Reinforcement Learning: A Case Study in Model Generalization

John Morrow[*]

Abstract

This project explores the ability of a model trained with reinforcement learning (RL) to generalize, i.e., produce acceptable results when presented with data it was not exposed to during training. The application in this study is an industrial process with multiple controls that determine the effect on a product as it transitions through the process. Determining optimal control settings in this environment can be challenging. For example, when there are interactions between the controls, adjusting one setting can require the readjustment of other settings. Also, a complex relationship between a control and its effect complicates finding an optimal solution. The results presented here show that a model trained by an RL process performs well in this environment. Further, with proper definitions of the state and reward functions in the RL process, the trained model is able to generalize to conditions different from those used for training.

1 Introduction

In this project, an RL model is trained to find the optimal control settings for a reflow oven used for soldering electronic components to a circuit board (Figure 1 & Figure 2). The oven's moving belt transports the product (i.e., the circuit board) through multiple heating zones. This process heats the product according to a temperature-time target profile required to produce reliable solder connections.

One of the challenges in finding optimal control settings is that the timing of the target profile's zones does not necessarily align with the oven's heating zones as the product moves through the oven. The model must cope with this misalignment to produce a product profile close to the target profile.

[*] www.linkedin.com/in/johnmorrow1000

Figure 1: **Reflow oven**
(image licensed from Adobe)

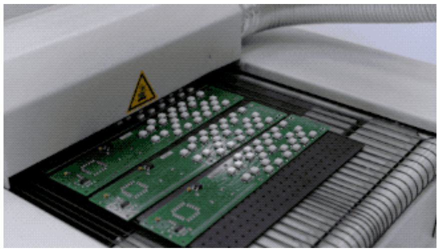

Figure 2: **Circuit boards on oven belt**
(image licensed from Adobe)

The reflow oven in this project has eight heating zones, each with a control for setting the temperature of the zone's heater (Figure 3). Sensors record the temperature of the product at three hundred points as it travels through the oven.

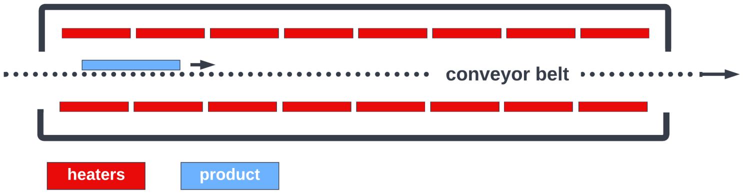

Figure 3: **Reflow oven schematic diagram**

2 Process optimization

A human operator typically takes the following steps to determine the heater settings required to solder circuit boards successfully:

- run one pass of the product through the oven
- observe the resulting temperature-time profile from the sensor readings
- adjust the heater settings to improve the profile toward the target profile
- wait for the oven temperature to stabilize to the new settings
- repeat this procedure until the profile from the sensor readings is acceptably close to the target profile

2.1 Learning the policy

An RL system replaces the operator steps with a two-stage process. In the first stage, an agent learns the dynamics of the oven and creates a policy for updating the heater settings under various oven conditions.

Since considerable time is required to stabilize an oven's temperature after making a change to the heater settings and then to pass the product through the oven, an oven simulator is used to speed up the learning process [Morrow, 2022][5]. The simulator emulates a single pass of the product through the heating profile in a few seconds instead of the many minutes required by a physical oven.

In each pass of the learning stage, the agent takes an action from its current state by sending the simulator new settings for the eight heaters. After the simulation run, the simulator reports back the product temperature readings (three hundred readings taken at 1-second intervals).

The agent is rewarded for its action based on the difference between the returned readings (oven_pts) and the target temperature-time profile (profile_pts). If the difference for the current run is less than the previous run, the reward is positive; otherwise, it is negative.

A subset of the readings determines the new state of the system. The agent starts the next pass of the learning stage by taking action from the new state.

2.2 Planning with the policy

In the second stage, the agent follows the learned policy to find optimal heater settings. These settings will produce the closest match between the actual product profile and the target temperature-time profile. Figure 4 shows the result of the agent following the policy to find optimal settings. The blue trace is the target temperature-time profile, and the red trace is the actual profile produced by the optimal settings.

