Computers and Chemical Engineering 114 (2018) 111–121

Contents lists available at ScienceDirect

Computers and Chemical Engineering

journal homepage: www.elsevier.com/locate/compchemeng

Machine learning: Overview of the recent progresses and implications for the process systems engineering field

Jay H. Lee[a,*], Joohyun Shin[a], Matthew J. Realff[b]

[a] Chemical and Biomolecular Engineering Department, Korea Advaced Institute of Science and Technology, Daejeon, Korea
[b] Chemical and Biomolecular Engineering Department, Georgia Institute of Technology, Atlanta, GA, USA

ARTICLE INFO

Article history:
Received 22 May 2017
Accepted 10 October 2017
Available online 13 October 2017

Keywords:
Machine learning
Deep learning
Reinforcement learning
Process systems engineering
Stochastic decision problems

ABSTRACT

Machine learning (ML) has recently gained in popularity, spurred by well-publicized advances like deep learning and widespread commercial interest in big data analytics. Despite the enthusiasm, some renowned experts of the field have expressed skepticism, which is justifiable given the disappointment with the previous wave of neural networks and other AI techniques. On the other hand, new fundamental advances like the ability to train neural networks with a large number of layers for hierarchical feature learning may present significant new technological and commercial opportunities. This paper critically examines the main advances in deep learning. In addition, connections with another ML branch of reinforcement learning are elucidated and its role in control and decision problems is discussed. Implications of these advances for the fields of process and energy systems engineering are also discussed.

1. Introduction

Machine learning has recently resurged as a popular technology. In the '90 s and 2000s, the internet transformed the way we live and do business, and in the process generated many petabytes of data. Now machine learning and predictive analytics are revolutionizing our society again by transforming that data into useful predictions. Already a number of significant commercial applications have appeared, including recommendation engines, speech and handwriting recognition systems, content identification, image classification/retrieval, automatic captioning, spam filters, and demand forecasting.

IBM Watson is an excellent example of a modern machine learning application; this question answering computer system, communicating in natural language, entered the public imagination through its winning performance in the quiz show Jeopardy, and is now used in commercial applications like lung cancer treatment and heart failure identification. Google, Microsoft and Amazon have made significant investments in the requisite infrastructure (e.g., cloud computing). For example, Google is said to have launched more than 25,000 software projects in 2015 alone based on a key AI technology called Deep Learning. A large number of new startups have sprung up specializing in algorithmic machine learning for applications ranging from e-commerce and security to robotics/auto and healthcare. Not just confined to the commercial world, machine learning has been applied for scientific data analysis in many areas, bioinformatics being the most notable one. In many universities, machine learning courses are highly popular with students. Clearly there is a need to evaluate the potential of this new wave to influence the process systems field.

Machine learning can be categorized into supervised learning, unsupervised learning, and reinforcement learning. Supervised learning uses labeled examples (X and Y) to predict their relationship (i.e., P(Y|X)) and is used in classification and regression. On the other hand, unsupervised learning uses unlabeled examples (X only) to learn about their distribution (i.e., P(x)) and can be useful for tasks such as clustering, compression, feature extraction, etc. Between them, there is another category of semi-supervised learning which has become popular in recent years. This uses unlabeled examples to help learn the probability distribution over the input space P(X) and jointly optimizes the prediction over the labeled and unlabeled examples, i.e., P(Y|X) and P(X) are optimized together in a weighted combined objective. Finally, reinforcement learning uses the data of input (X) and critic (U) to learn about an optimal relationship (for a given objective), either between the input and the decision or the input and performance index. Reinforcement learning can combine the learning and acting phases simultaneously into on-line learning and gives a self-optimizing feature.

* Corresponding author.
E-mail addresses: jayhlee@kaist.ac.kr (J.H. Lee), sinnis379@kaist.ac.kr (J. Shin), matthew.realff@chbe.gatech.edu (M.J. Realff).

https://doi.org/10.1016/j.compchemeng.2017.10.008
0098-1354/© 2017 Elsevier Ltd. All rights reserved.

Many researchers engaged in machine learning have promoted recent developments like Deep Learning as revolutionary inventions that will have a transformative effect. According to Andrew Ng at Stanford, "I have worked all my life in Machine Learning, and I've never seen one algorithm knock over benchmarks like Deep Learning." Geoffrey Hinton at Google boasts, "Deep Learning is an algorithm which has no theoretical limitations of what it can learn; the more data you give and the more computational time you provide, the better it is." On the other hand, some notable experts of the field have expressed concerns and skepticism over what they see as yet another wave of overhyped emerging technologies (Gomes, 2014). They point out that the idea of backpropagation is not new and recent progress has not involved any major new conceptual breakthroughs, but a series of refinements that already existed in the 70 s and 80s. In addition, the often-used parallel of Deep Learning with the human brain is a gross oversimplification. Deep Learning is just a tool which is successful in certain, previously very challenging domains like speech recognition, vision, and natural language processing. In addition, big data with a large number of features naturally lead to a very large hypothesis set and overfitting (leading to many false positives) is inevitable; applications of ML techniques to big data without the theoretical backings of statistical analysis and validation are bound to fail.

In this paper, we will make a critical assessment of the recent developments in machine learning, Deep Learning and reinforcement learning in particular. We will see what motivates the use of a deep network architecture, why it has not seen use until now, and what recent progress has been made that enables its current success. We will also examine the impact of the recent developments in reinforcement learning using the example of Alpha-Go, which is a recent success story in automated game playing that caught the world's attention. We will then examine how these developments may impact the fields of process and energy systems of engineering and point out some promising directions for their application.

The rest of the paper is organized as follows. In Section 2, we present a critical assessment of Deep Learning including its motivation, early problems, and some recent resolutions as well as remaining challenges. In Section 3, the idea of reinforcement learning and its history are briefly introduced and the specific ML approaches used to train Alpha-Go are discussed. Some similarities between the Go game and the multi-stage decision problems in industries are brought out, pointing to the practical potential of the overall approach. In Section 4, some potential applications in the process and energy systems engineering domains are introduced. Naturally, this section carries the authors' own bias and opinions. Section 5 concludes the paper with a summarizing perspective.

2. Deep learning

2.1. Historical review

The neural network community reached the conclusion in the early 1990′s that training multi-layered networks with backpropagation, or really any gradient following algorithm, was essentially impossible, and the solutions obtained with deeper neural networks starting from random initialization perform worse than the solutions obtained for networks with 1 or 2 hidden layers (Bengio et al., 2007; Tesauro, 1992). Specifically it was shown that the weights in multi-layer networks would tend to shrink to zero or grow without bound (Hochreiter, 1991) and that the networks would have a very high ratio of saddle points compared to local minima (Bengio et al., 1994). Confining neural networks to shallow architectures essentially meant that features had to be coded before the network input, or the specific kernel-type mappings had to be embedded into the shallow architecture. Time series data was particularly difficult to handle. There was a clear need to find an approach where the learning of features and the learning of the mapping of the features to classification or action could somehow be separated as combinations of these two learning tasks led to poor performance.

There was a breakthrough in 2006 with regards to how to train multi-layer networks. The breakthrough was inspired by the concept of hierarchical feature extraction, where each "layer" of the network robustly extracted features based on the output representation of the layer closer to the data. This was said to be biologically motivated by the architecture of visual image processing. The training was essentially unsupervised (Hinton et al., 2006). One architecture was particularly successful, the Convolution Neural Network (CNN) LeCun et al. (1990). (LeCun et al., 1998). This leads to the term "deep learning" where "deep" referred to the number of layers in the network (LeCun et al., 2015).

Deep networks are therefore a neural network architecture in which the lower layers of the network engaged in feature extraction and then the last layer then mapped these features to classification or action. This allowed for the concept of basic features being combined into more complex features, and for enabling certain important invariances in features to be embedded into the way the layer operated on the inputs from the layer below (Hinton et al., 2006; LeCun et al., 2015). The stage was now set for dramatic improvement in multi-layer network performance – but there was one more innovation that had an important synergy – hardware development.

In addition to the advance in the layer-by-layer training of the network, the use of GPU computational architectures to carry out the training led to an order of magnitude speed up. The combination of hardware and the explosion of data of various types, but particularly visual image and text, provided an additional boost to the effectiveness of deep neural network learning. This has led to some substantial successes in image recognition, handwriting tasks, and in game playing (Mnih et al., 2013; Mnih et al., 2015). Game playing has proved to be a very fruitful area because a game has well-defined rules and hence can be simulated exactly, this allows for simulated play and hence the generation of data and the bootstrapping of performance.

