Supervised Feature Selection using Sparse Evolutionary Training and Neuron Strength

Karolis Girdžiūnas

University of Twente
P.O. Box 217, 7500AE Enschede
The Netherlands

July 11, 2021

Abstract: Feature selection has been used to battle the ever-increasing dimensionality of datasets used for machine learning applications. Many feature selection methods, such as the Chi-Square test and Laplacian score, determine feature importance via a hand-crafted metric, often tailored to a specific type of dataset. This paper proposes a method for deciding feature importance by training a supervised sparse neural network model using *Sparse Evolutionary Training* and scoring features depending on *Neuron Strength*. The features are selected in one shot after a network has been trained and, can outperform Chi-Square test feature selection, performing best in image recognition tasks.

1 Introduction

Feature selection is an important field of research of data-driven learning. It focuses on finding the ”best” subset of features for classification, clustering or regression tasks. Depending on the type of dataset used, feature selection can be categorized as supervised, semi-supervised or unsupervised [1]. Semi-supervised and unsupervised learning refers to learning from data points that have (partially) unknown ground truth labels, as opposed to supervised learning, in which the ground truth labels are all known.

Feature selection is often used to find a smaller subset of variables while still capturing the essential information for label prediction. This becomes essential when analyzing very high-dimensional data, as it is not uncommon to run into computational limitations both in memory and speed when considering all, primarily redundant, features [1].

Data features used for training machine learning models have a significant impact on the performance achieved. Irrelevant or partially relevant features can have a negative influence over model performance as well as introduce memory bottlenecks. Feature selection can aid in the following:

- avoid overfitting and improve model performance
- provide faster and more cost-effective models
- gain a deeper insight into the underlying processes that generated the data [2]

A standard tool for building models for high-dimensional data is to pass it through an artificial neural network. Today’s computational advancements in efficient matrix computation, parallelization on GPUs and offloading the compute to the cloud have allowed for the training of deep neural network architectures for very high dimensional datasets. Artificial Neural Networks (ANNs) are among the most successful artificial intelligence methods today. The use of ANNs has led to significant breakthroughs in deep reinforcement learning [3], computer vision [4], natural language processing [5] and more [6]. Though made infeasible for applications for autonomous agents as the mentioned methods require extensive computing facilities [7].

1.1 Reducing the size of the model

To reduce the number of parameters in an ANN, the connections can be pruned after training to reduce the parameter count by over 90% while maintaining model accuracy [8]. Though such methods still require the training of an initially fully-connected dense neural network to make use of the overparameterizing power of neural networks in the training phase. This aids in reducing compute in the inference stage of the

model, though still requiring the whole computation to train a dense neural network.

1.2 Sparse neural networks

Mocanu et al. [9] propose the Sparse Evolutionary Training method for training a sparse neural network from scratch, starting with a randomly initialized graph and pruning connections after each training epoch based on weight magnitude. The same proportion of pruned connections are regrown at random in the network, maintaining a fixed sparsity level but dynamically changing its architecture.

Atashgahi et al. [10] propose a method for extracting important features, called QuickSelection, by considering the combined weights of each input neuron and assigning importance based on this metric. The selected features proved to help find the best subset of features for unsupervised learning problems implemented using an autoencoder neural network.

1.3 What this paper focuses on

This paper combines Sparse Evolutionary Training and Neuron Strength used in the QuickSelection algorithm as the importance metric to perform supervised feature selection, in contrast to the unsupervised feature selection proposed in [10].

The paper aims to answer the following research questions:

- Can Sparse Evolutionary Training (SET) in combination with Neuron Strength be used to perform supervised feature selection on various datasets?
- How do the selected features using SET compare to standard statistical methods for feature selection?

2 Background

2.1 Feature selection

Feature selection aims to select a subset of features from a dataset to reduce the memory and computation time required for the model, capturing relevant features while discarding redundant or insignificant features. In many classification problems, it is challenging to learn a good classifier if the dataset is riddled with redundant/irrelevant features. Reducing the number of features can drastically reduce training time and yield a more general classifier [11].

Feature selection methods come in several forms: filter, wrapper and embedded methods. The filter method selects features based on an importance score assigned to each feature, selecting only k highest scoring features. Examples of scoring (importance) metrics of each feature include information gain, the correlation between features and labels, Chi-Square score, Fisher score, Mutual Information.

