Exploring classification, clustering, and its limits in a compressed hidden space of a single layer neural network with random weights

Meiyan Xie and Usman Roshan

Department of Computer Science, New Jersey Institute of Technology, Newark NJ 07090, USA,
mx42@njit.edu,usman@njit.edu

Abstract. Classification in the hidden layer of a single layer neural network with random weights has shown high accuracy in recent experimental studies. We further explore its classification and clustering performance in a compressed hidden space on a large cohort of datasets from the UCI machine learning archive. We compress the hidden layer with a simple bit-encoding that yields a comparable error to the original hidden layer thus reducing memory requirements and allowing to study up to a million random nodes. In comparison to the uncompressed hidden space we find classification error with the linear support vector machine to be statistically indistinguishable from that of the network's compressed layer. We see that that test error of the linear support vector machine in the compressed hidden layer improves marginally after 10,000 nodes and even rises when we reach one million nodes. We show that k-means clustering has an improved adjusted rand index and purity in the compressed hidden space compared to the original input space but only the latter by a statistically significant margin. We also see that semi-supervised k-nearest neighbor improves by a statistically significant margin when only 10% of labels are available. Finally we show that different classifiers have statistically significant lower error in the compressed hidden layer than the original space with the linear support vector machine reaching the lowest error. Overall our experiments show that while classification in our compressed hidden layer can achieve a low error competitive to the original space there is a saturation point beyond which the error does not improve, and that clustering and semi-supervised is better in the compressed hidden layer by a small yet statistically significant margin.

1 Introduction

Single layer neural networks are known to approximate any non-linear function within a threshold of error [1, 2]. We also know from Cover's theorems [3] that there exists a linearly separable feature space given sufficient nodes in the single layer. Their applicability to classification is limited though because we usually do not know the non-linear function underlying the data and Cover's work doesn't tell us exactly how to obtain node weights that give a linearly separable space.

As a result we initialize with random weights and use the back propagation algorithm to fit the weights on training data [4].

It is well known that optimizing a single layer network can overfit on the training data and give poor results on test data [5]. To relieve this overfitting we use methods like dropout [6] and stochastic gradient descent [7] while training the model. These methods introduce randomness into the training process with the intent to relieve overfitting.

On the extreme end we can use entirely random weights in the single hidden layer. In fact random weights have been studied as early as Rosenblatt's original paper on perceptrons [8]. Recent studies have shown that random weights in the hidden single layer followed by trained weights in the output layer give highly accurate classification results [9–12]. Of these the random bit regression and random bit forest give the highest accuracies [13, 12]. This work is different from random projections [14, 15] because here we apply a non-linear activation function on the hidden layer output whereas random projections work on the actual output.

In this paper we further explore classification in the feature space given by the hidden layer of a single layer random network. Specifically we compress the hidden layer with bit-encoding to reduce memory requirements and thus allowing for up to a million random nodes in the network. We compare the test error of the linear support vector machine and other classifiers in the compressed hidden layer compared to the original input space as well as k-means clustering and k-nearest neighbor semi-supervised learning.

In conclusion we see that adding random nodes reaches a saturation point and the error begins to increase when we are at a million random nodes. We see that the linear support vector machine in the hidden layer achieves classification errors comparable to state of the art methods in the original and compressed hidden space. We also see that both clustering and k-nearest neighbor semi-supervised learning are better in the compressed hidden layer by statistically significant margins (except for the adjusted rand index in k-means clustering). Overall our study shows that a single hidden layer with random weights improves classification and clustering by moderate statistically significant margins.

2 Methods

2.1 Single layer random weights network

In Figure 1 we show a basic single layer neural network. Each node in the middle layer represents a linear classifier (or peceptron). The weights of each node are usually determined by the back propagation algorithm [4]. Briefly, this algorithm starts with initial random weights and optimizes each node at a time with gradient descent (here the gradient is given by the chain rule [4]). It iterates over the entire network until we have convergence.