PSE researchers engaged with neural networks in the 1980′s but the efforts died out in the early 90′s due to precisely the problems that the AI community had identified. There have been attempts to use neural network approaches for process control (Saint-Donat et al., 1991; Ydstie, 1990) and fault detection (Naidu et al., 1990). PSE researchers recognized some of the features of these approaches and made significant contributions: notably auto-associative neural network (Kramer, 1992) and Wave-net (Bakshi and Stephanopoulos, 1993). The auto-associative neural network which is composed of several layers (mapping, bottleneck and de-mapping) is a realization of a general nonlinear PCA (Kramer, 1991). These auto-assocative structures can be stacked, such that the output of one is the input to the next triplet of layers, leading to the extraction and recombination of successively higher level features, which is the general approach of the deep learning networks. The hierarchical multiresolution wavelet based network, named wave-net, was first introduced by chemical engineering society (Bakshi and Stephanopoulos, 1993), where the wavelet transform played the role of the feature extraction and the last layer combined the features.

PSE community continued to work on regression approaches that can be seen as shallow architecture learning. Kernel PCA, LLE, etc. are all examples of this (Schölkopf and Smola, 2002). The main issue is whether the mapping has essentially a local field or a global field. For example, Gaussian kernels are local in the sense that the weights die away the further you are from the mean, which is often taken to be placed on the data itself (Bengio et al., 2006).

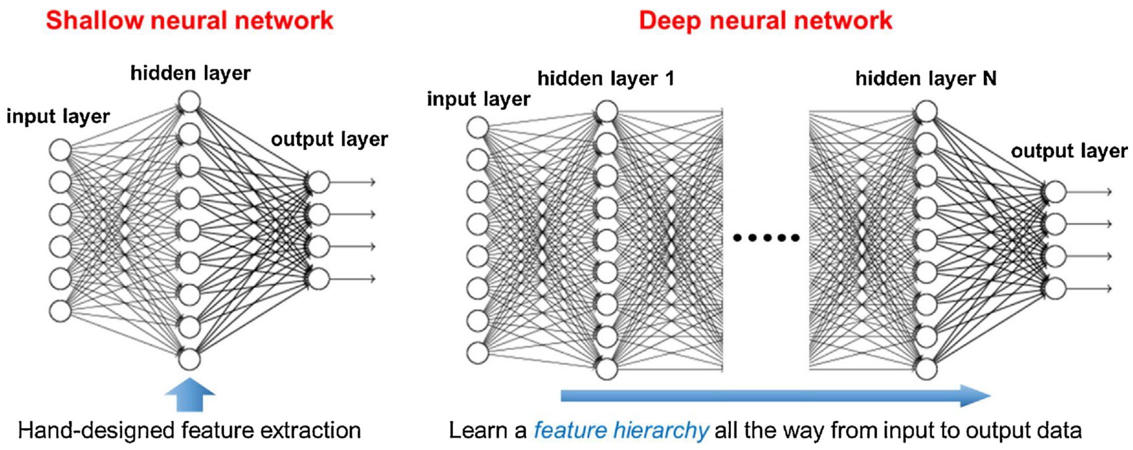

Fig. 1. The structure of shallow neural network (left) and deep neural network (right).

There are problems with this as it can be shown that although shallow architectures can in theory learn the same functions as deeper architectures they do so at the expense of potentially requiring exponentially larger numbers of internal nodes (Bengio, 2009).

2.2. *Shallow vs deep networks, and motivations for learning deep networks*

We distinguish deep networks from shallow by the number of hidden layers, shallow networks have one hidden layer that have nodes that perform a transformation of their inputs and sums them up, whereas deep networks have at least four or more and may have mixtures of types of layers (Fig. 1). Shallow networks encompass a wide variety of representation models for regression. For example, if the inputs are transformed through a basis function and then weighted this encompasses a wide range of efforts that have the same underlying representation but which differ in the way they train the network (Stulp and Sigaud, 2015). The general form of the node transformation is a local linear weighting of the basis function and with E hidden nodes this leads to the equation.

$$f(x) = \sum_{e=1}^{E} \phi(x, \theta_e).(a_e^T x + b_e) \tag{1}$$

The basis function computes the weighting for each linear regression model and hence captures local weighted linear regression, or with $a_e^T = 0$, this is a weighted basis function network.

Shallow networks can be conceptualized as using a combination of feature extraction, through the basis function or linear regression, and then weighting those features. Often the success of shallow networks has come through either already having appropriate features extracted and used as inputs to a relatively simple network that sums and thresholds the result, or through having basis functions that are effective for the feature extraction. For example, if $\theta_e = x_i$, x_i defined as one example of the training set, then $\phi(x, \theta_e) = K(x, x_i)$ computes the kernel operation, and efficiently allows high dimensional feature spaces to be employed (Schölkopf and Smola, 2002). The training then adjusts the weights associated with each kernel function and we recover the Support Vector Machine (SVM) representation and training approach.

A problem with shallow networks is that domain-specific features require expertise and insight of the task to be extracted (hand-crafted) and they can require an exponentially larger number of nodes in one fewer layers to represent a function (Bengio and LeCun, 2007) compared to a deeper network. This means that the number of nodes, and hence parameters, is much greater for the shallower network and the generalization properties can be expected to be poorer. On the other hand, deep networks can organize and store features in a hierarchical manner so as to allow reuse of those features that occur multiple times. Therefore, in contrary to the common misconception, the number of nodes and parameters needed to represent a function can be much smaller for a deep network compared to a shallow network (Fig. 2). This means there is an optimal depth for each given function.

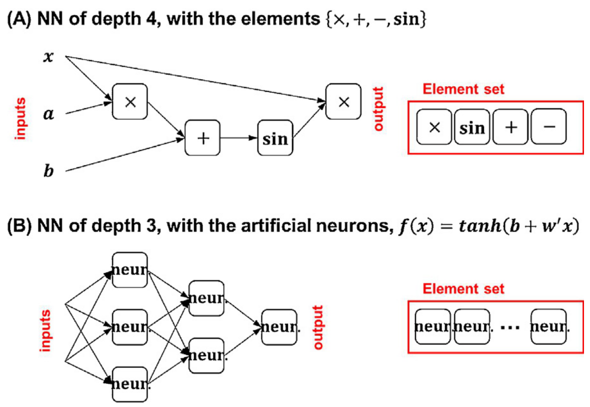
(A) NN of depth 4, with the elements {×, +, −, sin}
x
a
b
inputs
×
+
sin
×
output
Element set
× sin + −
(B) NN of depth 3, with the artificial neurons, f(x) = tanh(b + w′x)
inputs
neur.
output
Element set
neur. neur. ··· neur.

Fig. 2. Examples of function, $\boldsymbol{x} * \sin(\boldsymbol{a} * \boldsymbol{x} + \boldsymbol{b})$ represented by two different multi-layer NNs, where each node is taken in some element set of allowed computations (Bengio, 2009): (A) multi-layer NN of depth 4 with the elements are $\left\{\times, +, -, \mathbf{sin}\right\}$, and (B) multi-layer NN of depth 3 of which the elements are artificial neurons computing f(x)=tanh(b+w′x); each element in the set has a different parameter ($\boldsymbol{b}$, $\boldsymbol{w}$).

A second unrelated issue is that most kernel functions embody a distance metric over the training set examples and hence are inherently local. The need for non-local feature extraction, and the inefficiency of shallow network architectures, motivates the use of deeper networks (Bengio, 2009), along with biomimicry of biological neural networks that appear to have layers in the range of 5–10 just for visual processing. The problem, however, is that deeper networks have more complex and non-convex error functions, which makes them potentially harder to train with standard backpropagation and other optimization techniques.

2.3. *Deep networks – early problems and resolution*

In the early 1990′s the theoretical underpinnings of the reason behind the difficult training problem were established, a summary of which can be found in (Hochreiter and Schmidhuber, 1997). Essentially this demonstrated that the error propagated backward a significant distance in the network either decayed to zero, making training very slow, or blew up, making the training unstable.

Further theoretical analysis (Dauphin et al., 2014) showed that, for non-convex error functions in high dimensional spaces, naïve gradient descent was hampered by the occurrence of large numbers of saddle points, not local minima, with high error plateaus. This makes naïve gradient descent slow and gives the illusion of having found a local minimum. This type of error surface has been demonstrated to exist in multilayer neural network architectures (Baldi and Hornik, 1989). An alternative view of the difficulty of training deep networks is presented in Erhan et al. (2010) where the difference in the objective of training the layers was proposed as the key limitation. The layers nearest the data are posited to be extracting features whereas the upper layers are combining those features for the final task. These two different objectives combine poorly during training through back-propagation, and particularly the upper layers may over-fit the training set leading to poor generalization performance.

