SN Computer Science (2021) 2:103
https://doi.org/10.1007/s42979-021-00488-w

ORIGINAL RESEARCH

Using Reinforcement Learning for Generating Polynomial Models to Explain Complex Data

Niclas Ståhl[1] · **Gunnar Mathiason**[1] · **Dellainey Alcacoas**[1]

Received: 31 August 2020 / Accepted: 25 January 2021 / Published online: 19 February 2021

Abstract
Basic oxygen steel making is a complex chemical and physical industrial process that reduces a mix of pig iron and recycled scrap into low-carbon steel. Good understanding of the process and the ability to predict how it will evolve requires long operator experience, but this can be augmented with process target prediction systems. Such systems may use machine learning to learn a model of the process based on a long process history, and have an advantage in that they can make use of vastly more process parameters than operators can comprehend. While it has become less of a challenge to build such prediction systems using machine learning algorithms, actual production implementations are rare. The hidden reasoning of complex prediction model and lack of transparency prevents operator trust, even for models that show high accuracy predictions. To express model behaviour and thereby increasing transparency we develop a reinforcement learning (RL) based agent approach, which task is to generate short polynomials that can explain the model of the process from what it has learnt from process data. The RL agent is rewarded on how well it generates polynomials that can predict the process from a smaller subset of the process parameters. Agent training is done with the REINFORCE algorithm, which enables the sampling of multiple concurrently plausible polynomials. Having multiple polynomials, process developers can evaluate several alternative and plausible explanations, as observed in the historic process data. The presented approach gives both a trained generative model and a set of polynomials that can explain the process. The performance of the polynomials is as good as or better than more complex and less interpretable models. Further, the relative simplicity of the resulting polynomials allows good generalisation to fit new instances of data. The best of the resulting polynomials in our evaluation achieves a better R^2 score on the test set in comparison to the other machine learning models evaluated.

Keywords Reinforcement learning · Polynomial generation · Generalisation in machine learning · Steel making

Introduction

The basic oxygen furnace (BOF) process has multiple complex highly dynamic and non-linear interactions of chemical and physical reactions, of which all are not fully known. The purpose of the process is to reduce the carbon content by oxidisation which releases carbon and other chemical elements out of the melted material (the "heat"), which consists mainly of pig iron ("hot metal") and scrap steel. The end goal of this process is to reach a level lower than a certain required threshold for some elements, such as carbon and phosphorous, while also reaching a sufficiently high temperature.

The intense heat and the heavy reactions quickly destroy any sensor, so continuous direct in-process monitoring of the heat during the process is difficult. With the lack of rich continuous data, operators need to rely on a few available data points, their experience, and rough heuristic for their control of the process. Some heat data is known in advance of the process start, such as the actual weights and temperatures of the scrap and the hot metal to be used, in total some 50 parameters. Using this data in combination with visual observations and a few other indirect data points, operators tune the process during the process. This is done by inserting additional key materials, or adjusting the lance height and thereby changing the spraying area on the heat, resulting in energy and mass balance adjustments.

✉ Niclas Ståhl
niclas.stahl@his.se

[1] School of Informatics, University of Skövde, Box 408, 541 28 Skövde, Sweden

The existing heat management support system that is in use proposes a few key process parameter settings and a temperature prediction, which is based on the actual amounts of hot metal and scrap delivered for each heat, together with some other similar data points. These predictions are based on partly unknown hard-coded rules that give only rough estimates and a simplistic temperature prediction, so operators can use this information only as advice. A successful operator needs both explicit and tacit knowledge, based on experience gathered over a long time. It does, therefore, exist a great need for both better prediction systems as well as transparent models that can be understood and trusted by operators [14]. These two goals are often orthogonal to each other and while complex machine learning (ML) approaches can achieve the former they often lack transparency. In this work, we therefore aim to build complex ML models, while still keeping their decision process interpretable and transparent. Our approach base predictions on polynomials that express the connection between proposed process settings and the resulting target prediction. Operators and process developers can examine these polynomials, hence the impact of each parameter, and both judge about the accuracy of the prediction and also understand what process parameter settings cause a certain process outcome.

Using an ML approach allows for many more heat parameters to be used for target predictions, such as we have done in our earlier work [1]. Utilising the information from such high dimensionality data and for better prediction cannot easily be done otherwise. Without it, predictions are typically based on either hard coded rules or by human raw data analysis that uses known calculations based on few features. For many ML approaches all available features can be used for model training, such that there is no need to exclude any available data that possibly could contribute to a good prediction. From the available process data, we let the algorithm reduce the dimensionality and, by itself, pick the features that are important for good predictions. Moreover, our approach generates polynomials and uses these as prediction models. The advantage of generating polynomials and continuously evaluating their performance is that important features get discovered in the process. This is in contrast to depicting the important features through evaluation of an already trained model. Such continuous refinement of polynomials during training allow for multiple solutions to emerge, and thus multiple likely explanations that can be examined by operators and process experts.

Using the presented generative model, we manage to generate several polynomials that achieve a better R^2-score (0.72) on a test set than all other evaluated machine learning models, and are almost as good as the best models when it comes to predicting instances within a correct margin. The best of the generated polynomials predict 85% of all samples within 15 °C from the measured temperature, which is the acceptable range for when a prediction will be useful in real world production. This is just a few percentage units lower than what is achieved with more complicated models, such as SVMs and ANNs. Furthermore, the presented model can give several alternative explanations to the process, as well as recognising important features that are always a part of the polynomials.