In our work we use random weights in the hidden single layer followed by a trained linear support vector machine in the output layer. We describe our single

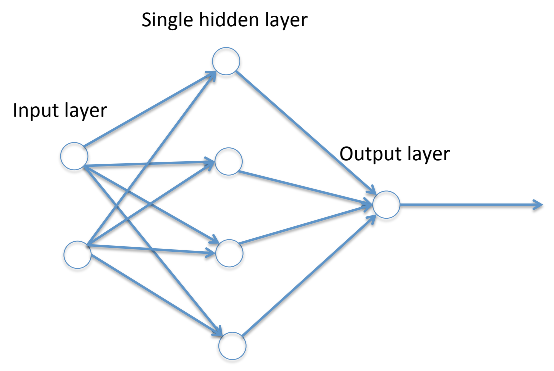

Fig. 1. Toy example of a single layer network. Here we see that the original input data is two dimensional. There are four nodes in the hidden layer which corresponds to a feature space of four dimensions. From this feature space we learn a classifier in the output layer.

layer network in detail in Algorithm 1. We see that our algorithm is similar to the random bit regression method [13] except for two things: the output layer and selection of offset. Our output layer is a support vector machine whereas they use logistic regression, and our offset is chosen randomly between projected points while theirs is an actual projected point (also randomly chosen).

Since the hidden layer has random weights the only optimization required is in the final layer. The outputs of the hidden layer represent a new feature space on which we optimize a linear support vector machine with the liblinear program [16].

3 Results

3.1 Experimental performance study

In order to evaluate the performance of the random weights network we study it on several dataset from the UCI repository at `https://archive.ics.uci.edu/ml/`.

Datasets: We obtained 52 datasets from the UCI repository. The datasets include data from different sources such as biological, medical, robotics, and business. Some of the datasets are multi-class and since we are studying only binary classification in this paper we convert them to binary. We label the largest class to be -1 and remaining as +1. We trim down excessively large datasets and ignore instances with missing values across the datasets. Thus, the number of instances in some of our datasets are different from that given in the UCI website `https://archive.ics.uci.edu/ml/`. For example the SUSY dataset originally has 5 million entries randomly ordered from which we choose the first 5000 for

Algorithm 1 Single layer random weight neural network

```
Input: Training data $x_i \in R^d$ with labels $y_i \in \{+1, -1\}$, the number of nodes $m \in N$ in the hidden layer
Output: Single layer network with random weights in the hidden layer and optimized SVM weights in the final output layer
Procedure:
  Single hidden layer:
  for node $k = 0$ to $m - 1$ do
    1. Create a random vector $w_k$ for the $k^{th}$ node such that $w_{ki} \sim Uniform[-1, 1]$
    2 . In order to set the offset parameter $w_0$ first let $w_k^T x_i, \forall i = 0, ..., n - 1$ be the projection of the training data on $w$. Determine $w_0$ by randomly picking it from $\{\frac{w^T x_0 + w^T x_1}{2}, \frac{w^T x_1 + w^T x_2}{2}, ..., \frac{w^T x_{n-2} + w^T x_{n-1}}{2}\}$ with uniform probability.
    3. For each training point $x_i$ the output given by node $k$ is $z_{ki} = sign(w^T x_i + w_0)$
  end for
  Final output layer: Learn a linear cross-validated SVM model (which we do with the liblinear program [16]) on the outputs given by the hidden layer. This classifier represents a single node in the output layer.
```

our study. We provide our data on the website `http://web.njit.edu/~usman/randomnet`.

Software We use the Python scikit-learn library version 0.19.1. to study other classification and clustering programs in the original and our hidden space, except for linear support vector machine. For that we use the fast liblinear version 2.20 software.

