DOI: http://dx.doi.org/10.7551/978-0-262-32621-6-ch076

Online Extreme Evolutionary Learning Machines

Joshua E. Auerbach[1], Chrisantha Fernando[2] and Dario Floreano[1]

[1]Laboratory of Intelligent Systems, École Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland
[2]School of Electronic Engineering and Computer Science, Queen Mary, University of London, London, UK
joshua.auerbach@epfl.ch

Abstract

Recently, the notion that the brain is fundamentally a prediction machine has gained traction within the cognitive science community. Consequently, the ability to learn accurate predictors from experience is crucial to creating intelligent robots. However, in order to make accurate predictions it is necessary to find appropriate data representations from which to learn. Finding such data representations or features is a fundamental challenge for machine learning. Often domain knowledge is employed to design useful features for specific problems, but learning representations in a domain independent manner is highly desirable. While many approaches for automatic feature extraction exist, they are often either computationally expensive or of marginal utility. On the other hand, methods such as Extreme Learning Machines (ELMs) have recently gained popularity as efficient and accurate model learners by employing large collections of fixed, random features. The computational efficiency of these approaches becomes particularly relevant when learning is done fully online, such as is the case for robots learning via their interactions with the world. Selectionist methods, which replace features offering low utility with random replacements, have been shown to produce efficient feature learning in one class of ELM. In this paper we demonstrate that a Darwinian neurodynamic approach of feature replication can improve performance beyond selection alone, and may offer a path towards effective learning of predictive models in robotic agents.

Introduction

The notion of the brain as fundamentally a prediction machine is an old idea (Helmholtz, 1860) that has recently been gaining traction within the cognitive science community (see e.g. Clark, 2013; Hawkins and Blakeslee, 2007). A consequence of this idea is that if we wish to build robots capable of exhibiting intelligent behavior, and adapting to different circumstances, then prediction must be a fundamental part of their cognitive architecture. If the world behaved in a fundamentally linear way, than this would be an easy problem to solve: linear models operating directly on raw sensorimotor data could be learned in a straightforward and efficient manner. Unfortunately, the world is noisy and full of non-linear interactions that make learning difficult (Enns, 2010). In order to make accurate, non-linear predictions it is necessary to find appropriate data representations from which to learn. Finding such data representations or features is a fundamental challenge for machine learning.

Often the domain knowledge of human experts is leveraged to design useful features for specific problems (LeCun et al., 1998). While this may be an effective means of making learning tractable in many instances, it is not an ideal solution. Using human expertise is problem-specific, expensive, injects potentially sub-optimal biases into the solution, and for many robotics applications (especially those employing soft materials or other unconventional components, e.g. Germann et al., 2014) the relevant expertise may not exist. For these reasons, learning representations in a domain independent manner is highly desirable.

One common method of predicting non-linear relationships is to train multilayer feed-forward neural networks, which have been proven to be universal function approximators (Cybenko, 1989; Hornik, 1991). Most frequently these networks are trained offline from a pre-compiled training-set of input and target output values by gradient-descent via the backpropagation algorithm (Rumelhart et al., 1988). This offers one approach to feature learning: by backpropagating the supervised error signal, features can be adjusted in the gradient-descent direction. While this method may work successfully in many applications, it often learns slowly and may require large datasets to be effective (Ciresan et al., 2010).

Many other approaches for automatic feature extraction have been proposed in the literature. One approach that has recently proven quite successful involves the use an unsupervised "pre-training" step followed by further refinement through error backpropagation (Hinton and Salakhutdinov, 2006). However, this method is computationally expensive– usually involving extended computation time even when specialized hardware is employed. Moreover, the necessity of doing extensive "pre-training" on a data set cannot be applied when learning must be done fully online.

In online learning, data is learned from as it is received. In this regime, previously seen data points cannot be revis-

ited, and training gradients must be estimated from one (or a small subset) of the most recently seen data points. However, learning online from data as it is obtained is crucial in robotics domains where it is not possible to collect data *a priori* to be used in an offline batch mode. Moreover, even if possible, batch learning is not always desirable, because robotic agents may be operating in non-stationary environments within which they must continuously adapt. Finally, the volume of sensorimotor data obtained by agents may easily exceed the storage capacities of their onboard computers, especially if the agents continuously operate for extended time periods. For these reasons, this work concerns itself with online learning of predictors.