Related Works

There is a body of literature on how to predict BOF process targets. The data that are used in these studies are, however, in most cases, kept secret, which make it difficult to replicate the presented experiments. Furthermore, the amount of data used in the conducted studies varies as well as the stability of the production processes in the different production sites where the data are collected. These factors make it problematic to compare the results from previously presented works, on more than a conceptual level.

One example where ML is used to predict the end temperature is Gao et al. [4] that proposes an improved version of a support vector machine regression using a weighted matrix and coefficient vector, optimised by wavelet transform to estimate the carbon content and the end temperature end-point. Other approaches use novel and durable sensors for better in-process estimate the actual process state, such as Xu et al. [20] that uses the data from a spectrum distribution of the flame at the mouth of the BOF vessel to better estimate the current heat state, using a support vector machine algorithm. This idea was further developed by Shao et al. [16] using a combination of support vector classifier and a support vector regression for process state estimation using flame radiation at the mouth of the BOF vessel. Using additional sensors such as these improve prediction accuracy, and may also extend process understanding, but still lacks the transparency to explain influential factors of the collected data. In our previous research, we compared the prediction accuracy between several conventional ML algorithms, trained on a full-variance data set that includes all types of heats from years of production [1]. We found that many earlier publications use limited data sets with a lower variance in the training examples and the target outcomes. In our work, a subset of the conventional ML algorithms were able to capture the full variance and prediction accuracy relatively well, in particular when raw data is augmented with expert-based informed features based on that data.

Most of the existing literature support that ML models can give useful predictions for BOF processes. However, the selection of ML model has a significant impact on the performance. Support vector machines (SVM) and artificial neural networks (ANN) perform well, better than many other conventional algorithms, since they are good

at capturing the complexity of the process. Some conventional ML algorithms are relatively good at explaining model predictions, such as decision trees, *k*-nearest neighbours. These have been used for temperature prediction in BOF-processes, but with a less satisfactory result [1]. Further, machine learning models that are often considered as easy to interpret, such as linear regression and decision trees, may still be impossible for humans to interpret when the data is high dimensional [18].

Machine learning algorithms can take advantage of data that has many dimensions, while most humans can only comprehend and distinguish patterns among a few dimensions. Thus, for a human analysis it is necessary to aggregate model prediction explanations into a few comprehensible dimensions. Preferably, a good model explanation presents only a few process parameters as being influential factors of the target prediction, with the interrelations between them explained as well. This explanation should further closely resemble how the trained model behaves. Recent approaches to explaining ML models by attribution of features to predictions include SHAP and LIME [9, 15] which examine already trained models. Their advantage is that they are model-agnostic and can explain the influencing variables for any ML model. However, since complex ML models learn high-dimensional connections between the input variables and the target variables, the approaches that visualise these must also be able to visualise high-dimensional connection between variables, something which is a challenging problem by itself. Hence, these high-dimensional patterns must be transferred from a high-dimensional space into human-interpretable visualisations. A good understanding of ML and visualisation techniques for high dimensional data is therefore needed. This is something that typically industrial operators and process experts lack. Rather, they analyse data through traditional calculations typically based on known properties of the process. Even extensive calculations are often still simplifications of the actual complexity, consisting only of just a few terms.

In summary, expressive algorithms cannot explain their predictions well, and algorithms with explanation abilities are not expressive enough to capture the complexity for good predictions.

Methods

This section describes how the generative model is implemented. First a short description of the data is given. This is followed by a description of the architecture of the deep neural network that acts as an agent in the RL setup.

Exploration to Understand the Basic Oxygen Furnace Process

Due to the complexity of the BOF process, a fully detailed model or simulation of the process is infeasible. In spite of many efforts to model this complexity, there is still much uncertainty around how input parameters determine the result. Our approach to find this relation is based on the exploration of the possible combinations, using an approach that simulates informed attempts to traverse the solution space of possible combinations. Knowledge development for process analysis has traditionally been done by experts by trying out combinations of experiments that has a potential of yielding explanations for process behaviours. We mimic this approach by letting an agent generate possible steps towards promising combinations, guided towards reasonable explanations. The agent-based algorithm automatically generates polynomials step-wise that express how input parameters relate to process outcomes. However, the generated polynomials are not necessarily anchored in physics or known process calculations, but should rather be considered a consequence of what the used data represents. Therefore, polynomials should be used as an indicator to both as whether useful data has been collected, and as suggestions for hypothesis of how the process can be explained. Hence, this allows process experts and engineers to continue to work exploratory with process analysis, but now with the support of models that are able to quantify the impact of much more information than those grounded in the physics of the process alone.

Production Data

The data used in this study consists of real production data from the BOF process, run by SSAB in Sweden. The data is collected between 2014 and 2017 and consist of 9710 heats, where 112 different features are recorded, with one value for each heat. The original data contains around 17,000 heats of data for the entire time period, with corrupt or missing values. The 112 features consist of both initial heat data, sensor measurements of the initial heat state, calculated descriptors, and also aggregated statistics of sequential data available during the process. The data is split into a training set of 70%, a 15% validation set for model selection, and a 15% hold-out test set for model evaluation.

