bias–variance trade-off." Belkin, et al., 2019.

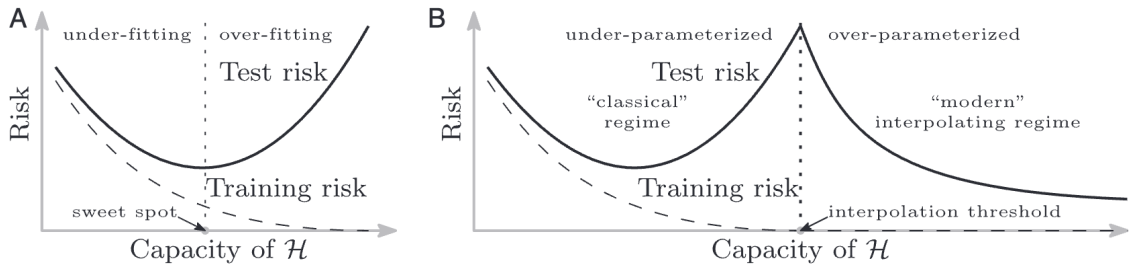

Figure 1: "Curves for training risk (dashed line) and test risk (solid line). (A) The classical U-shaped risk curve arising from the bias–variance trade-off. (B) The double-descent risk curve, which incorporates the U-shaped risk curve (i.e., the "classical" regime) together with the observed behavior from using high-capacity function classes (i.e., the "modern" interpolating regime), separated by the interpolation threshold. The predictors to the right of the interpolation threshold have zero training risk" (Belkin, et al., Fig. 1).

threshold," which is often equal to the number of total observations. Finding this minimum is the goal of "classical" regression model design, but can mean excluding certain predictive variables from your data set all together.

In what Belkin calls the "modern interpolating regime," we see that the test-loss of overparameterized models only decreases as model complexity goes to infinity - in most cases improving on the optimal loss found in the under-parameterized regime. This finding is lauded as one of the most promising and counterintuitive aspects of deep neural network modeling, as the only limitation of such a model would be the scope of the data itself and the computational resources. Large models require a lot of computer memory[10] so I will not be able to fully explore the overparameterized regime in this project. However, future iterations where more data is available and computation is outsourced, I can perform a similar analysis with even larger models and compare performance on a variety of predictive tasks.

2.1 Literature Review

The predictive modeling literature has been consistently shifting towards more modern technological advances such as Natural Language Processing (NLP),[11] Artificial Neural Networks (ANN),[12] among other statistical and machine learning processes.[13] However, for the specific task of predicting

[10]Training considerably large model with 1 billion parameters requires roughly 24 GB of GPU RAM: (Fregly et al., 2023. Ch. 4.)

[11](Tsai, et al., 2019); (Alvi, et al., 2023)

[12](Brito & Adeodato, 2020); (Chan, et al., 2021)

[13](Campbel & Mann, 1996) ; (Yang, William & Mary)

election outcomes, each of these approaches limits their model's application to *one* specific form of election related data. The recent advancements in NLP and ANN models for election prediction cited here focus primarily on twitter and social media data for sentiment analysis, and more traditional predictive models utilize polling and survey data exclusively.

In a 2023 paper, political scientist Lukasz Wordliczek comprehensively examined the potential for uses of neural networks in the social sciences, stressing the importance of large data sets (either through a large time-horizon or a wider range of variables) and concluding that there is vast potential for creative applications to "entirely new problems and the setting of new horizons."[14] Many recent publications like those cited above offer new perspectives on both old and new questions, but they only scratch the surface of this burgeoning new field. There appears to be significant room for exploration with multi-modal data sets both using both ANNs and other machine learning processes.

3 Walk-Through

3.1 Data Collection and Processing[15]

As will become clear through the description of the model in the next section, data must be in a very specific form to be processed by this model prototype. While this is ultimately one of the fundamental weak-points of this model prototype and will likely be the focus of future iterations, I hope to convince you - as I have convinced myself - that the benefit of utilizing these architectures is worth the momentary convolution of political data.

In the spirit of treating social science data similar to how we might treat text data in similar contexts, it may be useful to think of each row of data as a unique sentence whose words are comprised of the values of its columns. In order to analyze these sentences, we first need to define a dictionary of the possible words we may come across. This requires us to treat our concrete, numerical data rather unintuitively.

