Online Representation Search and Its Interactions with Unsupervised Learning

Ashique Rupam Mahmood, Richard S. Sutton
Department of Computing Science
Reinforcement Learning and Artificial Intelligence Laboratory
University of Alberta
Edmonton, Alberta, Canada
{ashique,rsutton}@ualberta.ca

Abstract

We consider the problem of finding good hidden units, or features, for use in multilayer neural networks. Solution methods that generate candidate features, evaluate them, and retain the most useful ones (such as cascade correlation and NEAT), we call *representation search* methods. In this paper, we explore novel representation search methods in an online setting, compare them with two simple unsupervised learning algorithms that also scale online. We demonstrate that the unsupervised learning methods are effective only at the initial learning period. However, when combined with search strategies, they are able to improve representation with more data and perform better than either of search and unsupervised learning alone. We conclude that search has enabling effects on unsupervised learning in continual learning tasks.

1 Introduction

In this paper, we consider an online supervised learning setting, where data arrives continually as a stream of inputs with associated desired outputs. On arrival, each input is mapped into a large number of features in some nonlinear way, and then the features are linearly combined to produce an output that approximates the desired output. Learning the weights of the linear combination in the second stage of this process is straightforward and can be done online using simple gradient descent methods. The nonlinear mapping in the first stage is often fixed and designed by experts using domain knowledge, but can also be learned, as we explore here. This two-stage architecture is popular and, given enough or the right features, powerful. It is used, for example, in support vector machines, in radial basis functions, and in tile coding or CMACs. The features produced by the first stage can be considered a representation used by the second stage, and thus, learning their nonlinear mapping can be considered a form of *representation learning*. We refer to the two-stage architecture as one of *expanded representations* (Sutton & Whitehead 1993).

Online learning of expanded representations is important yet less explored area of research. Representation learning is commonly studied in the opposite setting, where features are learned prior to their actual use on a fixed held-out data set. Overfitting is a prominent problem for learning from limited data. Hence a compressed representation is often desired. However, in online learning where data is never exhausted, overfitting is less critical, and an expanded representation can also be useful.

Online learning of expanded representations is associated with some natural desiderata. As a continual stream of data has to be processed here, storing any past data is impractical, and the overall computation and memory should be constant for each example. Moreover, an expanded representation also requires the computation and memory to scale linearly with the number of features. The backpropagation algorithm fulfills this desiderata. There are many representation learning methods that take different approaches than backpropagation and are commonly studied in a batch setting,

for example, unsupervised feature learning and evolutionary approaches. For online learning of expanded representations, these approaches can be adopted and modified so that they fulfill the desiderata.

We consider a search-based approach to representation learning and adopt it for online learning of expanded representations. In this approach, useful features are searched by generating features randomly and testing to selectively retain them. Few existing methods apply a search-based approach to representation learning. One of the first applications is by Uhr and Vossler (1961) who used a generate and test approach for a pattern recognition task. Evolutionary approaches to representation, such as NEAT (Stanley & Miikkulainen 2002) or EPNet (Yao & Liu 1997) generate features using evolutionary techniques and then use supervised criteria to test and select useful features. The cascade correlation method by Fahlman and Lebiere (1990) generates a pool of candidate features for evaluation and adds them sequentially. The work by Klopf and Gose (1969) explicitly explores a generate and test approach in an evolutionary framework. Evolutionary approaches typically maintain a pool of learners, where evaluation of a learner at each generation is done in a batch of data, which is not suitable to online learning.

We propose a search-based approach to representation learning that is well suited for online learning and scales linearly with the number of features. Using idealized problems, we demonstrate that our proposed methods can continually improve representation as more examples are seen. Although, representation search is not a mainstream approach and used commonly in batch settings, we show that it is at its most powerful in online settings and can be an integral part of general representation learning methods. Using idealized problems, we demonstrate that our proposed methods can continually improve representation as more examples are seen.

We study two online unsupervised learning algorithms by Sutton and Whitehead (1993) to investigate how they interact with representation search. Our experiments reveal that the unsupervised learning algorithms are effective only at the initial stage of learning, and after that, they are not able to adapt the representation with more data. However, when the unsupervised learning algorithms are combined with our online representation search strategies, they retain their initial effectiveness and also become able to continually adapt the representation. Their combination performed better than either of search and unsupervised learning alone. We suggest that, in an online learning setting with continual stream of data, the effect of unsupervised learning alone can be limited, but they may be reenabled by utilizing search.