Wrapper methods approach feature selection as a search problem, attempting to determine the best combination of features for a classification [11] or clustering [12] problems. Wrapper methods generally follow three steps [11]:

1. Search for a subset of features,
2. Evaluate the selected subset by the performance of the classifier,
3. Repeat 1. and 2. until the desired performance is reached.

Embedded methods rely on extracting important features while the model is being trained. A common type of embedded method is regularization, introducing constraints on certain features and reducing overfitting.

2.2 Sparse Evolutionary Training

Sparse Evolutionary Training, as proposed by Mocanu et al., uses a dynamically changing neural network architecture with a fixed sparsity level.

A sparse artificial neural network is initialized as a Erdős–Rényi random graph [9], in which the probability of a weight existing from i^{th} neuron in the $(k-1)^{th}$ layer to the j^{th} neuron in the k^{th} layer is given by:

$$p(W_{ij}^k) = \frac{\varepsilon(n^k + n^{k-1})}{n^k n^{k-1}}, \tag{1}$$

where n^k refers to the number of neurons in the k^{th} hidden layer, and ε controls the sparsity.

After each training epoch, a fraction ζ of the smallest weights in magnitude is removed from each layer. The same proportion of weights is then randomly reinitialized in each layer to provide a fixed sparsity level. Once the loss of the model converges, the training is stopped.

2.3 QuickSelection

Atashgahi [10] performs unsupervised feature selection by training a sparse denoising autoencoder using Sparse Evolutionary Training. To select the important features from the trained network, a *Neuron Strength* metric for determining feature relevance is proposed. The *Neuron Strength* is defined as the sum of the absolute values of weights outgoing from a given input neuron:

$$s_i = \sum_{j=1}^{n^1} |W_{ij}^1|, \tag{2}$$

where n^1 corresponds to the number of neurons in the first hidden layer and W_{ij}^1 to the weights connecting i^{th} neuron in the input layer to the j^{th} neuron in the first hidden layer.

The computed strength for each input neuron is ranked from highest to lowest, and a subset of k highest-scoring neurons, corresponding to the important features, is used to construct a new dataset. It is important to note that this kind of feature selection computes the important features in one shot after a sparse neural network has been trained, allowing for computationally inexpensive feature selection provided the sparse neural network is trained efficiently.

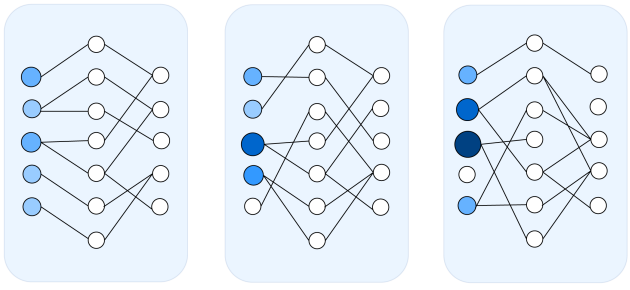

Figure 1: Overview of the "QuickSelection" algorithm, the color depth indicating the increasing strength of neurons in the input layer as the sparse topology changes during training after 5 and 10 epochs [10]

3 Related work

3.1 Sparse neural networks

There has been an extensive body of research in the field of sparse neural networks. LeCun et al. 1990 [13] propose removing unimportant weights from a neural network to improve performance. In 2015 Han et al. [14] had demonstrated a technique for retraining the network after pruning while reducing the storage and computation required by order of magnitude without affecting the accuracy.

In 2019, Frankle and Carbin [8] formalized a hypothesis, stating that if a dense neural network can be trained to a certain accuracy, a subset of that network with the same random initialization can be trained in isolation to match the accuracy of the dense counterpart. Reducing the parameter count by over 90%, drastically reducing the storage and compute requirements of the network in the inference stage. Though promising, the hypothesis does not provide steps to guessing the initial sub-network architecture and initialization that converges without training a dense neural network architecture first. Mocanu's et al. [9] proposed method addressed the problem by training a sparse neural network architecture from scratch with a fixed sparsity, removing unimportant connections at each epoch and redistributing new connections at random in the same number as had been removed.