For example, the Jacobson-Carson dataset contains a variable 'fr,' which can take on a few values depending on the whether that county has elected a freshman member to Congress in recent years.[16] Categorical variables like this one (i.e. variables whose value indicates belonging to an abstract category, rather than an objective value) are easily understood as being flexible and open

[14](Wordliczek, 2023)

[15]Data cleaning available at: https://github.com/brosenzweig27/POAN5990/blob/main/clean_jacobson.ipynb

[16]See 'Jacobson Carson Codebook.docx' in the GitHub repository.

to interpretation. To fully treat this data as we would text data, we must similarly abstract non-categorical variables like 'year' or 'dexp' (financial expenditures of the Democratic candidate) from their objective meaning. Thus, the first step of data processing will be to factorize (i.e. turn into categorical variables) each variable in the data set into useful categories, from which we can extract a full "dictionary" of factors that can be used to describe any election.

After basic housekeeping outlined in the notebook linked to this section, I create explicit functions which place different types of continuous variables into bins, and then replace the variable with the factor variables associated with those bins. In this case, variables representing vote-shares (continuous percentages on [0.0, 100.0]) were placed into one of 34 bins which get progressively smaller around 50%. Candidate expenditures (continuous dollar amounts on [0.00, 3e7]) were placed into one of 28 bins spaced roughly according to expenditure distribution, as well as a separate less granular set of bins with 13 distinct categories. The first five rows of the before and after of this initial cleaning process are shown in Figure 2.

	year	stcd	inc	pwin	dv	dvp	fr	po1	po2	redist
0	1946.0	101.0	1.0	1.0	-999999999.0	-999999999.0	0.0	5.0	2.0	0.0
1	1946.0	102.0	1.0	1.0	-999999999.0	-999999999.0	0.0	5.0	2.0	0.0
2	1946.0	103.0	1.0	1.0	-999999999.0	-999999999.0	0.0	5.0	2.0	0.0
3	1946.0	104.0	1.0	1.0	88.1	84.5	0.0	0.0	0.0	0.0
4	1946.0	105.0	1.0	1.0	-999999999.0	-999999999.0	1.0	5.0	2.0	0.0

	dexp	rexp	dpres
0	-999999999.0	-999999999.0	-999999999.0
1	-999999999.0	-999999999.0	-999999999.0
2	-999999999.0	-999999999.0	-999999999.0
3	-999999999.0	-999999999.0	-999999999.0
4	-999999999.0	-999999999.0	-999999999.0

	year	inc	pwin	fr	po1	po2	redist	state	dist	dexp_cat_gran
0	1946	1	1.0	0	5.0	2.0	0	1	1	NaN
1	1946	1	1.0	0	5.0	2.0	0	1	2	NaN
2	1946	1	1.0	0	5.0	2.0	0	1	3	NaN
3	1946	1	1.0	0	0.0	0.0	0	1	4	NaN
4	1946	1	1.0	1	5.0	2.0	0	1	5	NaN

	rexp_cat_gran	dexp_cat	rexp_cat	dpres_cat	dvp_cat	dv_cat
0	NaN	NaN	NaN	NaN	NaN	NaN
1	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	31.0	32.0
4	NaN	NaN	NaN	NaN	NaN	NaN

Figure 2: Raw data (top) compared with factorized data (bottom). Transformations include 1. Converting all negative entries to NaNs, 2. Splitting 'stcd' into 'state' and 'dist,' 3. Factorizing vote-share variables ('dv,' 'dvp,' 'dpres') and expenditure variables ('dexp,' 'rexp') into bins denoted by '_cat.' Expenditures have been split into two factors, with the more granular (i.e. more categories) denoted with '_gran.'

To create the desired dictionary of possible "words" (or categories of data points) that can be used to describe each election, I distinguish each column with an identifying character. This ensures that a categorical value of 1 from the 'state' column gets a different dictionary value than a categorical value of 1 from any other column. This is the last step which allows us to create a single dictionary containing every unique value present in the dataset, which is essential for training discussed in section 3.3 below.

The next stages of data processing involve deciding what to do with missing data and splitting the data into inputs and outputs. The outputs I will consider for this project are the variables representing the results of the elections, which in this case are the 'dv' (Democrat vote-share) and 'pwin' (party win) variables. These will be considered separately by the model so I set them aside for now. Lastly, I consider the possible options for dealing with missing data in the code notebook.

	year	inc	fr	po1	po2	redist	state	dist	dexp_cat_gran	rexp_cat_gran	\
0	a1946	b1	c0	d5.0	e2.0	f0	g1	h1	i2342.0	j2342.0	
1	a1946	b1	c0	d5.0	e2.0	f0	g1	h2	i2342.0	j2342.0	
2	a1946	b1	c0	d5.0	e2.0	f0	g1	h3	i2342.0	j2342.0	
3	a1946	b1	c0	d0.0	e0.0	f0	g1	h4	i2342.0	j2342.0	
4	a1946	b1	c1	d5.0	e2.0	f0	g1	h5	i2342.0	j2342.0	

	dexp_cat	rexp_cat	dpres_cat	dvp_cat
0	k2342.0	l2342.0	m2342.0	n2342.0
1	k2342.0	l2342.0	m2342.0	n2342.0
2	k2342.0	l2342.0	m2342.0	n2342.0
3	k2342.0	l2342.0	m2342.0	n31.0
4	k2342.0	l2342.0	m2342.0	n2342.0

	pwin	dv_cat
0	a1	<NA>
1	a1	<NA>
2	a1	<NA>
3	a1	b32
4	a1	<NA>
...	...	...
16963	a0	b6
16964	a0	<NA>
16965	a0	b7
16966	a0	<NA>
16967	a0	b4

