Deep learning for sentiment analysis of movie reviews

Hadi Pouransari
Stanford University

Saman Ghili
Stanford University

Abstract

In this study, we explore various natural language processing (NLP) methods to perform sentiment analysis. We look at two different datasets, one with binary labels, and one with multi-class labels. For the binary classification we applied the bag of words, and skip-gram word2vec models followed by various classifiers, including random forest, SVM, and logistic regression. For the multi-class case, we implemented the recursive neural tensor networks (RNTN). To overcome the high computational cost of training the standard RNTN we introduce the low-rank RNTN, in which the matrices involved in the quadratic term of RNTN are substituted by symmetric low-rank matrices. We show that the low-rank RNTN leads to significant saving in computational cost, while having similar a accuracy as that of RNTN.

1 Introduction

Sentiment analysis is a well-known task in the realm of natural language processing. Given a set of texts, the objective is to determine the polarity of that text. [9] provides a comprehensive survey of various methods, benchmarks, and resources of sentiment analysis and opinion mining. The sentiments can consist of different classes. In this study, we consider two cases: 1) A movie review is *positive* (+) or *negative* (-). This is similar to [2], where they also employ a novel similarity measure. In [10], authors perform sentiment analysis after summarizing the text. 2) A movie review is *very negative* (- -), *somewhat negative* (-), *neutral* (o), *somewhat positive* (+), or *very positive* (+ +).

For the first case, we picked a Kaggle[1] competition called "Bag of Words Meets Bags of Popcorn". The challenge consists of two main parts. In the first part, we try a variety of basic sentiment analysis techniques. This provides a reasonable baseline to asses further complex methods. In the second part, we try different variants of the basic models. The objective of this part is to train a binary classifier for movie reviews (i.e., output classes are positive/negative). As in many natural language tasks, the first task here is to clean up, and convert the input texts (movie reviews) into numbers. This can be done using a variety of methods such as *bag of words*, *word to vector*, etc. Afterwards, we train the classifier.

For the second case we used the data set used in [6]. Each example in this dataset (both training and test) is decomposed using a recursive tree. We use different variations of the recursive neural networks (RNN) (see [3, 4]) for the classification. Unlike the first case, here we have a multi-class (5 classes) problem.

[1] `www.kaggle.com`

2 Datasets

2.1 Binary classification dataset

We use the data provided in [1], which is publicly available on Kaggle. Here is a description of the data, provided by Kaggle:

> The labeled data set consists of 50,000 IMDB movie reviews, specially selected for sentiment analysis. The sentiment of reviews is binary, meaning the IMDB rating < 5 results in a sentiment score of 0, and rating ≥ 7 have a sentiment score of 1. No individual movie has more than 30 reviews. The 25,000 review labeled training set does not include any of the same movies as the 25,000 review test set. In addition, there are another 50,000 IMDB reviews provided without any rating labels.

2.2 Multi-class dataset

To study RNNs we need to have the reviews stored in the form of parse trees. We used parse trees in the tree bank introduced in [6]. The training set contains 8544 reviews labeled with - -/-/o/+/+ +, stored in labeled recursive trees. Test set contains 1101 reviews.

3 Technical Approach and Models

3.1 Binary classification approaches

We have tried several basic sentiment analysis methods as described below. For all methods, we performed a preprocessing step to clean up the data. This includes removing the HTML tags (using the Python package BeautifulSoup), unnecessary punctuation (using the Python package Regular-Expression), convert to lower case, and removing stop words[2] if needed (Python Natural Language Toolkit).

To convert a cleaned sequence of words to numerical feature vectors we tried the following methods:

- Bag of Words (BOW)
  Bag of words is probably the simplest way to numerically represent texts. Given a text T, we assign a vector $v_T \in \mathbb{N}^d$ to it, such that v_{Ti} is the number of times the i'th word of the vocabulary has appeared in the text T. d is the size of our vocabulary, which consists of all words in the set of reviews except for very rare words (we use the 5000 most frequent words). After learning the BOW vectors for every review in the labeled training set, we fit a classifier to the data.
- Word2Vec
  Another way to numerically represent texts is to transform each word of the text to a vector [7]. This transformation should preserve the semantics of words, that is if the meanings of two words are close, their vectors should be close as well (in an L_2-distance sense). One important aspect of the `word2vec` task is that it is independent of the main objective (here sentiment analysis), and does not require a labeled dataset. Therefore, here we used all of the 75,000 reviews (25,000 labeled and 50,000 unlabeled training sets) as the corpora to train word vectors. Aside from the regular preprocessing steps on the raw reviews, to train word vectors we need to split paragraphs into sentences, since `word2vec` algorithm accepts sentences as input (since words not in the same sentence as the current word are not part of its relevant context).