3.2 Feature selection

Feature selection methods are used in data-preprocessing to achieve efficient data reduction. Since an exhaustive search for the best subset of features is rarely feasible, a large body of research has been conducted over the past 50 years to extract important feature sets by means of an *importance metric*.

Backwards elimination feature selection was first introduced in 1963 [15], and since then, numerous methods have focused on extracting 'important features'. The tutorial by Huang [16] provides a summary of feature selection techniques and the definitions of feature 'importance', and a survey by Jović et al. [1] elaborates on the best applications for varied learning tasks.

A common technique for feature selection is by assigning an 'importance' score to the features.

Chi-Square scoring and Information Gain is often used for text classification tasks [17], along with the commonly used Fisher score and Generalized Fisher Score for feature selection as proposed by Gu et al. [18]

3.3 Chi-Square

As a baseline to compare to, we consider features selected using the Chi-squared χ^2 score. The Chi-squared score is computed as in 3 for each feature.

$$\chi^2 = \sum_{i=1}^{N} \frac{(O_i - E_i)^2}{E_i}, \tag{3}$$

where O refers to the observation (or ground truth label) and E is the expected value for the hypothesis that the features are independent.

Again, the features are ranked according to their score, and a subset of top k features is selected.

The standard Python library *scikit-learn* provides many tools for selecting important features based on various scoring metrics using the *SelectKBest* method. For example, *SelectKBest* can select features for classification tasks based on the Chi-Square score [19], ANOVA F-value score [20] or by Mutual Information [21].

4 Proposed method

The proposed method takes several steps, initially requiring to train a neural network and selecting important features from a trained network. Feature importance is determined from the trained network's inherent weights and sparse architecture. We begin by training a supervised multi-layer perceptron with a sparse architecuture using Sparse Evolutionary Training [22] and determining feature importance via Neuron Strength metric proposed by Atashagi, used in unsupervised feature selection [10]. The important features are then used to transform the original high-dimensional dataset to a lower dimensional one for classification.

4.1 Sparse Neural Network model

We begin by training a sparse multi-layer perceptron network using Sparse Evolutionary Training on a supervised classification problem as follows:

1. Initialize a network as an Erdős–Rényi random graph, where each bipartite connection between layers has a probability of existing given by Equation 1
   For each epoch perform:
2. Forward and backward propagation minimizing loss, such as MSE given in Equation 4

$$\mathcal{L}_{MSE} = \frac{1}{N} \sum_{i=1}^{N} (\mathbf{y} - \hat{\mathbf{y}})^2, \tag{4}$$

   where $\mathbf{y}$ is the the ground truth label and $\hat{\mathbf{y}}$ the prediction label output of the neural network
3. Remove ζ of the smallest weights in magnitude and add the same number of connections at random to the network
4. Repeat until training loss converges.

Mocanu, Stone, Nguyen, et al. [9] provides additional information into the direct implementation of the SET algorithm.

The trained networks weights are stored for later feature selection.

4.2 Feature selection

Once the network has been trained, the inherited architecture and weights can be used to select important features from the dataset in a single shot.

Important features are gauged by considering the first hidden layer weights, and computing the 'strength' of each input neuron corresponding to a feature. Neuron Strength is defined in Equation 2, and each corresponding feature is ranked. From the original set of features $\mathbb{F} = \{f_1, f_2, \ldots, f_n\}$ we construct a new set of k features with the highest 'strength' $\mathbb{F}_s \subset \mathbb{F}$, $\mathbb{F}_s = \{f'_1, f'_2, \ldots, f'_k\}$, $|\mathbb{F}_s| = k$.

If the network was initially trained on a dataset $\mathbf{X} \in \mathbb{R}^{m \times n}$, where m corresponds to the number of training instances and n to the number of features in the untampered dataset.