Figure 3: The final input (top) and output (bottom) datasets that will be used for training and evaluation. Letters serve as column identifiers and the values following them serve only as a label of a categorical variable. Each data point is thus a string.

For the purposes of this project I have decided to simply treat missingness as a possible factor for each variable by replacing each NaN with an arbitrary value not present in the data (I choose 2342 which carries not meaning outside of personal) and similarly appending the column identifiers. The heads of the final processed data sets are shown in Figure 3.

3.2 Model Design[17]

In the spirit of treating social science data as we would treat text data in similar contexts, this model is centered around the self-attention mechanism outlined in the seminal 2017 paper "Attention is all you need," which greatly advanced the field of natural language processing. In short, attention maps parameters containing information about data input ("queries") to a set of parameters containing information about the desired outputs ("keys" and "values") with the purpose of modeling relationships between the components of a potentially high-dimensional input.[18] These architectures are unprecedentedly good at capturing the nuances of highly complex classes of data such as text, but require large quantities of parameters to do so. Even what are now considered relatively "small" language models have parameter counts well in the *billions*.[19]

[17] Model code available at: https://github.com/brosenzweig27/POAN5990/blob/main/model.py

[18] (Vaswani, et al., 2017)

[19] (Hugging Face, 2023) ; (Touvron, et al., 2023)

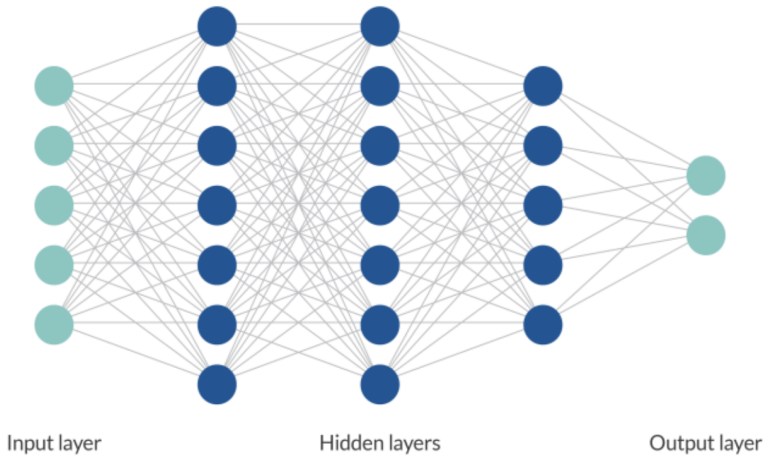

Figure 4: Example of a basic neural network. Both nodes ("neurons") and edges in this graph contain model parameters called biases and weights, respectively. The total number of weights in this example is the total number of edges connecting the neurons: $(5*7)+(7*7)+(7*5)+(5*2)=129$. The total number of biases is the number of non-input neurons: $7+7+5+2=21$. Therefore, this model would contain $129+21=$ **150** total learnable parameters.

In many ways, social science data is similar to language: individual pieces of data can take on multiple different meanings, and the context of additional data can significantly alter interpretations. As such, common language model architectures like generative pre-trained transformers (GPTs) can be thought of as a flexible tool with direct applications to social science beyond language modeling.

This project's model's complexity comes in various forms of learnable parameters, including simple hidden layers (Figure 4) and more complex attention mechanisms (Figure 5). Upon the creation of a model instance, each learnable parameter is randomly initialized, and over the course of training, these parameters are tuned in incremental optimization steps. During a single training step - also referred to as an "epoch" - the model is fed a subset of inputs taken from the data set. This data is then transformed by the model parameters into a guess of what the corresponding outputs should look like. This guess is then compared to the real outputs seen in the data, and the accuracy of the prediction is quantified by a loss function. The optimizer algorithm used for this model then aims to minimize this loss, and does so through gradient descent and back propagation through all of the parameters of the network.[20]

[20] I will not be detailing these processes in this report, but an excellent explanatory video is linked here: 3blue1brown, "What is backpropagation really doing? — Chapter 3, Deep learning". For a technical overview of the optimizer used for this model, see the documentation for AdamW from pytorch, which minimizes the cross-entropy loss calculated at each training step.