Recurrent Neural Networks

To model the agent that generates polynomials and acts in the RL framework presented in Sect. 3.5, a recurrent neural network (RNN) is used. A recurrent neural network is a special case of an artificial neural network (ANN) where there exist cyclic connections between neurons [12]. This allows

the network to keep an inner state, allowing it to make use of information that spans several steps of a sequence. In our approach, we make use of a special type of RNN, specifically a long short-term memory (LSTM) network. Compared to general RNNs, LSTMs use special cell that has a variable that holds the current state of the network. This cell enables the network to have a memory of past steps in the sequence and hence, the information stored in memory can be used in how to act on new data.

A trained LSTM network can not only be used to analyse (for instance to classify) a sequence of data, but it can also be used to generate new sequences based on the trained knowledge. Such sequences follow the same distribution as the training sequences used. Several examples of how to generate text using this approach are given in Graves [5]. The sequence is first initialised by the the insertion of a special start vector at the first step in the sequence. This vector is then propagated trough the network, giving the conditional probability over all possible symbols. A symbol is then sampled from this distribution and appended to the sequence. The sampled symbol is also transformed into an one-hot encoded vector which is propagated trough the network, resulting in that a new symbol is sampled. The process is repeated, adding another symbol at a time to the end of the sequence, until a special stop symbol is sampled or when the length of the sequence reaches a predefined limit. A visual description of this process is shown in Fig. 1.

Polynomial Generation from Recurrent Networks

In this section we describe how generated polynomials are encoded in a way that they can be generated by an RNN. It is described how the goodness of these polynomials are calculated, so that an RL-model can learn how to generate good polynomials. We continue with how generated polynomials are evaluated, and how this result compares to alternative models.

Polynomials are constructed by the algorithm to make use of a specific subset of all the features in the original dataset, with the intention to reduce the complexity of the solution. There is a balance needed between complexity and expressiveness of the polynomial, and the interpretability. This reduction of features can be conducted using a multitude of different search methods [8]. In our work we use an RL-approach, and similar approaches have been used by others, such as Fard et al. [3] and Piñol et al. [13]. While these publications target the extraction of a subset of features for any model, we try to avoid generating too many terms by limiting the terms of our polynomials.

To generate polynomials to describe the process, we instead use a generative RL-based approach, where each symbol in a polynomial is generated one at a time. The RL technique has proven useful in many different applications. For instance, generative RL-approaches have been used for improving hyper-parameter choices for neural network configurations [2, 6, 22]. In Khurana et al. [7] RL is used to learn transformations that can be applied to ML input data to get a lower error rate, both for classification and regression problems. Our generative approach also draws inspiration from using similar generative models to find very complex molecule structures that have certain desired properties, such as when generating molecules for drug discovery [11]. However, to the best of our knowledge, there is no similar work with the focus on generating simple and interpretable machine learning models.

Encoding Polynomials

In this paper, we consider polynomials as a sequence of coefficients and variables, separated by either addition or multiplication. These polynomials are encoded as one-hot representation. However, in the generation process, we only consider the placement of variables and addition, while the placement of the coefficients and multiplication is implicit. Thus, as shown in Fig. 2, there is only need for tokens representing each variable and the addition. Beside these tokens, there will also be a special token representing the start of the polynomial and one representing the end. Hence, there will be 114 different tokens that the agent can select from when generating polynomials, 112 of them representing the different variables and one representing addition and a special token representing the end of a polynomial. In addition,

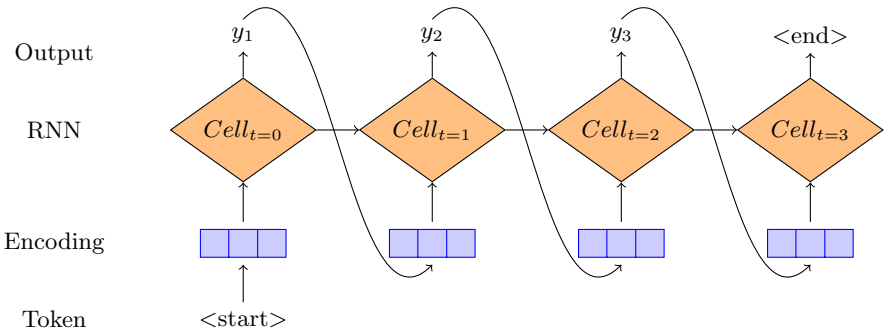

Fig. 1 Using a RNN to generate new sequences. At every time step t, a new output token y_t is sampled from the distribution that is proposed by the network. This token is then used as the input in the following step. The generation ends when the special end token is sampled or when the generation reaches the maximum number of allowed steps

	$x1$	$+$	$x1$	$x2$	$+$	$x3$	$+$	$x4$
x_1	1	0	1	0	0	0	0	0
x_2	0	0	0	1	0	0	0	0
x_3	0	0	0	0	0	1	0	0
x_4	0	0	0	0	0	0	0	1
$+$	0	1	0	0	1	0	1	0

Fig. 2 A simplified depiction of the one-hot representation derived from the polynomial $x_1 + x_1x_2 + x_3 + x_4$. A reduced vocabulary of variables is shown in this example, and in the real implementation there are 115 different tokens, 112 variables and 3 special characters, that can be used in the formation of polynomials

there is, also, a special start token, which the agent cannot select, adding up to 115 different tokens in total.

