Representation Search through Generate and Test

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

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 continually improves. We study an important case where learning is done fully online (i.e., on an example-by-example basis) from an unending stream of data. In the presence of an unending stream of data, the computational cost of the learning element should not grow with time and cannot be much more than that of the performance element. Few methods can be used effectively in this case. 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 learning representations. Our methods are fully online and add only a small fraction to the overall computation. They constitute an important step toward effective and inexpensive solutions to representation learning problems.

Introduction

Data representations are fundamental to artificial intelligence and machine learning. Learning systems require data to learn, and 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. It is more desirable that the representational elements such as features themselves are learned from data. 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, Bengio et.al. 2007, LeCun & Bengio 2007). Some methods use several levels of abstractions to capture features that are invariant to low level transformations (Hinton 2007). 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 semilinearly 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.

Here 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 to such continually learning systems. Online learning of representations can be effective in avoiding such a problem. One important case is a fully online learning setting where learning has to be done on an example-by-example basis. In the presence of an unending stream of data, the computational cost of a fully online learning method should be small and should not grow as more data is seen. 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 represen-

tation learning methods (e.g., supervised gradient-descent learning) can be used fully online. In general, how representations can be learned effectively in an online learning setting is not well understood.

In this work, we take a search approach to representation learning which fits naturally with continual learning. In this approach, good representations are searched through generating and testing features, while the base system is performing on its original task. A large number of candidate features are generated in this approach, 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 learning or supervised gradient-descent learning can be viewed as different ways of generating candidate features within this approach.

Search through generate and test is not a new idea; similar ideas existed for a long time, often under different names. For example, some feature selection methods (Blum & Langley 1997, 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 of evolutionary computation (Goldberg 1989). Except for some of the recent works (Whiteson 2007), they were seldom viewed as representation learning methods. Efforts have been made to extend existing representation search methods to online variants (Whiteson & Stone 2006, Vamplev & Ollington 2005), however, a fully online method for representation search is still absent in the literature. In general, search through generate and test is not fully developed for representation learning.

We develop new representation search methods that can utilize generate and test for representation learning in a simple and effective way. Our methods are fully online, that is, they change the representation on each example, but add only a fixed small fraction to the overall computation of the system. Using a supervised learning setting, we demonstrate that our methods can effectively learn the representation by continually improving it with more data. We show that representation search can also utilize existing representation learning methods such as gradient descent. These results indicate 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 methods 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, 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 setting, the generate and test process should be able to operate 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 reliably identified. In an online 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 heterogenous 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 cheaply.

We develop several representation search methods that overcome these two challenges. To demonstrate their performance, we use a series of experiments in an online supervised learning setting.

We use an online supervised learning setting for our experiments where data arrives as a series of examples. The kth example is presented as a vector of m binary inputs $x_k \in \{0,1\}^m$ with elements $x_{i,k} \in \{0,1\}, i = 1, \ldots, m$ 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 in 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 ith feature and the jth input, and θ_i is the threshold for the ith 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 θ_i is set in such a way that the ith feature activates only when at least β 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 ith feature. The threshold parameter β is tunable.

The output is produced by linearly mapping the features: $\widehat{y}_k = \sum_{i=0}^{n} w_{i,k} f_{i,k}$, where $f_{0,k}$ is a bias feature always having the value of 1, and $w_{i,k}$ is the output weight for the ith 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:

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

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. The objective of the base system is to approximate the target output as well as possible, which can be measured using a window or a running average of δ_k^2.

The cost for mapping each input vector to a feature vector is $O(mn)$, and producing the linear map from a feature vector to an output costs $O(n)$. Therefore, the total cost of the overall map 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 $O(mn)$.

We introduce three representation search methods that search features on an example-by-example basis. Each method searches for features through generate and test. All of the methods use the same generator that generates features randomly. The three methods differ by their testers.

We first describe what is common between these methods. All the methods start with the same 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 ρ of the features that are least useful with newly generated features. The replacement parameter ρ 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 ρn features does not require sorting all features. It only requires finding the ρnth order statistic and all the order statistics that are smaller, which can be computed in $O(n)$. Generating ρn features randomly requires $O(\rho nm)$ computation. Note that ρ is a small fraction.