The poor practical performance of the multi-layered neural networks (MLNN) architecture discouraged applications in the 1990′s and early 2000′s. However, there was a breakthrough in training multilayer architectures that occurred in 2006 (Hinton et al., 2006). This approach called Deep Belief Networks (DBN) used unsupervised "pre-learning" of each layer, using a Restricted Boltzmann Machine (RBM), to extract useful features that were then fed to the next layer (Fig. 3). Here each layer is trained such that it reproduces its input on its output but compresses it through the node functions/weights (i.e. there are not enough weights or the algorithm is specifically designed to avoid finding the identity function). This connects to early chemical engineering literature and specifically Mark Kramer's work (Kramer, 1992) on recognizing the connection between Hopfield type networks (Hopfield, 1982) and PCA. The successively trained layers were then fine-tuned by the traditional supervised approach. The success of this technique has subsequently spurred successful applications of deep learning architectures to problems such as image classification (Huang et al., 2012; Lee et al., 2009), playing video games (Mnih et al., 2015) and the game of Go (Silver et al., 2016).

The success of the pre-learning phase of deep learning was explored experimentally in Erhan et al. (2010). This proposes that pre-learning finds features that are predictive of the main variations in the input space, i.e. P(X), and learning P(X) is helpful for learning the discriminative P(Y|X). Erhan observed that if the number of nodes in the early layers was not large enough then the pre-learning made the error worse, suggesting that although features may have been extracted, there were not enough of them to form good predictors for the final network output. The explicit focus on learning P(X) separates the objective of learning good features from the objective of learning how those features map to good classification. It has also been demonstrated that learning generative models, P(X,Y), has better generalization error than discriminative learning. This connects deep learning to shallow learning architectures where the input to the shallow network has undergone pre-processing to find robust features of the raw input space, such as PCA.

The recurrent neural network (RNN) architecture (Tsoi and Back, 1997) has been proposed as a way to learn models of time series. The main idea is that RNN's are like deep NN's because we can think of each iteration of the recurrence as if it was another layer in the network. Therefore, similar to multilayer neural networks, training RNN's has been hard using back-propagation and there are some fundamental limitations (Zhang et al., 2014). The key difficulty is discovering long term dependencies, and the fixed points of the dynamic system are hard to converge to. Bengio et al. (1994) demonstrates that a system can be efficiently trainable by gradient descent but not simultaneously stable and resistant to noise for a simple task that requires information to be stored for an arbitrary duration. This results from the fact that storing information resistant to noise makes the gradient with respect to past events small and that small changes in the weights are felt by events that are close to the current time.

2.4. Extensions and open challenges

However, deep architectures can only go so far. When the dimension of the raw input space becomes too large, or the time delays that occur between the important information and the need to use it become long, we need to have a yet further refinement to the architecture that enables attention to be shifted to different parts of the input space (Ba et al., 2014), and to have long term memory elements (Hochreiter and Schmidhuber, 1997).

The idea of shifting attention to different parts of the input space is essentially the exploration vs exploitation tradeoff that is seen in many problems. We need to balance looking hard in the current location for the solution versus moving to another part of the space in which we may find it. Exploitation vs exploration is a meta-level algorithmic component which must be explicitly designed into the agent as essentially it implies a change in the objective function of the problem. If you have an optimization formulation, you cannot do this within the formulation, you have to do it by some procedure outside of the optimization algorithm itself. It is essentially saying "we need to change behavior because our current behavior is inappropriate to the situation." These meta-algorithmic decisions make us uncomfortable.

The implementation of long term memory gates information into an element that preserves it and then must be gated out when the conditions in the environment indicate that the information needs to be used. This is a meta-architecture component. The inclusion of memory elements into the architecture of the neural network is something that has to be selected, similar to deciding to including convolution and down-selection layers in the network. These hyper-parameter and structural decisions make us uncomfortable.

3. Reinforcement learning

3.1. Introduction and history

Supervised or unsupervised learning methods learn to perceive/encode/predict/classify patterns, but they do not learn to act or make decisions. Reinforcement learning (RL), on the other hand, by actively interacting with an uncertain and dynamic environment, learns an optimal decision policy mapping the system state to the optimal action. RL does this by (1) observing the real-time responses of the environment when random or non-optimal actions are taken and (2) learning, either implicitly or explicitly, the cost associated with the given state (and possibly the action). The typical setting is one of *dynamic decision-making* meaning a series of decisions are to be made in a dynamic, and possibly uncertain, environment to minimize overall cost (e.g., sum of stage-wise costs). Such sequential decision-making problems are found in various fields such as automatic control, scheduling, planning, and logistics. This is a popular subject in PSE as most problems in control, real-time optimization, process scheduling, and supply chain operation areas share these features.

RL differs from SL in that it uses evaluative feedbacks from the environment to estimate real-valued rewards or costs versus instructive feedback used in SL through classification errors, and input events affect decisions made at subsequent times and the resulting output events, which is inherent in the nature of a dynamical system. An iterative step of RL is generally composed of two-steps: policy evaluation, wherein the long-term consequences of current decisions are characterized by a critic, followed by pol-

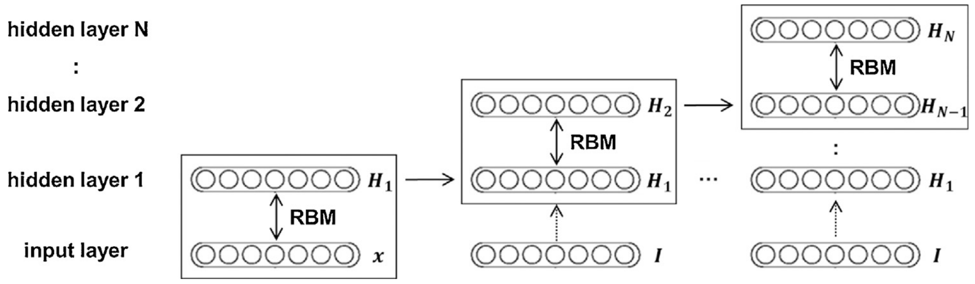

Fig. 3. A typical DBN with one input layer and N hidden layers $\mathbf{H}_1, \ldots, \mathbf{H}_N$. The layer-wise reconstruction is implemented by a family of restricted Boltzmann machine (RBM).

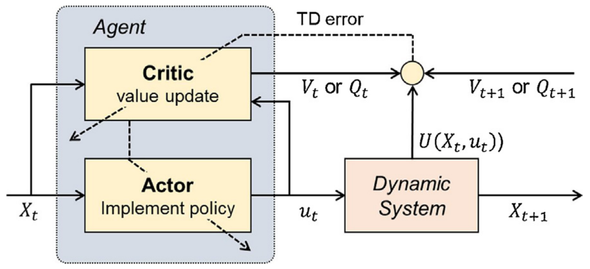

Fig. 4. The structure of typical RL algorithm: TD learning (equivalent to HDT) or Q-learning (equivalent to ADHDT).

icy improvement, which tries to modify the current policy based on the critic's evaluative feedback (Fig. 4). Naturally, RL can be applied to control problems involving temporally extended behavior.

In the literature, RL is studied and referred to by several alternative names (Sutton and Barto, 1998) including approximate dynamic programming (ADP) (Powell, 2007), neuro-dynamic programming (NDP) (Bertsekas and Tsitsiklis, 1995), and adaptive critic designs (ACDs) (Prokhorov and Wunsch, 1997). Werbos used a broad and inclusive term, reinforcement learning approximate dynamic programming (RLADP), to unite these overlapping strands of research and technology (Werbos, 2012). ADP broadly encompasses all computationally feasible tools for finding the most accurate possible solution to the Bellman equation, as required for optimality associated with dynamic programming. In the perspective of control theory, RL may be seen as adaptive optimal control, which provides an adaptive controller that converges to the optimal one (Lewis and Vrabie, 2009).

The general discrete-time stochastic system of RL, or feedback control, tasks is expressed as the 1-step Markov model $X_{t+1} = F(X_t, u_t, w_t)$, wherein the completely observed system X_t is evolved based u_t and the realized random variable w_t represents the uncertainty or disturbance realization. The objective at each time t is to find π indicating which decision should be made given a state so as to minimize (or maximize) the sum of all future stage-wise cost (or reward) functions from time t:

$$\min_{\pi(\cdot)} E\left[\sum_{\tau=t}^{\infty} \gamma^{\tau-t} U(X_\tau, u_\tau = \pi(X_\tau)) | X_t\right] \tag{2}$$

where $0 < \gamma \le 1$ is discounted factor, and $U(\cdot, \cdot)$ is the one-step cost function or performance measure. Mathematical formulation of such a sequential decision problem is called the Markov decision process (MDP) (Puterman, 2014).