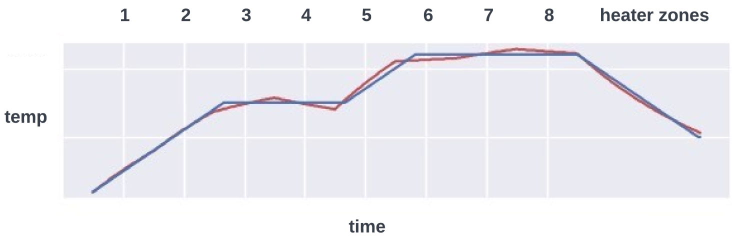

Figure 4: **Example planning result** Blue trace: target profile. Red trace: actual product profile.

3 Reinforcement learning system

As discussed above, an RL system comprises an agent taking actions in an environment to learn a policy for reaching the target goal. The environment responds to each action with a reward indicating whether the action was good or bad toward reaching the goal. The environment also returns the state of the agent in the environment. The agent consists of two neural networks: the model network and the target network. The agent's goal is to find heater settings that will produce a product time-temperature profile that is very close to the target profile. The environment is the reflow oven simulator. (Figure 5)

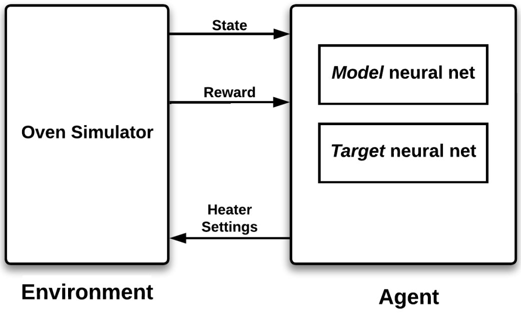

Figure 5: **Reinforcement learning system**

3.1 Generalization: state and reward definition

The state and reward definitions are critical to the RL model's ability to generalize to new environments where the target profile and product parameters differ from those used during training. Specifically, both the state and reward are defined in terms of the relative difference between the product and target profile temperatures and normalized by the maximum range of allowed heater values.

3.1.1 State definition

State parameters are defined at the centers of the eight heater zones. Each state parameter, m, is defined as the normalized difference between the temperature at the center of the product and the temperature of the profile at the center of each heater zone:

$$state\_params_m = state\_scale \cdot \left(\frac{profile\_pts_m - oven\_pts_m}{maximum\_temperature\_range} \right) \tag{1}$$

The $state\_scale$ factor controls the magnitude of input values presented to the neural networks. The $maximum\_temperature\_range$ factor is the maximum range of allowed heater values.

3.1.2 Reward definition

When the agent performs an action, the environment returns a reward indicating the effectiveness of the action in achieving the agent's goal. The reward is based on whether the action reduced the total temperature difference between the oven_pts and profile_pts. The total temperature difference is calculated by the error function, and then further processed by the reward function to produce the final reward presented to the agent.

Error function: The error function reports the summed differences between the oven points and their corresponding target profile points. The total error comprises two parts, the error for oven_pts above the target profile and the error for points below the target profile:

$$current\_error_A = \sum_{n=0}^{num\ pts} (profile_n - oven_n), \quad for\ profile_n > oven_n \tag{2}$$

$$current\_error_B = \sum_{n=0}^{num\ pts} (oven_n - profile_n), \quad for\ oven_n \geq profile_n \tag{3}$$

Reward function: The total reward is the sum of two terms: reward_A and reward_B. Each of these terms comprises two factors (Figure 6):

- the difference between the current error and the previous error
- a weighting factor determined by the average of the current error and previous error

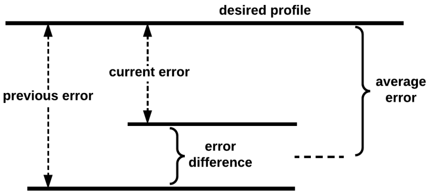

Figure 6: **Reward elements** The reward is based on the difference between the current and previous errors and the average of the two errors.