Note that for the sentiment analysis we need a feature vector for each review. This can be done through a variety of methods. Here, we explain some basic methods that we have tried so far.

- Words to reviews: Averaging
  Perhaps the simplest way to assign a feature vector to a set of words (a review) is to average

[2]Stop words are highly frequent words such as ”the”, ”is”, etc. that do not have specific semantics. The NLTK package recognizes the stop words through a statistical analysis on some large corpora.

the word vectors of all words. We ignore the stop words in the averaging process, since they do not contribute to the polarity of the whole paragraph. After assigning a feature vector to all reviews, we train a binary classifier using the labeled training set.

- Words to reviews: Clustering
  Having the words transformed to vectors, we can find similarities (distances) among different words. This allows us to cluster similar words together. Using the K-means algorithm we form clusters of words (with the average size of 5 words per cluster) that are represented by a centroid. Now, we can represent each review as a set (bag) of centroids, similar to the bag of words idea. After learning the bag of centroids vectors for every review in the labeled training set, we fit a classifier to the data.

Once we have the feature vectors for each review, we use a binary classifier to learn the sentiments. We applied the following classifiers:

- Random forest
- Logistic regression
- SVM

3.2 Recursive neural networks

Even though the above models may result in reasonable accuracies, all of them suffer from loosing the order of words appearing in a sentence, and therefore cannot capture delicate semantics from the input reviews. Recursive neural networks, on the other hand, allow us to account for the order of words in a sentence.

Although RNNs can achieve relatively high classification accuracies, they RNNs still cannot express certain relations, such as negated positives or negated negatives. Finding extensions to the basic RNN that are capable of dealing with negation errors is an active area of research. Recursive matrix-vector spaces [5] and recursive neural tensor networks (RNTN)[6] are two examples of such works. In this study, we experimented with RNTNs, and developed a more numerically efficient extension to RNTN called **Low-Rank RNTN**.

Here, we briefly describe the RNTN approach. In RNTN each node in the parse tree is represented by a hidden vector of length d (similar to word2vec). The hidden vectors at the leaf level (i.e., vectors corresponding to single words) are in fact the word-vectors. For a non-leaf node, the hidden vector is obtained from the hidden vectors of its left and right children through the following process:

$$p = f\left(\begin{bmatrix} b \\ c \end{bmatrix}^{\mathsf{T}} V^{[1:d]} \begin{bmatrix} b \\ c \end{bmatrix} + W \begin{bmatrix} b \\ c \end{bmatrix}\right), \tag{1}$$

where we assume p is the hidden vector of the parent node, and b and c are the hidden vectors of the children. W has size `(d,2d)` , and the tensor V has a `(d,2d,2d)` shape. Function f is the activation function of the neurons, which is assumed to be tangent hyperbolic `tanh` here. The term involving the tensor V is designed to produce non-linear (quadratic) features. This can help to overcome the issue of negated phrases.

However, tensor V introduces an extra $\mathcal{O}(d^3)$ new parameters to the model. This is not favorable for several reasons. Firstly, it is computationally expensive (both in terms of requires FLOPS and required memory). Secondly, having a lot of parameters makes the optimization problem much more difficult to solve, and increases the risk of over-fitting.

One remedy for the above issues is to use low-rank approximations of matrices $V^{[t]}$ for $t = 1, 2, \ldots, d$. For each t consider a matrix $U^{[t]}$ with size `(2d,r)` where $r \ll d$, and replace $V^{[t]}$ with $U^{[t]}U^{[t]\mathsf{T}}$. Note that the conventional RNN and the standard RNTN are both special cases of low-rank RNTN with ranks 0 and $2d$, respectively. This has two important consequences. One consequence is that the matrices $V^{[t]} = U^{[t]}U^{[t]\mathsf{T}}$ remains symmetric positive semi-definite. More importantly, the number of extra parameters is reduced from $\mathcal{O}(d^3)$ to $\mathcal{O}(rd^2)$. When we consider full-rank matrices (the original RNTN method) the bottle-neck of the calculation comes from the quadratic term ($\mathcal{O}(d^3)$). Therefore, reducing the number of parameters from $\mathcal{O}(d^3)$ to $\mathcal{O}(rd^2)$ accelerates the calculations significantly.