An alternative approach to the above methods, which has proven surprisingly effective, both for offline as well as online learning, is to randomly generate a large number of features on which to learn. Extreme Learning Machines (ELMs) (Huang et al., 2012) are a recently introduced formalization of single-hidden-layer feed-forward neural networks, where the feature-mappings are not tuned, but rather are chosen stochastically. This has the advantage that the only model parameters that are trained are the connection weights from hidden units to outputs, therefore simplifying learning to a linear regression problem[1]. The intuition here is that these fixed random features create a dimensionality expansion on top of which it is often possible to fit a linear model.

Recent work has demonstrated that it is possible to automatically search for effective features in online learning scenarios through a generate and test procedure (Mahmood and Sutton, 2013). In that work, it was demonstrated that, in a form of ELM-like artificial neural network (ANN), predictive accuracy could be greatly improved by regularly discarding features that offer low utility and replacing them with new stochastically generated features. This is essentially a selectionist approach, whereby poor features are selected for elimination and new features are generated in a completely random manner devoid of any information transfer.

In this work we extend Mahmood and Sutton's approach by taking inspiration from the Neuronal Replicator Hypothesis (Fernando et al., 2010, 2012), which posits that "replication (with mutation) of patterns of neuronal activity can occur within the brain using known neurophysiological processes." Specifically, instead of introducing new features at random as Mahmood and Sutton have done, new features are created through a Darwinian process of replication + variation of existing features, which have been discovered to be useful for solving the given prediction problem[2]. We dub this learning architecture an Online Extreme Evolutionary Learning Machine (OEELM). We demonstrate that OEELMs are capable of achieving lower error than the purely selectionist approach employed in (Mahmood and Sutton, 2013) with a much smaller number of features. Moreover we demonstrate that this method compares favorably to backpropagation.

The remainder of this paper is structured as follows. The following section describes the OEELM method, and describes the experimental setup used for comparing this approach to existing methods. Next, the results of these experiments are presented and analyzed. A discussion of this method is then presented, followed by conclusions and directions for future research.

Methods

Taking inspiration from Mahmood and Sutton (2013), we investigate the problem of automatically searching for useful features in a fully online learning scenario. In this formulation, an ANN is attempting to learn by adjusting its parameters to better fit the observations emanating from a noisy data stream. The underlying learning architecture is an ELM-like, single-hidden-layer feed-forward ANN. Following the implementation described in (Mahmood and Sutton, 2013) the ANN architecture consists of an input layer fully connected to a hidden layer with nonlinear activation functions (all features have access to all inputs), which is then fully connected to a linear output unit that produces a prediction of a target value.

The nonlinearities in the hidden layer are achieved by means of Linear Threshold Units (LTUs) adopted from (Sutton and Whitehead, 1993). Specifically, the output of feature i is given as follows:

$$f_i(x^{(t)}) = \begin{cases} 1 & \text{if } \sum_{j=1}^{m} v_{ji}^{(t)} x_j^{(t)} > \theta_i^{(t)} \\ 0 & \text{otherwise} \end{cases} \quad \forall i = 1, ..., n \quad (1)$$

for a network with m inputs and n features. Here, $x^{(t)}$ is the input vector at iteration t, $v_{ji}^{(t)}$ is the weight from input j to feature i at iteration t, $x_j^{(t)}$ is the jth component of $x^{(t)}$, and $\theta_i^{(t)}$ is the threshold of feature i at iteration t.

The prediction of the network at iteration t is given by

$$\hat{y}^{(t)} = \sum_{i=0}^{n} w_i^{(t)} f_i(x^{(t)}) \quad (2)$$

where $f_0(x)$ is a bias feature that is always set to 1.

At each iteration t, the network is presented with a single observation $(x^{(t)}, y^{(t)})$ from a noisy data stream and the

[1] Echo State Networks (Becker and Obermayer, 2003) and Liquid State Machines (Maass et al., 2002) (collectively known as Reservoir Computing) employ a similar idea for recurrent neural networks: the output connections from a dynamical reservoir of stochastically generated neurons is trained to fit a teaching signal via linear regression

[2] It is worth stressing that, here, we evolve a population of features for a single predictor, rather than a population of predictors as in (Arthur, 1994; Bongard and Lipson, 2007).

output weights w are updated in order to reduce the mean squared error between the observed target value $y^{(t)}$ and the predicted value $\hat{y}^{(t)}$ by means of the Delta-rule (Widrow and Hoff, 1960):

$$\Delta w_i^{(t)} = \eta(y^{(t)} - \hat{y}^{(t)}) f_i(x^{(t)}) \quad (3)$$

$$w_i^{(t+1)} = w_i^{(t)} + \Delta w_i^{(t)} \quad (4)$$

where η is a free parameter known as the learning rate.

