May 2021

Fixed-policy value-function prediction for stable control applications using machine learning

Pablo Barrera González

Ivo Creusen

Fixed-policy value-function prediction for stable control applications using machine learning

ABSTRACT

Dynamic systems and processes can be controlled to drive them to a desired state (e.g., optimal). Control can be done with anything ranging from a simple model to a complex algorithm, depending on the dynamic system. In real world problems we can collect real data and use machine learning (ML) approaches to develop these control systems. A common ML approach used for these problems is reinforcement learning. In this paper we present a simplified method for problems that have three characteristics: a relatively low number of actions; we can predict the output state given an action; and for each state there is a default action that will maintain the system stable unless there are external factors. Our approach permits fast model training and uses a relatively simple control process. As an example, the jitter buffer of a real-time telecommunications system that encounters stochastic packet losses and delays can be controlled by this approach.

DESCRIPTION

Control theory relates to control of dynamic systems and processes, in which a model or algorithm is developed for governing the control of system or process inputs to drive the system or process to a desired state, while reducing delay, overshoot, or steady-state error and ensuring a level of control stability of the system or process. Control theory can be applied to many processes and systems, including, for example, temperature control, automatic gain control of an amplifier (e.g., an amplifier that receives input from a microphone), airplane and ship control, etc.

Control systems are based on developing an understanding of the model governing the behaviour of the system. For most interesting real-world problems, developing or finding the right model might be a difficult task. Instead, as discussed herein, machine learning techniques can be applied to develop an approximation that models the dynamic process and that can be used to control the process. In real world problems, data from the dynamic system itself can be used to train the control system. This approach is based on reinforcement learning.

In this paper we present a simplified solution to this problem when three conditions are met:

1. the number of actions applied during the control process is relatively low,
2. the next state after executing the action can be predicted, and
3. for each state there is a default action that keeps the system in a stable state (e.g. the system stays in the optimal state) unless there are external factors.

In this case we can simplify the method to reduce the training and dynamic system simulation complexity. This approach permits fast model training and uses a relatively simple model.

In order to understand how to apply these conditions in a control problem we will use the following example: maintaining a jitter buffer in a real-time telecommunications system that encounters stochastic packet losses and delays. Communication equipment or software that enables audio and/or video calls over the Internet uses a jitter buffer to buffer incoming data packets. The jitter buffer ensures that there is always enough media available for playback, e.g., to send to the sound card, even when there is jitter in network delay, e.g., under conditions of changing network delay.

The jitter buffer size is set as a trade-off between latency and interruptions. Too small a jitter buffer size results in buffer under-run, e.g., no media available to send to the sound or video card. Too large a jitter buffer size results in an increase in user-perceived delay, impeding real-time communication. Thus, having an optimally-sized jitter buffer throughout a communication is important to quality of experience. For example, when an ongoing call over Wi-Fi is switched to cellular data, e.g., LTE, there is a significant change in network conditions. Such significant change warrants a change in jitter buffer size. In general, a poor network (with high jitter) requires a larger jitter buffer while a good network (low jitter) requires a smaller jitter buffer. An adaptive jitter buffer can adjust the buffer size to respond in real-time to the actual network conditions, so that an optimal jitter buffer size that balances low latency vs. infrequent interruptions is maintained through a communication.

We describe techniques to control the size of the jitter buffer based on network conditions, and to adjust the jitter buffer size dynamically during a call. During the call, the buffer size will increase when a new packet arrives and will decrease after a playback event. We can not control at what time the packets will arrive, but we can control the playback rate to increase or reduce the speed at which the jitter buffer is depleted. A reward function can be defined to measure the performance of the model. For example, a reward function, R = A*d + B*u, can be used to measure the quality of the jitter buffer over a short time interval, where R expresses a balance between average delay (d) in delivering packets through the buffer and an underrun duration (u), and A and B are parameters that provided a weight to the importance of each term.

Previous reinforcement learning (RL) approaches to this problem focused on training a model to apply a corrective action (e.g., increasing, decreasing, or maintaining a playback rate to change the jitter buffer size) based on an observed Markov state of the system (i.e., the history of incoming packets to the jitter buffer of the communications system) to drive the jitter buffer size to an optimal value. Such RL approaches (e.g., using algorithms such as DQN) use a simulator to generate training data that is used to learn an action-value function (Q-function) connected to a greedy-policy that estimates discounted cumulative rewards at a given state for a given corrective action applied to the state. The learned action-value function can be applied to real world data (e.g., telecommunications network data), so that corrective actions are taken to optimize the system. However, a problem with such approaches is both the simulator and the Q-function itself are part of the training loop. We train the system to learn the Q-function, and while we learn it we change it. Changes to the Q-function alter the behaviour in the simulator forcing us to generate new data after each training iteration. This results in a high computational cost and slow training. Furthermore, predictability or similitude of inputs is an important factor for the success of RL techniques, so traditional techniques may have difficulty converging when inputs have a high degree of stochasticity, such as is the case of a real telecommunication network.