The walk-through presented in the file test_myfunctions.py allows for significant freedom in setting the hyperparameters which dictate the structure of the model. These include the following:

3.2.1 n_embd

At this point, one row of our data looks like a row of strange strings that represent the different variables' factor levels. In order for the model to learn what these strings mean, I assign each unique string in the data set to a vector of learnable parameters of size n_embd, called an *embedding*. For data units with multiple meanings or interpretations (words in text data, for example) n_embd can be very large.[21] The default n_embd for this project is 12.

3.2.2 n_head

For high-dimensional embeddings, the attention mechanism becomes computationally costly. This hyperparameter allows us to split the input data across $\frac{\text{n\_embd}}{\text{n\_head}}$ different attention "heads" that can then be run in parallel. The effectiveness of parallel computing is hardware-dependant, so this will not greatly affect this project's analysis. The default n_head is set to 4.

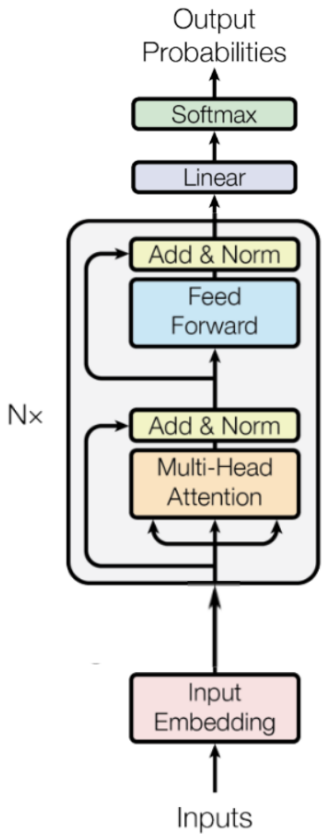

Figure 5: This is the general architecture of the model used for this project. Boxed terms represent layers of neurons or sequences of layers that serve specific functions; arrows represent weights connecting the layers. This visualization is adapted from Vaswani, et al., 2017.

3.2.3 n_layer

This dictates the number of sequential "layers" or "blocks" that contain multi-head attention layers, as well as normalization layers and "feed forward" mechanisms. The feed forward mechanism used here consists of two linear layers of neurons, one non-linear layer, and parameter dropout (see 3.2.4).

[21] For a language model like ChatGPT-4, each unique token gets an embedding of size 1,536, but the majority of smaller models employ embeddings of size ranging from 12 to 256 (Open AI, 2022).

Linear layers behave exactly like the hidden layers in Figure 4, and they are separated by a non-linear "ReLU" layer which functionally *turns off* a subset of neurons.[22] n_layer corresponds to N in Figure 5, and the default is set to 3.

3.2.4 dropout

During training, dropout randomly zeroes some elements of an input vector with probability equal to this hyperparameter. This acts as a regularization technique that prevents the model from overfitting.[23] Dropout serves a similar purpose to the parameter λ in a standard ridge regression, which penalizes parameters from growing too large and hampering the generalizability of the model. For this model, dropout occurs after each multi-head attention step, as well as after each feed forward sequence, and the default hyperparameter is set to 0.2.

3.2.5 width

This hyperparameter, when multiplied by n_embd, determines the number of nodes contained in the two linear layers in each feed forward mechanism discussed in 3.2.3. Specifically, if we consider n_embd = 5 (which would correspond to an input layer of size 5 like that seen in Figure 4) and width = 2, the linear layers (now taking the place of the hidden layers in Figure 4) would each contain $5 * 2 = 10$ nodes. Increasing either of these parameters is an easy way to increase the complexity of the model. If we consider only these input and linear layers with n_embd = 5 and width = 2, we obtain a total parameter count of 170.[24] By doubling width (i.e. width = 4) we more than triple the parameter count to 540.[25] The default width for this model is set to 6.

3.3 Training

Once the data has been processed, hyperparameters have been chosen, and a model instance has been initialized, we can begin training. We do this by executing the "train" function in myfunctions.py, which itself takes the following inputs:

[22] Details and added benefits of ReLU activation layers can be found in (Gupta, 2024).
[23] (Hinton, et al., 2012)
[24] Weights: $(5 * 10) + (10 * 10) = 150$. Biases: $10 + 10 = 20$. Total: $150 + 20 = 170$.
[25] Weights: $(5 * (5 * 4)) + ((5 * 4) * (5 * 4)) = 500$. Biases: $(5 * 4) + (5 * 4) = 40$. Total: $500 + 40 = 540$.