2 Expanded Random Representation

All methods in our study have expanded representations with fixed number of features and initialize them randomly. The first method that we study has fixed representations, that is, the features are not learned. We refer to it as an *Expanded Random Representation* (ERR). It has long been argued that an ERR can be an effective form of representations (Rosenblatt 1962, Gallant & Smith 1987, Kanerva 1988, Prager & Fallside 1988). Sutton and Whitehead (1993) conducted a detailed study of ERRs in an online supervised learning setting and showed that ERRs can perform as well as nearest-neighbor methods. Jarrett et al. (2009) and Pinto et al. (2010) demonstrated their effectiveness on object recognition tasks. Saxe et al. (2010) argued that random representations reflect intrinsic properties of the architecture and should be used for baseline comparisons. We use ERRs as the baseline representations for comparison as well as the starting representations for our representation search methods. The basic structure of it is shown in Figure 1.

We consider an online supervised learning setting, where data consists of a series of examples. The kth example is presented as a vector of m binary inputs $x_{1,k}, \ldots, x_{m,k}$ and as a single target output $y_k \in \mathbb{R}$. The inputs are mapped into n binary features, where n is fixed and much larger than m. We consider each feature to be a Linear Threshold Unit (LTU). Each of the LTU-based features $f_{i,k}$, $i = 1, \ldots, n$, is connected to all the inputs $x_{j,k}$, $j = 1, \ldots, m$, by input weights $v_{ji,k}$. Initially, input weights v_{ji} are either $+1$ or -1 and they remain fixed for ERRs. Each feature has an input prototype, for which the feature responds maximally. For the ith feature $f_{i,k}$, if the weighted sum of the inputs, $\sum_{j=1}^{m} v_{ji,k} x_{j,k}$, is greater than a threshold θ_i, then the value of the feature is one; otherwise it is zero. The thresholds θ_i is set as $\theta_i = m\sigma - S_i^{min}$, where S_i^{min} is the number of negative input weights for the ith unit, and the tunable parameter σ signifies the proportion of input

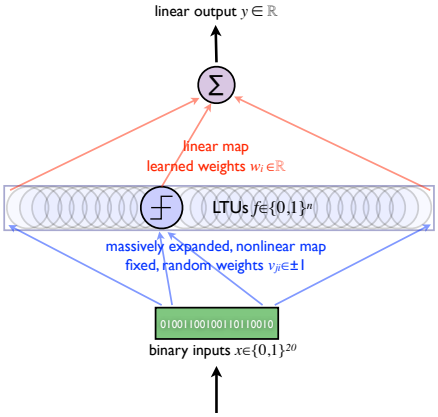

Figure 1: The basic structure of expanded random representation used in this paper. Each example is a vector of binary inputs and it is nonlinearly mapped into a massively expanded feature representation of LTUs. The features are then linearly mapped to approximate a target output.

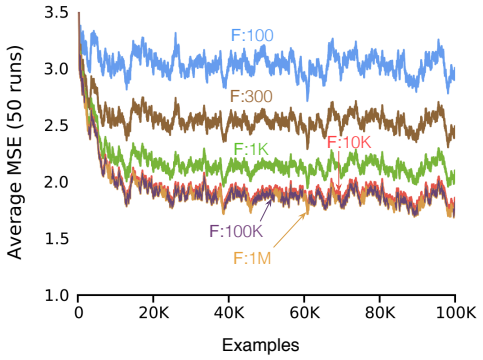

Figure 2: Fixed, expanded random representations perform better in online learning with more features. Best performance is achieved by a fixed representation with one million features (F:1M), but the performance increase compared to the ten times smaller representation (F:100K) is neglible over this time span.

bits that have to match the prototype in order to activate the unit. Representation learning in this setting corresponds to learning the input weights and thresholds.