The MDP formulated problem is solved exactly by dynamic programming (DP), in which the value function (or cost-to-go) evaluating the sum of all discounted future costs under optimal decision policy is introduced in the following temporally recursive form (Bellman optimality equation)

$$V^*(X_t) = \min_{\pi(\cdot)} \left\{ U(X_t, \pi(X_t)) + \gamma E\left[V^*(X_{t+1}) | X_t, \pi(X_t)\right] \right\} \tag{3}$$

Once the value function is obtained by solving the above, the corresponding optimal policy can be easily derived by solving the following single step optimization:

$$\pi^*(X_t) = arg \min_{\pi(\cdot)} \left\{ U(X_t, \pi(X_t)) + \gamma E\left[V^*(X_{t+1}) | X_t, \pi(X_t)\right] \right\} \tag{4}$$

Note that the above is for an infinite horizon problem and there exist finite-horizon counterparts. In a finite horizon decision problem, the Bellman equation (above Eq. (3)) can be solved by backward induction generally applicable for off-line planning problems. On the other hand, iterative methods are developed based on a fixed-point estimation of Eq. (3) leading forward-in-time algorithms such as value iteration (VI), policy iteration (PI), or generalized policy iteration (GPI). However, the exact solution by DP is thought to be inapplicable to most practical problems due to the computational challenge brought by huge dimensions of state and action space, which is referred to as the 'curse of dimensionality'. RLADP has been developed as a practical method for evolving a decision policy towards the optimal (or near-optimal) one online, forward in time, by using measured performance feedbacks along the system trajectories.

The Bellman equation is replaced by a recursion equation and the value function is iteratively updated based on the time varying prediction error, which is referred to as 'temporal difference (TD)'. Such a procedure is a form of bootstrapping, and it leads to a forward-in-time algorithm, which serves as a basis for constructing an on-line incremental policy in response to actions applied to the system. Sutton called this TD learning (Sutton and Barto, 1998). Furthermore, some sort of function approximation architecture, *e.g.,* a user-specified parametric model, a neural network, a nearest neighbor averager, Taylor series, etc., is employed to represent the value function or action policy, which is the reason for the names like ADP or NDP.

Werbos introduced a family of approaches for ADP, which he called adaptive critic designs (ACDs). ACD was defined broadly to include those employing any approximator of the value function with tunable weights or parameters and for which there is a general method to train, adapt, or tune the weight or parameter values. He provided a general algorithm for training the value function, called heuristic dynamic programming (HDP) (Werbos, 1977), which is essentially the same as TD learning by Sutton. Werbos also introduced the dual heuristic programming (DHP) algorithm for estimating the gradient of the value function $\lambda_t = \partial V(X_t)/\partial X_t$ (called 'costate function'), rather than the function value itself (Werbos, 1990). The costate function is interpreted as a marginal value of the state, so it carries information more than just the function value.

All these require a model of system dynamics, but in many real problem, system dynamics may be incompletely known and difficult to identify empirically. To resolve this issue, a value function with the argument of state-action pair, known as the Q (quality) function, was introduced (Watkins, 1989).

$$Q^*(X_t, u_t) = U(X_t, u_t) + \gamma \min_{u'} E\left[Q^*(X_{t+1}, u')|X_t, u_t\right] \tag{5}$$

$$\pi^*(X_t, u_t) = arg \min_{u(\cdot)} Q^*(X_t, u_t) \tag{6}$$

Therefore, by introducing a more general form of the value function taking the additional argument (Werbos, 1992), model-free designs can be established which directly learn optimal policy using observed data $\{X_t, X_{t+1} U(X_t, \pi(X_t))\}$. Werbos called this action-dependent HDP (ADHDP), which is better known as Q-learning in the RL community. NN for approximating the Q function can be deeply extended, which is called 'deep Q-network (DQN)' (Mnih et al., 2015). DQN uses a representation generator capturing features from the observed states as well as an action scorer to produce scores for all actions and argument objects.

Another important question in RL is which decision the RL agent (the decision-maker or the controller) should conduct to quickly learn the uncertain information (e.g., the value function parameters) that govern the behavior of resulting policy. This leads to the classical trade-off between exploration and exploitation. To assure the convergence of Q-function, all the state-action pairs should be visited infinitely often (in an asymptotic sense), which is only satisfied when all actions have nonzero probability of being selected by the decision policy during the training. That is, from the viewpoint of learning, decisions should be made for the purpose of attaining more valuable information about the model and uncertainty at hand ('exploration'). On the other hand, to achieve good results now, one should choose the best option based on the current knowledge ('exploitation'). Optimal balancing between the two is needed to achieve the best overall performance.

Bayes-Adaptive MDP (BAMDP) (Duff and Barto, 2002; Martin, 1967) is a MDP problem formulation that incorporates the knowledge on the uncertain parameters or states (e.g., their probability distributions) as state variables (referred to as knowledge or belief state to distinguish them from the regular state). The knowledge state may acquire the extra information given the data observed so far, and evolve to the posterior belief distribution over the dynamics. Suppose that uncertain cost U is multi-variate, normally distributed, and a knowledge state with n measurements is the current mean and covariance to represent the beliefs, i.e., $X^n = (\mu^n, \Sigma^n)$. The state is then updated by using the Bayesian rule after taking a decision u^n and observing a measurement obtained from the corresponding decision: $X^{n+1} \longleftarrow Bay(X^n, u^n, \hat{\mu}^{n+1}_{u^n})$. The objective is to find decision policy π that minimizes the expected sum of the cost over a time horizon.

$$\min_{\pi} E^{\pi}\left[\sum_{n=1}^{N} \gamma^n \mu_{u^{\pi,n}(X^n)}\right] \tag{7}$$

Optimal learning (better known as the dual control problem in the control community (Feldbaum, 1960)) techniques have been developed to explicitly optimize the knowledge evolvement through the input decisions by modeling a trade-off between the objective to maximize the current reward and that to increase the value of information (Powell and Ryzhov, 2012). There are several heuristic algorithms for balancing the objectives of exploitation and exploration. In the simplest case, one explores with probability ε and exploits with probability 1-ε. Here, random decisions are chosen, with uniform probability (ε-greedy policy), or weighted probability (Boltzmann exploration or a soft-max policy). Interval estimation can also be used for tuning an interval of possible (or probable) values of an unknown population parameter, in contrast to point estimation.

The Gittins index policy (Gittins, 1979) is known as the optimal policy for an infinite horizon on-line problem with independent beliefs (diagonal covariance), known as 'multi-armed bandit'. The Gittins index policy is to select an option having the highest Gittins index, which is a measure of the reward that can be achieved by a random process evolving from its present state onward towards a termination state, under the option of terminating it at any later stage with the accrual of the probabilistic award from that stage up to the termination state. For evaluating the Gittins index, proper approximation architectures are studied (Brezzi and Lai, 2002; Chick and Gans, 2009).

The knowledge gradient (KG) algorithm is one step roll-out policy, first introduced by Gupta and Miescke (1996) and further developed by P. Frazier et al., 2008. The main idea is to use the marginal value of information gained by a measurement. The *knowledge gradient* of each decision at time n is defined by expectation of the incremental random value of the newly made measurement, $\nu_u^{KG,n} = E_u^n\left[\min_{u'} \mu_{u'}^{n+1} - \min_{u'} \mu_{u'}^n\right]$, and the decision is then made by optimizing both μ_u^n and exploration bonus computed from $\nu_u^{KG,n}$.

Unlike the index policies, the value of choosing decision u depends on the knowledge about all other decisions $u' \neq u$ by the KG policy. Thus the KG policy does not have the theoretical strength of the most index policies, but it is more suitable for a specific problem class since it can capture the correlation between different decisions (Ryzhov et al., 2012). The exact algorithm for computing $\nu_u^{KG,n}$ is specified in Frazier et al. (2008, 2009) for independent and correlated belief models, respectively. Instead of a lookup representation of a belief model, a parametric belief model has been used to deal with the curse-of-dimensionality, as in the case of ADP (Negoescu et al., 2011).

For general MDP problems, physical state or exogenous information variables should be considered as well as the knowledge state. With the view of the value function capturing all the effects of uncertainty, a Bayesian prior belief is introduced on Q or V itself, rather than on specific parameters. Dearden et al., 1998 proposed a model-free BAMDP algorithm with Q-value distribution, called Bayesian Q-learning (BQN), and they extended this work to the case where there is an explicit model with uncertain parameters (Dearden et al., 1999). Based on these works, there have been studies to apply optimal learning techniques to DP as exploration strategies, involving the Bayesian DP (Strens, 2000) and the BEETLE algorithm (Poupart et al., 2006). Recently, Ryzhov and Powell (2010, 2011) adapted the KG policy to problems with a physical state, in which the total infinite horizon discounted reward has a normal distribution, which leads to relatively simple belief states capable of capturing correlations.