Feature Selection

Mahmood and Sutton (2013) demonstrated that the predictive accuracy of this class of model could be improved if, instead of using a fixed random set of feature weights v, selection is employed to discard features offering low utility and replace them with new features during the course of online learning. One problem with implementing such a selectionist method in an online setting is that it may be difficult to quantify the utility of individual features. As argued in that work, if a batch learning system is optimized until convergence (see e.g. Schmidhuber et al., 2007) then the utility of features can easily be evaluated, but in an online setting this evaluation must be able to function on a per-example basis. Additionally, it is argued in Mahmood and Sutton (2013) that in online settings, where new data points are constantly arriving, the process of evaluating feature utilities must be computationally efficient and not add to the overall computational complexity of the learning system.

Mahmood and Sutton (2013) present a method that overcomes these limitations. For each iteration of online learning, a small fraction ρ of existing features is selected for elimination and replaced with newly generated, random features. They demonstrate that such an approach can be implemented such that the total computational complexity is no greater than a base system that implements the Delta-rule for learning the readout weights: $O(mn)$ for computing the output of the ANN, and $O(n)$ for updating the readout weights. The reader is referred to that work for further details of this derivation.

Finally, that work presented three alternative approaches for estimating the utility of a feature in the online learning scenario. All three of the approaches are based upon the idea that the relative utility of a feature is related to the magnitude of its readout weight. The intuition behind this idea is that the magnitude of a feature's readout weight determines how much that feature contributes to the output of the network. Since the readout weights are trained to approximate the observed data, the magnitude of a feature's readout weight will therefore serve as a proxy for how much that feature serves to explain the observations.

The problem with using the actual readout weight magnitude of a feature is that newly introduced features will initially have small output weight magnitudes[3], and without a mechanism to allow them time to prove their usefulness they will immediately be selected for elimination. Here, we adopt one of the procedures described in that work for overcoming this difficulty. This is described next.

In the employed approach, the utility of a feature u_i is calculated as an exponential moving average (EMA) of the magnitude of its readout weight:

$$u_i^{(t+1)} = \alpha_u u_i^{(t)} + (1 - \alpha_u)|w_i^{(t+1)}| \quad (5)$$

where α_u is the decay rate of the EMA, here chosen through a tuning procedure to be 0.9.

When a new feature k is introduced, u_k is set to the median utility value of all features so that it does not get replaced immediately. If this new feature is not useful then its actual readout weight w_k will remain near zero, and therefore u_k will shrink over time. This will lead to feature k eventually getting replaced. On the other hand, if feature k is useful then w_k will increase in magnitude and the feature will remain in the network. We choose this particular procedure because it performed competitively with the other approaches described in (Mahmood and Sutton, 2013), and is straightforward to implement.

Feature Evolution

The technique of (Mahmood and Sutton, 2013) described above is purely selectionist: features with poor utility are selected for elimination and are replaced with features that are generated at random. However, it is known that purely selectionist search methods have limitations that may be overcome through the use of replicators (Fernando et al., 2010). Additionally, there is a growing body of evidence which suggests that there is a process of replication occuring within individual brains (Fernando et al., 2010, 2012). Taking these ideas as inspiration, we suggest that a Darwinian process of feature evolution (with replication) will be a more powerful search method than the purely selectionist approach.

Extending the above selectionist method into a Darwinian one is fairly straightforward. The process of estimating feature utility and selecting features for removal remains unchanged. The main difference is that this utility estimation becomes the fitness function on which an online (steady-state) evolutionary algorithm operates. Now, instead of introducing new features purely at random, eliminated features are replaced with mutated copies of other features, which have themselves proved to be useful for explaining the observed data.

Specifically, when a feature is selected for removal, a binary tournament is conducted to choose a "parent" feature for reproduction. Two features from the population are chosen at random and the one with higher fitness creates a copy of itself. Each gene of that feature (the weights v_{ji}) are then

[3]Newly introduced features are initially given a readout weight of 0 so that they do not to contribute to the prediction before the learner has a chance to adjust this weight.