The target output y_k is approximated as a weighted sum of the features, $\widehat{y}_k = \sum_{j=0}^{n} w_{i,k} f_{i,k}$, where $w_{i,k}$ is the output weight for ith feature. Here, $f_{0,k}$ is not a LTU but a bias feature, that is, it always has the value of 1. The online supervised learning task is to learn the output weights on each example. The output weights are learned incrementally and online using a simple gradient descent rule, which is the standard Least Mean Squares (LMS) algorithm in our setting. For the kth example, the output weights are updated as

$$w_{i,k+1} = w_{i,k} + \alpha \delta_k f_{i,k}, \tag{1}$$

for $i = 0, \ldots, n$. Here, δ_k is the estimation error $y_k - \hat{y}_k$ and α is a positive scalar, known as the step-size parameter. Output weights are always initialized at zero. We set the step-size parameter to $\frac{\gamma}{\lambda_k}$ for the kth example, where $0 < \gamma < 1$ is a small constant, that we refer to as the *effective step-size parameter*. Here, λ_k is an incremental estimate of the expected squared norm of the feature vector $\widehat{\mathbb{E}}\left[\sum_{i=0}^{n} f_{i,k}^2\right]$. The effective step-size parameter γ is set to 0.1 for all the experiments.

We study how well ERRs perform in an online supervised learning problem. Figure 2 shows the performance of fixed ERRs of different sizes (from 100 up to one million features) over one hundred thousand examples. Performance is measured as a running estimate of Mean Squared Error (MSE). Performance is averaged over 50 runs. Results show that ERRs with more features perform better. However, as number of features is increased, the increase in performance becomes smaller and smaller. Similar results were also found by Sutton and Whitehead (1993) in their work on online learning with random representations. Here we used much larger number of examples and features. Description of data (i.e., how inputs and outputs are generated) in our experiment is also different.

Data in the experiment consisted of a series of 20-dimensional i.i.d. input vectors (i.e., $m = 20$). Inputs are binary, chosen randomly between zero and one with equal probability. The target output value was computed by linearly combining 20 target features, which were generated from the inputs using 20 fixed random LTUs. The threshold parameter σ of these LTUs was set to 0.6. The target output y_k was then generated as a linear map from the target features $f_{i,k}^*$ as $y_k = \sum_{i=1}^{n} w_i^* f_{i,k}^* + \epsilon_k$, where $\epsilon_k \sim N(0,1)$ is a random noise. The target output weights w_i^* were randomly chosen from a normal distribution with zero mean and unit variance. Their values were chosen once and kept fixed for all examples. The learner only observed the inputs and the outputs. If the features and output

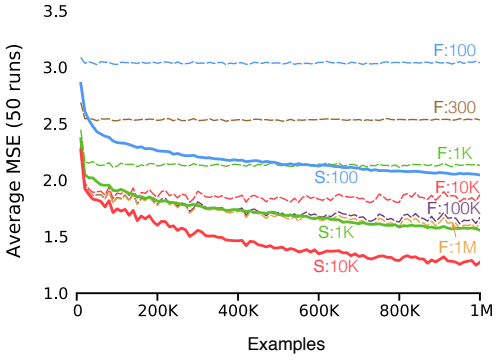

Figure 3: Our simple representation search method outperforms much larger, fixed representations. This method outperforms a fixed representation with one million features, F:1M, and continues to improve. Search with larger representations perform even better, approaching the minimum possible value of 1.0.

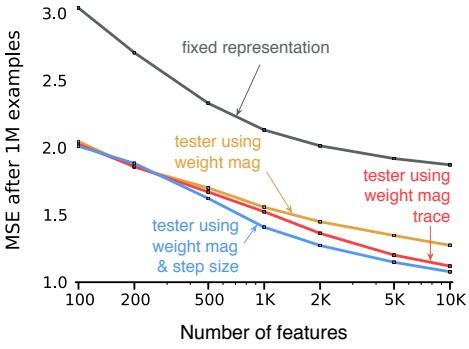

Figure 4: The choice of a tester has a significant effect on the performance of search. Search with the basic tester performed substantially better than fixed representation for different number of features. Here, the best performance is achieved when a more complex tester is used on 10,000 features.

weights are equal to the target features $f^*_{i,k}$ and target output weights w^*_i, respectively, then the MSE performance $\mathbb{E}\left[(y_k - \hat{y_k})^2\right]$ of the learner would be at minimum, which is 1 in this setting.

Random representations perform better with larger number of features, because larger random representations are more probable to contain the useful features. Although more features bring better performance, computational resource is always limited, and, typically, many features are far less useful. Therefore, many of them can be expended, and more useful features can be searched.