Learning Generic Polynomials

To prevent the agent from exploring a tremendous amount of states before it figures out how well formed polynomials look we will give the agent a warm start by training it to generate polynomials in a supervised way. To this end, 100,000 polynomials are generated by a simple set of rules and the network is trained in a supervised way so that the likelihood of generating these polynomials is maximised. The polynomials that are used in this pre-training are generated in a step wise manner. The first step is to randomly select a variable. The polynomial is then constructed by repeatedly and probabilistically selecting one of the four possible continuations until the maximum length is reached or the stop symbol is sampled. In the first out of four possible cases, the last variable is repeated. In the second case, the variable is followed by another randomly selected variable. In the third case, the variable is followed by an addition and a random variable after the addition. In the last case, a stop symbol is added and the generative process is terminated. The probabilities for each of these four cases are arbitrary selected as; 0.2, 0.1, 0.65 and, 0.05 respectively. During the generative process, the random selection of variables is uniform and thus, all variables are equally likely.

Evaluation of Polynomials

The aim of the presented research is to apply RL in order to generate as good polynomials as possible. A good polynomial is considered to be a polynomial that explains the process sufficiently well and should be given a high reward to the RL agent. Since the process is considered to be rather stochastic and noisy, it would be infeasible to perfectly predict the outcome in all the cases. Instead, the aim is to get as many temperature predictions as possible within a given acceptable range from the measured temperature. Hence, the goodness of a polynomial p is scored by;

$$S(p \mid X, Y) = \frac{1}{m} \sum_{i}^{m} 1 - H\big(|p(x_i) - y_i| - \delta\big), \tag{1}$$

for a given set of data (X, Y), where m is the number of samples in the dataset. In Eq. (1), $H(x)$ is the unit step function, which is 1 if $x > 0$ and 0 otherwise. Furthermore, δ is the maximum temperature (°C) that would be considered to be an acceptable divergence from the measured value, in order for the prediction to be useful in a real-world setup. Hence, the usefulness of the model, is measured as the percentage of predictions within $\pm\delta$ °C of the measured outcome.

The reason behind the selection of this scoring function is that errors within this, pre-defined, margin can easily be corrected and the heat can still be delivered to the following production step without any delay. Larger errors, on the other hand, would require cumbersome and costly efforts to restore. Hence, this particular evaluation metric is defined to evaluate the usefulness of the model in a real production line. In the conducted experiments, δ is defined as 15 °C.

The score, given in Eq. (1) is normalised, to prevent a quick convergence to sub-optimal configurations of the agent during the training process, leading to the following, final, scoring function:

$$S(p \mid X, Y, \nu) = S(p \mid X, Y) - \nu, \tag{2}$$

where ν is the expected reward from a randomly sampled polynomial generated by the process described in the previous section.

However, before a polynomial can be evaluated, the optimal coefficients for that polynomial must be inferred. This is done by minimising the mean squared error between the predicted outcome of the polynomial and all recorded observations in the training data;

$$\text{MSE}(X, Y) = \frac{1}{m} \sum_{i}^{m} (p(x_i) - y_i)^2. \tag{3}$$

Hence, the coefficients of the polynomials are adjusted to minimise Eq. (3), given the training dataset. All redundant terms, terms that occur multiple times in the polynomial, are removed in the step, to make the training of the model more stable. The inferred configuration of coefficients is then evaluated, using the scoring function given in Eq. (2), on a separate validation set.

Training with Reinforcement Learning

Consider that the generative network, described in Sect. 3.3, is an acting agent in an environment. The whole environment that this agent is acting in is shown in Fig. 3 and as in most RL approaches, we frame the targeted problem as a Markov decision processes. This means that the current state contains all information that the agent may

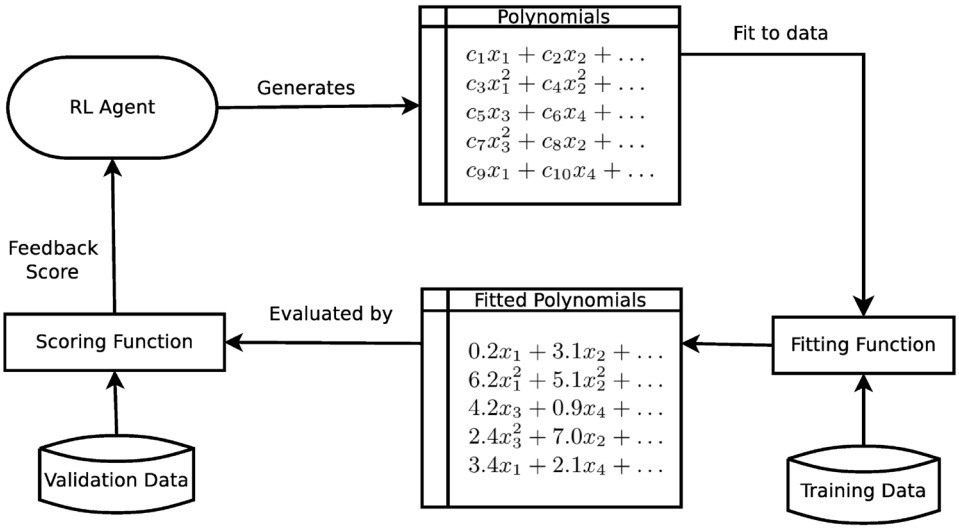

Fig. 3 Framing of the presented RL scenario. Here an agent generates polynomials that are fitted to the training data. The fitted polynomials are then applied to the validation set, and the goodness of fit for this set is used as the reward, which are fed back to the agent