$$\mathbf{X} = \begin{array}{c} \begin{array}{ccccc} f_1 & f_2 & f_3 & \cdots & f_n \end{array} \\ \begin{pmatrix} \_\_ & \_\_ & x^{(1)} & \_\_ & \_\_ \\ \_\_ & \_\_ & x^{(2)} & \_\_ & \_\_ \\ \_\_ & \_\_ & \vdots & \_\_ & \_\_ \\ \_\_ & \_\_ & x^{(m)} & \_\_ & \_\_ \end{pmatrix} \end{array} \tag{5}$$

then we transform the dataset $\mathbf{X}$ to a new dataset $\mathbf{X}'$, removing all features but the ones contained in the set $\mathbb{F}_s$

$$\mathbf{X}' = \begin{array}{c} \begin{array}{ccccc} f'_1 & f'_2 & f'_3 & \cdots & f'_k \end{array} \\ \begin{pmatrix} \_\_ & \_\_ & x^{(1)} & \_\_ & \_\_ \\ \_\_ & \_\_ & x^{(2)} & \_\_ & \_\_ \\ \_\_ & \_\_ & \vdots & \_\_ & \_\_ \\ \_\_ & \_\_ & x^{(m)} & \_\_ & \_\_ \end{pmatrix} \end{array} \quad (6)$$

5 Experiment and Results

5.1 Setup

The architecture used contains 3 hidden layers, containing 3000 neurons each. Each training sample minimizing the MSE loss of the network defined in Equation 4.

All hidden layers using the $ReLU(x) = max(0, x)$ activation function, except for the final output using the sigmoid $\sigma(x) = \frac{1}{1+e^{-x}}$ activation. The sparse network is trained for 40 epochs for each dataset and the input layer - first hidden layer weights are stored. The value for 40 epochs was chosen after training the network for 100 epochs, and observing convergence for loss at around 30 epochs for the chosen datasets. All hyperparameters for training the sparse neural network are presented in Table 2.

For each input neuron corresponding to a feature, we consider the strength as defined in Equation 2 as select the subset of features with the highest-ranking strengths. Original dataset $\mathbf{X}$ is transformed to only consist of columns containing highest-ranking features. With a significantly reduced dataset $\mathbf{X}'$, a classifier can be trained to predict the labels of the dataset. A support vector machine using the standard Python scikit-learn library is used with a radial basis function for the kernel for this implementation [23].

Chi-Square often being used for text classification problems [17] [24] [25], the text datasets in 1 are chosen for direct comparison to the proposed method, along with more varied datasets provided in [26].

Datasets considered are presented in Table 1.

Dataset	Examples	Features	Classes	Data type
BASEHOCK	1993	4862	2	Text
PCMAC	1943	3289	2	
RELATHE	1427	4322	2	
orlraws10P	100	10304	10	Face images
warpAR10	130	2400	10	
ORL	400	1024	40	
COIL20	1440	1024	20	
Isolet	1560	617	26	Spoken letters
madelon	2600	500	2	Artificial

Table 1: Datasets used with the number of instances, features and classes for feature selection

Hyperparameter	
# of hidden layers	3
Activation functions	$ReLU \rightarrow ReLU \rightarrow ReLU \rightarrow \sigma$
Loss function	MSE (Equation 4)
Batch size	10
Epochs	40
Learning rate	0.01
Momentum	0.9
Weight decay	0.0002
ζ *(fraction removed)*	0.3
k *(max features)*	100
ε *(sparsity)*	20

Table 2: Hyperparameters for training used for all datasets presented in Table 1

5.2 Training

The training and test accuracy for each of the datasets is presented in Figure 2, with each network's sparsity and training time presented in an Appendix in Table 8. The training time is recorded when run on a local machine with an Intel Core i5-8250U CPU @ 1.6GHz and 8Gb of RAM.

5.3 Feature selection

Up to $k = 100$ features are selected for classification using an SVM [27] with a Radial Basis Function kernel for both the features selected using both methods implemented via scikit-learn [19]. The results for classifier accuracies are presented in Figure 3. Best test accuracies and the corresponding number of features are presented in Table 3, along with the final accuracy of trained sparse multi-layer perceptron. The presented results are for general investigation when not fine-tuning the hyperparameters. For a more rigorous analysis, avoiding overfitting on the test set, a validation set should be considered.