As a future extension of this work, one can exploit the low-rank representation of tensors (see [11]) instead of matrices. For example, consider the `(d,2d,2d)` tensor V. Using three vectors x, y, and z (with sizes d, $2d$, and $2d$, respectively) we can write the first rank representation of V, as follows:

$$V_{i,j,k} = x_i y_j z_k \tag{2}$$

This reduces the number of parameters required for the quadratic term even further to $\mathcal{O}(d)$. However, the linear term in (1) has still $\mathcal{O}(d^2)$, when a full-rank matrix W is used.

Note that the back-propagation step of the Low-Rank RNTN is almost identical to the original RNTN. The only new derivatives that we need to compute are the derivatives of cost (J) with respect to the matrices $U^{[t]}$, for $t = 1, 2, \ldots, d$, which are as follows:

$$\frac{\partial J}{\partial U^{[t]}} = \left(\left[\frac{\partial J}{\partial V^{[t]}} \right] + \left[\frac{\partial J}{\partial V^{[t]}} \right]^{\mathsf{T}} \right) U^{[t]}. \tag{3}$$

4 Results

4.1 Binary classification results

In this section, we report the baseline results (i.e., accuracy on the test data set) we obtained using the above methods, with a different classifiers (using the Python package scikit-learn).

The performance of a classifier on the movie reviews can be evaluated by looking at the fraction of correctly classified reviews from the test set. This is done through the submission framework of Kaggle, which evaluates the model on the aforementioned test set.

In the following we present results we obtained from different models, as well as the results of hyper-parameter tuning. For hyper-parameter tuning we used hold-out cross validation.

- **Bag of Words + Random Forest Classifier**: Obtained accuracy = 0.84352
- **Bag of Words + SVM/Logistic regression** : To achieve the best possible accuracy, we first tried elastic-net [9] regularization, in which we add the following penalty term to the cost function:

  $$\lambda \left((1-\alpha) \|\mathcal{P}\|_2^2 + \alpha \|\mathcal{P}\|_1 \right) \tag{4}$$

  Where α is the *elastic-net ratio* which specifies the relative importance of the L_1 and L_2 penalty terms, and $\mathcal{P}$ is the vector of all parameters. Figure 1(left) shows the development set accuracy versus elastic-net ratio, α. Evidently, for the BOW model $\alpha = 0$ (i.e., plain L2 regularization) is optimal.

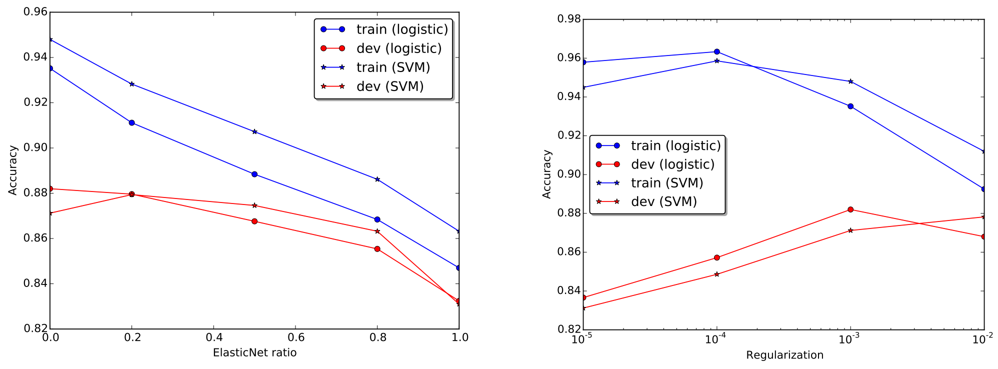


Figure 1: Left: Tuning the elastic-net ratio, α. Right:Tuning the regularization parameter, λ.