require to decide which action to take. Hence, at each step the agent is in a specific state $s \in \mathbb{S}$ and takes an appropriate action $a \in \mathbb{A}(s)$, which is only based on the current state. Here $\mathbb{S}$ is the set of all possible states and $\mathbb{A}(s)$ is the set of all possible actions that the agent may take when it is in state s. To determine which action a the agent should take in a given state s, the agent follows a policy $\pi(a|s)$ which maps a state to the probability of each action. The objective of the agent is to find an optimal policy, such that the expected total reward is maximised. To this end, we denote the weights of the neural network, which controls the behaviour of the agent, θ. For any configuration of θ, the expected total score can be denoted as:

$$J(\theta) = \sum_{s \in \mathbb{S}} d^{\pi_\theta}(s) \sum_{a \in \mathbb{A}} Q^{\pi}(a, s) \pi_\theta(a|s) \tag{4}$$

where d^{π_θ} is the stationary distribution of states in the Markov chain, for an agent with the policy π_θ. Thus, $\pi_\theta(a|s)$ is the probability of taking action a when in state s. $Q(a, s)$ is the expected reward for taking the action a when in the state s. The aim of the learning process is to find an optimal configuration of θ that would maximise the score in Eq. (4). In the presented work, we use the REINFORCE algorithm [19] to find an optimal configuration θ^*, following the same approach as, for example, Yu et al. [21] and Silver et al. [17]. In the proposed approach, the starting state is always the same and the acting agent gets a reward when the generative process terminates. For such a process, the expected total score, using a given policy, can be estimated by several unbiased traces from the process while following that policy. The expected total reward for a policy, parameterised by θ, is be estimated by:

$$J(\theta) = \sum_{\tau \in \mathsf{T}} R(\tau) \prod_{t=0}^{T} \pi_\theta(a_t|s_t). \tag{5}$$

Here $\pi_\theta(a_t|s_t)$ is the probability of taking action a_t, which was the action taken in the trace at time t, in the state s_t and $R(\tau)$ is the total accumulated reward for the whole trace. In the polynomial generation process, which is described earlier, the reward is only given in the final step of the generative process and hence, the total reward, $R(\tau)$, corresponds to the function given in Eq. (2). The aim of the REINFORCE algorithm would, hence, be to maximise the expression in Eq. (6) by gradient ascent. Thus, the expected total score is increased by changing the parameterisation of the policy using the gradient of Eq. (6), which is:

$$\nabla_\theta J(\theta) = \sum_{\tau \in \mathsf{T}} R(\tau) \sum_{t=0}^{T} \log \pi_\theta(a_t|s_t). \tag{6}$$

This process would raise the probability for polynomials with a high reward and decrease the probability for those with a low reward.

Experiment Parameters

The LSTM network, as described in Sect. 3.3, is used to model the action policy for selecting the next term of the polynomial. The trained network predicts the probabilities attributed to each of the possible next terms that are chosen. The first layer of the trained network is a fully connected hidden layer with 512 neurons that use the leaky relu function as the activation function. The second layer consists of 128 LSTM cells and the final layer consists of 114 neurons,

which is the same as the number of possible actions. This layer has the softmax function as an activation function and the activation can hence be considered as an approximation of probabilities. The weights of this network are adjusted, during the training phase, using stochastic gradient descent, with a learning rate of 0.01. In the proposed experiment, the agent is trained for 5000 epochs where each epoch consists of 4096 independently sampled traces.

Experiment Evaluation

We compare the expressiveness and accuracy of the generated polynomials on how to model the BOF process, with alternative ML approaches. Among the several ML methods compared to, artificial neural network (ANN) and a support vector machine (SVM) have previously been shown to be the top performing models for endpoint temperature prediction in the BOF process [1]. We also compare polynomials to machine learning methods that are known to provide some interpretability about their models. These are a random forest (RF), decision trees (DT), and k nearest neighbour (KNN).

More specifically, the algorithms that we use for comparison are: an ANN with two hidden layers consisting of 128 and 64 neurons respectively; a SVM with a radial basis kernel; a random forest with 100 trees; a decision tree with a maximal depth of 10; and a KNN where k is equal to 5. These models make use of all available parameters in data and some of them are therefore expected to outperform the generated polynomials, when it comes to prediction accuracy, since the generated polynomials make use of only a subset of all input parameters. However, these models, with the exception of the decision tree, are known to be difficult to interpret, especially when applied to many variables.

Results

Generated Polynomials

The agent succeeds in learning how to generate polynomials that are as useful as more complex and less interpretable models are for the steel operators. The increase in quality of the generated polynomials during the learning process is depicted in Fig. 4. In this figure, it is shown that the agent manages to generate increasingly better polynomials over time. The spread in performance of the generated polynomials decreases over time and, in the last epochs, almost all generated polynomials have a high predictive power. The spread in performance among the polynomials shows one of the benefits of the presented approach, namely, that this approach creates several possible polynomials, which all have the possibility to predict the outcome of the process.