3.3.1 data_in & data_out

These are the inputs and outputs (respectively) of data that the model is shown during training. There are multiple reasons why we might choose to train the model on a subset of the total data set. For the purpose of predicting election results, I exclude the most recent elections from training (in the Jacobson-Carson data set this is all House election in 2022) and evaluate the performance of the model by how well it can predict the outcomes of this unseen data (i.e. the 2022 elections). All subsetting (with additional examples) is performed in test_myfunctions.py.

3.3.2 target_var

This specifies the outcome variable from data_out that the model will try to predict. In our case, data_out contains "pwin" (winning party) and "dv_cat" (democratic voteshare) and I have chosen to predict "pwin" only for the demonstration below. It is possible to have the model predict multiple or all outcome variables simultaneously, but for the purposes of this project, target_var = "pwin."

3.3.3 test_in & test_out

During training, it is helpful to keep track of how well the model is learning by having it report the loss calculated at each step. We can evaluate this loss not only for the training data, but also on the data reserved for testing. At certain increments during the training process, we can show the model data from the test set, have it predict the corresponding election result, and calculate the loss with the true observations. test_in and test_out are the dataframes containing the inputs and outputs from these test elections, which in the example below are all elections from 2022. It is important to note that once the test loss is calculated, no optimization step occurs so we can be sure that the model is not learning from the data it is not supposed to see.

3.3.4 in_vocab & out_vocab

These are the dictionaries that map each unique factor level in the data set to an integer value, as discussed in section 3.1 above. These mappings make it easy to create large tables called "embedding tables," where each unique factor level gets associated with a vector of n_embd learnable parameters. The functions used to create these dictionaries are in myfunctions.py and the implementation is in test_myfunctions.py.

3.3.5 rows & n_batch

If you want the model to train on a specific subset of rows from the input and output data sets, include this list of row indices in the rows input. If rows is not specified, the model will choose an n_batch number of rows randomly from the training data at each training step. In a single training step, the model minimizes the loss calculated across this "batch" of input rows simultaneously. If n_batch equals the length of the training data, or you specify every row of the training data in rows, the model will see every data point in each step which is commonly referred to as a complete forward pass, or a single training epoch.

3.3.6 epochs & eval_iters

The epochs input specifies the number of steps you would like the model to take when you run the training function. Since the model parameters update after each step, no training progress will be lost if you run multiple separate training sessions. The number of epochs required to reach some desired loss depends greatly on the size of the model as well as the training data being used, so some trial and error is best here - at least for experimental purposes.

eval_iters specifies the increments at which we would like to evaluate the training and test losses in order to track and visualize training progress. We can do this after each training step (i.e. eval_iters = 1), but this can significantly slow down the training process and is often not necessary to get a good idea of how training is progressing. While the training function runs, it saves the training and test loss values calculated at each evaluation to be plotted when training concludes, and prints the following statement at the i^{th} training step:

$$\text{Losses at iteration } i = [(\text{training loss}), (\text{test loss})]$$

3.3.7 lr

Lastly, we can specify the learning rate (lr) of the model. This is a hyperparameter which dictates how much the model parameters should change at each training step. Large learning rates can lead to fast, but imprecise training, and small learning rates can increase the number of epochs needed to fully train a model. More advanced models than this prototype employ variable learning rates for different types of parameters and different stages of training.[26] The default learning rate is set to 0.0001.

[26](Montigatti, 2023)

