Examining Recursive Feature Machines in Text Generation

Arunav Gupta
Halıcıoğlu Data Science Institute
UCSD
San Diego, CA
arg002@ucsd.edu

Mehdi Boussami
Halıcıoğlu Data Science Institute
UCSD
San Diego, CA
mbouassa@ucsd.edu

Rohit Mishra
Halıcıoğlu Data Science Institute
UCSD
San Diego, CA
r1mishra@ucsd.edu

William Luu
Halıcıoğlu Data Science Institute
UCSD
San Diego, CA
wjluu@ucsd.edu

Abstract

Although neural networks have proved very effective in solving a range of technological and scientific problems, our knowledge of how they work hasn't kept up with their progress in practice. We can develop fresh and efficient strategies to enhance neural network performance by identifying the underlying mechanics that underlie these systems. In order to construct recursive feature machines (RFMs), which are kernel machines that learn features, we focus our study on the essential mechanism underlying feature learning in fully connected neural networks. Our results demonstrate the superior performance of RFMs over a variety of models and a variety of data types.

1 Introduction

1.1 Background

In recent years the world has seen large, billion-parameter models achieve extremely high levels of accuracy at tasks previously thought to be difficult for machine learning models. However, researchers begin to question why these large parameter models work significantly better than more traditionally-sized models. One approach to this problem is to analyze kernel methods, specifically kernel functions which are a type of algorithm used to solve non-linear problems with linear classifiers. Different types of kernels such as the polynomial or Laplacian kernel transform the data into higher dimensions to solve the linear separability problem. Additionally, we want to compare neural networks to recursive feature machines which are simply kernel machines that learn features from deep fully connected neural networks and fill the gap between kernel machines and fully connected networks. A key mechanism between feature learning in neural networks can be isolated by connecting feature learning with a statistical estimator for feature selection which is known as the expected gradient outer product. Recursive feature machines can capture features learned by deep networks and can achieve state-of-the-art performance on some tabular data that we will compare against both kernel methods and neural networks and comment on the mathematical basis of our results. In this paper, we compare n-gram models and Laplacian kernel machines to recursive feature machines that utilize

Preprint. Under review.

a feature matrix that is a driving feature in neural networks for text data and gain an understanding of the performance differences between the three methods. We also look at the theory behind recursive kernel machines by analyzing the eigenvalues of the kernel matrix.

1.2 Related Work

Previous work has been done in analyzing kernels for machine learning (Hofmann et al., 2008). The authors of the paper explain that kernel methods are based on the idea of using a kernel function to map data from the original input space to a higher-dimensional feature space, where the data can be more easily separated by a linear classifier.

The authors describe different types of kernel functions, such as polynomial, Gaussian, and sigmoidal kernels, and discuss their properties and applications. They also describe how kernel methods can be used in different machine learning tasks, such as support vector machines, kernel principal component analysis, and kernel density estimation.

However, no research has been done on trying to understand why certain datasets are better suited for kernels than others. Further research has also been done the use of GPUs for large batch training on kernels (Ma and Belkin, 2018). Ma's work creates the basis for parallelizing kernel training by loading data and weight matrices onto a GPU where linear calculations can happen orders of magnitude faster. Some research has also been done on understanding the phenomenon of transition to linearity of certain neural networks as their width approaches infinity (Liu et al., 2020). This provides a new perspective on the NTK's role as an approximator of (linear) neural networks. However, the transition to linearity only holds when the last layer of the neural network is linear.

Previous work has also been conducted to develop kernel machines that learn features. Since kernel machines are classical learning models that are generally easy to implement (Adityanarayanan Radhakrishnan, 2022). Neural Tangent Kernels are a specific kernel that have been effective at classifying images but have under performed compared to neural networks. Recursive Feature Machines aim to learn features and become data-adaptive to estimate a predictor with a kernel machine and average gradient outer product to update the feature matrix which causes feature learning.

1.3 Datasets

We will be using a text novel dataset in this project:

1. 1984 (Orwell): 1984 by George Orwell is a ebook novel with 9456 lines with 64 characters each and utilizes a context size of 48 characters.

The text will be webscraped to extract raw words from the document and build a vocabulary of 50 words with alphanumeric characters to tokenize the text. The characters will be one hot encoded to create a matrix with the number of samples for dimensions rows (N) and the token size (64) by vocabulary size (50).

