Position Paper:
Representation Search through Generate and Test

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

Abstract
Learning representations from data is one of the fundamental problems of artificial intelligence and machine learning. Many different approaches exist for learning representations, but what constitutes a good representation is not yet well understood. In this work, we view the problem of representation learning as one of learning features (e.g., hidden units of neural networks) such that performance of the underlying base system improves. We study an important case where learning is done continually on an example-by-example basis from an unending stream of data where the computational cost of the learning element cannot grow with time or cannot be much more than that of the performance element. We show that a search approach to representation learning can naturally fit with this setting. In this approach, good representations are searched by generating different features and then testing them for utility. We develop new representation-search methods and show that the generate-and-test approach can be utilized in a simple and effective way for continual improvement of representations. Our methods are fully online and add only a small fraction to the overall computation. We believe online representation search constitutes an important step toward effective and inexpensive solutions to representation learning problems.

Introduction
Data representations are fundamental to artificial intelligence and machine learning. Performance of a learning system depends heavily on how the data is represented to it. Typically, human experts hand design a large part of the data representation using domain knowledge. However, it would be more desirable to learn the representational elements from data themselves. This would reduce the amount of human labor required, and learning systems would scale more easily to larger problems. However, what constitutes a good representation is not well understood. This makes learning representations from data a challenging problem.

Different approaches have been proposed to solve the problem of representation learning. Supervised learning through error backpropagation, one of the most popular methods for representation learning, learns the representation by reducing the supervised error signal toward the gradient-descent direction. Although this method is proved successful in several applications, it often learns slowly and poorly in many problems. Other methods for representation learning have also been proposed. Many researchers hold that good representations can be learned by fulfilling some unsupervised criteria such as sparsity (Olshausen & Field 1997), statistical independence (Comon 1994) or reproduction of the data (Hinton & Salakhutdinov 2006). Some methods use several levels of abstractions to capture features that are invariant to low level transformations (Bengio et al. 2012). Despite the existence of different approaches, it is yet unclear what is the right approach to representation learning.

We view the problem of representation learning as one of learning features such that the underlying base system performs better. Here, by features we refer to representational elements, such as hidden units in neural networks, kernels in support vector machines or elements of function approximation in reinforcement learning, that are combined semi-linearly to form the final output of the base system. The base system learns the appropriate combination of the features in order to perform well on a given task, such as classification, regression or policy optimization. The problem we focus here is how the features themselves can be learned from data so that the performance of the base system improves.

In this work, we study how representations can be learned online from an unending stream of data. In many AI systems such as life-long learning robots, data arises abundantly as a series of examples through their sensors, and learning occurs continually. As more data is seen, a pre-learned representation may become less useful with time to such systems, and continual learning of representations might be helpful in this case. One important case is a fully online learning setting where learning has to be done on an example-by-example basis, and the computational cost of the learning element cannot grow with more data or cannot be much more than that of the performance element. Here we study how representations can be learned fully online as well. Most representation learning methods consider a fixed batch of data, and pass through it several times in order to learn from it. Only a few representation learning methods (e.g., supervised gradient-descent learning) can be used fully online. In gen-
eral, how representations can be learned effectively in online learning settings is not well understood.

Here, we show that a search approach to representation learning fits naturally to continual learning problems. In this approach, good representations are searched through generating and testing features continually, while the base system is simultaneously performing on its original task. A large number of candidate features are generated, and they are then tested for their utility in the original task. Features that are more useful are preserved, and less useful features are replaced with newly generated ones. We refer to this approach as representation search. Although our approach is different than the conventional approaches, it is not opposed to them. The existing approaches such as unsupervised feature learning or supervised gradient-descent learning can be viewed as different ways of generating candidate features within this approach.

We develop a representation search method that utilizes the generate-and-test approach in a simple and effective way. Our method changes the representation on each example, but adds only a fixed small fraction to the overall computation of the system. Using a supervised learning setting, we demonstrate that our method can effectively learn the representation by continually improving it with more data. It indicates that representation search can be a potential and computationally effective solution for representation learning problems.