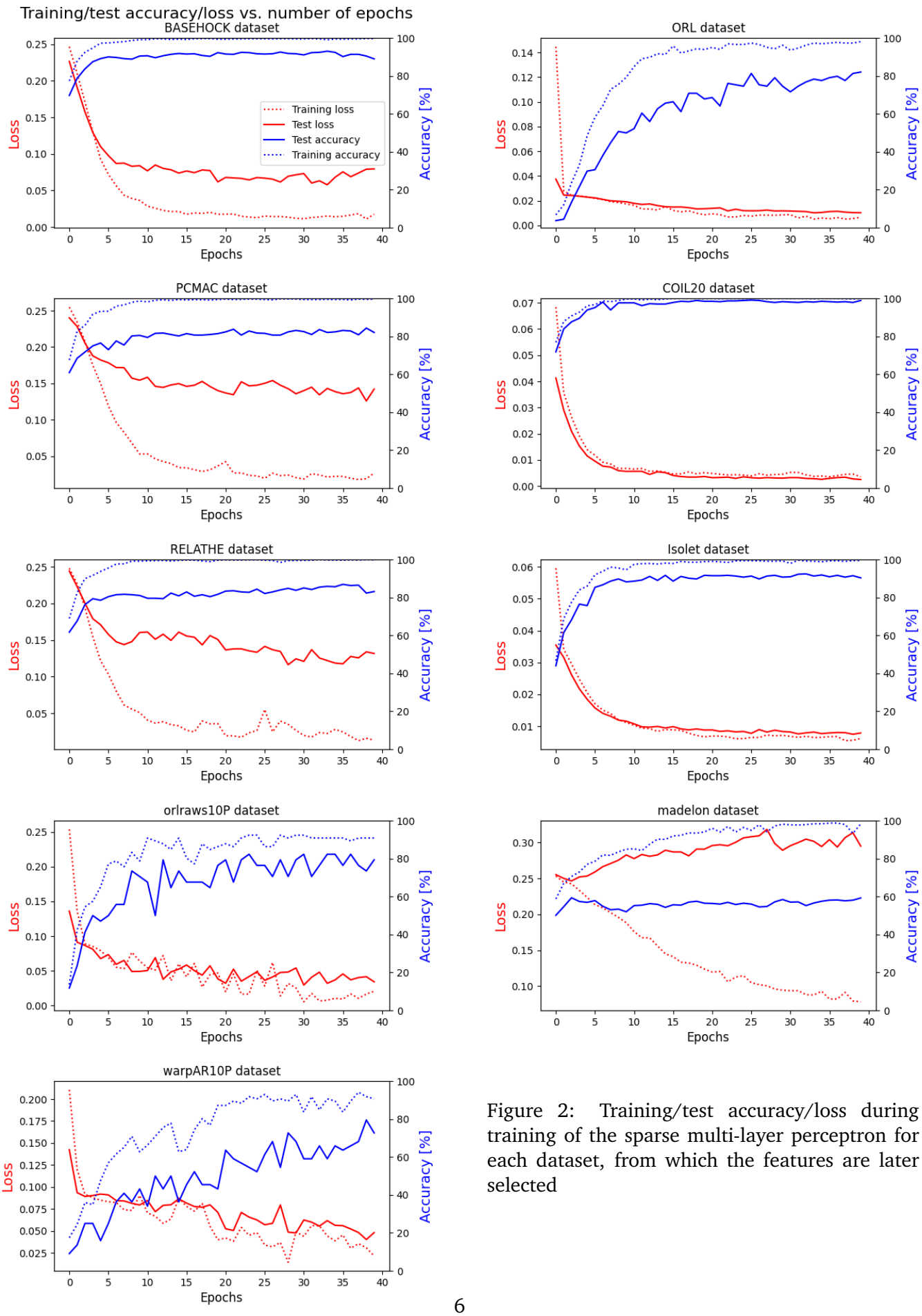

Figure 2: Training/test accuracy/loss during training of the sparse multi-layer perceptron for each dataset, from which the features are later selected