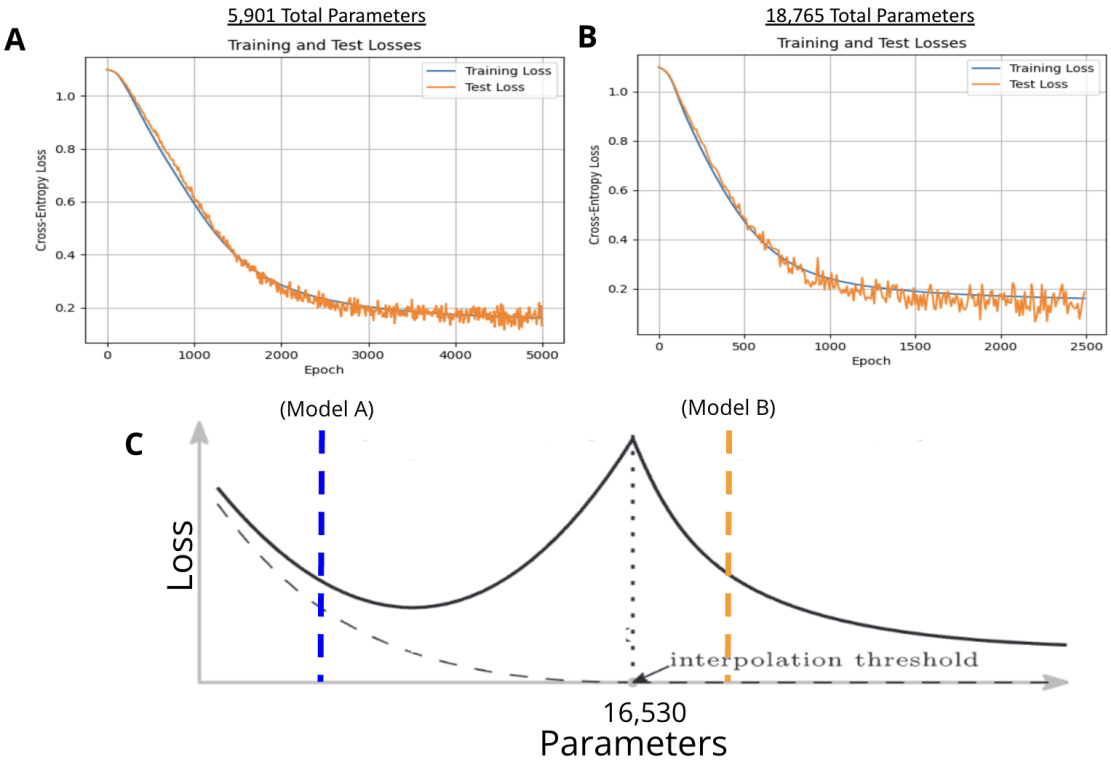

Figure 6: Training evolutions of two model instances, and roughly where they fall on the double descent plot. (A) This model is underparameterized, reaching a minimum training loss of 0.185. (B) This model is overparameterized, reaching a minimum training loss of 0.181. (C) This is the same general double-descent plot from Figure 1, with the approximate placements of the models from A and B.

4 Results

After cleaning, the Jacobson-Carson data set contains 16,530 rows of election data prior to 2022. As discussed earlier, the double-descent interpolation threshold occurs roughly when the number of parameters (P) equals the number of data observations (N = 16,530). The implication of this literature is that there exists a local minimum test loss attainable by a model with fewer than 16,530 parameters, and an even smaller test loss attainable by a highly overparameterized model. Figure 6 plot C revisits the general double-descent plot from Belkin, et al. shown in Figure 1, but it is important to note that the observed shape of double-descent varies by model, and the true double-descent plot for this project's model is currently unknown.[27]

[27]For more intuition on this topic as well as examples of real double-descent plots, see "Understanding 'Deep Double Descent.'" AI Alignment Forum, 2019..

Rather, Figure 6 plot C conveys roughly where the models whose evolutions are shown in plots A and B fall on the spectrum of parameterization. The model from plot A is underparameterized, containing 5,901 total parameters.[28] The model from plot B is overparameterized, containing 18,765 total parameters. The hyperparameters and training inputs are identical between these two models, except model A has n_embd = 6, and model B has n_embd = 12. When evaluating the predictive capabilities of these models, both successfully categorize $\frac{\mathbf{408}}{\mathbf{435}} = \mathbf{93.79}\%$ of the unseen 2022 elections.

While it seems counterintuitive that these models with dramatically different parameterizations would ultimately have the same predictive accuracy, double-descent allows us to understand how this is possible, and helps point us towards better models. For this project, all models were trained on a 2020 MacBook Pro, which is limited both in computational efficiency and in memory. This means that training large models can take up to an hour, and can easily expend the computer's virtual memory. In future work, ideally with outsourced computation and memory, I can more efficiently train models along a wide range of parameterizations and clearly identify the optimal underparameterized model, and the potential improvements we can make through overparameterization. I would also then be able to perform analysis like that in *Deep Double Descent*, from OpenAI, which not only looks for optimal parameterizations, but considers the optimal number of training data samples to show to models of different sizes.[29]

In playing around with different model parameters, the best performing neural network model I trained had 10,130 total parameters, achieved a minimum training loss of 0.116, and correctly predicted $\frac{\mathbf{418}}{\mathbf{435}} = \mathbf{96.09}\%$ of the unseen 2022 elections.

4.1 Other Predictive Modeling Approaches

For the purpose of comparison, I trained two other common types of predictive models for this same task. For both, the same 16,530 rows of pre-2022 election data were shown to the model for training, and the performance was assessed by the accuracy of "pwin" predictions given the remaining set of unseen 2022 elections.[30]

[28] The general formula for calculating the total number of parameters is:

$$P = (vn) + l(5n + 5n^2 + 2nw + 2n^2w + n^2w^2) + n + o + no + 2c + 1$$

Where v ="vocab_len"; n ="n_embd"; l ="n_layer"; w ="width"; o ="out_space"; c ="n_col". Additional constants not mentioned in section 3 are defined in test_myfunctions.py.

[29] (Open AI, 2019)

[30] This modeling was done in R and is available upon request.