**Effectiveness of Search**

We view a representation search method as an auxiliary to a base system the objective of which is to perform well on a given learning task. In order to perform its task, a base system typically takes input examples and produces outputs. We consider a particular form of base systems, in which, each input example is mapped nonlinearly into a number of features, and the features are then mapped to produce an output. Once an output is produced, the base system receives an error or a feedback, based on which the system updates the maps. Typically, the base system only updates the output map. But, the base system may also update the input map using conventional representation learning methods such as unsupervised learning or supervised feature learning through gradient descent. Under this framework, the objective of representation search is to search for good features so that the base system performs better.

The basic idea that underlies our representation search method is generate and test. A representation search method uses a tester that estimates the utility of each feature. Based on the estimate, the method eliminates a small fraction of the features that are least useful. A generator then generates new features, and those are added to the feature pool for the base system’s use.

The generate and test process can be executed either online or in a batch. If executed in a batch, first the base system can learn the maps, perhaps until convergence, on a fixed batch of data, and then the generate and test process can be applied. In an online learning setting, the generate and test process should be able to operate simultaneously with the base system on an example-by-example basis.

There are two important challenges using a generate and test process on an example-by-example basis. First, it is difficult to estimate the utility of the features reliably when learning online. In a batch setting, the base system can learn the maps until convergence, at which point all the estimates become stable, and hence the least useful features can be identified reliably. In an online learning setting, new examples may always arrive, making it difficult to obtain reliable estimates. Moreover, as the generate and test process operates on each example, the feature representation may contain different kinds of features among which some are old and some are just newly generated. Among such a heterogeneous group of features, estimating the utility is much more difficult.

Second, in order to execute a generate and test process on an example-by-example basis, a representation search method must fulfill some computational constraints that are typically more severe than in a batch setting. In a fully online learning problem, data arrives frequently and unendingly as a stream of examples. As examples arrive in a frequent manner, the overall system has a limited time to process each example. As examples arrive unendingly, per-example computation of a system must not grow with more data. Hence, the per-example computation of the system should be small and constant. Typically representation learning is seen as a computation-intensive process. But in online learning settings, it has to be done inexpensively.

We develop a representation search method that overcomes these two challenges. To demonstrate its performance, we use a series of experiments in an online supervised learning setting.

In our online supervised learning setting data arrives unendingly as a series of examples. The $k$th example is presented as a vector of $m$ binary inputs $x_k \in \{0,1\}^m$ with elements $x_{i,k}, i=1,\ldots,m$ and a single target output $y_k \in \mathbb{R}$. Here the task of the base system is to learn the target output as a function of the inputs in an online manner, that is, the learning system can use each example only once and can spend a small, fixed amount of computation for each example.

The base system approximates the target output as a nonlinear function of the inputs. To achieve this, the inputs are mapped nonlinearly into a number of features, which are then linearly mapped to produce the output. In order to keep the per-example computation constant, the number of features must remain fixed over the course of learning. We denote the number of features as $n$.

The nonlinear map from the inputs to the features is achieved using Linear Threshold Units (LTU). The particular form of the representation is adopted from Sutton and Whitehead’s (1993) work. Each feature is computed as follows:

$$f_{i,k} = \begin{cases} 1 & \sum_{j=1}^{m} v_{ij,k}x_{j,k} > \theta_i \\ 0 & \text{otherwise} \end{cases}$$

where $v_{ij,k}$ is the input weight for the $i$th feature and the $j$th input, and $\theta_i$ is the threshold for the $i$th feature. The input weights are initialized with either $+1$ or $-1$ randomly, and they remain fixed in the absence of representation learning.
The task of representation learning is to learn these weights. The threshold \( \theta_i \) is set in such a way that the \( i \)th feature activates only when at least \( \beta \) proportion of the input bits matches the prototype of the feature. This can be achieved by setting the thresholds as \( \theta_i = m\beta - S_i \), where \( S_i \) is the number of negative input weights (\( -1 \)) for the \( i \)th feature. The threshold parameter \( \beta \) is tunable.