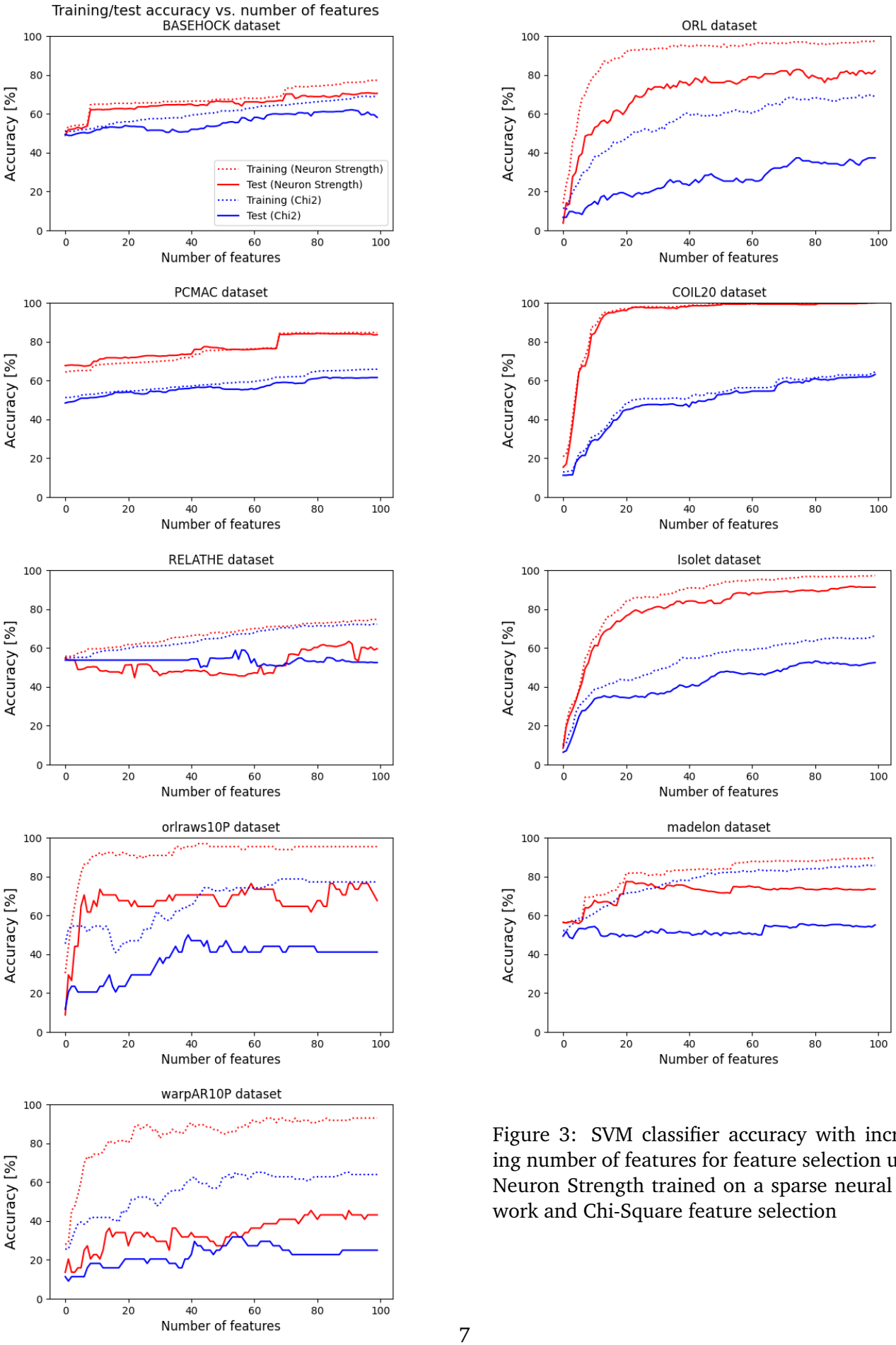

Figure 3: SVM classifier accuracy with increasing number of features for feature selection using Neuron Strength trained on a sparse neural network and Chi-Square feature selection

Dataset	Sparse MLP final test accuracy [%]	Chi-Square		Neuron Strength	
		Best test accuracy [%]	Corresponding # of features	Best test accuracy [%]	Corresponding # of features
BASEHOCK	89.02	62.11	91	71.83	96
PCMAC	**82.09**	**62.73**	**82**	**84.41**	**80**
RELATHE	83.19	59.03	56	63.45	90
orlraws10P	79.41	50.00	39	76.47	59
warpAR10P	72.73	31.82	53	45.45	78
ORL	**82.09**	**37.31**	**74**	**82.83**	**74**
COIL20	**98.96**	**63.13**	**99**	**100**	**97**
Isolet	**90.38**	**53.27**	**80**	**91.73**	**91**
madelon	**59.28**	**55.71**	**76**	**77.51**	**20**

Table 3: Best accuracies for the SVM classifiers along with the corresponding number of features achieving the best test accuracy. Comparing with the trained sparse multi-layer perceptron final test accuracy (considering all features)

6 Discussion

The proposed method for feature selection correctly identifies features that correlate to the output, with overlapping features selected by the Chi-Squared method. Upon training for 40 epochs for each dataset, the selected feature showed to perform better on a SVM classifier than the ones selected by Chi-Square scoring on almost all datasets, with the exception of RELATHE dataset.