Our three methods have three different testers. Our first tester 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 a_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 μ. Therefore, the selection of ρn least-useful features occurs only among the features for which $a_i \geq \mu$. The maturity threshold μ is a tunable parameter. Age statistics a_i can be kept and updated using $O(n)$ time and memory complexity.

Our second tester uses the trace of the past weight magnitudes instead of the instantaneous ones. The trace is estimated as an exponential moving average, which can be updated incrementally. Instead of using an age statistic for each feature, the trace of a newly generated feature is initialized using a particular order statistic of all the existing traces (e.g., the median of all traces), so that newly generated features do not get replaced immediately. If a feature is irrelevant, its weight will have a near-zero value, and its trace will also get smaller with time, making the feature eligible for replacement. The decay rate of the exponential average and the order statistic for initializing the traces are tunable.

Our third tester uses the instantaneous output weight magnitudes for estimating the utility, but also uses learned step sizes as measures of how reliable the weight estimates are. No age statistic is used in this tester. We use the Autostep method by Mahmood et al. (2012) that learns one step size for each feature online without requiring any tuning of its parameters. Higher confidence is ascribed to a weight estimate if the corresponding feature has a smaller step size. The initial step size of a newly generated feature is set to a particular order statistic of all step sizes. A feature is eligi-

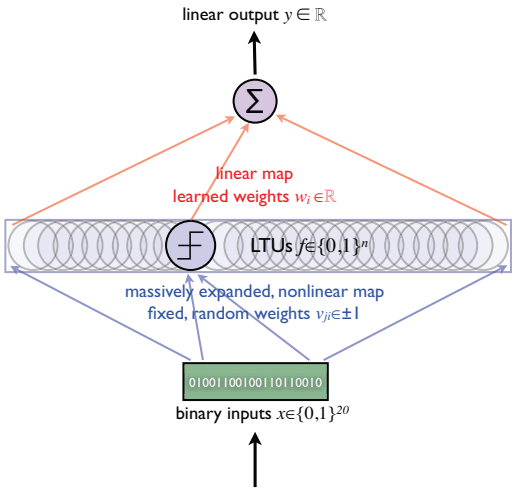

Figure 1: The general architecture of the base system. A binary input vector is nonlinearly mapped into an expanded feature representation. The features are linear threshold units, which are linearly mapped to produce a scalar output. The base system learns the output weights whereas representation search learns the input weights.

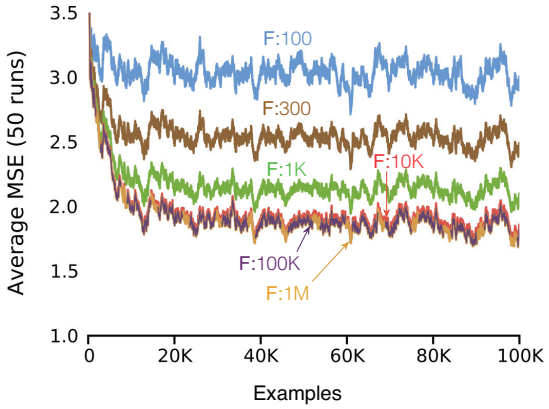

Figure 2: The base system with fixed representation performs better in online learning with larger representations. Best performance is achieved by a fixed representation with one million features (F:1M), but the performance increase is neglible compared to the ten times smaller representation (F:100K).

ble for replacement only if its step size is smaller than that statistic. The order statistic is a tunable parameter.

Per-example computation cost for all three testers is $O(n)$, hence our online representation search methods use a total of $O(n) + O(\rho nm)$ computation. Therefore, the order of per-example computation of the representation search methods is not more than that of the base system. If we choose ρ always to be less than $1/m$, then the total cost becomes $O(n)$. Moreover, each tester overcomes the difficulty of reliably estimating the feature utility by using different measures (age statistics, traces and step sizes).

Experiments and Results

Here we empirically investigate whether our representation search methods are effective in improving representations. The base system performs a supervised regression task, and the task of a representation search method is to improve the performance by searching and accumulating better features.

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 β 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 $\mathbb{E}\left[(y_k - \widehat{y}_k)^2\right]$ of the learner would be at minimum, which is 1 in this setting.

For all the methods except the third representation search method, the step-size parameter has been set 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 $\widehat{\mathbb{E}}\left[\sum_{i=0}^{n} f^2_{i,k}\right]$. The effective step-size parameter γ is set to 0.1 for all the experiments. The replacement rate ρ is set to $1/200$, which stands for replacing one feature in every 200 for every example. The rest of the parameters of the representation search methods are roughly tuned.