3.2. Alpha-go and its impact

In 1997, IBM's Deep Blue defeated the then reigning world chess champion Garry Kasparov by 3.5-2.5 in a six-game match. It marked the first time that a computer defeated a reigning world champion in a standard time-controlled chess match. Deep Blue took advantage of the brute force computing power rather than computer intelligence; being one of the world's most powerful supercomputers at the time, it typically searched to a depth of six to eight moves to as many as twenty or more in some situations. Though successful in the game of chess, many experts doubted that such a brute-force approach would be extendable for more complex games like Go. In fact, typical depth and number of candidate moves in the Go game are much higher, ~150 and ~250 respectively, compared to chess. This makes the number of possible threads to be approximately

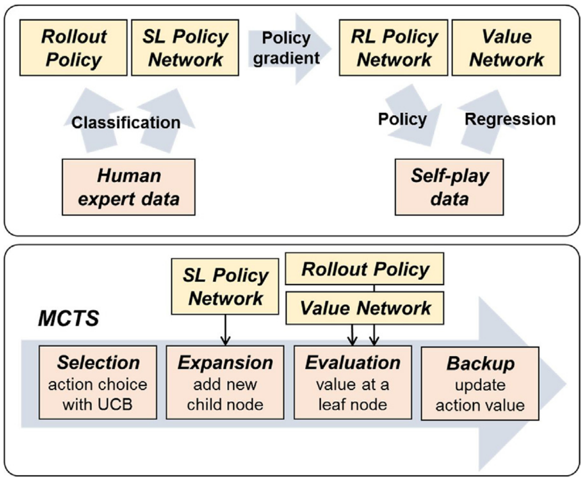

Fig. 5. AlphaGo algorithmic framework for NNs training (above) and search algorithm (below) (Silver et al., 2016).

$250^{150} \approx 10^{360}$, which is clearly beyond the realm of computation. Deep Blue's approach relying on sheer computational power rather than intuition or learning would not be effective for the Go game (Browne et al., 2012) and a new paradigm was needed.; All the successful Go-playing computer programs developed over the last decade relied on the use of smart sampling (Monte Carlo simulation) for efficient exploration and curbing the depth and breadth of tree expansion (Chang et al., 2016).

Recently, AlphaGo developed by Google DeepMind made history when it defeated South Korea's Lee Sedol, an 18-time world Go champion. AlphaGo uses an algorithm to combine Monte Carlo tree search (MCTS) and two deep NNs. Policy network is to estimate a probability distribution P(u|s) over legal moves u, taking a board configuration s as input, and value network is to predict the expected outcome (win or defeat) v(s) in board positions (Silver et al., 2016). The policy network allowed for an efficient search over a huge number of legal moves and the value network helped reduce the depth of moves to be searched.

Details of the algorithm of AlphaGo is published in Silver et al. (2016) (Fig. 5, above). In a prior learning step, a policy network (13-layer CNN) is trained by SL (classification) based on data sets containing 28.4 million positions from human expert games. Subsequently, the trained SL policy network is continually improved by policy gradient RL via self-play with a randomly chosen previous iteration of the policy network. The policy network helps reduce the number of action candidates (breadth reduction). Then, a value network (15-layer CNN) is trained to approximate the value function of the RL policy network by regression from 30 million positions, each drawn from a unique game of self-play. The fact that a system can play against itself (self-play) and improve over time means that the number of games that are explored by AlphaGo are many orders of magnitude higher than those of a human player. This complements the deficiency that the prior knowledge of the processing system is not as good.

AlphaGo provides an effective action search algorithm through intelligent simulation using MCTS (Fig. 5, below) with upper confidence bounds (UCBs), and policy and value networks. Along a search tree (from the root node to a leaf node), the action with higher action values plus an exploration bonus (UCB) is selected at each node, $arg\max(Q(s_t, a_t) + u(s_t, a))$ (selection). The value at the leaf node is then obtained by the value network and further propagating the simulation with a fast rollout policy (trained a priori) until a terminal state is reached (evaluation). Then, the action values Q and visit counts of all visited edges are updated (Backup). If the visit count exceeds a threshold, the successor state is added to the search tree of which prior probability is computed by the SL policy network (expansion). At the end of simulation, the most visited move is chosen from the root position.

3.3. Extensions and challenges

Many games of perfect information, such as chess, checkers, othello, backgammon, and Go, may be defined as alternating Markov games (Littman, 1994) with appropriate components (e.g. state, action, state transition, reward, objective function) definition by a set of established rules to attain specified outcomes. In the case of AlphaGo (Silver et al., 2016), the randomness in the state transition of the MDP by an opposing player is eliminated by the zero sum game assumption, and reward is zero for all non-terminal time-steps. The outcome of the game is the terminal reward at the end of the game from the perspective of the current player at current time-step: +1 for winning and −1 for losing. Zero sum games have a unique optimal value function that determines the outcome from each state following perfect play by both players.

As explained in the previous sections, MDPs are powerful analytical tools used for formalizing sequential decision making under uncertainty and have been widely used in many industrial and manufacturing applications. AlphaGo is considered a major achievement by the AI field, but its efficient algorithm results from several research streams such as NNs, learning algorithms, MCTS and multi-armed bandit models, performed by various research communities including operations and systems engineering. Chang et al. (2016) pointed to the practical potential of the overall approach of AlphaGo as following: "*What makes the work perhaps more impressive is that the machinery behind AlphaGo can be easily adapted to other games and applications. In other words, the general NN/MCTS architecture can be viewed as a very general-purpose tool for games or sequential decision-making under uncertainty, much like regression is used for the social sciences.*"

MCTS, which is a core element in AlphaGo, is based on the upper confidence bounds for trees (UCT) algorithm (Kocsis and Szepesvári, 2006). The two 2006 AI conference papers (Coulom, 2006; Kocsis and Szepesvári, 2006) received all the publicity (and citations) of this study. However, according to their citation, the adaptive multi-stage sampling (AMS) algorithm for estimating the value function in finite-horizon MDPs, proposed by Chang et al., 2005 in Operations Research, is the first work to explore the idea of UCB-based exploration and exploitation in constructing sampled/simulated trees, which is clearly the main basis for UCT.

MCTS attempts to balance considerations of exploration and exploitation. According to the review paper on MCTS (Browne et al., 2012), while full-depth minimax, which is optimal in the game-theoretic sense (used for Chess game), depends significantly on the heuristic evaluation function, MCTS does not need domain-specific knowledge, so it can be used in the cases where it is difficult to formulate useful heuristics due to the huge size of state space like in the Go game. Another benefit of MCTS is to allow the tree grow asymmetrically as the method concentrates on the more promising nodes. Thus it achieves better results than classical algorithms in games with a high branching factor.

MCTS, being effective for decision-making problems with large state space through effective exploration, has been applied to problems of many domains beyond games ranging from planning, scheduling, and optimization to function approximation (De Mesmay et al., 2009; Kocsis and Szepesvári, 2006), production management (Chaslot et al., 2006), sample-based planning with large state space (Walsh et al., 2010), and feature selection (Gaudel and Sebag, 2010). While the basic algorithm of MCTS has proven to be effective for a wide range of problems, continuous efforts are needed to adapt it to each specific problem, and at the same time

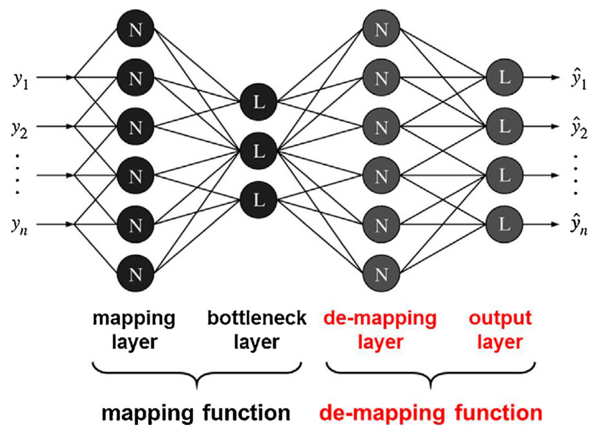

Fig. 6. The structure of associated neural network which is composed of several hidden layers: mapping, bottleneck, and de-mapping layer (Kramer, 1991).

to generalize so that an improvement in one domain can extend to others (Browne et al., 2012).