A sharp increase in accuracy in the beginning as the number of selected features increases can be seen in Figure 3, which is not present in BASEHOCK, PCMAC or RELATHE. This is due to the nature of the dataset, as the features are text tokens and accuracy steadily increases as we consider more of the text. A similar pattern can be seen for the Chi-Square method evaluated on the text datasets.

Feature selection using Neuron Strength seems to perform best in image recognition tasks, since naturally the trained sparse network will have strong connections closer to the center of the image, quickly finding a pattern to disregard pixels close to the borders, and thus rapidly and monotonically increasing the accuracy. A similar pattern of important features arising towards the center of the image are observed in Atashgahi's work on unsupervised feature selection [28]. Though interesting to note is that Chi-Squared method, in this case, performs much worse.

Only the madelon dataset failed to converge to a low loss value, as the test loss increased to a steady level as presented in Figure 2. In an attempt to resolve the issue, various values for batch size were used ranging from 4 to 64, as well as training for more epochs with a reduced learning rate and varying the momentum, but without much improvement. Despite this, Figure 3 shows $\approx 70\%$ test accuracy as compared to the $\approx 50\%$ accuracy when using Chi-Squared. The tapering off of accuracy at around 20 features is explained by the nature of the dataset, as madelon is an artificial dataset of 500 features, only 20 of which are relevant for the label prediction and the remaining 480 are artificial noise [29].

6.1 Further research

The time required to select features using Neuron Strength seems to pose a disadvantage, as it requires to train a neural network from scratch. Though when scaled to higher dimensional data may provide an alternative approach to feature selection and could scale better than the Chi-Square methods, or more computationally expensive methods, such as Fisher score or Laplacian score. Further research could go into finding optimal cases for the use of Neuron Strength for supervised feature selection over other methods for very high-dimensional data.

7 Conclusion

The presented results show that a sparse neural network trained for supervised learning prob-

lems possesses characteristics that help select important features from a dataset by using Neuron Strength as the importance metric. Compared with the Chi-Square feature selection method, the features selected using Neuron Strength outperforms Chi-Square with the chosen datasets, performing best in image recognition tasks.

Further research is still required to determine the the scalability of the method as well as the further investigation of the circumstances under which the proposed method outperforms other supervised feature selection methods with handcrafted importance metrics.

8 Appendix

Dataset	1st Layer		2nd Layer		3rd Layer		Output Layer		Training time
	Parameters	Density (%)	Parameters	Density (%)	Parameters	Density (%)	Parameters	Density (%)	
BASEHOCK	156368	1.07	119203	1.32	119202	1.32	5999	99.98	0:19:15
PCMAC	124957	1.26	119224	1.32	119217	1.32	6000	100	0:21:28
RELATHE	145615	1.12	119197	1.32	119205	1.32	5999	99.98	0:14:27
orlraws10P	264935	0.86	119206	1.32	119172	1.32	25963	86.54	0:02:20
warpAR10P	107176	1.48	119243	1.32	119191	1.32	25927	86.4	0:01:27
ORL	79465	2.58	119199	1.32	119185	1.32	47599	39.66	0:01:34
COIL20	79470	2.58	119194	1.32	119182	1.32	38059	63.43	0:03:55
Isolet	70951	3.83	119192	1.32	119180	1.32	42166	54.06	0:03:45
madelon	68414	4.56	119178	1.32	119189	1.32	6000	100	0:20:40

Table 4: Network architecture for each dataset and the training time required when run on local machine

References

[1] A. Jovic, K. Brkic, and N. Bogunovic, "A review of feature selection methods with applications," in *2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. IEEE, pp. 1200–1205. [Online]. Available: http://ieeexplore.ieee.org/document/7160458/

[2] review of feature selection techniques in bioinformatics | bioinformatics | oxford academic. [Online]. Available: https://academic.oup.com/bioinformatics/article/23/19/2507/185254

[3] B. Baker, I. Kanitscheider, T. Markov, Y. Wu, G. Powell, B. McGrew, and I. Mordatch, "Emergent tool use from multi-agent autocurricula." [Online]. Available: http://arxiv.org/abs/1909.07528

[4] S. R. Richter, H. A. AlHaija, and V. Koltun, "Enhancing photorealism enhancement," p. 16.