3 Effectiveness of Search

We investigate how an effective representation search can be performed incrementally and online, starting from an expanded random representation. Our first representation search approach uses a simple feature generator and tester. The generator simply generates features randomly, and the tester uses the magnitude of current output weights to evaluate features. The features are ranked with respect to the magnitude of their output weights, which signifies the relevance of the corresponding features to the given supervised learning task. For each example observed, a small fraction of features with the smallest magnitude of weights are then replaced with new randomly generated features, and the output weights of them are set to zero. As these new features would be ranked among the lowest for the next example, a maturity threshold is used and these features are not eligible for replacement as long as their ages are less than the maturity threshold. The maturity threshold is tuned to 2. We refer to the fraction of features replaced as the *replacement rate* ρ, and it is tuned to be one in every 200 features per example.

Figure 3 shows performance of our simple representation search method in comparison to fixed ERRs over one million examples on the same problem as in Figure 2. Performance is measured as an estimation of MSE. Each point of the curves is the squared estimation error δ^2_k averaged over last 10,000 examples and 50 runs. Performance is shown for every 10,000-examples interval. Standard errors of the estimated MSE were in the order of 10^{-3}. Our simple representation search method performed substantially better than fixed representations and continued to improve as more examples are seen. Performance of the fixed representation with 100 features (F:100) settled at a certain level, but representation search with the same number of features (S:100) outperformed it at an early stage and continued to improve. Representation search with 1,000 features (S:1K) outperformed fixed representation with 1,000 times more features (F:1M).

We developed two more representation search methods by combining two new testers with the random generator. A drawback of our first tester is that the weight magnitude is a noisy measure and can fluctuate greatly from one example to other. Our new testers address these issues. Our second tester uses the trace of the weight magnitudes for ranking the features. Before the first example is seen the trace is initialized to zero. Hence all features are equally eligible for replacement at the beginning. After the first example is seen, the trace for a newly generated feature is initialized with the median of all the traces, so that a newly generated feature do not get replaced immediately.

Our third tester is an extension of our first tester but uses learned step sizes as a measure of the confidence about the estimated weights. The features are ranked using the weight magnitudes as in our first tester, but a feature is not replaced if the confidence about the estimate of the weight is low. One step-size parameter for each feature is learned and the value of the learned step size is used as the confidence about the estimate of a weight. Higher the value of the step size, lower is the confidence about the weight estimate. We used Autostep by Mahmood et al. (2012) that adapts one step size for each feature without requiring any tuning of its parameters. Before the first example is seen, all the step sizes are initialized to the same value, which is roughly equal to one tenth of the inverse of estimated expectation of the squared feature norm. After the first example is seen, the initial step size of a newly generated feature is set to the median of all step sizes. A feature is eligible for replacement only if its step size is smaller than the median.

Figure 4 shows the comparison among the represent search methods with different testers. Performance after observing one million examples is plotted against different number of features. In this experiment, the tester that uses learned step sizes as a confidence measure performed the best. The tester using the trace of weight magnitudes also performed better than the simple tester. For all search methods, performance was better for larger representation.

The random generator and the testers are computationally inexpensive. The testers only need to determine an order statistic of a given order and find the features that have smaller orders, the overall computational complexity of which is linear on the number of features. The random generator generates a small number of features for each example, and computational complexity of each feature generation is linear on the number of inputs, which is much smaller than the number of features in our setting. So the total computational complexity of the random generator, and the testers for each example is precisely $O(\rho nm + n)$. Note that, ρ is a small number and can be chosen smaller than $1/m$.

4 Search with Unsupervised Learning

In this section we investigate how representation search interacts with unsupervised feature learning. We adopt two algorithms from Sutton and Whitehead (1993) that perform unsupervised learning to achieve sparsity in the representation. These algorithms operate online and scale linearly with the total number of weights. Sutton and Whitehead showed that they outperform fixed ERRs.

The first algorithm modifies the frequency of the activation of each feature. For this, it computes the running estimate of the activation frequency of each feature. The threshold θ_i is incrementally changed so that the estimated frequency is within a small bound around a target frequency. This unsupervised learning algorithm has one main parameter: the target frequency. We experimented with different target frequencies between 0.0 and 1.0. Following Sutton and Whitehead (1993), The tolerance limit around the target frequency is always set to 0.05. The threshold parameter σ determines the initial value of θ_i, which eventually gets modified by frequency adaptation. For initialization, we used different σ for each different features, chosen randomly between 0.0 and 1.0.