The output is produced by linearly mapping the features: \[
\hat{y}_k = \sum_{i=0}^{n} w_{i,k} f_{i,k},
\]
where \( f_{0,k} \) is a bias feature always having the value of 1, and \( w_{i,k} \) is the output weight for the \( i \)th feature. The output weights are initialized to zero. The overall structure of the representation is shown in Figure 1.

In the absence of representation learning, the feature representation is always a fixed map of the inputs. Then the base system only learns the output weights using the Least Mean Squares (LMS) algorithm:

\[
w_{i,k+1} = w_{i,k} + \alpha \delta_k f_{i,k},
\]
for \( i = 0, \ldots, n \). Here, \( \delta_k \) is the estimation error \( y_k - \hat{y}_k \), and \( \alpha \) is a positive scalar, known as the step-size parameter. The objective of the base system is to approximate the target output as well as possible, which can be measured using a window or running average of \( \delta_k \).

The total cost of the overall map from an input vector to an output is \( O(mn) \) for each example, that is, proportional to both the number of inputs and features, and remains constant over examples. The computational cost for learning the output weights using LMS is \( O(n) \) for each example. Therefore, the total per-example computation used by the base system is \( O(mn) \).

Our representation search method searches features on an example-by-example basis. We first describe the steps our method takes on each example. This method starts with the same representation as the base system with a fixed representation. After each example is observed, the base system executes its operations once. First the input example is mapped to produce the output, and the output weights are then updated using the LMS algorithm (Eq. 1). When representation search is not used, only these steps are repeated for each example. A representation search method does the following in addition to the operations of the base system. The tester first estimates the utility of each feature. The search method then replaces a small fraction \( \rho \) of the features that are least useful with newly generated features. The replacement parameter \( \rho \) is a constant and has to be tuned. Input weights \( v_{ij} \) of the new features are set with either \( +1 \) or \( -1 \) at random. The output weights \( w_{i} \) of these new features are set to zero. This process is repeated for each example. Note that selecting \( \rho m \) features does not require sorting all features. It only requires finding the \( \rho \)th order statistic and all the order statistics that are smaller, which can be computed in \( O(n) \). Generating \( \rho m \) features randomly requires \( O(n\rho m) \) computation. Note that \( \rho \) is a small fraction.

The tester in our method uses the magnitude of the instantaneous output weight as an estimate of the utility of each feature. This is not an unreasonable choice, because the magnitude of the output weights is, to some extent, representative of how much each feature contributes to the approximation of the output. When magnitudes of the features are of the same scale, then the higher the output-weight magnitude is, the more useful the feature is likely to be. Features that are newly generated will have zero output weights, and will most likely become eligible for replacement on the next example, which will be undesirable. In order to prevent this, we calculate the age \( \alpha_i \) of each feature, which stands for how many examples are observed since the feature is generated.
A feature is not replaced as long as its age is less than a maturity threshold $\mu$. Therefore, the selection of $\rho$ least-useful features occurs only among the features for which $a_i \geq \mu$. The maturity threshold $\mu$ is a tunable parameter. Age statistics $a_i$ can be kept and updated using $O(n)$ time and memory complexity. Therefore, the total cost of the per-example computation of our method is $O(n) + O(mn)$, which is no more than that of the base system. If we choose $\rho$ always to be less than $1/m$, then the total cost becomes $O(n)$. Moreover, the problem of reliable estimation of feature utility is also taken care of by keeping age statistics.

**Experiments and Results**

We used a simple experiment to investigate the effectiveness of our method. Here the base system performs a supervised regression task. Data in our experiment was generated through simulation as a series examples of 20-dimensional i.i.d. input vectors (i.e., $m = 20$) and a scalar target output. Inputs were binary, chosen randomly between zero and one with equal probability. The target output was computed by linearly combining 20 target features, which were generated from the inputs using 20 fixed random LTUs. The threshold parameter $\beta$ 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 weights of the learner are equal to the target features $f_{i,k}^*$ and target output weights $w_i^*$, respectively, then the MSE performance $E[(y_k - \hat{y}_k)^2]$ of the learner would be at minimum, which is 1 in this setting. The replacement rate $\rho$ was set to $1/200$, which stands for replacing one feature in every 200 for every example.