- XGboost ver 0.81: a Python implementation of gradient boosting [17] for decision trees. We use the scikit-learn API called XGBClassifier.
- Multi-layer perceptron: Python sci-kit [18] implementation with a single layer of 100 hidden nodes.
- Liblinear ver 2.20: a fast linear support vector machine program [16] with cross-validated value of C from the set $\{0.000001, 0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100\}$.
- K-nearest neighbor: Python sci-kit [18] implementation with cross-validated value of number of neighbors k from the set $\{1, 2, 5, 7, 10, 20, 50\}$ for the semi-supervised study. For the classification we use $k = 5$ neighbors.
- K-means clustering: Python sci-kit [18] implementation with 1000 random restarts. We cross-validate the number of clusters k from the set $\{2, 3, 4, 5, 6, 7, 8, 16, 20, 24, 28\}$ for the respective accuracy measure.

Experimental platform: We run our experiments on a cluster of computing nodes equipped with Intel Xeon E5-2660v2 2.27GHz processors with one method and dataset exclusively on a processor core.

Train and test splits: For each dataset we create a single random partition into training and test datasets in the ratio of 90% to 10%. We run all programs on each training dataset and predict the labels in the corresponding test set.

Measure of accuracy: We measure error as the number of misclassifications divided by the number of test datapoints. We use two measures for clustering accuracy: the adjusted Rand index [19] (ARI) and purity [20]. Briefly, the Rand index is a pairwise similarity measure defined by $RI = \frac{TP+TN}{TP+FP+FN+TN}$ whereas purity is the classification accuracy given by labeling clusters to maximize the accuracy. We use the adjusted Rand index (ARI) that accounts for pairs occurring due to chance.

3.2 Compressed hidden layer with bit-encoding

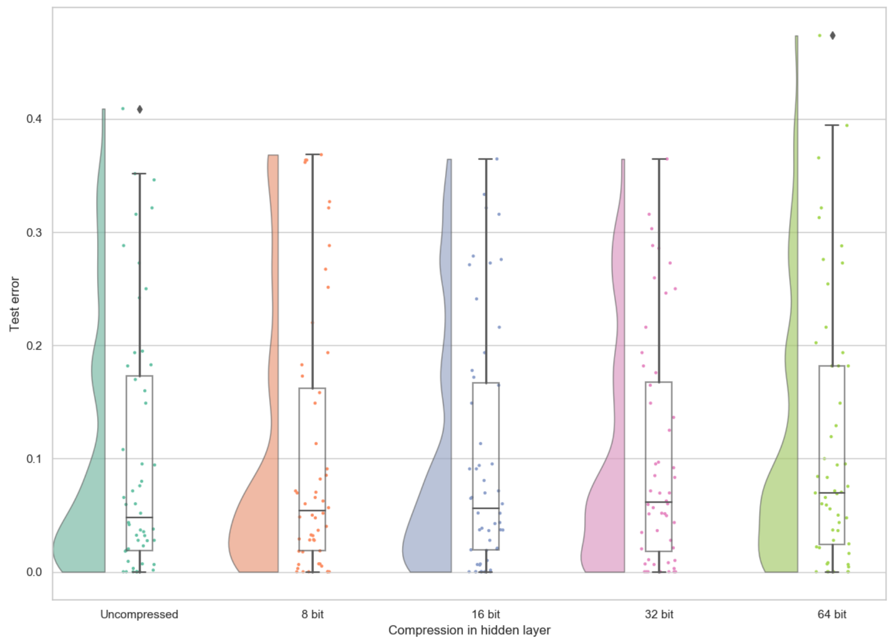

Fig. 2. Raincloud plot showing distribution of errors across the 52 datasets as well as the five summary statistics of min, max, first and third quartiles, and median. The graph shows errors of the linear support vector machine in the uncompressed hidden space vs. 8, 16, 32, and 64 bit compressions.

Since the output of each hidden layer is a 0 or 1 we perform a simple bit encoding. We divide the hidden nodes into sets of k and replace each set of k

outputs with their bit encoded single value. For example for a set of all 0's except for 1's in position 0 and 2 we would give it the number $2^0 + 2^2 = 5$. A 64 bit compression on 100,000 features would thus give us 1563 features in the end.