2 Methods

For points that are randomly distributed, classifying them using a linear classifier is a complicated task as they are generally not linearly separable. The problem of linear separability scales with the dimensionality of the dataset, as linear separability must produce a hyperplane as a linear combination of all dimensions.

Kernel methods come in handy in these situations because they transform the points with respect to the rest of the dataset before classification. The basic structure of a radial kernel (the focus of this paper) is as follows:

$$K(x, y) = \exp\left(\frac{-||x - y||_{\Omega}^{t}}{\gamma}\right)$$

The key parameters in this kernel (Ω, t, γ) can be modified to produce different kernels. For example the Laplacian kernel (which is used in all experiments in this paper) is created when $\Omega = 1$ and $t = 1$. The Gaussian kernel (sometimes called the RBF kernel), is produced when $\Omega = 2$ and $t = 2$. γ is the bandwidth parameter – it governs how "wide" the kernel is, and will affect the classification performance based on the dataset.

After rewriting our data points using the formula above, we get regression model weights:

$$\hat{\alpha} = (K(X, X) + \lambda I_n)^{-1} y$$

Where λ is a regularization parameter. The above formula also gives us some insight into the effect of γ; namely, when $\gamma \to 0$, $K \to I$, which causes $\hat{\alpha} \to y$. On the other hand, when $\gamma \to \infty$, $K \to \mathbf{1}$.To get the training and test predictions, we use $\hat{\alpha}$ as follows:

$$\hat{y} = \text{sign}(K(X, X)\hat{\alpha})$$
$$\hat{y'} = \text{sign}(K(X', X)\hat{\alpha})$$

Note that these pairwise kernel machines require the storage of the entire training dataset X, even for inference. This imposes a severe performance restriction on them as $K \in \mathbf{R}^{n \times n}$, which when n is large, makes it difficult to store K on most modern computers, much less invert in order to find $\hat{\alpha}$. In future portions of this project we will explore more advanced kernel machines that use random fourier features to approximate K using Monte-Carlo sampling. However, one should note that these kernel machines are highly efficient when d is large and n is small, since they effectively reduce the dimensionality (and therefore the size) of the dataset in such domains.

In deep neural networks, the *expected gradient outer product* is a statistical estimator for feature selection in feature learning. Recursive Feature Machines (RFMs) are a class of kernel machines that learn features using this expected gradient outer product as denoted below.

$$\Gamma(g) = \frac{1}{n} \sum_{i=1}^{n} \nabla g(x_i) \nabla(x_i)^T \in \mathbb{R}^{d \times d}$$

Kernel machines also must use a kernel function in the form K: $\mathbf{R}^d \times \mathbf{R}^d \to \mathbf{R}$ so with that kernel function and a dataset (X,y) $\in \mathbb{R}^{d \times n} \times \mathbf{R}^{1 \times n}$ the predictor $\hat{f}$ is depicted as below:

$$\hat{f} = \hat{\alpha} K(X, x); \hat{a} = y K(X, X)^{-1}$$

M will be a positive semi-definite and symmetric matrix that will be a learnable parameter in the kernel function. We then consider kernels of the form

$$K_M(x, z) := \phi(||x - z||_M).$$

Now the kernel regression with the kernel function K_M can be used estimate a predictor then using the average gradient outer product to update the feature matrix M.

Using the algorithm above, we analyze the *1984* dataset by predicting a specific character based on previous characters. We use a weighted bag of word method in which we start by tokenizing our text into characters. We then encode the characters into numerical representations by assigning vectors to each characters. When predicting a character, we also assign more importance/weight to characters closer to the the item we are trying to predict.

2.1 N-Gram Model Baseline

The N-Gram model is a language model that finds the probability distribution over word sequences. We will develop two N-gram models of 3 and 4 grams respectively which calculates the probability of a word based on the previous 3 words. It will utilize a context size of 48 words which is the window of words to predict the context of each word.

2.2 Laplacian Kernel Method

The Laplacian kernel machine utilizes an encoded matrix where each token is replaced with its index in the vocabulary. Then it is one hot encoded into multiple matrices split for training and testing data and its respective labels. We can obtain the sequence of the next characters and train the Laplacian kernel by finding the solution of the kernel and the next character and call this â. We then obtain the resulting matrix by solving that kernel at â to obtain ŷ. We can plot this ŷ to visually see the next-character accuracy in the figure below.