The search method performed substantially better than fixed representations and continued to improve as more examples are seen. Results are shown in Figure 2. Performance was measured as an estimate of MSE averaged over last 10,000 examples and 50 runs. 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 until the end of the sequence. Representation search with 1,000 features (S:1K) outperformed fixed representation with 1,000 times more features (F:1M).

**Related Works**

Search through generate and test is not unheard of in prior literature; similar ideas existed for a long time, often under different names. Klopf and Gose’s work (1969) is one of the earliest examples on the use of a generate and test approach for searching representations in connectionist networks. Some feature selection methods (Guyon & Elisseeff 2003) such as those called wrappers (John et al. 1994) share a similar idea with representation search. Other methods often fall under the umbrella term evolutionary computation (Goldberg 1989). These methods are applied to adapt either the network architecture or the weights or both. The main emphasis of evolution-based approaches is often not as much about utilizing the power of search for learning representations as it is about mimicking the biological evolution through pools of learners and recombinations.

Some other methods, such as constructive methods (Parekh et al. 2000) and pruning methods (Reed 1993), search the network structure through addition or deletion of representational elements. The cascade-correlation method (Fahlman & Lebiere 1990) is a notable example of constructive methods. It randomly generates and learns a pool of candidate features, and includes some of the best candidate features to the working set.

Most of the generate-and-test-based methods have only been applied to batch settings. Efforts have been made to extend the existing methods to online variants. Whiteson and Stone (2006) adopted an existing evolutionary method in a way so that it can utilize the online nature of RL problems. Vamplev and Ollington (2005) used the cascade-correlation method in an online manner for RL problems. Anderson (1993) developed a method for learning representations in RL problems where the least useful feature is replaced with the prototype of the current input vector, when a large increase occurs in the error level.

Our method can be seen as a demonstration of how generate and test can be used to search representations in the extreme case of online learning where representation is learned from an unending stream of data on an example-by-example basis. Other online methods, such as Vamplev and Ollington’s (2005) or Anderson’s (1993) do not change the representation on each example, but rather the representation change is triggered by the occurrence of an uncertain or infrequent event such as stagnation or sudden change in the error level. On the other hand, our method demonstrates that a small and effective change can be made to the representation on every example.

Changing the representation on an example-by-example basis is not only an extreme form of online learning, it is also a practical way of fulfilling the computational constraints faced by online learning problems with frequent and unending stream of data. When data is arriving at a fairly constant rate (e.g., through robot sensors), the overall system can only use a small, constant amount of time between two examples to complete all the per-example mappings and learnings. The computation for representation change has to be allotted within this small period of time. If the representation changes only once in a while, the allotted time would be wasted for those times when the representation does not change. It would be desirable, in that case, to amortize the computation for representation change throughout all examples. It is also one of the main motivations for using a fixed number of features on every example instead of taking a constructive or pruning approach.

**Conclusion**

In this work, we proposed a new method to search representations through generate and test. Although some prior
works used similar ideas, our study focused directly on the issues of representation learning and demonstrated how a simple and effective representation search method can be developed for continual learning problems. We studied an important online learning setting, where data arrives frequently and unendingly, hence the learning system is computationally constrained. We showed that the idea of generate and test fits naturally with such a setting, and can search for features in an inexpensive way. With a small addition to the overall computation of the system, representation search can continually improve the representation, and make the base system perform better.

Learning representations through search is a simple and intuitively appealing idea. In a sense, many machine learning researchers and practitioners already search for representations through generate and test as part of their experimental procedure. When deciding which representations or representation learning methods to use, they try different ways on their task of choice and declare victorious whichever method performs best on that task. Our study on representation search attempts to automate this largely-manual effort of the experimenter by delegating it to the learning system itself. We aimed at an extreme form of this automation, where the learning system is able to run this generate-and-test cycle on every example using a small computation. This idea may also be utilized to search other forms of abstractions in online learning settings given that generate and test can be facilitated. We believe that representation search would be able to provide a prominent solution to the problem of representation learning in AI systems.

Acknowledgments

The authors are grateful to the anonymous reviewers for their valuable suggestions. This research was supported by iCORE and Alberta Ingenuity, both part of Alberta Innovates Technology Futures.

References