The second algorithm modifies the total activation of the feature set so that it achieves a target density. If the density of the feature set at any moment goes below a certain limit of the target density, one of the inactivate features is chosen with a small probability (0.0001), and then one of its input weights that do not match with their corresponding input bits is selected randomly. The sign of that input weight is then flipped. The corresponding threshold θ_i is then decreased by 1 to reduce the effect of this change on the frequency of the feature. If the density of the feature set at any moment goes above a certain limit of the target density, one of the activate features is chosen with a small probability (again, 0.0001), and then one of the input weights that match with their corresponding input bits is selected randomly. The sign of that input weight is then flipped. The threshold is also

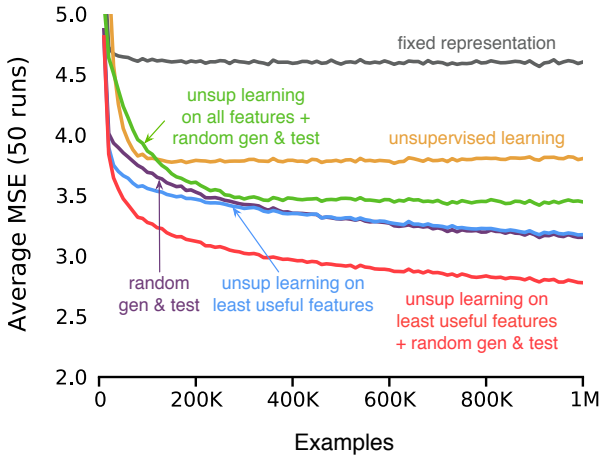

Figure 5: Representation search improves the performance of unsupervised learning. When some of the useful features are preserved, and the unsupervised learning algorithms are applied only on the rest of the features, it worked better than applying them on all features. When the random generation of features was included to it, it performed the best among all variations.

increased by 1 in this case. Here, the main tunable parameter is the target density. We experimented with different the target densities between 0.0 and 1.0. The tolerance limit around the target density is again set to 0.05.

Figure 5 shows the results of unsupervised learning and its different combinations with representation search strategies. When an unsupervised learning method was used, both algorithms were used together. Data was generated using a similar problem as in the previous experiments (described in section 2) except the activation probability of each input was 0.2, instead of 0.5. There were also 100 target LTU features, instead of 20, and the thresholds for the target LTUs were chosen randomly between 0.0 and 1.0, instead of 0.6 in previous experiments. Here, all the representations had 200 features, and the initial thresholds of all the LTUs were set randomly between 0.0 and 1.0 in all cases. Results are shown for best target frequency and density values. The best values for these parameters were between 0.2 and 0.3.

The unsupervised learning method performed substantially better than the fixed representation. This reconfirms the results by Sutton and Whitehead (1993). However, the method was effective only at the initial stage of learning. After that, they settled at a certain level of performance and did not improve with more data. It also performed substantially worse than one of our search methods that used random generation and testing with traces of absolute output weights.

Different combinations of unsupervised learning with search strategies performed better than unsupervised learning alone. We combined them in three ways. For the first combination, we used search only for testing the features and preserving the most useful ones. Random generation of features was left out. The tester with the trace of the absolute output weights was used to rank the features. For each example observed, the tester was used to preserve the top 40% of the features for that example, and the unsupervised methods were used to modify the rest of the 60% features. This combination performed better than the unsupervised learning method alone and continued to improve with more examples. This combination performed as well as the representation search method with random generation and test in the long run, but the combination performed better than that at the initial stage. This combination also outperformed the standalone unsupervised learning method during the initial period of learning. This suggests that the combination retained the initial advantage provided by the unsupervised learning method and also made use of the selective retention of search to improve continually.

For the second and third combinations, we added the random generation of features in the unsupervised learning method for replacing some of the least useful features. For each example observed, the output weights were learned using the LMS algorithm, the features were ranked using the tester, and the least useful features were replaced with randomly generated features. Then the features were updated using the unsupervised learning algorithms. For the second combination, all the features were updated using the unsupervised learning algorithms, which is typically the case when unsupervised learning is used. For the third combination, the unsupervised learning algorithms were applied only on the bottom 60% of the features for that example.