This produces a reward that represents not only the increase or decrease in the error but also the magnitude of the error. For example, an error reduction for a large average error would produce a greater reward than the same error reduction for a smaller average error. In other words, the agent is rewarded more for reducing large average errors.

The factor, k_weight, can change the relative importance of errors above or below the target profile. The factors (reward_scale_factor, weight_scale, and weight_normalize) are used to scale the range of reward values.

$$reward = reward\_scale\_factor \cdot ((reward_A + k\_weight \cdot (reward_B)) \tag{4}$$

where,

$$reward_A = weight_A \cdot (prev\_error_A - current\_error_A) \tag{5}$$

$$reward_B = weight_B \cdot (prev\_error_B - current\_error_B) \tag{6}$$

$$weight_A = weight\_scale \cdot \left(\frac{prev\_error_A + current\_error_A}{weight\_normalize} \right) + 1 \tag{7}$$

$$weight_B = weight\_scale \cdot \left(\frac{prev\_error_B + current\_error_B}{weight\_normalize} \right) + 1 \tag{8}$$

$$weight\_normalize = 2 \cdot num\_profile\_pts \cdot (maximum\_temperature\_range) \tag{9}$$

3.2 Learning process

The model is trained with a DQN [4] SARSA process [Sutton & Barto, 2018, p.129] [6]. The DQN process is modified [2] to use a mellowmax update target (Appendix A). A diagram of the learning process is presented in Figure 7 and the steps of the process are detailed in Algorithm 1.

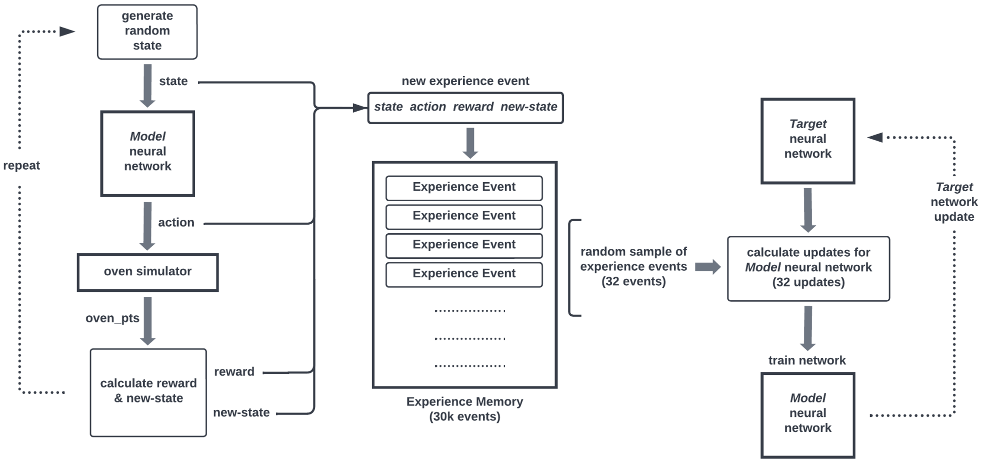

Figure 7: **Learning process**

3.2.1 Experience memory

The experience memory is used to implement experience replay [3] [4]. Experience replay provides a number of advantages including the efficiency of using each experience in multiple training updates and providing uncorrelated update batches for training the model neural network.

Experiences are collected from the environment (oven simulator) and stored in the experience memory. Each experience starts from a random state and an action is chosen in the state based on epsilon-greedy selection. The agent takes the action to move to a new state and receives a reward based on how good the action was toward achieving the agent's goal. Each experience is stored in the experience memory as (state, action, reward, new-state).

Typically, experiences are collected in episodes that start from a random state followed by a sequence of experiences generated by following an epsilon-greedy policy. For this project, the best performance resulted when using one experience per episode (Appendix B).

3.2.2 Model network training

Periodically, as the agent collects experiences, a random sample of thirty-two experiences is drawn from the experience memory and a batch of thirty-two training updates is calculated from the samples. The model neural network is then trained with the batch of updates comprising states as inputs and their corresponding action-value updates as training targets.