Meanwhile, deep Q-Network (DQN) introduced by Mnih et al. (2015) inaugurated a new field of deep RL. To stabilize the training of Q-function approximation by a nonlinear function like deep NNs, DQN uses experience replay (sampling random mini-batch from the replay memory data) and fixed Q-target (computing Q-learning target with respect to the fixed parameters obtained from the previous batch). Another contribution of DQN is to use an end-to-end learning approach requiring minimal domain knowledge as inputs (Li, 2017). The original DQN is further improved through double Q-learning (Van Hasselt et al., 2016), prioritized experience replay (Schaul et al., 2015), and dueling network (Wang et al., 2015) to deal with problems such as overestimation and convergence of Q-function.

Li (2017) reviewed recent studies related to policy-based and model-based as well as value-based deep RL; he also mentioned some opportunities for theoretical and conceptual advances including the following research directions: how to explore the large state space, how to have a policy that involves actions of different timescales, or has subgoals (hierarchy), and how to combine RL with unsupervised learning.

4. Implications and opportunities for process systems engineering

4.1. Nonlinear PCA for data compression and dimensionality reduction

Autoencoder (AE) is an artificial neural network used to learn an efficient representation (encoding) of a given data set. It is a form of unsupervised learning. It shares similarities to principal component analysis (PCA) used for dimensionality reduction of data. Whereas PCA typically finds representations consisting of linear combinations of the variables (or preselected features), AE generalizes this to nonlinear combinations, so it can be viewed as a nonlinear generalization of PCA. AEs can have multiple hidden layers and therefore are deep networks.

It is an interesting historical note that the PSE community has seen a related concept in the early 90's. Kramer (Kramer, 1991) proposed auto-associative neural networks (AANNs), which are feedforward networks trained to produce an approximate identify mapping between inputs and outputs. The key feature is a dimensionality bottleneck introduced by the middle layer, which compresses and encodes the input information on a lower dimensional space (Fig. 6). In fact, he referred to this as nonlinear PCA and pointed to potential applications in a variety of data screening tasks such as data rectification, gross error detection, fault identification and inference.

Though useful in concept, the work did not see much progress, mainly due to the aforementioned difficulty of training such a network by backpropagation (BP). There are some fundamental issues with using BP for training networks with many hidden layers, as mentioned earlier. Once errors are backpropagated through the first few layers, the error gradient becomes essentially zero, thus halting the training process. It wasn't until Hinton (Hinton et al., 2006) developed the technique for pretraining deep networks by treating each neighboring set of two layers as a restricted Boltzmann machine, which is a form of autoencoder. As said before, this technique can find a good starting point to further train the networks by BP. Now, with the recent developments to train deep networks (e.g., AANNs), we are in position to reexamine these concepts for many applications.

Besides data compression, AANNs can serve a useful role in nonlinear model reduction. PCA is the basis for a data-based nonlinear model reduction technique called proper orthogonal decomposition (POD) (Chatterjee, 2000), which has been mainly used to obtain low-dimensional descriptions of discretized fluid flow equations, esp. turbulent, multi-dimensional ones (Berkooz et al., 1993). In POD, conventional PCA is used to find orthogonal vectors that represent 'model shapes' or basis functions based on simulation data. Once a representation of the data in significantly lower dimension is developed, the dynamic model can also be re-expressed (or fitted) in the lowered dimension. With the use of conventional PCA, however, the modal shapes or the basis functions get fixed and are not allowed to change with the operating range. By using the nonlinear PCA afforded by the AE or AANN, one can therefore generalize the POD so that it can explain the nonlinear dependency of the model shapes.

4.2. Hierarchical feature learning of time-series or image data for monitoring and diagnosis

A notable and attractive aspect of deep learning is that important features of the input data are extracted in a hierarchical manner as they are processed through the front layers of the network, before mapped to the output. Hence, integral to deep learning is hierarchical feature learning, which bears some similarity to multi-resolution analysis through wavelet decomposition of time series data or image data. Such data decomposition with varying details of features can be very useful in process monitoring and diagnosis.

The idea of using resolution-resolved features of signals for enhanced monitoring and diagnosis appeared in the early '90 s in the process control community. Bakshi and Stephanopoulos (1993) introduced the Wave-net architecture, which is a neural network with one hidden layer of nodes, whose basis functions are wavelets (Fig. 7). They claimed that the time-frequency localization characteristics of wavelets enable efficient multi-resolution learning of input-output maps as well as hierarchical representation of signals leading to effective fault detection and diagnosis. However, the Wave-net is a shallow network with preselected features (wavelet basis functions), which can be an inefficient representation of large time-series or image signals. As mentioned, deep learning automatically generates highly efficient and compact hierarchical representations of input data. Such generated hierarchical features can potentially provide some new information about the signal that can be useful for fault detection and identification.

4.3. Integration of planning and scheduling

A longstanding challenge in PSE is the integration of the layers in the vertical operation hierarchy (Grossmann, 2012). An example is the integration between planning and scheduling (Maravelias

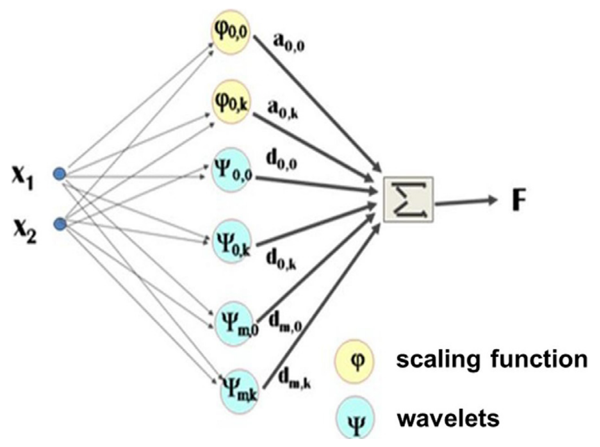

Fig. 7. Typical structure of Wave-net (Bakshi and Stephanopoulos, 1993).

and Sung, 2009). The difficult stems from the fact that time scales involved in different hierarchies can be quite different. Hence, by writing the problem of two or more adjacent layers into a single mathematical program can lead to a very large problem that is difficult to handle computationally. Considering the uncertainty exacerbates the issue significantly. Scenario-branching used in stochastic programming can explode (in an exponential manner) as the number of time steps or stages increases. This is the reason why stochastic programming has seldom seen applications beyond two stage problems.

In integrating planning and scheduling, the high-level decision problem (i.e., planning) typically has recurring dynamics and decisions and are dominated by uncertainties. Therefore, they are naturally expressed as Markov decision processes (MDPs), for which (stochastic) dynamic programming provides a solution. On the other hand, the lower-level decision problem (i.e., scheduling) tends to have more complex state representation and dynamics and is better suited for the mathematical programming approach; in some cases, certain heuristic rules exist that are already proven highly effective. Since RL or ADP solves MDP by simulating the stochastic environment and iterative improving a decision policy by learning, it is a very fitting approach for integrating the two layers. The whole integrated system can be simulated easily, regardless of whether a mathematical programming based solution or heuristic rules are used for the lower layer. Such approaches have been explored extensively by the authors over the last decade or so, and have shown some success and potential (Lee and Lee, 2006; Shin and Lee, 2016; Shin et al., 2017).

4.4. Control

Optimal control problems also employ Markov models (e.g., state-space models with random noise inputs) and stagewise cost functions, and therefore can be viewed as MDPs. The main difference from those studied in the operations research field is that the state and action spaces are continuous, leading to very high-dimensional state and action space when discretized. Hence, the curse-of-dimensionality is much worse for typical control problems. Nevertheless, RL can provide a useful route to solving stochastic optimal control problems, which are generally not solvable by current other techniques, including the ever popular model predictive control (MPC) technique. Efforts have been made in using ADP to solve control problems with stochastic uncertainty. In the PSE domain, the authors have used the approach to handle some challenging problems like the dual adaptive control problem and more general nonlinear stochastic optimal control problems, and have shown that it can provide better-performing solutions compared to other classical optimal control techniques for these types of problems (Lee and Wong, 2010; Lee and Lee, 2004, 2005, 2009).

One of the keys to successfully applying RL to control problems is choosing a proper form of the value function approximation. In general, because the state space is continuous and the state variables tend to evolve in a highly correlated manner, the piecewise linear, separable approximation popular in planning problems is not effective. Instead neural networks and nonparametric approximators like the k-nearest averager have been used for control problems (Bertsekas and Tsitsiklis, 1995; Lee et al., 2006). Here deep learning can provide a tool for an effective function approximation. Note that Alpha-go used deep neural networks with 10–20 hidden layers for the policy and value approximations.