Now, with $\alpha = 0$, we tuned the hyper-parameter λ in equation (4). The accuracy of the development and training sets are shown in figure 1(right).

- **Word2Vec + Averaging + Random Forest Classifier**: In order to train the word vectors using the skip-gram model, we need to pick a context-size and a a dimension for the word-vectors. To do so, we did hyper-parameter tuning on the development set. The results are shown in figure 2. As expected, for the sentiment analysis problem, which is more about semantics than syntax, the larger the context-size the more accurate the classifier. Also, it is evident from the result that a word-vector dimension of 100 is sufficient for our purpose.

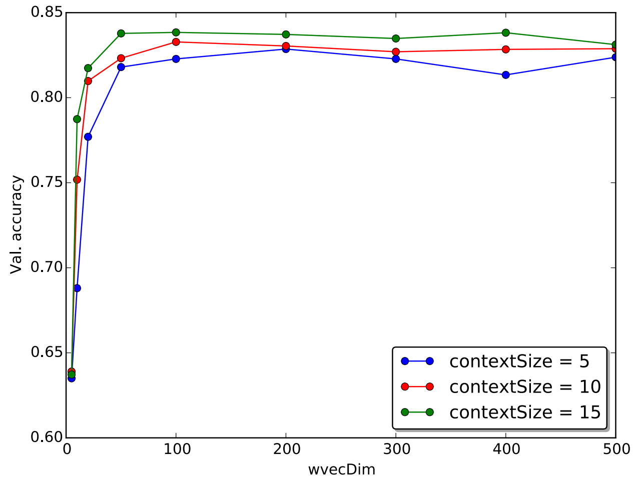


Figure 2: Tuning the skip-gram model hyper-parameters.

The final test accuracy using random forest, SVM, and logistic regression classifiers are 84.0%, 85.8%, and 86.6% respectively.

- **Word2Vec + Clustering + Random Forest Classifier**: Obtained accuracy = 0.83528, which is not a gain compared to averaging method.

4.2 Recursive neural networks result

We begin with our results on the standard recursive neural tensor network (RNTN). The history of the training and development errors as a function of number of epochs is illustrated in figure 3(left). We used *adaGrad* stochastic gradient descent with a mini batch-size of 30, and L_2 regularization with a strength of 10^{-6}, and a learning rate of 10^{-2}. As it can be seen from the figure, after 100 epochs the optimization does not seem to have converged yet. Note that due to the quadratic term, each step of the optimization is very expensive (this will be quantified in the next section).

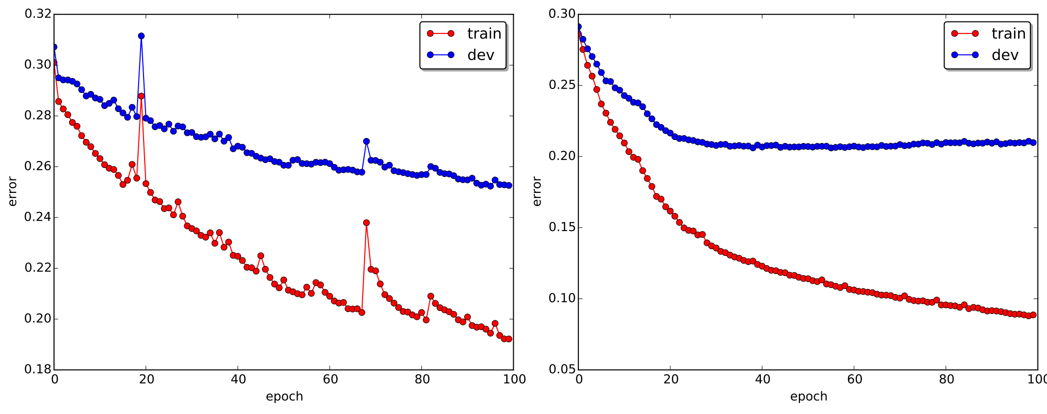


Figure 3: Evolution of the training and development errors for the standard RNTN (left) and the low-rank RNTN with rank = 5 (right)

As opposed to the standard RNTN for which the optimization procedure involves a large number of (expensive) steps, the low-rank RNTN converges relatively fast, i.e., fewer (and cheaper) epochs. Figure 3(right) depicts the history of training and development errors for a low-rank RNTN with rank 5. The optimal development accuracy is reached after roughly 40 epochs, which is much smaller compared to that of standard RNTN. Note that obtaining the best accuracy needs some thorough hyper-parameter tuning on the learning rate, regularization strength, and rank.