In Figure 2 we see Raincloud plots showing the five summary statistics of median, min, max, first, and third quartile as well as the distribution of errors across the 52 datasets for different k-bit encodings. The uncompressed hidden layer has 100,000 random nodes. We see that the median error increases somewhat marginally as we increase the number of bits in the encoding. At 64 bits the median error is statistically indistinguishable by the t-test from the uncompressed version even though it is a little higher.

3.3 Effect of number of nodes on the accuracy of the linear support vector machine

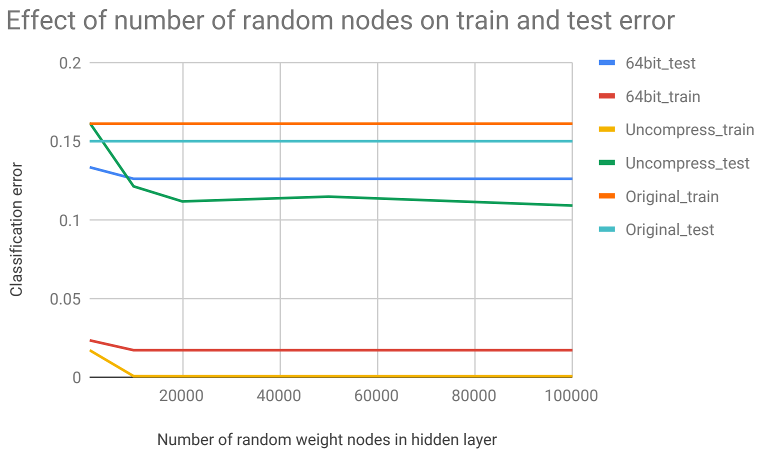

Fig. 3. Average error of the linear support vector machine as a function of number of random weight nodes in the uncompressed and 64-bit compressed hidden layer, and in the original input feature space

In Figure 3 we see that as we increase the number of nodes both the train and test error in the uncompressed and 64-bit encoded compressed layers decrease but become flat after 10,000 nodes. With the compressed features we can now study classification accuracy in the hidden layer up to a million random nodes compressed with 64 bits. In Table 1 we see that the error of both the linear support vector machine and k-nearest neighbor increases as we go from 100,000 to a million random nodes.

Classifier	10^5 random nodes	10^6 random nodes
Linear support vector machine	11.4%	11.8%
K nearest neighbor	13.4%	13.9%

Table 1. Average error in the 64-bit compressed hidden layer with a million random nodes originally uncompressed

3.4 Performance of k-means clustering in original and 64 bit compressed hidden layer

In Figure 4 we see the Raincloud plot of adjusted rand index (ARI) and the purity of k-means in the original and 64 bit encoded hidden layers. In the original uncompressed layer we have 100,000 random nodes. In both cases the improvement in median ARI and purity over the original space is small (about 2%) but statistically significant for purity under the t-test (p-value < 0.01).

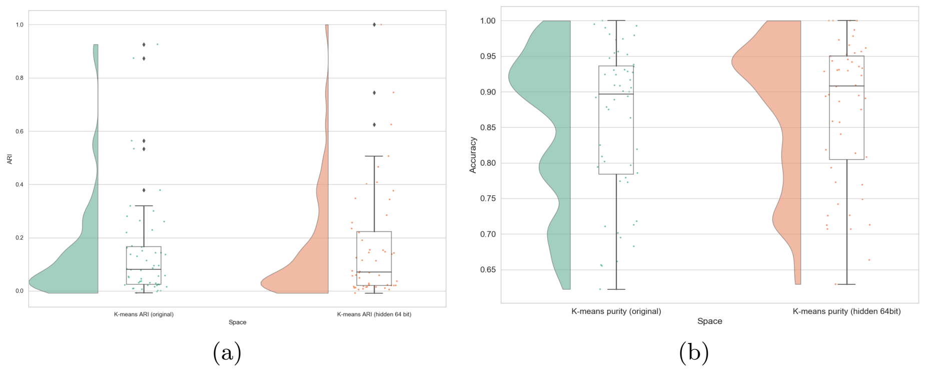

Fig. 4. Raincloud plots of k-means adjusted rand index and purity across the 52 datasets in the original space and the 64 bit compressed hidden layer