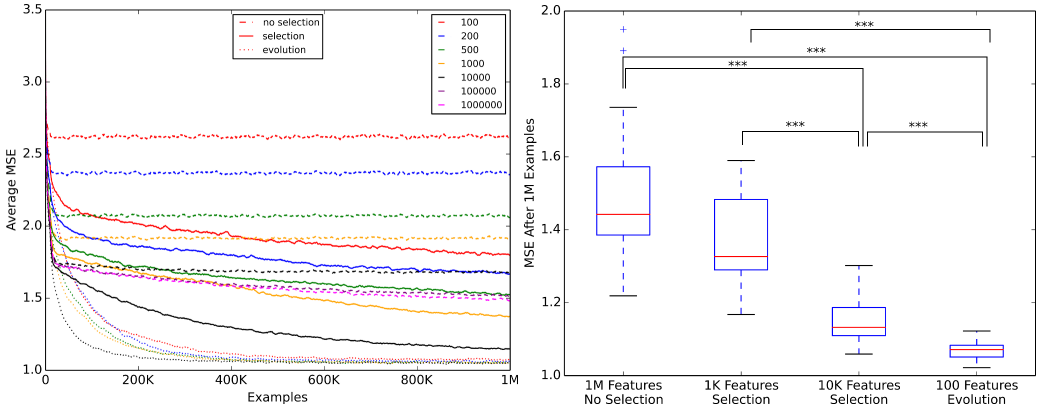

Figure 1: Comparison of learning errors across regimes and number of features. Left: this plot depicts how predictive-error (calculated as the mean across 30 independent runs of a sliding window estimate of mean squared error) varies over training time for all experimental setups investigated. Right: boxplots comparing the predictive-errors across regimes after being exposed to 1,000,000 examples. Asterisks denote statistical significance (*** = p-value < 0.001, Mann-Whitney U test with Bonferroni correction).

mutated with probability p_{mutate} and the resulting feature replaces the removed one. As is the case in the selectionist method, the readout weight of this feature is initialized to 0, and its fitness is initialized to the median fitness of the population. Conducting a binary tournament takes constant time (since the fitnesses are already in memory) and mutating the winner takes no more time than creating a new feature at random: $O(m)$. Therefore, the evolutionary approach is no more computationally expensive than the selectionist one.

Experiments

We first reproduce the results presented in (Mahmood and Sutton, 2013) before comparing the selectionist approach with the evolutionary approach. These experiments are described here.

Refer back to Eqn. 1. In these experiments the system is assumed to take a binary input vector $x \in \{0,1\}^m$ and produce a scalar prediction $\hat{y} \in \mathbb{R}$. The input weights v_{ji} are initialized with either $+1$ or -1 uniformly at random. The threshold θ_i is set as $\theta_i = m\beta_i - S_i$, where S_i is the number of negative input weights of feature i and β_i is a free parameter. This formulation ensures feature i activates when at least $m\beta_i$ input bits match the prototype defined by the feature's weights. Following the procedure of (Mahmood and Sutton, 2013)[4] β_i is set to 0.6 $\forall i$.

In the absence of any feature search procedure the values of these parameters will remain fixed for the entire duration of an online learning task–the network is essentially an ELM with binary features. However, when either the selectionist or the evolutionary approach is employed, its task is to find a set of features that is appropriate for explaining the observed data.

[4]Clarified in (Mahmood, 2014).

The online learning task is conducted in simulation. At each iteration a binary, 20-dimensional input vector $x^{(t)}$ is generated uniformly at random. This input is fed through an ANN of the type described above containing 20 fixed random LTU target features f_i^* each having threshold parameter $\beta_i = 0.6$. Next, the outputs of these features are linearly combined with output weights drawn from a normal distribution having mean 0 and unit variance ($w_i^* \sim N(0,1)$). The output of this network is then injected with Guassian noise $\epsilon_t \sim N(0,1)$ drawn independently at random for each iteration. Summarizing, the target value at iteration t, y_t is computed as:

$$y_t = \sum_{i=1}^{20} w_i^* f_i^* + \epsilon_t \tag{6}$$

The Gaussian noise makes the task more resemble real-world online learning tasks such as those found in robotics applications, and implies that if the learning network exactly learns the target function than the expected value of its mean squared error will be 1.

At each iteration, the learning rate η is set to $\frac{\gamma}{\lambda^{(t)}}$, where $\gamma \in (0,1)$ is the effective learning rate[5], and $\lambda^{(t)}$ is an EMA estimate of the squared norm of the feature vector:

$$\lambda^{(t)} = \alpha_\lambda \lambda^{(t-1)} + (1-\alpha_\lambda)(f^{(t-1)} \cdot f^{(t-1)}) \tag{7}$$

Unless otherwise specified, all reported results employ a decay rate $\alpha_\lambda = 0.999$, and an effective learning rate $\gamma = 0.1$.

Results

The above online learning problem is investigated under several experimental regimes. The first regime: "no selec-

[5]Called the *effective step-size* in Mahmood and Sutton's terminology.