When it comes to the usefulness of the model, it is measured as the percentage of predictions within ±15 °C of the real measured outcome, see Table 1 for this score for all evaluated models. Here, small errors can be ignored, while large errors would have a negative impact on the score. Therefore, we would prefer a model with many small prediction errors over one with few, but large, prediction errors. As seen in Fig. 5, this is what the best polynomial, which is discovered by the RL agent, achieves. This behaviour is

Table 1 Performance on the leave-one-out test set for the evaluated algorithms and the simple heuristic of always guessing the mean value

% of predictions within ±15 ℃							
ANN	SVM	RF	DT	KNN	Mean guess	Best polynomial	Average polynomial
86.8%	86.6%	82.1%	67.2%	72.2%	58.3%	84.9%	83.6%

The performance is measured as the percentage of predictions within ±15 °C. The rightmost column is the average performance of all generated polynomials in the last epoch. The best polynomial is decided to be the polynomial which had the best performance on the internal validation set during all epochs

Table 2 R^2 score on the training- and the leave-one-out test set for the evaluated algorithms

R^2-score on training set							
ANN	SVM	RF	DT	KNN	Mean guess	Best polynomial	Average polynomial
0.89	0.84	0.90	0.55	0.72	0.0	0.73	0.73
R^2-score on test set							
ANN	SVM	RF	DT	KNN	Mean guess	Best polynomial	Average Polynomial
0.72	0.70	0.49	0.44	0.15	0.0	0.74	0.73

Here, the great generalisation capability of generated polynomials, which, on average, score almost the same on the test and the training set

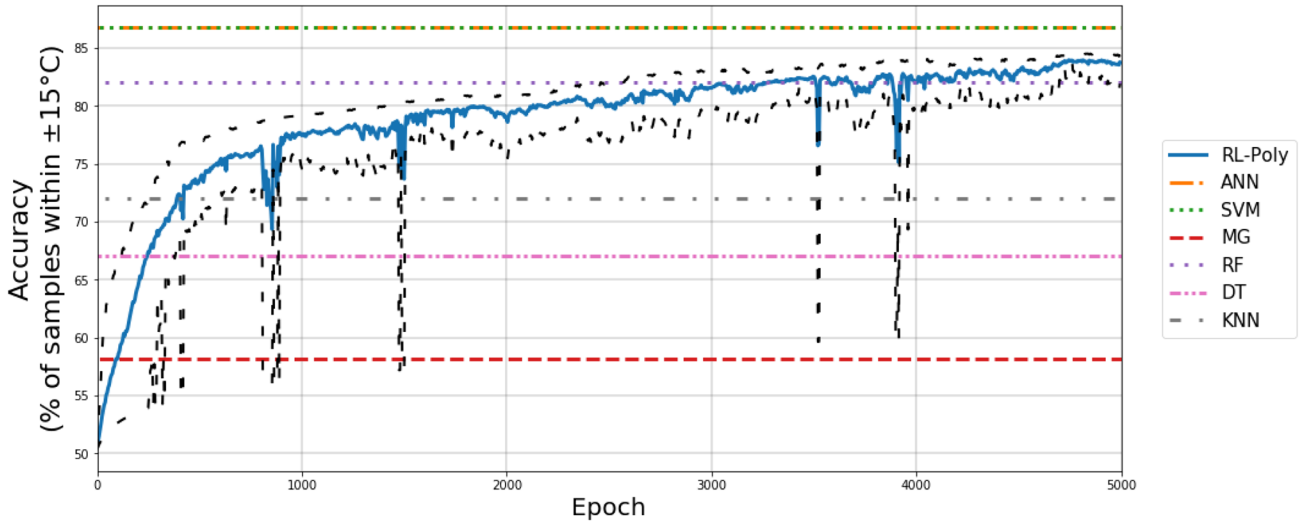

Fig. 4 The evolution of the generated polynomials during the training process. The average performance is shown by a blue solid line, surrounded by two dashed black lines representing the 10th and the 90th percentile. The result achieved by the heuristic approach of guessing the mean is shown by a red dashed line (straight line at the bottom), as well as the result from the other evaluated machine learning models

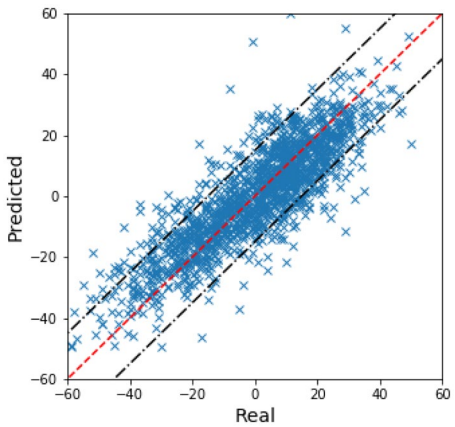

Fig. 5 Predictions for the test set made by the best polynomial that is generated over the real observed value, where both are shifted to a mean of 0. All predictions would land on the red diagonal line if the model manages to perfectly predict all instances. The polynomial model that is used for predictions has a R^2 score of 0.73

inherent to simplistic models, such as the generated polynomials, since they are not as prone to overfitting as more complex models and, therefore, most often generalise well. This strength is shown in Table 2 where R^2-scores for the training- and test sets are shown. The generated polynomials have the highest R^2-score for the test set while, by a great margin, also have the smallest difference between the score for the training set and the test set. Thus, generalises the best of all models.