In our example we consider the state of the system as a collection of network statistics and the buffer size itself. The limited set of actions are the changes in playback rate that can be applied during the communication.

The techniques disclosed herein simplify this problem by recognizing that no action needs to be applied (and the current playback rate should be maintained) when the jitter buffer is in an optimal state. This default action (neither expanding nor contracting the buffer size) does not apply any correction to the system when no correction is needed. Using that policy, we can reformulate the control problem as one in which a corrective action is applied when the model detects that another state exists that is better as compared with the current state. This information is provided by the value function, V(s), rather than an action-value function. The value function returns the expected cumulative reward for a given state following a fixed policy. But considering that we can keep the model in an optimal state by applying a default action (no action in our example) in the absence of external perturbations, we can use the value function of a policy that always applies such default action. Given a value function, V(s), that is correctly trained, it should return a high value for the optimal state and a lower expected cumulative reward for states with a buffer size that is either too high or too low. Using V(s), we can explore the state space. The state space is composed of the history of network statistics (which are fixed) and information about the jitter buffer. Because the history of network statistics is fixed, exploration of the state space results in exploration of different jitter buffer sizes. By exploring points in the state space, an approximation to an optimal buffer size can be determined.

Thus, control of the buffer size can be reframed from learning from a Markov Decision Process, in which corrective actions to take on a given state are learned based on an action-value reward function, to learning from a Markov Reward Process, in which a state of the communications network and the jitter buffer is evaluated to learn the expected discounted cumulative reward of proceeding from a given state with a fixed-policy of neither increasing nor decreasing the buffer size. The Markov Reward Process is quite efficient because it removes the actions taken on the system as a parameter of the function to be evaluated and requires only generating sequences of training data to use in a supervised learning algorithm to train the machine learning model. Because actions are not included in the sequences, a machine learning model can be trained using multiple, parallel evaluations of a state of the communications network and the jitter buffer in a reinforcement learning process, which significantly simplifies the training process and reduces the number of computations needed to train the model.

Sequences of training data can be generated from actual VoIP calls, e.g., audio and/or video calls over a packet-switched network such as the Internet, upon permission from the user for use of log data. Such calls generate logs that are descriptive of traffic or network conditions during the call. The logs do not include any user data or identifiable information. For example, the logs include real time communications event logs (RTC event logs), and include, e.g., timing information of incoming real time transport protocol (RTP) packets, RTP headers, timing information of the sound card requesting media, etc. Packet jitter, which is the difference in timing between packet arrival and request by sound card, is computed for each packet in a call using the event log.

The RTC event logs are used to simulate the incoming network packets and the dynamics of the telecommunications system (e.g., how often and when it requires audio packets for playback) to create the training data. In each simulation, a simulator applies the same naïve policy π of doing nothing (i.e., neither expanding nor contracting the buffer size) unless the buffer size is less than a threshold value required by the model, which forces an emergency expansion of the buffer size. In an implementation, input sequences are cut in chunks of nine sections with a hop size of six seconds, and for each sequence chunk, fifty different simulations using initial buffer sizes of zero to fifty packets are started, and a discounted reward is calculated for each simulation. For example, the discounted reward can be equal to an average delay multiplied by a first discount plus an underrun duration multiplied by a second discount. For sequences exhibiting high numbers of packet loss, the buffer can be depleted, and once this happens, that simulation can be terminated for efficiency purposes.

From the simulations, a set of training data is created that includes information about 1) states of the system, including information about network statistics and a jitter buffer size, and 2) a discounted reward for each state. The discounted reward for a state is based on the past, present, and future jitter information for the state obtained from the event logs, and the discounted reward can be determined based on the reward function. The discounted reward for a state represents a labeled training datapoint and these data points can be used to train the value function as a supervised learning problem.

Accordingly, a machine learning model then can be trained as a supervised learning model over this set of training data. In an example implementation, a fully connected network with three layers of the same size and a final layer that outputs the expected discounted reward can be used. In experiments conducted by the authors, tanh was used as an activation function, adam was used as an optimization algorithm, and L2 regularization was used.