[5] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners." [Online]. Available: http://arxiv.org/abs/2005.14165

[6] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," vol. 521, no. 7553, pp. 436–444, number: 7553 Publisher: Nature Publishing Group. [Online]. Available: https://www.nature.com/articles/nature14539

[7] D. C. Mocanu, E. Mocanu, T. Pinto, S. Curci, P. H. Nguyen, M. Gibescu, D. Ernst, and Z. A. Vale, "Sparse training theory for scalable and efficient agents," p. 5.

[8] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks." [Online]. Available: http://arxiv.org/abs/1803.03635

[9] D. C. Mocanu, E. Mocanu, P. Stone, P. H. Nguyen, M. Gibescu, and A. Liotta, "Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science," vol. 9, no. 1, p. 2383. [Online]. Available: http://www.nature.com/articles/s41467-018-04316-3

[10] Z. Atashgahi, G. Sokar, T. van der Lee, E. Mocanu, D. C. Mocanu, R. Veldhuis, and M. Pechenizkiy, "Quick and robust feature selection: the strength of energy-efficient sparse training for autoencoders." [Online]. Available: http://arxiv.org/abs/2012.00560

[11] J. Tang, S. Alelyani, and H. Liu, "Feature selection for classii¬cation: A review," p. 33.

[12] S. Alelyani, J. Tang, and H. Liu, *Feature Selection for Clustering: A Review*.

[13] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," p. 8.

[14] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks." [Online]. Available: http://arxiv.org/abs/1506.02626

[15] T. Marill and D. Green, "On the effectiveness of receptors in recognition systems," vol. 9, no. 1, pp. 11–17.

[16] S. H. Huang, "Supervised feature selection: A tutorial," vol. 4, no. 2, p. 22, number: 2. [Online]. Available: http://www.sciedupress.com/journal/index.php/air/article/view/6218

[17] Y. Yang and J. O. Pedersen, "A comparative study on feature selection in text categorization," p. 9.

[18] Q. Gu, Z. Li, and J. Han, "Generalized fisher score for feature selection," in *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence*, 2011, pp. 266–273.

[19] sklearn.feature_selection.chi2 — scikit-learn 0.24.2 documentation. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.chi2.html#sklearn.feature_selection.chi2

[20] sklearn.feature_selection.f_classif — scikit-learn 0.24.2 documentation. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.f_classif.html

[21] sklearn.feature_selection.mutual_info_classif — scikit-learn 0.24.2 documentation. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.mutual_info_classif.html#sklearn.feature_selection.mutual_info_classif

[22] D. Mocanu, "dcmocanu/sparse-evolutionary-artificial-neural-networks," original-date: 2018-03-02T21:47:05Z. [Online]. Available: https://github.com/dcmocanu/sparse-evolutionary-artificial-neural-networks

[23] 1.4. support vector machines — scikit-learn 0.24.2 documentation. [Online]. Available: https://scikit-learn.org/stable/modules/svm.html

[24] Y. Zhai, W. Song, X. Liu, L. Liu, and X. Zhao, "A chi-square statistics based feature selection method in text classification," in *2018 IEEE 9th International Conference on Software Engineering and Service Science (ICSESS)*, pp. 160–163, ISSN: 2327-0594.

[25] P. Meesad, P. Boonrawd, and V. Nuipian, "A chi-square-test for word importance differentiation in text classification," vol. 6.

[26] Datasets | feature selection @ ASU. [Online]. Available: https://jundongl.github.io/scikit-feature/datasets.html

[27] sklearn.svm.SVC — scikit-learn 0.24.2 documentation. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

[28] Z. Atashgahi, "zahraatashgahi/QuickSelection," original-date: 2020-11-16T08:43:56Z. [Online]. Available: https://github.com/zahraatashgahi/QuickSelection

[29] UCI machine learning repository: Madelon data set. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Madelon