During the last training epoch, the agent generates 121 unique polynomials out of 4096 trials. The most common polynomial is generated 948 times and there are 7 polynomials that the model generates more than 100 times. The polynomial with the best performance over the validation set is one of these and that particular configuration is generated 123 times. Only 2.1% of all the generated polynomials contain a second order term but none of these have a higher predictive performance on the validation data compared to the average score of the generated polynomials in the last epoch. Therefore, in this particular application, the capability of the agent to generate such terms are of lesser importance.

No full investigation of the interpretability of the generated polynomials is conducted within this work. However, the generated polynomials have been discussed with process experts. These experts had no problems to interpret what the polynomials did and deemed them to describe the process reasonably well. The ability, of the proposed method, to generate multiple polynomials is especially appreciated.

Discussion and Future Work

In recent years, reinforcement learning (RL) has brought major advances in application areas that previously were difficult to address. We argue that we apply RL in a novel way, where we let an agent build polynomial models for temperature prediction. The purpose of this paper is to demonstrate this application. The hyperparameters that are used in this study are, therefore, arbitrarily selected and most likely not optimal. We would, therefore, suggest that future studies investigate if there are other choices that would be more beneficial in regards to accuracy or time consumption. The

presented RL application is a method that to the best of our knowledge has not been used before on a similar problem, a complex steel making process with high dimensional data. RL-based model generation has a potential both in generation of more complex models such as with neural networks, for example, as in Bello et al. [2], Zoph et al. [22], Jomaa et al. [6], and also in generation of simpler models, such as our case.

The successful accuracy we get in the application case we use for this study, the BOF steel making process, can probably be explained with that for this case, the dependency between the input and the output is relatively smooth and locally stable and could therefore be well approximated by a polynomial. Furthermore, the proposed polynomials mostly contain first order terms and only a few contain second order terms. It would, therefore, be interesting to investigate if the proposed method could manage to learn more complex polynomials where the dependency between the input and the output is even more complex. Another avenue of future work would be to generalise the approach to other simple models which are even easier to interpret for operators, such as a decision tree or a rule based system. This would either require that the generative process is redefined or that such models can be formulated using a linear representation.

Our choice of using a policy gradient approach for training the RL-agent allows for sampling of many different models, all with a high predictive power, explaining the studied phenomenon. This is in contrast to a common Q-learning approach that strives for one optimal solution, such as the method used by Khurana et al. [7]. Sampling of multiple models makes it possible for system operators to extract polynomials that are well based in the physical description of the system, since polynomial can be assesses from their existing process knowledge. The extracted polynomial gives transparency and can be used to further understanding the process and how to improve process control in order to achieve process targets.

In this work, we constrict the polynomials to only have a few variables to improve interpretability for a process expert, who can choose the most useful alternative probabilistically plausible polynomials generated. Further, the relevance of these features in the data set can be identified by analysing the coefficients of each variable of a polynomial. The ability to explore and select the most suiting polynomial is a major improvement over using other commonly used variable selection- and regularisation techniques for polynomials, such as LASSO and elastic net [10]. The possibility to sample many different polynomials can also be used in an ensemble approach, where many polynomials are sampled and the average prediction of all polynomials is the final prediction.

One drawback of the presented approach is that the training time is much longer compared to the other evaluated models. However, the training of our model, as well as all other models, is done offline and the time it takes is therefore of lesser importance. For any trained model used in an industrial application, it is critical that the model execution is fast and predictable. For a model based on polynomials, the execution time is constant, since it depends only on setting the values of the free variables. The implementation of a model trained with our approach is, therefore, a matter of implementing the polynomials as a single function, which is typically much easier to implement in any industrial infrastructure or control system, than implementing a solution using a third party machine learning package. The implementation of function that calculates a polynomial is also easily done in any programming language, compared to ML solutions which may require specific programming languages and libraries. The presented method has, however, not been tested in production and there is a need to study the impact of the inclusion of a new predictive system.

We see that very few of the generated polynomials contain higher-order terms, and we believe that this is due to the risk of overfitting the trained model when such terms are used. The model becomes more true to the training data, but there is a risk of decreasing the general predictive power when a linear term is changed to an arbitrary second order term. This likely discourages the agent to generate second order terms, such that it after a few epochs removes them and chooses to search for more linear terms. However, when the model is applied to more complex problems, higher order terms are most likely needed and generated, so potentially our approach can perform as good as more complex alternative ML algorithms. It is therefore an interesting future research direction to find out how the generating agent can be refined to generate terms that even better captures higher complexity.

Conclusions

This paper presents a novel RL-based method to generate polynomials for the prediction of the end temperature in a BOF-process. The presented method performs feature selection from a large set of features and includes high-order terms of these features in the selection process. This is a major advantage compared to most other feature selection methods, such as LASSO, which would face an explosion in the dimensionality of the input space when higher order terms are added.