Equation 10 defines the procedure for updating action-values in a SARSA model ([Sutton & Barto, 2018, p.129] [6]) with a mellowmax policy (Appendix A).

$$\underbrace{Q(s_t, a_t)}_{\text{new value}} \leftarrow \underbrace{Q(s_t, a_t)}_{\text{current value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount}} \cdot \underbrace{mm\_value(s_{t+1})}_{\text{mellowmax value}} - \underbrace{Q(s_t, a_t)}_{\text{current value}} \right) \tag{10}$$

This procedure is implemented in the model neural network by applying values from Equation 10 to the cost function of the gradient descent algorithm used to train the network. The cost function is given by:

$$J(\theta) = 1/2\,(y - \hat{y})^2 \tag{11}$$

where y is the current value output by the network:

$$y = Q(s_t, a_t) \tag{12}$$

and $\hat{y}$ is the desired output:

$$\hat{y} = r_t + \gamma \cdot\ mm\_value(s_{t+1}). \tag{13}$$

Learning rate, α The learning rate in Equation 10 becomes the learning rate used in the gradient descent algorithm.

Gamma, γ The γ hyperparameter in Equation 10 controls the influence of future rewards. The greater the value of γ, the greater the influence of rewards from distant steps. After investigating a range of values for γ, it was found that $\gamma = 0.99$ produces action values that are extremely close to each other in value and frequently result in unstable behavior, i.e., diverging errors in the planning process. Setting $\gamma = 0.81$ spreads out the action values delivering stable behavior.

3.2.3 Target network

The target neural network is a frozen copy of the model network that is periodically refreshed by updating it with the weights from the model network. This network provides the update targets for the DQN update process. The use of a separate target network improves the stability of the learning process [4].

3.3 Planning process

As discussed in Section 2.2, the agent uses the trained model neural network to find oven settings that produce a product profile close to the target temperature-time profile. The agent starts from an initial state and takes a greedy action step from that state and from each successive state until it reaches a minimum error between the product profile and the target profile. A diagram of the planning process is presented in Figure 8 and the steps of the process are detailed in Algorithm 2.

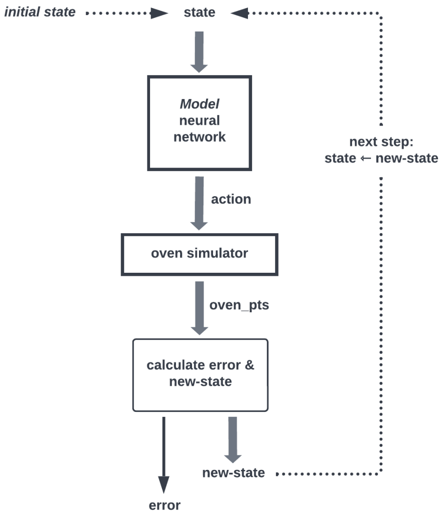

Figure 8: **Planning process**

An example of the error reduction as the agent follows the policy is presented in Figure 9.

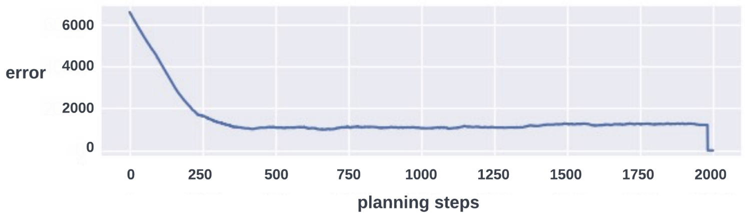

Figure 9: **Planning error**

4 Results

Following are the test results of running the planning process on various configurations of product materials and temperature-time profiles. All of the tests were run with a model neural network trained with the following product and profile parameters:

Oven Parameters	
transport belt speed	0.0075 m/s
heat transport coefficient (air)	50.0 W/degK-m**2
heater zones (1-8) lengths	0.225, 0.225, 0.225, 0.225, 0.225, 0.225, 0.225, 0.225 m
top heaters	on
bottom heaters	off