Now, we establish the performance of the low-rank RNTN and compare it to that of the standard RNTN. Figure 4 shows the time spent on each epoch in the optimization process as a function of the rank.

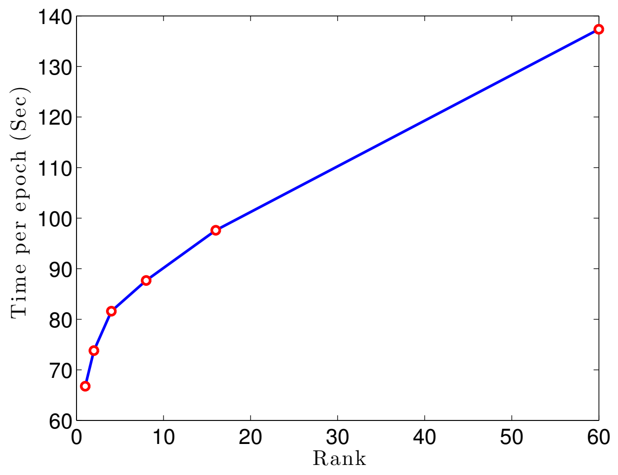


Figure 4: Elapsed time per epoch in the optimization of low-rank RNTN for various ranks .

4.3 Ensemble averaging

In this section, we show how we can take advantage of the fast and cheap convergence of the low-rank RNTN method by training multiple models, and ensembling the predictions.

Suppose we are given n models, $M_1, M_2, \ldots, M_n$, and we want to form a combined model (in our study $n = 10$). Also, assume there are a total of m classes (here, $m = 5$).

For a given image, x and a model M_i, $\mathbb{P}(x = y_j; M_i)$ represents the probability that the input image x is classified as y_j $(j = 1, 2, \ldots, m)$ under the model M_i. The simple ensemble average is then:

$$\mathbb{P}(x = y_j; \{M_1, \ldots, M_n\}) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{P}(x = y_j; M_i) \qquad (5)$$

In Figure 5 we show the ensemble model development accuracy, when k different models are combined. For each k there are $\binom{10}{k}$ possible combinations of models. Each point on the plot shows the accuracy of one combination (this accuracy is normalized by dividing by the average of all 10 models). We can see from the figure that ensemble-averaging leads to an about 1.5% improvement in accuracy. In figure 6 the confusion matrix of the combined model (i.e., $k = 10$) is plotted.

5 Conclusion

In this work, we studied a wide range of NLP classification models. Our investigations consisted of two main parts.

In the first part, we used the dataset provided by Kaggle and applied the bag of words, and skip-gram word2vec models to represent words numerically. We then used several classifiers, including random forest, SVM, and logistic regression to perform the binary classification task. When we use

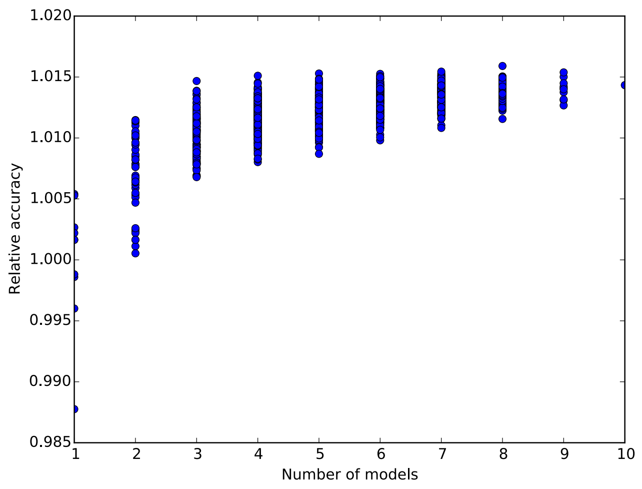
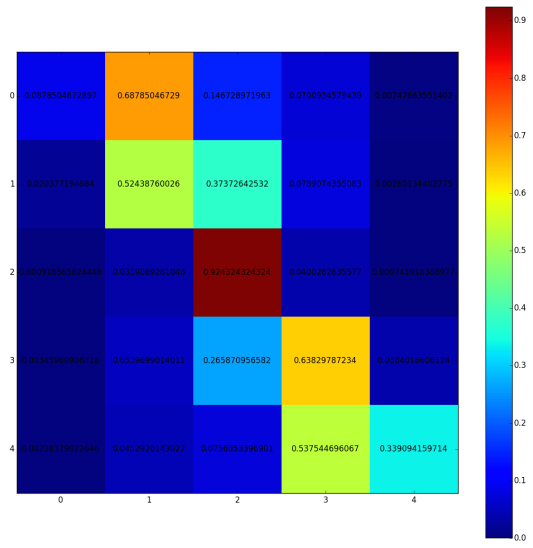