4.5. Identification of uncertain mapping and dynamics

Most deep networks inherently represent probabilistic input-output relationships. That is, these networks map the input to the probability distribution of the output (which are typically discrete) (Baum and Wilczek, 1987). Hence, they can be useful in representing complex uncertain relationships. In particular, such feature can be used to identify Markov state transition dynamics that may be too complex or nonlinear to represent by a simple additive noise. For example, suppose we are interested in identifying Markov transition mapping $X_t \rightarrow X_{t+1}$ where X occupies a continuous state space and the transition is stochastic. One can use deep learning to create a mapping between X_t and the probability distribution of X_{t+1} in an appropriately discretized state space using simulation or real data. Such a mapping can naturally serve as a Markov transition model of the process dynamics.

5. Conclusion

Some significant recent developments in machine learning like deep learning and reinforcement learning open up new possibilities in many application domains. Deep learning enables efficient training of neural networks with a large number of hidden layers, which in turn allow for hierarchical feature learning of the input data. This will become increasingly effective as data streams are generated and curated at different scales and plant locations. Developments in reinforcement learning like approximate dynamic programming allow us to obtain optimal or near optimal policy for multi-stage stochastic decision problems. Both of these learning techniques are embarrassingly parallel, and thus able to take advantage of the trends in hardware development that emphasize parallel multicore computation versus more capable single cores. In the PSE domain, these developments represent opportunities to push the technological boundaries in important problems like fault detection/diagnosis, nonlinear model reduction, integrated planning and scheduling, and stochastic optimal control.

So the question is whether it is time for the PSE community to re-engage in this space and with what research problems in mind. The following are a few that come to the authors' minds:

a Exploration vs exploitation – this is an old problem, but one which has not attracted the community's attention except with the concepts of dual control. This is a problem that we need to return to with the idea that advances in the computational power make this feasible. The problem is that we are unlikely to have crisp theoretical results beyond those already established, unless we can find specific sub-problem structures that are relevant to PSE.

b Deep neural network architectures – is it time for the community to see if these can be usefully applied in PSE problems and which types of problems would they seem best suited to? There are lots of choices in the structure of networks and types of nodes to use in these networks; these decisions, much as earlier less complex decisions about the number of nodes and layers to include, are not easy to make and are disconcerting. Furthermore, what is the academic value?

c Reinforcement Learning – this has the exploitation vs exploration tradeoff built into its learning, but may require changing the architecture to explicitly recognize the decision of switching between the two modes. This uses features to action (or to value) mapping and hence can be combined with the deep learning to come up with potentially interesting decision-making for stochastic multi-stage decision problems. The use of simulation, even with imperfect physical rules, can provide inputs that allow for better performance to be learned cheaply relative to experiments on real systems.

Our view is that investment in these areas by the PSE community will pay healthy dividends.

Acknowledgement

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government(MEST) (No. NRF-2015R1A2A2A01007102).

References

Ba, J., Mnih, V., & Kavukcuoglu, K. (2014). Multiple object recognition with visual attention. *arXiv preprint arXiv:1412.7755*.

Bakshi, B.R., Stephanopoulos, G., 1993. Wave-net: a multiresolution, hierarchical neural network with localized learning. AIChE J. 39 (1), 57–81.

Baldi, P., Hornik, K., 1989. Neural networks and principal component analysis: learning from examples without local minima. Neural Netw. 2 (1), 53–58.

Baum, E.B., Wilczek, F., 1987. Supervised Learning of Probability Distributions by Neural Networks. Paper Presented at the NIPS.

Bengio, Y., LeCun, Y., 2007. Scaling learning algorithms towards AI. Large-Scale Kernel Mach. 34 (5), 1–41.

Bengio, Y., Simard, P., Frasconi, P., 1994. Learning long-term dependencies with gradient descent is difficult. IEEE Trans. Neural Netw. 5 (2), 157–166.

Bengio, Y., Delalleau, O., Le Roux, N., 2006. The curse of highly variable functions for local kernel machines. Adv. Neural Inf. Process. Syst. 18, 107.

Bengio, Y., Lamblin, P., Popovici, D., Larochelle, H., 2007. Greedy layer-wise training of deep networks. Adv. Neural Inf. Process. Syst. 19, 153.

Bengio, Y., 2009. Learning deep architectures for AI. Found. Trends® Mach. Learn. 2 (1), 1–127.

Berkooz, G., Holmes, P., Lumley, J.L., 1993. The proper orthogonal decomposition in the analysis of turbulent flows. Ann. Rev. Fluid Mech. 25 (1), 539–575.

Bertsekas, D.P., Tsitsiklis, J.N., 1995. Neuro-dynamic programming: an overview. Paper presented at the Decision and Control, 1995. Proceedings of the 34th IEEE Conference.

Brezzi, M., Lai, T.L., 2002. Optimal learning and experimentation in bandit problems. J. Econ. Dyn. Control 27 (1), 87–108.

Browne, C.B., Powley, E., Whitehouse, D., Lucas, S.M., Cowling, P.I., Rohlfshagen, P., . . . Colton, S., 2012. A survey of monte carlo tree search methods. IEEE Trans. Comput. Intell. AI Games 4 (1), 1–43.

Chang, H.S., Fu, M.C., Hu, J., Marcus, S.I., 2005. An adaptive sampling algorithm for solving Markov decision processes. Oper. Res. 53 (1), 126–139.

Chang, H.S., Fu, M.C., Hu, J., Marcus, S.I., 2016. Google Deep Mind's AlphaGo. OR/MS Today.

Chaslot, G., De Jong, S., Saito, J.-T., Uiterwijk, J., 2006. Monte-Carlo tree search in production management problems. Paper Presented at the Proceedings of the 18th BeNeLux Conference on Artificial Intelligence.

Chatterjee, A., 2000. An introduction to the proper orthogonal decomposition. Curr. Sci. 78 (7), 808–817.

Chick, S.E., Gans, N., 2009. Economic analysis of simulation selection problems. Manage. Sci. 55 (3), 421–437.

Coulom, R., 2006. Efficient selectivity and backup operators in Monte-Carlo tree search. Paper Presented at the International Conference on Computers and Games.

Dauphin, Y.N., Pascanu, R., Gulcehre, C., Cho, K., Ganguli, S., Bengio, Y., 2014. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. Paper Presented at the Advances in Neural Information Processing Systems.

De Mesmay, F., Rimmel, A., Voronenko, Y., Püschel, M., 2009. Bandit-based optimization on graphs with application to library performance tuning. Paper Presented at the Proceedings of the 26th Annual International Conference on Machine Learning.

Dearden, R., Friedman, N., Russell, S., 1998. Bayesian Q-learning. Paper Presented at the AAAI/IAAI.

Dearden, R., Friedman, N., Andre, D., 1999. Model based bayesian exploration. Paper Presented at the Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence.

Duff, M., Barto, A., 2002. Optimal Learning: Computational Procedures for Bayes-adaptive Markov Decision Processes. PhD Thesis. University of Massachusetts Amherst.

Erhan, D., Bengio, Y., Courville, A., Manzagol, P.-A., Vincent, P., Bengio, S., 2010. Why does unsupervised pre-training help deep learning? J. Mach. Learn. Res. 11 (February), 625–660.

Feldbaum, A., 1960. Dual control theory. I. Avtomatika i Telemekhanika 21 (9), 1240–1249.

Frazier, P.I., Powell, W.B., Dayanik, S., 2008. A knowledge-gradient policy for sequential information collection. SIAM J. Control Optim. 47 (5), 2410–2439.

Frazier, P., Powell, W., Dayanik, S., 2009. The knowledge-gradient policy for correlated normal beliefs. Informs J. Comput. 21 (4), 599–613.

Gaudel, R., Sebag, M., 2010. Feature selection as a one-player game. Paper Presented at the International Conference on Machine Learning.

Gittins, J.C., 1979. Bandit processes and dynamic allocation indices. J. R. Stat. Soc. Ser. B (Methodol.), 148–177.

Gomes, L., 2014. Machine-learning maestro michael jordan on the delusions of big data and other huge engineering efforts. IEEE Spectr. 20 (October).

Grossmann, I.E., 2012. Advances in mathematical programming models for enterprise-wide optimization. Comp. Chem. Eng. 47, 2–18.

Gupta, S.S., Miescke, K.J., 1996. Bayesian look ahead one-stage sampling allocations for selection of the best population. J. Stat. Plann. Inference 54 (2), 229–244.

Hinton, G.E., Osindero, S., Teh, Y.-W., 2006. A fast learning algorithm for deep belief nets. Neural Comput. 18 (7), 1527–1554.