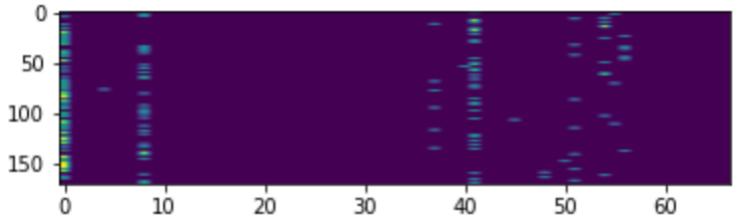

Figure 1: Next Character Accuracy Plot

We then calculate this ŷ with the context size for the train dataset and decode the text and slide the window forward and return the transpose for the results. We then join it all together to receive a text of the results and append the Bleu Score and Perplexity metrics to our findings.

2.3 Recursive Feature Machine

The Recursive Feature Machine requires the Training features *X*, training labels *y*, Kernel Function *K* and number of iterations *T*. We create the feature matrix called *M* that is a prominent feature of neural networks that is a square matrix whose size is *d x d* where *d* is the number of features. We then train the Kernel Function *K* with *M*.

$$K_M(x,z) = \exp\left(\frac{-\|x-z\|_M}{\sigma}\right) \qquad \text{where } \|x-z\|_M := \sqrt{(x-z)^T M (x-z)}$$

$$\text{Thus,} \quad \nabla K_M(x,z) = \frac{Mx - Mz}{\sigma \|x-z\|_M} K(x,z)$$

We update M where $M = \frac{1}{n}\sum_{x \in X} \nabla f(x) \nabla f(x)^T$ where $\nabla f(x) = \alpha \nabla K_M(X,x)$. We then cross-validate and repeat the previous steps until it converges. After training, we receive an α and generate a text kernel with the kernel RFM function, alpha, and the training and testing encoded matrices. We join the output and create a list of text that we can evaluate our bleu and perplexity scores on.

2.4 Scoring Metrics

Bleu Score - Bilingual Evaluation Understudy is a metric to measure and compare the similarity of translated texts ranging from 0 (perfectly mismatched) to 1 (perfectly matched). It compares the original and translated corresponding n-grams to each other and averages all the n-grams to compute the final Bleu Score.

Perplexity - A metric that measures the amount of "chaos" that exists in the text dataset. The lower the perplexity score is, the more "predictable" the generation of the text is. Conversely, the higher the perplexity score, the more likely the less "predictable" the generation of the text is.

3 Results

3.1 Score Comparison

Comparing the RFM to the bigram/trigram model and the Laplacian kernel model, we get the results in the table below:

Results

	N-gram	Laplacian	RFM
Bleu Score	0.449	0.453	**0.472**
Perplexity	15.36	9.08	11.53