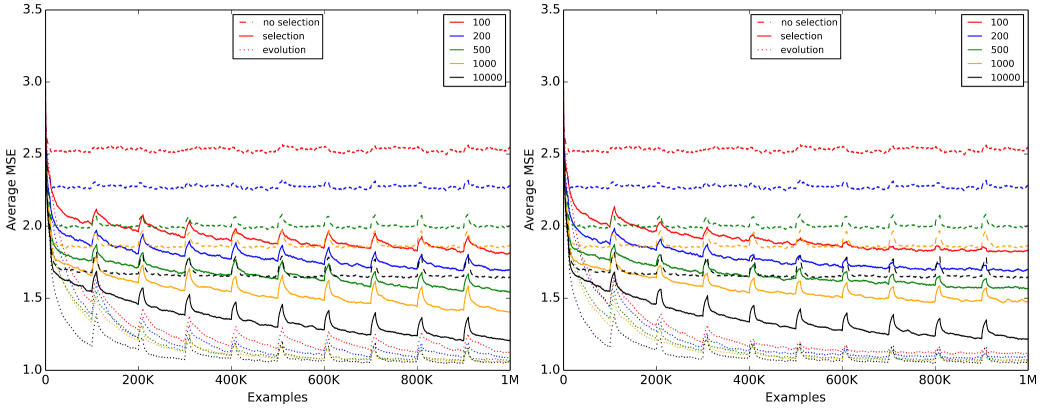

Figure 2: Exploring non-stationary environments. Here the environment (the target function) switches between two randomly generated functions every 100,000 iterations. Left: the same learning algorithms as are used for stationary environments. Right: for both the "selection" and "evolution" regimes features may be added to a growing archive that is immune from selection. These plots depict how predictive-error (calculated as the mean across 30 independent runs of a sliding window estimate of mean squared error) varies over training time.

tion" employs a fixed, random feature set, as is traditionally used in ELMs. The second regime: "selection" uses the purely selectionist approach of Mahmood and Sutton (2013). The third regime: "evolution" uses the evolutionary approach based on Darwinian neurodynamics introduced above (OEELMs). Each regime is investigated for different, fixed, number of features n. For each regime and value of n, 30 independent runs of the online learning scenario are conducted. Each learning scenario lasts for 1,000,000 iterations.

Under both the "selection" and "evolution" regimes, the fraction $\rho = 0.005$ of the existing features having lowest estimated utility are selected for elimination at each iteration. Under the "evolution" regime, the input weights of copied features are each mutated with probability $p_{\text{mutate}} = 0.1$, chosen to optimize performance. Reported results are robust to small variations of this value.

Fig. 1 compares the performance of these regimes as the learning experiments progress. On the left, the accuracy of each experimental setup (regime and n value) is plotted for each iteration t as follows. Within each run, the current mean squared error (MSE) is estimated using a sliding window approach: the current error estimate is the mean of the individual squared errors of the past 10,000 data points. Due to the noise inherent in the data stream, the sliding window provides a better estimate of a predictor's accuracy at a given time than its error on any individual data point. The means of these MSE estimates are then taken across the 30 independent runs of each experimental setup. On the right, we show a boxplot comparing the most relevant final MSE estimates after 1,000,000 iterations.

With a fixed, random feature set (the "no selection" regime) performance improves as a function of the number of features, but continuing to increase the feature count has diminishing returns. As demonstrated by Mahmood and Sutton (2013), and confirmed here, a purely selectionist approach to searching for features (the "selection" regime) with only 1,000 features can outperform a fixed feature set of 1,000,000 features. However, by using the OEELM approach described above, near optimal error can be achieved very rapidly. Moreover, using only 100 features with "evolution" not only outperforms all "no selection" formulations investigated, but also all "selection" formulations as well. Using 100 features with "evolution", the estimated MSE becomes significantly smaller (p-value < 0.001, Mann-Whitney U test) than that of all "no selection" and "selection" setups by iteration 104,900 and remains that way for the duration of the learning scenarios.

Non-stationary Environments

Often robotic agents are operating in non-stationary environments to which they must continuously adapt. In order to investigate whether feature evolution is also useful under changing environmental conditions the above online learning task is altered as follows. Instead of having a single target function that a network is attempting to predict, two different target functions are created. Since it is likely that different environmental conditions will have many similarities (e.g. the laws of physics remain unchanged across environments), the two functions are related to each other. Specifically, target function 1 is constructed exactly the same way as described above. Target function 2 is constructed from target function 1 by replacing 25% of its hidden nodes. For each node to be replaced, a new input weight vector is created uniformly at random, and a new readout weight is chosen from a Gaussian distribution with zero mean and unit variance.