The first model I trained was a standard ridge regression with optimal feature selection. Compared to linear regression, ridge regression is a more favorable strategy for modeling data like Jacobson-Carson, where multicollinearity - or the tendency for two or more columns of a data set to be highly correlated - requires consideration. To combat multicollinearity, ridge regression incorporates a penalty term λ to its objective function, which discourages model parameters from getting large. The optimal λ is found via the grid-search algorithm cv.glmnet from the glmnet package in R. This model additionally included optimal feature selection using the regsubsets function of the leaps package in R. This function systematically tests regression models using different subsets of predictors and reports the one which minimizes test loss. This optimal ridge regression model correctly predicted $\mathbf{\frac{376}{435}} = \mathbf{86.43}\%$ of the unseen 2022 elections.

The second alternative modeling technique I employed was a random forest, which aggregates the predictions of various decision trees that each break a subset of the data into smaller and smaller subgroups which it finds to be most predictive of the outcome variable. This type of model is particularly useful for categorical and factor data, as the data is already split neatly into levels that the model can distinguish between, and is thus appropriate for this project. I used the randomForest function from the randomForest package in R to perform this analysis on the Jacobson-Carson data. This model correctly predicted $\mathbf{\frac{416}{435}} = \mathbf{95.63}\%$ of the unseen 2022 elections. A summary of the performance of each model is in the table below.

	Ridge Regression	Random Forest	**Neural Network**
Correct Predictions	$\frac{376}{435} = 86.43\%$	$\frac{416}{435} = 95.63\%$	$\frac{418}{435} = 96.09\%$

5 Conclusion

I hope to have demonstrated in this report that deep neural networks with attention are not only appropriate for social science analyses, but have potential beyond what was demonstrated in earlier sections. Even without reaping the benefits of highly overparameterized models due to limitations in computing power, and even without incredibly large corpora of data as is common for training large models like these, this model architecture was able to out-perform a random forest. While this improvement is marginal (an increase in predictive accuracy of only 0.46 percentage points), it is highly unlikely that the network used to make this prediction was optimally parameterized.

Without access to specialized GPUs and increased memory, it will be difficult to test the full

predictive potential of this network architecture. However, there are multiple reasons to be hopeful that the added benefits will outweigh the costs. The first is the concept of predictors with "zero training risk" which comes from Figure 1 of Belkin et al. and underscores the benefit of working with overparameterized models. This is the idea that because the model has so many extra parameters, adding predictors (i.e. extra columns or variables to the training data set) will *only* work to decrease the test error. This idea is also supported by De Veaux & Ungar in their 1994 paper,[31] who argue that column multicollinearity does not have negative effects in overparameterized regimes, as model depth and parametric redundancy "makes individual weights unimportant." Similarly, columns with near-zero covariance with the outcome variable should get down-weighted by the network, and thus can do not predictive harm, so to speak.

The upshot of this is that these large models will ultimately be better suited than traditional models for analyzing data sets with hundreds of predictive variables or more. This is because the attention architecture is specifically designed to analyze *all of the predictors together*. In order to accommodate large data sets, modeling techniques like regression require optimal feature selection, whereby a subset of predictors are dropped from the analysis all together. Random forests are able to utilize all predictors, but each decision tree comprising the random forest is only ever privy to a small subset. Not only is every predictor considered at each training step of a neural network, but the attention mechanism is specifically designed to find the intricate relationships that exist between each and every one.

We are living in a unique time where unprecedented new technologies are being met with unprecedented access to data. Currently, the cost of computer memory and specialized GPUs is significant, yet their integration is rapidly becoming ubiquitous as never seen before. If we are, as it seems, ultimately beholden to Moore's law, models significantly larger than those discussed in this project are poised to become commonplace on any personal computer within only a few years. Personally, I am excited for this reality and hope that the architecture proposed in this report can be a meaningful step in that direction.

- Benny Rosenzweig

[31] "Multicollinearity: A tale of two nonparametric regressions," De Veaux, 1994

Works Cited

Alvi Q, Ali SF, Ahmed SB, Khan NA, Javed M, Nobanee H. On the frontiers of Twitter data and sentiment analysis in election prediction: a review. PeerJ Comput Sci. 2023 Aug 21;9:e1517. doi: 10.7717/peerj-cs.1517. PMID: 37705657; PMCID: PMC10495957.

Belkin, Mikhail, et al. "Reconciling modern machine-learning practice and the classical bias–variance trade-off." Proceedings of the National Academy of Sciences, vol. 116, no. 32, 24 July 2019, pp. 15849–15854, https://doi.org/10.1073/pnas.1903070116.

Brito, Kellyton, et al. "Please stop trying to predict elections only with Twitter." DG.O 2022: The 23rd Annual International Conference on Digital Government Research, 15 June 2022, https://doi.org /10.1145/3543434.3543648.