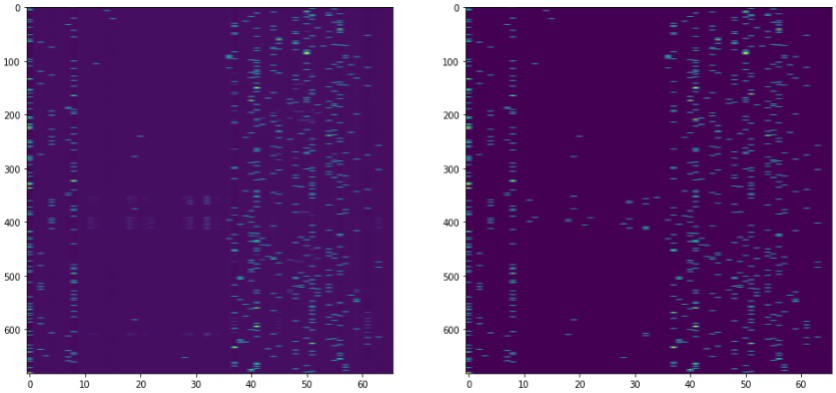

Figure 2: Next Character Accuracy Plot for RFM

3.2 Scaling Test Findings

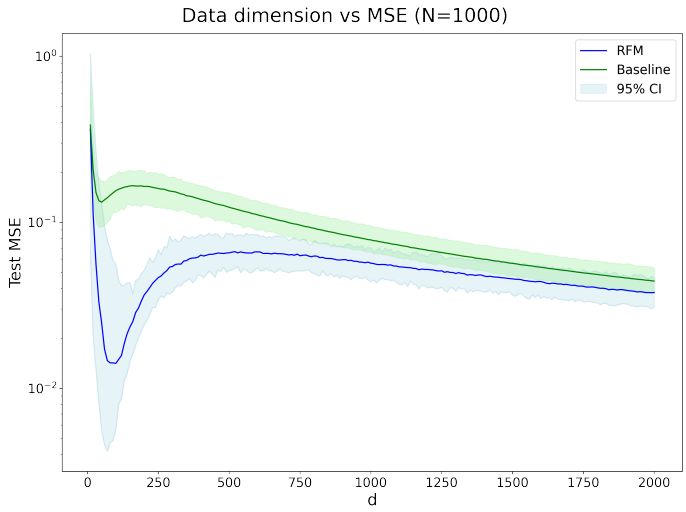

Figure 3: Scaling Plot Test with Baseline

4 Conclusion

4.1 Baseline Methods vs RFM Results

From these results table between the three methods, we see that the RFM has the highest Bleu score, meaning that it performed best at predicting text. However, the Laplacian Kernel Machine had the lowest perplexity meaning that it produced the most "predictable" text out of the three methods.

4.2 Recursive Feature Machine Next Character Accuracies

Figure 2 compares the next character accuracy of the RFM generated characters (left) with the actual next character accuracy of the original text (right). Each row corresponds to a one-hot encoded vector denoting the next character in the sequence, with lighter colors corresponding to a higher predicted probability.

4.3 Scaling Test Findings

In Figure 3, we wanted to understand how the performance of an RFM scaled with the model and feature size. In order to do so, we generated 1000 random datapoints $x_i \in R^d \to X$ and used the following target function: $f(x) = 5x_1^3 + 10x_2^2 + 2x_3$ We then trained an RFM and a Laplacian kernel and calculated their respective MSEs.

What is interesting here is that the test MSE for the RFM is much smaller than the Laplacian kernel's MSE when *d* is between 20 to 200. However, when *d* is greater than 500, the performances of both models are very similar even though the RFMs MSE is still a little smaller.

More peculiarly, the MSE curves look highly irregular, as they descend first, then ascent until about $d = 500$, and then descend again. This corresponds roughly to the idea of double descent as seen in deep neural networks Nakkiran et al. (2019). The fact that the RFM appears to also exhibit this indicates that the RFM model and a classic deep neural network share similar properties that lead to interpolation in high-feature regimes.

5 Appendix

References

Parthe Pandit Mikhail Belkin Adityanarayanan Radhakrishnan, Daniel Beaglehole. Feature learning in neural networks and kernel machines that recursively learn features. 2022. URL https://arxiv.org/abs/2212.13881.

Thomas Hofmann, Bernhard Schölkopf, and Alexander J. Smola. Kernel methods in machine learning. *The Annals of Statistics*, 36(3):1171 – 1220, 2008. doi: 10.1214/009053607000000677. URL https://doi.org/10.1214/009053607000000677.

Chaoyue Liu, Libin Zhu, and Mikhail Belkin. On the linearity of large non-linear models: When and why the tangent kernel is constant. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS'20, Red Hook, NY, USA, 2020. Curran Associates Inc. ISBN 9781713829546.

Siyuan Ma and Mikhail Belkin. Kernel machines that adapt to gpus for effective large batch training, 2018. URL https://arxiv.org/abs/1806.06144.

Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: Where bigger models and more data hurt. 2019. doi: 10.48550/ARXIV.1912.02292. URL https://arxiv.org/abs/1912.02292.

George Orwell. 1984. *Planet Ebook*. URL https://www.planetebook.com/free-ebooks/1984.pdf.

5.1 Bochner's Theorem

The Fourier transform of a given distribution is a positive semi-definite function (one that is non-negative at any non-zero vector). Conversely, a positive-semi-definite function must also be a Fourier Transform of the same distribution.

k(x-x$_0$) $= k(x - z)$

k(x-x$_0$, $z - z_0$) $= k(x - z)$

$\Sigma_{ij} a_i k(x_i - x_j) a_j \geq 0$

k(x-z) = $\int e^{jt(x-z)}\, d\mu(t)$

$\mu(t) = \int k(s) e^{-jst}\, ds$

E$_\mu e^j T(x - z)$

T $\mu(cdf)$

$E_\mu e^j T(x - z)$ is the characteristic function of T

5.2 Neural Network Gaussian Process

In a neural network Gaussian process, the kernel function is used along with the neural network to make predictions based on inputs during inference.

When a Wide Neural Network is randomly initialized, it approximates to a Neural Network Gaussian process We also only train the last layer of this network.