First we study how well the base system with fixed representations performs with different size of representations. Figure 2 shows the performance of fixed representations with 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 fixed representations with more features perform better. However, as the 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.

The result of our first representation search method is shown in Figure 3 over one million examples. This result is on the same problem as in Figure 2. Performance is measured as an estimate of MSE averaged over last 10,000 examples and 50 runs. The search method performed substantially better than fixed representations and continued to im-

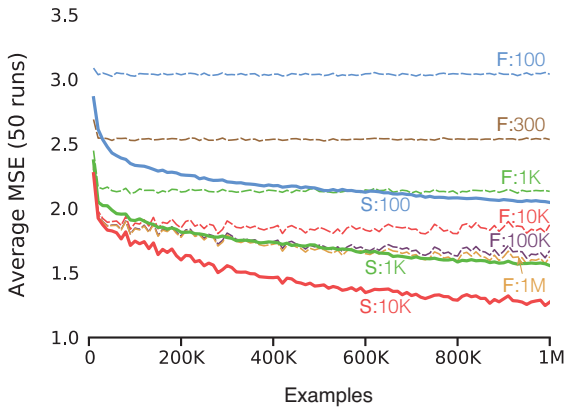

Figure 3: A simple representation search method outperforms much larger, fixed representations. With 1,000 features (S:1K), it outperforms a fixed representation with one million features (F:1M) and continues to improve.

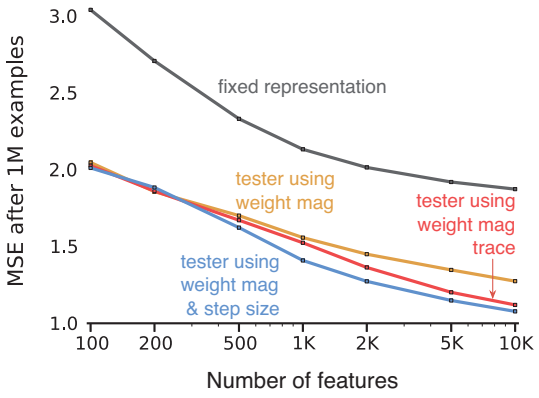

Figure 4: The choice of a tester has a significant effect on the performance of representation search. Our simplest tester using weight magnitudes is outperformed by testers that use more reliable estimates of feature utility.

prove 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 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).

Figure 4 compares the three representation search methods on the same problem as previous. Performance after observing one million examples is plotted against different number of features. The simple tester is outperformed by the other testers. The tester with learned step sizes performed the best.

Search with Gradient Descent Learning

In this section, we study the effects of combining search with the supervised Gradient-Descent (GD) learning through error backpropagation. The backpropagation algorithm is one of the most-popular supervised learning methods for representation learning and is well suited for online learning. We use online backpropagation to minimize the squared error δ^2. Online backpropagation uses a stochastic gradient-descent rule to learn both input and output weights.

In order to compare search with GD learning, we tuned the GD learning method in various ways and obtained the best variant. We experimented with logistic functions, hyperbolic tangent functions and LTUs as features. The GD update of input weights requires computing the derivative of the features. As LTUs are step functions, its partial derivative is zero everywhere except at the threshold. Therefore, the exact GD update for LTUs will not be useful. In order to overcome this problem, we used a modified backpropagation for LTUs. Whenever the derivative of a LTU is needed, the derivative of the logistic function is used instead, with the inflection point set at the threshold of the LTU. We tuned both the slope of the sigmoid functions and the initial variance of the input weights. We also used an additional variation. The input-weight update of the backpropagation algorithm is proportional to the output weight, and this leads to a problem: the update tends to modify the most useful features the fastest. To alleviate this problem, we used a simple modification, where the input-weight update uses only the sign of the output weight, but the update is not proportional to its magnitude. We refer to it as the *modified gradient update*.

When we applied search and GD learning in combination, the GD learning is regarded as the base system. Therefore, for each example, first the backpropagation algorithm updates both the input and the output weights, then the generate and test process is executed. We used the random generator and the second tester for search in this experiment.