3.5 Performance of semi-supervised k-nearest neighbor in original and 64 bit compressed hidden layer

In this setup we randomly pick 90% (and 50% separately) of the training data and set it aside as unlabeled. We use the label propagation method [21] based on k-nearest neighbor to determine labels of the remainder of the training dataset. We then predict the test dataset with k-nearest neighbor.

In Figure 5 we see the Raincloud plot of test error across the 52 datasets in the original hidden space of 100,000 random nodes and the compressed one. We see that when 90% of the training data is unlabeled there is a statistically significant improvement in the median error. However, if 50% are unlabeled they are statistically indistinguishable.

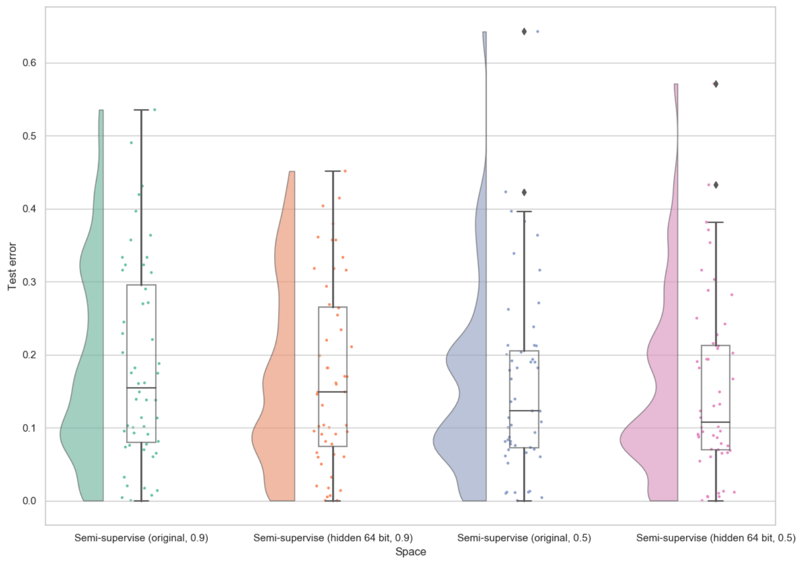

Fig. 5. Raincloud plots of k-nearest neighbor test error across the 52 datasets in the original space and the 64 bit compressed hidden layer

3.6 Accuracy of linear support vector machine, xgboost, multi-layer perceptron, and *k*-nearest neighbor in the compressed hidden layer

For our final experiment we compare several classifiers in the original hidden layer of 100,000 nodes and its 64-bit compression. In Figure 6 we see that the linear support vector machine in the 64-bit compressed encoded space has the lowest median error of all methods across the original and hidden layer feature spaces. We also see that the linear support vector machine, MLP classifier, and k-nearest neighbor all improve in the new feature space by statistically significant margins under the t-test (p-value < 0.01). However xgboost performs slightly better in the original space and is statistically indistinguishable in the hidden space under the t-test.

4 Conclusion

We see that classification with the linear support vector machine in a random network's compressed hidden layer achieves low errors competitive with state of the art classifiers but does not decrease considerably beyond 10,000 random nodes. Both semi-supervised and clustering are better in the network's compressed hidden layer by statistically significant margins except for the case of adjusted rand index in clustering.