Brownlee, Jason. "How to Avoid Overfitting in Deep Learning Neural Networks." MachineLearningMastery.Com, 6 Aug. 2019, machinelearningmastery.com/introduction-to-regularization-to-reduce-overfitting-and-improve-generalization-error/.

Campbel, James E., Mann, Thomas E. "Forecasting the Presidential Election: What Can We Learn from the Models?" Brookings, 1 Sept. 1996, www.brookings.edu/articles/forecasting-the-presidential-election-what-can-we-learn-from-the-models/.

Chan, Ellison Yin Nang, et al. "Deep Learning-Based Election Results Prediction Using Twitter :." World Scientific, 8 Apr. 2021, www.researchgate.net/publication/356706869_Deep_learning-based_election_results_prediction_using_Twitter_activity.

Deep Double Descent, Open AI, 5 Dec. 2019, openai.com/research/deep-double-descent.

De Veaux, R.D., Ungar, L.H. (1994). Multicollinearity: A tale of two nonparametric regressions. In: Cheeseman, P., Oldford, R.W. (eds) Selecting Models from Data. Lecture Notes in Statistics, vol 89. Springer, New York, NY. https://doi.org/10.1007/978-1-4612-2660-4_40

Fregly, Chris, et al. Generative AI on AWS Building Context-Aware Multimodal Reasoning Applications. O'Reilly Media, 2023.

Gross, Ted. "How to Achieve Order from Chaos: Analyzing Big Data." Medium, 30 Jul. 2015, tedwgross.medium.com/how-to-achieve-order-from-chaos-analyzing-big-data-3eb6626e488.

Guo, M., Zhang, Y., & Liu, T. (2019). Gaussian Transformer: A Lightweight Approach for Natural Language Inference. Proceedings of the AAAI Conference on Artificial Intelligence, 33(01), 6489-6496. https://doi.org/10.1609/aaai.v33i01.33016489

Gupta, Dishashree. "Fundamentals of Deep Learning - Activation Functions and When to Use Them?" Analytics Vidhya, 25 Mar. 2024, www.analyticsvidhya.com/blog/2020/01/fundamentals-deep-learning-activation-functions-when-to-use-them/.

Hinton, Geoffrey E., et al. "Improving Neural Networks by Preventing Co-Adaptation of Feature Detectors." arXiv.Org, 3 July 2012, arxiv.org/abs/1207.0580.

Lever, J., Krzywinski, M. & Altman, N. Model selection and overfitting. Nat Methods 13, 703–704 (2016). https://doi.org/10.1038/nmeth.3968

Monigatti, Leonie. "A Visual Guide to Learning Rate Schedulers in Pytorch." Medium, Towards Data Science, 4 Dec. 2023, towardsdatascience.com/a-visual-guide-to-learning-rate-schedulers-in-pytorch-24bbb262c863.

M. -H. Tsai, Y. Wang, M. Kwak and N. Rigole, "A Machine Learning Based Strategy for Election Result Prediction," 2019 International Conference on Computational Science and Computational Intelligence (CSCI),

Las Vegas, NV, USA, 2019, pp. 1408-1410, doi: 10.1109/CSCI49370.2019.00263.

"New and Improved Embedding Model." Open AI, 15 Dec. 2022, openai.com/index/new-and-improved-embedding-model.

OpenAI GPT2, Hugging Face, 2023. huggingface.co/docs/transformers/main/en/model_doc/gpt2.

Schaeffer, Rylan, et al. "Double Descent Demystified: Identifying, Interpreting & Ablating the Sources of a Deep Learning Puzzle." arXiv.Org, 24 Mar. 2023, arxiv.org/abs/2303.14151.

Touvron, Hugo, et al. "Llama: Open and Efficient Foundation Language Models." arXiv.Org, 27 Feb. 2023, arxiv.org/abs/2302.13971.

"Understanding 'Deep Double Descent.'" LessWrong, AI Alignment Forum, 5 Dec. 2019, www.lesswrong.com/posts/FRv7ryoqtvSuqBxuT/understanding-deep-double-descent.

Vaswani, Ashish, et al. "Attention is All you Need." Advances in Neural Information Processing Systems, edited by I. Guyon et al., vol. 30, Curran Associates, Inc., 2017, https://proceedings.neurips.cc/ paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

Wordliczek, Łukasz. (2023). "Neural Networks and Political Science: Testing the Methodological Frontiers." Empiria. Journal of Social Science Methodology , (57), 37–62. https://doi.org/10.5944/ empiria.57.2023.36429

Yang, Chaoran, "Forecasting US Presidential Election Result with Machine Learning Algorithm," William & Mary, https://cklixx.people.wm.edu/teaching/math300/ChaoranY.pdf