Figure 5: Development accuracy of the ensemble model using different number (and combination) of models.



	0	1	2	3	4
0	0.0878504672897	0.68785046729	0.146728971963	0.0700934579439	0.00747663551402
1	0.020377194884	0.52438760026	0.37372642532	0.0789074355083	0.00260134402775
2	0.000918565624448	0.0339869281046	0.924324324324	0.0400282635577	0.000741918388977
3	0.00345960906418	0.0539699014011	0.265870956582	0.63829787234	0.0384016606124
4	0.00238379022646	0.0452920143027	0.0756853396901	0.537544696067	0.339094159714

Figure 6: The confusion matrix of the combined model.

these classifiers, one of the challenges is to aggregate word vectors into a single feature vector for each review. We tried vector averaging, and clustering to produce the aggregated feature vectors. However, these suffer from loosing the order of words in sentences. This motivated the second part of our work.

In the second part, we implemented the recursive neural tensor networks to train a multi-class sentiment analyzer. The training of standard RNTN was computationally very expensive. This motivated us to introduce the low-rank RNTN. We showed that the low-rank RNTN can achieve comparable accuracies to that of standard RNTN much faster. This better training performance of the low-rank RNTN enables us to train several (in this case 10) different models, and use them for ensemble-averaging. A 1.5% accuracy improvement was achieved by ensemble-averaging.

References

[1] Maas, Andrew L., et al. ”Learning word vectors for sentiment analysis.” Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1. Association for Computational Linguistics, 2011.

[2] Pang, Bo, and Lillian Lee. ”Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales.” Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics. Association for Computational Linguistics, 2005.

[3] Socher, Richard, et al. ”Parsing natural scenes and natural language with recursive neural networks.” Proceedings of the 28th international conference on machine learning (ICML-11). 2011.

[4] Socher, Richard, et al. ”Semi-supervised recursive autoencoders for predicting sentiment distributions.” Proceedings of the Conference on Empirical Methods in Natural Language Processing. Association for Computational Linguistics, 2011.

[5] Socher, Richard, et al. ”Semantic compositionality through recursive matrix-vector spaces.” Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning. Association for Computational Linguistics, 2012. APA

[6] Socher, Richard, et al. ”Recursive deep models for semantic compositionally over a sentiment treebank.” Proceedings of the conference on empirical methods in natural language processing (EMNLP). Vol. 1631. 2013.

[7] Mikolov, Tomas, et al. ”Distributed representations of words and phrases and their compositionality.” Advances in Neural Information Processing Systems. 2013.

[8] Zou, Hui, and Trevor Hastie. ”Regularization and variable selection via the elastic net.” Journal of the Royal Statistical Society: Series B (Statistical Methodology) 67, no. 2 (2005): 301-320.

[9] Pang, Bo, and Lillian Lee. ”Opinion mining and sentiment analysis.” Foundations and trends in information retrieval 2, no. 1-2 (2008): 1-135.

[10] Pang, Bo, and Lillian Lee. ”A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts.” In Proceedings of the 42nd annual meeting on Association for Computational Linguistics, p. 271. Association for Computational Linguistics, 2004.

[11] Beylkin, Gregory, Jochen Garcke, and Martin J. Mohlenkamp. ”Multivariate regression and machine learning with sums of separable functions.” SIAM Journal on Scientific Computing 31, no. 3 (2009): 1840-1857.