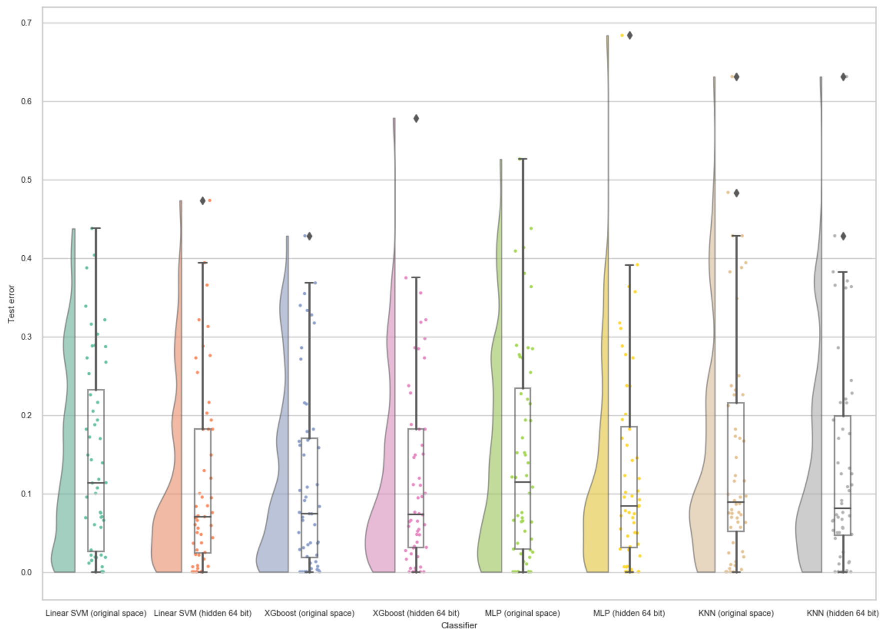

Fig. 6. Raincloud plots of test error of the linear support vector machine, XGboost, multi-layer perceptron, and k-nearest neighbor in the original space and 64 bit compressed hidden layer

References

1. Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.
2. George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.
3. Thomas M Cover. Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE transactions on electronic computers*, (3):326–334, 1965.
4. David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.
5. Rich Caruana, Steve Lawrence, and C Lee Giles. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. In *Advances in neural information processing systems*, pages 402–408, 2001.
6. Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.
7. Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.
8. F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386, 1958.

9. Wouter F Schmidt, Martin A Kraaijveld, and Robert PW Duin. Feedforward neural networks with random weights. In *11th IAPR International Conference on Pattern Recognition. Vol. II. Conference B: Pattern Recognition Methodology and Systems*, pages 1–4. IEEE.
10. Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. Extreme learning machine: theory and applications. *Neurocomputing*, 70(1-3):489–501, 2006.
11. Yi Wang, Yi Li, Momiao Xiong, Yin Yao Shugart, and Li Jin. Random bits regression: a strong general predictor for big data. *Big Data Analytics*, 1(1):12, 2016.
12. Yi Wang, Yi Li, Weilin Pu, Kathryn Wen, Yin Yao Shugart, Momiao Xiong, and Li Jin. Random bits forest: a strong classifier/regressor for big data. *Scientific Reports*, 6, 2016.
13. Yi Wang, Yi Li, Momiao Xiong, Yin Yao Shugart, and Li Jin. Random bits regression: a strong general predictor for big data. *Big Data Analytics*, 1(1):12, 2016.
14. Ella Bingham and Heikki Mannila. Random projection in dimensionality reduction: applications to image and text data. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 245–250. ACM, 2001.
15. William B Johnson and Joram Lindenstrauss. Extensions of lipschitz mappings into a hilbert space. *Contemporary mathematics*, 26(189-206):1, 1984.
16. Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. Liblinear: A library for large linear classification. *The Journal of Machine Learning Research*, 9:1871–1874, 2008.
17. Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.
18. F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
19. William M Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical association*, 66(336):846–850, 1971.
20. Christopher Manning, Prabhakar Raghavan, and Hinrich Schütze. Introduction to information retrieval. *Natural Language Engineering*, 16(1):100–103, 2010.
21. Xiaojin Zhu and Zoubin Ghahramani. Learning from labeled and unlabeled data with label propagation. Technical report, Citeseer, 2002.