The polynomials that are generated by the presented method, have a greater R^2 score than the other evaluated models, for example, an ANN, an SVM, and random forest. When the performance is measured as the percentage of predictions, in the desired range of $\pm 15°C$, the generated polynomials have most of their predictions within this error

margin and outperform some other evaluated models such as, a random forest and a decision tree. However, for this task, the polynomials perform slightly worse than the two best performing models, an ANN and a SVM. At the same time they are far more interpretable for operators and process experts. In our application case, this model interpretability improvement is well worth the trade-off for slightly more predictions outside the error margin when the goal is to understand a manufacturing process with the aim to improve it. Another major benefit of the presented method is that it generates multiple solutions (multiple polynomials) to the problem. Hence, human experts could investigate and compare these solutions and find the ones that align with their prior knowledge and understanding of the process.

While the presented model is applied in the domain of steel making, there is nothing that prevents it from being used for any other problem where a solution can be represented by polynomials. Thus, it should be relatively easy to generalise the method to be applicable in other domains and problems. One of the greater advantages of the presented approach is that the scoring function can be replaced with any arbitrary function for the evaluation of the polynomials. Hence, the presented method can be applied to other problems where there is a way to score the goodness of the polynomials.

Acknowledgements We would like to thank Juhee Bae and Yurong Li at the University of Skövde for the help with cleaning the data and for valuable discussions. We would also like to thank Niklas Kojola, Carl Ellström, Patrik Wikström, and Lennart Gustavsson at SSAB for the close collaboration in the Swedish Metal project under which this research is funded. The project is funded by the Knowledge Foundation in Sweden, under the Grant 20170297.

Funding Open Access funding provided by University of Skövde.

Compliance with Ethical Standards

Conflict of interest The authors of this article state that there are no conflicts of interest.

References

1. Bae J, Li Y, Ståhl N, Mathiason G, Kojola N. Using machine learning for robust target prediction in a Basic Oxygen Furnace system. Metall Mater Trans B. 2020;51.
2. Bello I, Zoph B, Vasudevan V, Le QV. Neural optimizer search with reinforcement learning. In: Proceedings of the 34th international conference on machine learning, vol 70. 2017. p. 459–468.
3. Fard SMH, Hamzeh A, Hashemi S. Using reinforcement learning to find an optimal set of features. Comput Math Appl. 2013;66(10):1892–904.
4. Gao C, Shen M, Wang L. End-point prediction of BOF steelmaking based on wavelet transform based weighted TSVR. In: 2018 37th Chinese control conference (CCC). IEEE. 2018. p. 3200–3204.
5. Graves A. Generating sequences with recurrent neural networks. 2013. arXiv:1308.0850.
6. Jomaa HS, Grabocka J, Schmidt-Thieme L. Hyp-rl: hyperparameter optimization by reinforcement learning. 2019. arXiv :1906.11527.
7. Khurana U, Samulowitz H, Turaga D. Feature engineering for predictive modeling using reinforcement learning. In: Thirty-second AAAI conference on artificial intelligence. 2018.
8. Kumar V, Minz S. Feature selection: a literature review. SmartCR. 2014;4(3):211–29.
9. Lundberg SM, Lee SI. A unified approach to interpreting model predictions. In: Advances in neural information processing systems. 2017. p. 4765–4774.
10. Ogutu JO, Schulz-Streeck T, Piepho HP. Genomic selection using regularized linear regression models: ridge regression, lasso, elastic net and their extensions. In: BMC proceedings. vol. 6. Springer; 2012. p. S10.
11. Olivecrona M, Blaschke T, Engkvist O, Chen H. Molecular de-novo design through deep reinforcement learning. J Cheminform. 2017;9(1):48.
12. Pineda FJ. Generalization of back-propagation to recurrent neural networks. Phys Rev Lett. 1987;59(19):2229.
13. Piñol M, Sappa AD, López A, Toledo R. Feature selection based on reinforcement learning for object recognition. In: Adaptive learning agent workshop. 2012. p. 4–8.
14. Rehse JR, Mehdiyev N, Fettke P. Towards explainable process predictions for industry 4.0 in the dfki-smart-lego-factory. KI-Künstliche Intelligenz 2019;33(2):181–187.
15. Ribeiro MT, Singh S, Guestrin C. "Why should i trust you?" explaining the predictions of any classifier. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining. 2016. p. 1135–1144.
16. Shao Y, Zhou M, Chen Y, Zhao Q, Zhao S. BOF endpoint prediction based on the flame radiation by hybrid SVC and SVR modeling. Optik. 2014;125(11):2491–6.
17. Silver D, Huang A, Maddison CJ, Guez A, Sifre L, Van Den Driessche G, Schrittwieser J, Antonoglou I, Panneershelvam V, Lanctot M, et al. Mastering the game of go with deep neural networks and tree search. Nature 2016;529(7587):484.
18. Stimson JA, Carmines EG, Zeller RA. Interpreting polynomial regression. Sociol Methods Res. 1978;6(4):515–24.
19. Williams RJ. Simple statistical gradient-following algorithms for connectionist reinforcement learning. Mach Learn. 1992;8(3–4):229–56.
20. Xu L, Li W, Zhang M, Xu S, Li J. A model of basic oxygen furnace (BOF) end-point prediction based on spectrum information of the furnace flame with support vector machine (SVM). Optik. 2011;122(7):594–98.

21. Yu L, Zhang W, Wang J, Yu Y. Seqgan: sequence generative adversarial nets with policy gradient. In: Thirty-first AAAI conference on artificial intelligence. 2017.
22. Zoph B, Vasudevan V, Shlens J, Le QV. Learning transferable architectures for scalable image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. 2018. p. 8697–8710.