For the experiment, we used the same problem as the previous one, this time with 500 target features and 1000 learnable features. When 20 target features were used, GD learning achieved a low error soon and left a little for search to improve on. We used more target features in this problem to make the problem harder. The results are shown in Figure 5. Here, 'GD' refers to the variant of GD where the features are hyperbolic tangent functions, and the modified gradient update is not used. The 'best GD' refers to the variant of GD where the features are LTUs, and the modified gradient update is used. This performed the best among all variants. All the differences in performance are highly statistically significant (the standard errors are smaller than the widths of the lines). The combination of search and the best GD learning reduced the final MSE by 13% more than the best GD alone. This improvement in performance is achieved through a negligible increase to the computational overhead. The extra runtime the combination took was less than 5% of the total runtime taken by the standalone backpropagation algorithm.

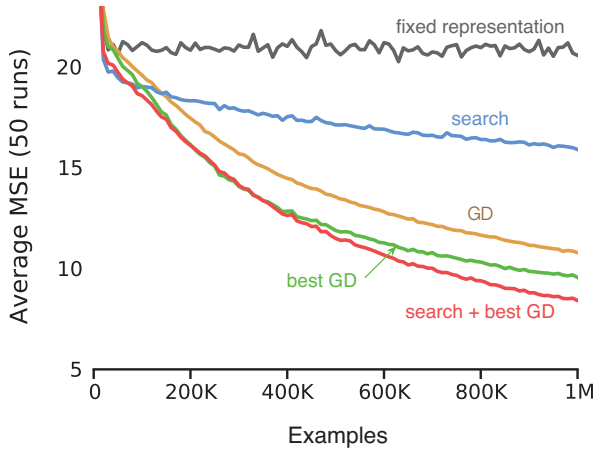

Figure 5: Combination of search with gradient descent performs better than using gradient descent alone.

Conclusions

In this work, we proposed new methods to search representations. Although some prior works used similar ideas, our study focused directly on the issues of representation search through generate and test and demonstrated how a simple and effective representation search method can be developed. We studied an important online learning setting, where data arrives frequently and unendingly, hence the learning system is computationally constrained. We demonstrated that the ideas of generate and test naturally fits with such a setting, and can search for features in an inexpensive way. With a negligible addition to the overall computation of the system, representation search can improve on an existing representation, and make the base system perform better. We showed the success of our methods on two important cases, a base system with no feature learning and a base system where features are learned through gradient descent. Representation search outperformed both when added to them. We believe that representation search may also improve other forms of representations, such as those being learned through unsupervised feature learning, as long as generate and test can be facilitated.

References

Bengio, Y., Lamblin, P., Popovici, P., Larochelle, H. (2007) Greedy Layer-Wise Training of Deep Networks, *Advances in Neural Information Processing Systems* 19, MIT Press, Cambridge, MA.

Blum, A., Langley, P. (1997). Selection of relevant features and examples in machine learning. *Artificial Intelligence*, 97(1-2):245271.

Comon, P. (1994). Independent component analysis, a new concept? *Signal Processing*, 36(3):287–314.

Goldberg, D. E. (1989). Genetic Algorithms in Search, *Optimization and Machine Learning*. Addison-Wesley.

Guyon, I., Elisseeff, A. (2003). An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3(Mar):1157–1182.

Hinton, G. E., Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science*, 313(5786), 504–507.

Hinton, G. E. (2007). Learning multiple layers of representations. Trends in Cognitive Sciences 11:428–434.

John, G. H., Kohavi, R., Pfleger, K. (1994). Irrelevant features and the subset selection problem. In *Proceedings of the 11th International Conference on Machine Learning*, 121–129.

LeCun, Y. and Bengio, Y. (2007) Scaling Learning Algorithms Towards AI. In Bottou et al. (Eds.) *Large-Scale Kernel Machines*, MIT Press.

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

Olshausen, B. A., Field, D. J. (1997). Sparse coding with an overcomplete basis set: A strategy employed by VI? *Vision research*, 37(23), 3311–3325.

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

Vamplev, P., Ollington, R. (2005). Global versus local constructive function approximation for on-line reinforcement learning. Technical report, *School of Computing, University of Tasmania*.

Whiteson, S. A. (2007). Adaptive representations for reinforcement learning, Ph.D. Thesis, Department of Computer Science, University of Texas at Austin, 2007.

Whiteson, S., and Stone, P. (2006). Evolutionary function approximation for reinforcement learning. *Journal of Machine Learning Research*, 7(May):877–917.