The experimental procedure above is repeated for this

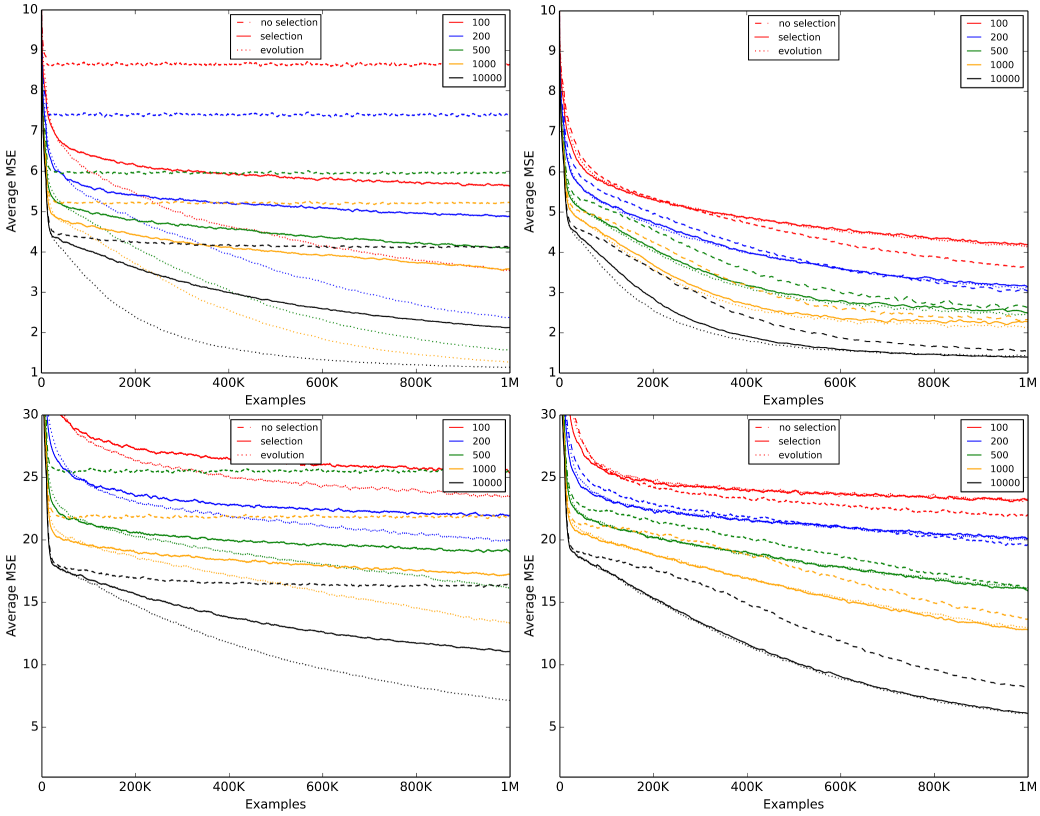

Figure 3: Increasing problem difficulty and combining feature evolution with backpropagation. Top: first, the difficulty of the problem is increased by going from 20 to 100 hidden units in the target function (left). Then the online learning algorithms are rerun in combination with a modified backpropagation procedure (right, see text for details). Bottom: The difficulty of the problem is increased further by using a target network with 500 hidden units, and the above is repeated. These plots depict how predictive-error (calculated as the mean across 30 independent runs of a sliding window estimate of mean squared error) varies over training time for this more difficult task.

new, non-stationary objective. First data points are sampled from target function 1 for the first 100,000 iterations, then data points are sampled from target function 2 for the next 100,000 iterations, and the target function continues to oscillate in this fashion for the duration of the task. As before, the data stream is noisy: target values are corrupted through the addition of Gaussian noise ($\epsilon_t \sim N(0, 1)$).

Fig. 2 (left) plots how estimated MSE (again taken as the mean across the 30 independent runs per setup of the sliding window estimate) varies by regime and n value for non-stationary environmental conditions. Once again the "evolution" regime with only 100 learnable features outperforms both the "no selection" and "selection" regimes using many more features. Here, the ("evolution", 100) setup achieves significantly smaller error (p-value < 0.001, Mann-Whitney U test) than all "no selection" and "selection" setups after being trained on 176,100 examples and continues to have significantly smaller predictive error for the duration of the learning scenarios.

Performance of both the "selection" and "evolution" regimes can be improved even further (especially for small numbers of learnable features) by incorporating a growing archive of useful features. This archive is initially empty. At each iteration there is a small probability ($p_{\text{archive}} = \frac{n}{1{,}000{,}000}$) that the best, currently non-archived feature is added to the archive. Once a feature has been added to the archive it is no longer eligible for replacement. So, even if a feature is not currently useful it may stick around if it was found useful in the past–the archive is a growing, long term memory of canalized features. The performance for the non-stationary task with the archive included for both the "selection" and "evolution" regime is depicted in Fig. 2 (right) (the "no selection" results are the same as in Fig. 2, left).