Hochreiter, S., Schmidhuber, J., 1997. Long short-term memory. Neural Comput. 9 (8), 1735–1780.

Hochreiter, S., 1991. Untersuchungen Zu Dynamischen Neuronalen Netzen. Diploma Thesis, Institut für Informatik, Lehrstuhl Prof. Brauer. technische universität münchen.

Hopfield, J.J., 1982. Neural networks and physical systems with emergent collective computational abilities. Proc. Natl. Acad. Sci. 79 (8), 2554–2558.

Huang, G.B., Lee, H., Learned-Miller, E., 2012. Learning hierarchical representations for face verification with convolutional deep belief networks. Paper Presented at the Computer Vision and Pattern Recognition (CVPR) 2012 IEEE Conference.

Kocsis, L., Szepesvári, C., 2006. Bandit based monte-carlo planning. Paper Presented at the European Conference on Machine Learning.

Kramer, M.A., 1991. Nonlinear principal component analysis using autoassociative neural networks. AIChE J. 37 (2), 233–243.

Kramer, M.A., 1992. Autoassociative neural networks. Comp. Chem. Eng. 16 (4), 313–328.

LeCun, Y., Boser, B., Denker, J., Henderson, D., Howard, R., Hubbard, W., Jackel, L., 1990. Handwritten digit recognition with a back-propagation network. Paper Presented at the Proceedings of Advances in Neural Information Processing Systems (NIPS).

LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., 1998. Gradient-based learning applied to document recognition. Proc. IEEE 86 (11), 2278–2324.

LeCun, Y., Bengio, Y., Hinton, G., 2015. Deep learning. Nature 521 (7553), 436–444.

Lee, J.M., Lee, J.H., 2004. Approximate dynamic programming strategies and their applicability for process control: a review and future directions. Int. J. Control Autom. Syst. 2, 263–278.

Lee, J.M., Lee, J.H., 2005. Approximate dynamic programming-based approaches for input–output data-driven control of nonlinear processes. Automatica 41 (7), 1281–1288.

Lee, J.H., Lee, J.M., 2006. Approximate dynamic programming based approach to process control and scheduling. Comp. Chem. Eng. 30 (10), 1603–1618.

Lee, J.M., Lee, J.H., 2009. An approximate dynamic programming based approach to dual adaptive control. J. Process Control 19 (5), 859–864.

Lee, J.H., Wong, W., 2010. Approximate dynamic programming approach for process control. J. Process Control 20 (9), 1038–1048.

Lee, J.M., Kaisare, N.S., Lee, J.H., 2006. Choice of approximator and design of penalty function for an approximate dynamic programming based control approach. J. Process Control 16 (2), 135–156.

Lee, H., Grosse, R., Ranganath, R., Ng, A.Y., 2009. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. Paper Presented at the Proceedings of the 26th Annual International Conference on Machine Learning.

Lewis, F.L., Vrabie, D., 2009. Reinforcement learning and adaptive dynamic programming for feedback control. IEEE Circuits Syst. Mag. 9 (3).

Li, Y. (2017). Deep reinforcement learning: An overview. arXiv preprint arXiv:1701.07274.

Littman, M.L., 1994. Markov games as a framework for multi-agent reinforcement learning. Paper Presented at the Proceedings of the Eleventh International Conference on Machine Learning.

Maravelias, C.T., Sung, C., 2009. Integration of production planning and scheduling: overview, challenges and opportunities. Comput. Chem. Eng. 33 (12), 1919–1930.

Martin, J.J., 1967. Bayesian Decision Problems and Markov Chains. Wiley.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., . . . Ostrovski, G., 2015. Human-level control through deep reinforcement learning. Nature 518 (7540), 529–533.
Naidu, S.R., Zafiriou, E., McAvoy, T.J., 1990. Use of neural networks for sensor failure detection in a control system. IEEE Control Systems Magazine 10 (3), 49–55.
Negoescu, D.M., Frazier, P.I., Powell, W.B., 2011. The knowledge-gradient algorithm for sequencing experiments in drug discovery. Informs J. Comput. 23 (3), 346–363.
Poupart, P., Vlassis, N., Hoey, J., Regan, K., 2006. An analytic solution to discrete Bayesian reinforcement learning. Paper Presented at the Proceedings of the 23rd International Conference on Machine Learning.
Powell, W.B., Ryzhov, I.O., 2012. Optimal Learning, vol. 841. John Wiley & Sons.
Powell, W.B., 2007. Approximate Dynamic Programming: Solving the Curses of Dimensionality, vol. 703. John Wiley & Sons.
Prokhorov, D.V., Wunsch, D.C., 1997. Adaptive critic designs. IEEE Trans. Neural Netw. 8 (5), 997–1007.
Puterman, M.L., 2014. Markov Decision Processes: Discrete Stochastic Dynamic Programming. John Wiley & Sons.
Ryzhov, I.O., Powell, W.B., 2010. Approximate dynamic programming with correlated Bayesian beliefs. Paper Presented at the Communication Control, and Computing (Allerton), 2010 48th Annual Allerton Conference.
Ryzhov, I.O., Powell, W.B., 2011. Information collection on a graph. Oper. Res. 59 (1), 188–201.
Ryzhov, I.O., Powell, W.B., Frazier, P.I., 2012. The knowledge gradient algorithm for a general class of online learning problems. Oper. Res. 60 (1), 180–195.
Saint-Donat, J., Bhat, N., McAvoy, T.J., 1991. Neural net based model predictive control. Int. J. Control 54 (6), 1453–1468.
Schölkopf, B., Smola, A.J., 2002. Learning with Kernels: Support Vector Machines, Regularization, Optimization and Beyond. MIT press.
Schaul, T., Quan, J., Antonoglou, I., & Silver, D. (2015). Prioritized experience replay. arXiv preprint arXiv:1511.05952.
Shin, J., Lee, J.H., 2016. Multi-time scale procurement planning considering multiple suppliers and uncertainty in supply and demand. Comp. Chem. Eng. 91, 114–126.
Shin, J., Lee, J.H., Realff, M.J., 2017. Operational planning and optimal sizing of microgrid considering multi-scale wind uncertainty. Appl. Energ. 195, 616–633.
Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., . . . Lanctot, M., 2016. Mastering the game of Go with deep neural networks and tree search. Nature 529 (7587), 484–489.
Strens, M., 2000. A Bayesian framework for reinforcement learning. Paper Presented at the ICML.
Stulp, F., Sigaud, O., 2015. Many regression algorithms: one unified model: a review. Neural Netw. 69, 60–79.
Sutton, R.S., Barto, A.G., 1998. Reinforcement Learning: An Introduction, vol. 1. MIT press, Cambridge.
Tesauro, G., 1992. Practical issues in temporal difference learning. Mach. Learn. 8 (3–4), 257–277.
Tsoi, A.C., Back, A., 1997. Discrete time recurrent neural network architectures: a unifying review. Neurocomputing 15 (3), 183–223.
Van Hasselt, H., Guez, A., Silver, D., 2016. Deep reinforcement learning with double Q-Learning. Paper Presented at the AAAI.
Walsh, T.J., Goschin, S., Littman, M.L., 2010. Integrating sample-Based planning and model-Based reinforcement learning. Paper Presented at the AAAI.
Wang, Z., Schaul, T., Hessel, M., van Hasselt, H., Lanctot, M., & de Freitas, N. (2015). Dueling network architectures for deep reinforcement learning. arXiv preprint arXiv:1511.06581.
Watkins, C.J.C.H., 1989. Learning from Delayed Rewards. University of Cambridge England.
Werbos, P.J., 1977. Advanced forecasting methods for global crisis warning and models of intelligence. Gen. Syst. Yearbook 22 (12), 25–38.
Werbos, P.J., 1990. A menu of designs for reinforcement learning over time. In: Neural Networks for Control., pp. 67–95.
Werbos, P.J., 1992. Approximate dynamic programming for real-time control and neural modeling. In: Handbook of Intelligent Control.
Werbos, P.J., 2012. Reinforcement learning and approximate dynamic programming (RLADP)—Foundations, common misconceptions, and the challenges ahead. In: Reinforcement Learning and Approximate Dynamic Programming for Feedback Control., pp. 1–30.
Ydstie, B., 1990. Forecasting and control using adaptive connectionist networks. Comput. Chem. Eng. 14 (4), 583–599.
Zhang, H., Wang, Z., Liu, D., 2014. A comprehensive review of stability analysis of continuous-time recurrent neural networks. IEEE Trans. Neural Networks Learn. Syst. 25 (7), 1229–1262.