Both the second and the third combinations performed better than the unsupervised learning alone. As the random generation of features provides a constant source of new features, unsupervised learning found its use beyond the initial period of learning. However, the second combination performed worse than the search method alone. It hints that selective retention should also be considered. The third combination performed better than the search method alone and continued to improve with more data. This method performed the best among all the methods. These results reveal that different components of search can be combined with unsupervised learning in different ways, and unsupervised learning can benefit from each of them separately.

5 Discussion and Conclusions

In this paper, we explored a search-based approach to representation learning. Its core strategies are based on random variation and selective retention. Our study showed that such simple ideas can be realized inexpensively and can provide an effective method for continual representation learning.

We studied two unsupervised learning algorithms and showed that they can greatly benefit from representation search in online learning settings. When the unsupervised learning algorithms were used without search, they were effective only at the initial period of learning. However, when the best features were preserved, unsupervised learning became able to generalize with more data. When the random generation of features were included, it provided a constant source of new features, and hence unsupervised learning found its use beyond the initial period of learning.

Both unsupervised learning and representation search have their unique strengths, but they alone might be of limiting use. In online learning tasks, it is not desirable to disregard the usefulness of features for too long. As unsupervised learning methods produce features disregarding their usefulness, they may produce many features that are irrelevant, and at the same time, they may also destroy some useful features in the process. Representation search can alleviate these problems by eliminating irrelevant features as well as protecting the useful ones. Our study suggests that these two representation learning approaches should be considered together in online learning problems.

All of our representation search methods are well suited for online learning, and they do not need to store any past data. The complexity of executing them for each example scales linearly with the number of features. Computationally, our online representation search methods are almost free.

References

Fahlman, S. E., Lebiere, C. (1990). The cascade-correlation learning architecture. *Advances in Neural Information Processing Systems*, pp. 524–532.

Gallant, S., Smith, D. (1987). Random cells: an idea whose time has come and gone ... and come again? *Proceeding of the IEEE International Conference on Neural Networks.*

Jarrett, K., Kavukcuoglu, K., Ranzato, M. A., LeCun, Y. (2009). What is the best multi-stage architecture for object recognition? *Proceeding of the 12th IEEE International Conference onComputer Vision*, pp. 2146–2153.

Kanerva, P. (1988). *Sparse Distributed Memory.* Cambridge, MA: MIT Press.

Klopf, A. H., Gose, E. (1969). An evolutionary pattern recognition network. *IEEE Transactions on Systems, Man, and Cybernetics 15*: 247–250.

Mahmood, A. R., Sutton, R. S., Degris, T., Pilarski, P. M. (2012). Tuning-free step-size adaptation. *Proceedings of the 2012 IEEE International Conference on Acoustics, Speech, and Signal Processing*, Kyoto, Japan, pp. 2121–2124.

Pinto, N., Doukhan, D., DiCarlo, J. J., Cox, D. D. (2009). A high-throughput screening approach to discovering good forms of biologically inspired visual representation. *PLoS computational biology*, 5(11).

Prager, R.W., Fallside, F. (1988). The modified Kanerva model for automatic speech recognition. *Computer Speech and Language 5*: 257–274.

Rosenblatt, F. (1962). *Principles of Neurodynamics*. New York: Spartan Books.

Saxe, A., Koh, P. W., Chen, Z., Bhand, M., Suresh, B., Ng, A. (2010). On random weights and unsupervised feature learning. *Workshop: Deep Learning and Unsupervised Feature Learning (NIPS)*.

Stanley, K. O., Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary Computation 10*(2):99–127. MIT Press.

Sutton, R. S., Whitehead, S. D., (1993). Online learning with random representations. *Proceedings of the Tenth International Conference on Machine Learning*, pp. 314–321.

Uhr, L., Vossler, C. (1961). A pattern recognition program that generates, evaluates and adjusts its own operators. *Proceedings of the Western Joint Computer Conference*, pp. 555–569.

Yao, X., Liu, Y. (1997). A new evolutionary system for evolving artificial neural networks. *IEEE Transactions on Neural Networks 8*(3):694–713. IEEE Press.