Backpropagation

Mahmood and Sutton (2013) suggested that combining feature selection with updating feature weights in the gradient descent direction via backpropagation (Rumelhart et al., 1988) could achieve better performance than either method alone. Following their example, here we explore how feature

evolution might synergize with backpropagation. Specifically, we employ the modified version of backpropagation introduced in that work: the output weights are adjusted using the Delta-rule as above, but since LTUs do not vary continuously, the gradient of each hidden unit is estimated using a logistic function centered around the threshold of its LTU. Then, instead of using the magnitude of the error multiplied by the output weights $(y^{(t)} - \hat{y}^{(t)})w_i^{(t)}$ when computing the feature gradients, only the sign of this quantity is taken, so that feature weights v_{ji} are updated as follows:

$$v_{ji}^{(t+1)} = v_{ji}^{(t)} + \eta_{\text{input}} * \text{sign}((y^{(t)} - \hat{y}^{(t)})w_i^{(t)}) * \sigma_i(x^{(t)})(1 - \sigma_i(x^{(t)}))x_j^{(t)}$$

where $\sigma_i(x^{(t)}) = \frac{1}{1+e^{-(v_{ji}^{(t)}x_j^{(t)} - \beta_i^{(t)})}}$ is the logistic function activation of the ith feature, and the learning rate η_{input} was selected by tuning to be a constant value of 0.01.

In order to investigate how feature evolution synergizes with backpropagation, we conduct further experiments. First, we make the learning problem more difficult by increasing the number of hidden units in the target network, then we supplement the above learning methods by first backpropagating the error before performing feature selection or evolution.

Fig. 3 shows how the performance of the experimental setups vary when the number of hidden units in the target network is increased to either 100 or 500. The left side of these figures uses the experimental setups as described above where only the readout weights are adjusted via the Delta-rule, the right side of these figures additionally incorporate the modified backpropagation just described. These results are discussed below.

Discussion

Looking at the left hand plots of Fig. 3 we see that as the problem is made more difficult, the advantage conferred by feature evolution appears to decrease. While the best performances are still found in the "evolution" regime, it now requires 500 learnable features to outperform selection with 10,000 learnable features in the 100 hidden target case, and requires 10,000 learnable features to do so when there are 500 hidden units in the target network. This could be for several reasons: one possibility is that as the prediction problem becomes more complex the necessity of maintaining a diverse set of features becomes more pressing. The "selection" regime naturally accomplishes this by introducing new features that are unrelated to existing features. Finding effective, computationally inexpensive methods for promoting feature diversity in the "evolution" regime will require further work.

For these experiments, altering the learning procedure to include backpropagation of error drastically improves the performance of the "no selection" regime (Fig. 3, right hand plots). This is essentially going from linear regression to the typical supervised ANN learning procedure. The performance of the "selection" regime is also either improved or left approximately the same. However, feature evolution no longer affects much improvement beyond purely selectionist methods.

This last point may be due to how feature weights are replicated. Here, the genome of a feature is considered to be its initial input weight vector, not the weight vector that has been altered through backpropagation–the evolutionary process is Darwinian rather than Lamarckian. So, any useful feature learning that arises as a result of backpropagation is not inherited through reproduction, and newly born features must then learn the backpropagated weight changes anew–just as under the selectionist method. Lamarckian evolution is known to be unstable in dynamic environments over phylogenetic timescales (Sasaki and Tokoro, 1997), but whether Lamarckian evolutionary neurodynamics combined with backpropagation also falls victim to the same pathologies remains to be tested in future work.

Conclusion

In this work we have introduced a method (Online Extreme Evolutionary Learning Machines) for automatically searching for features in an online learning task through feature replication. This process essentially makes the features (ANN hidden nodes) the members of a population undergoing an online, steady state evolutionary algorithm. We have demonstrated that using OEELMs results in significantly better predictive accuracy as compared to either using a fixed set of features or a purely selectionist search method, for both stationary and non-stationary environments.

Additionally, we have explored feature evolution as it relates to learning features through the backpropagation of error. Here, feature evolution is capable of achieving similar or better error than backpropagation in many instances (see Fig. 3). This result alone is interesting, because feature evolution may be more widely applicable than backpropagation: it does not require having known, differentiable features, but in principle could be used to evolve features of any form. Another natural next step is the application of evolution to convolutional neural networks (LeCun and Bengio, 1995). Here instead of hand-designing the shapes of the kernels that produce feature maps, the shapes and properties of kernels themselves can be evolved. Finally, it will be worthwhile to investigate how encoding features with a more evolvable, indirect encoding such as HyperNEAT (Stanley et al., 2009) may improve performance even further.