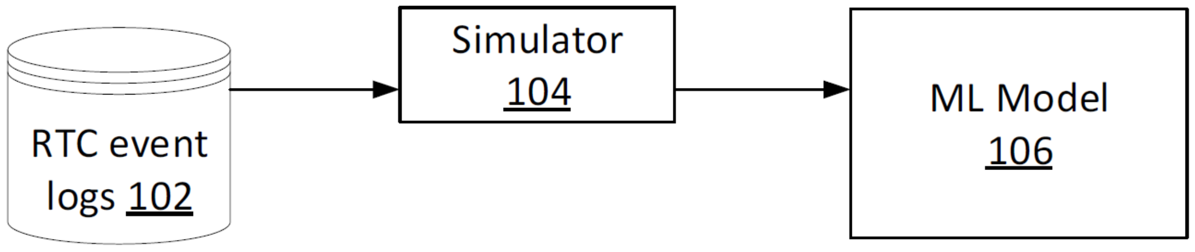

Fig. 1: Training the machine learning model

Fig. 1 illustrates the training of a machine learning model used to predict jitter buffer sizes. RTC event logs (102) of network states are passed to a simulator (104) that generates training data in the form of value functions $V_\pi(s)$, for the states, s, when the naïve policy is applied, where the states, s, describe the network history and a jitter buffer size. Because the value function data depend on the jitter buffer state and the network state, they can be used to select an optimal jitter buffer size for a given network state. The value functions are provided as a target for the machine learning model (106), along with the RTC event logs (102). The input features for the model (106) can include jitter data, e.g., inter-arrival times between incoming RTP packets from the RTC event log. By providing event logs and $V_\pi(s)$ data (104) from a large number of completed calls, the machine-learning model is trained to predict the optimal buffer size.

In an implementation, the training data can be generated faster and more efficiently compared to other techniques, and the trained model can produce fewer buffer underruns compared to other techniques to manage the buffer size, with a similar average buffer level. Further, the approach is completely data-driven since the model is trained on actual data of network conditions commonly experienced during calls. Different models also can be easily generated using different reward functions (e.g., different weights for the latency and underrun factors).

The machine-learning model can be implemented using a long short-term memory (LSTM) neural network. Other types of models, e.g., recurrent neural networks, convolutional neural networks, support vector machines, random forests, boosted decision trees, etc., can also be used.

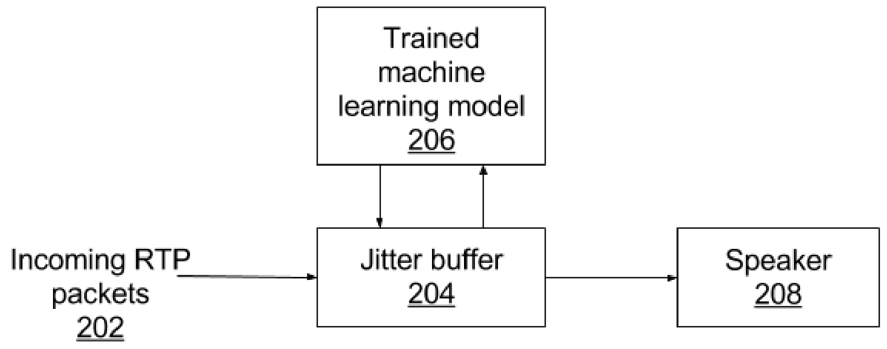

Fig. 2: Using a trained machine learning model to adjust jitter buffer size in real time

Fig. 2 illustrates the use of a machine learning model, trained using techniques of this disclosure, to control a jitter buffer size. Incoming RTP packets (202) enter the jitter buffer (204). A trained machine learning model (206) adjusts the jitter buffer size based on, for example, real-time information about the timing of incoming packets and outgoing data, e.g., to a sound card, for playback. The techniques of this disclosure can be used for calls conducted over the Internet, e.g., in audio and/or video conferencing or calling software.

Techniques of this disclosure perform data-driven control of dynamic systems. For example, a jitter buffer used for audio and/or video calls conducted over the Internet is controlled. Event logs that contain data about network conditions experienced during completed calls are used to calculate optimal jitter buffer size using a value function framework for a Markov state of the network, including the jitter buffer. The value functions are used as a target for offline training of a machine-learning model to calculate optimal jitter buffer size. The trained model is employed to control jitter buffer size during a call. Control of jitter buffer size using the techniques can reduce buffer under-runs without increasing average delay. Alternatively, depending on the weights in the optimization criterion, the techniques of this disclosure can reduce average delay with the same chance of buffer underrun.

Although the primary example described in this description relates to control of a jitter buffer, the techniques described herein can be applied more generally to control a system having limited inputs in the action space and/or relatively infrequent action space inputs.