Feature evolution, as inspired by the Neuronal Replicator Hypothesis, is a promising method for online learning tasks. We foresee it being of particular relevance for robotics applications, where robots must learn predictive models in order to operate in unknown and possibly non-stationary environments. Specifically, we foresee these methods be-

ing useful for a robot to learn forward and inverse models (Wolpert and Kawato, 1998) from which it may fantasize against in order to accomplish some task in a given environment. This should allow for a robot to easily adapt to damage (Bongard et al., 2006) or to changes in morphology brought about by evolution, development or self-reconfiguration. This promising area will be investigated in future work.

Source Code

The source code used for all experiments conducted in this paper is available online at `https://github.com/jauerb/OEELM`.

Acknowledgements

The authors thank Rupam Mahmood for his kind cooperation and advice.
The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 308943 and the "Bayes, Darwin, and Hebb" Templeton Foundation FQEB Grant.

References

Arthur, W. B. (1994). Inductive reasoning and bounded rationality. *American Economic Review*, 84(2):406–411.

Becker, S. T. S. and Obermayer, K., editors (2003). *Adaptive nonlinear system identification with echo state networks*. MIT Press Cambridge, MA.

Bongard, J. and Lipson, H. (2007). Automated reverse engineering of nonlinear dynamical systems. *Proceedings of the National Academy of Science*, 104(24):9943–9948.

Bongard, J., Zykov, V., and Lipson, H. (2006). Resilient machines through continuous self-modeling. *Science*, 314:1118–1121.

Ciresan, D. C., Meier, U., Gambardella, L. M., and Schmidhuber, J. (2010). Deep, big, simple neural nets for handwritten digit recognition. *Neural computation*, 22(12):3207–3220.

Clark, A. (2013). Whatever next? predictive brains, situated agents, and the future of cognitive science. *Behavioral and Brain Sciences*, 36(03):181–204.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314.

Enns, R. H. (2010). *It's a Nonlinear World*. Springer.

Fernando, C., Goldstein, R., and Szathmáry, E. (2010). The neuronal replicator hypothesis. *Neural computation*, 22(11):2809–2857.

Fernando, C. T., Szathmary, E., and Husbands, P. (2012). Selectionist and evolutionary approaches to brain function: a critical appraisal. *Frontiers in Computational Neuroscience*, 6(24).

Germann, J., Auerbach, J., and Floreano, D. (2014). Programmable self-assembly with chained soft modules: an algorithm to fold into any 2-d shape. In *Proceedings of the International Conference on the Simulation of Adaptive Behavior*. To Appear.

Hawkins, J. and Blakeslee, S. (2007). *On Intelligence*. Macmillan.

Helmholtz, H. v. (1860). Handbuch der physiologischen optik, vol. & trans. jpc southall.

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

Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257.

Huang, G.-B., Zhou, H., Ding, X., and Zhang, R. (2012). Extreme learning machine for regression and multiclass classification. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 42(2):513–529.

LeCun, Y. and Bengio, Y. (1995). Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Maass, W., Natschläger, T., and Markram, H. (2002). Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural computation*, 14(11):2531–2560.

Mahmood, A. R. (2014). Personal Communication.

Mahmood, A. R. and Sutton, R. S. (2013). Representation search through generate and test. In *Workshops at the Twenty-Seventh AAAI Conference on Artificial Intelligence*.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1988). *Learning representations by back-propagating errors*. MIT Press, Cambridge, MA, USA.

Sasaki, T. and Tokoro, M. (1997). Adaptation toward changing environments: Why darwinian in nature. In *Fourth European conference on artificial life*, pages 145–153. MIT Press.

Schmidhuber, J., Wierstra, D., Gagliolo, M., and Gomez, F. (2007). Training recurrent networks by evolino. *Neural computation*, 19(3):757–779.

Stanley, K., D'Ambrosio, D., and Gauci, J. (2009). A hypercube-based encoding for evolving large-scale neural networks. *Artificial Life*, 15(2):185–212.

Sutton, R. S. and Whitehead, S. D. e. a. (1993). Online learning with random representations. In *ICML*, pages 314–321. Citeseer.

Widrow, B. and Hoff, M. E. e. a. (1960). Adaptive switching circuits. In *IRE WESCON Conv. Rec.*, volume 4, pages 96–104. Defense Technical Information Center.

Wolpert, D. M. and Kawato, M. (1998). Multiple paired forward and inverse models for motor control. *Neural Networks*, 11(7):1317–1329.