Table 1: Oven parameters (training)

Target Profile Parameters			
zone	**start temp (degC)**	**slope (degC/sec)**	**duration (sec)**
1	20	2	65
2	150	0	60
3	150	2	35
4	220	0	80
5	220	-2	60

Table 2: Profile parameters (training)

Product Parameters	
material	FR4 circuit board
density	2000 kg/m3
specific heat	1300 J/kg-degC
thermal conductivity	0.8 w/m-degK
length	0.1 m
width	0.1 m
thickness	0.0016 m

Table 3: Product parameters (training)

As mentioned in the Introduction, one of the challenges in finding optimal control settings is that the timing of the target profile's zones does not necessarily align with the oven's heating zones as the product moves through the oven [Figure 10]. Although the model is able to minimize the resulting error, the misalignment will generally account for some unavoidable portion of the error reported in each test.

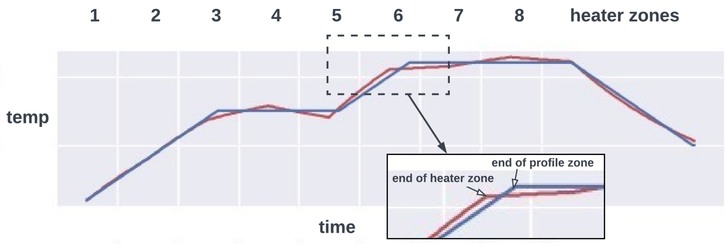

Figure 10: **Zone misalignment** Blue trace: target profile. Red trace: actual product profile.

The following notes apply to the errors reported in tests 1-6:

- The errors reported are single point errors, i.e., individual oven_pts compared to their corresponding profile_pts.
- The errors reported apply to the first four target profile zones. The fifth profile zone is beyond the eight heater zones and allows the product to cool to ambient temperature. This zone is outside the agent's control.

4.1 Test 1: baseline using training parameters

Test 1 is a baseline for testing the model's performance with the same parameters used to train the model [1, 2, 3]. Following are the test 1 errors, the optimal heat zone settings, and the target profile vs. actual temperature-time plot.

Test 1	
Average Error	1.9 %
Maximum Error	7.3 %

Optimal Heater Settings (°C)							
zone 1	zone2	zone3	zone 4	zone 5	zone 6	zone 7	zone 8
207	273	210	100	357	260	218	230

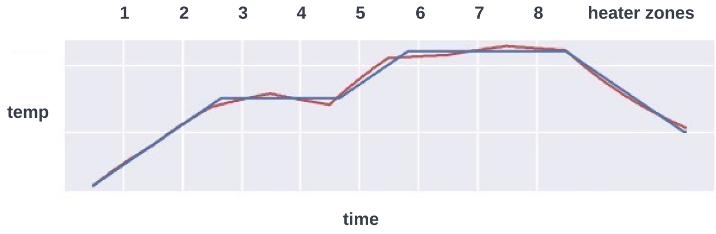

Figure 11: **Test 1** Blue trace: target profile. Red trace: actual product profile.

4.2 Test 2

Test 2 increases the size of FR4 product. The oven and profile parameter values are the same as used in the baseline of test 1, except that both the top and bottom heating elements are active. The following table reflects the product parameters used for this test (changes from the baseline training parameters are **bold**).

Product Parameters	
material	FR4 circuit board
density	2000 kg/m3
specific heat	1300 J/kg-degC
thermal conductivity	0.8 w/m-degK
length	0.2 m
width	0.15 m
thickness	0.0032 m

Table 4: Product parameters (test 2)

Following are the test 2 errors, the optimal heat zone settings, and the target profile vs. actual temperature-time plot.

Test 2	
Average Error	2.2 %
Maximum Error	10.0 %

Optimal Heater Settings (°C)							
zone 1	zone2	zone3	zone 4	zone 5	zone 6	zone 7	zone 8
195	275	212	99	351	256	222	233

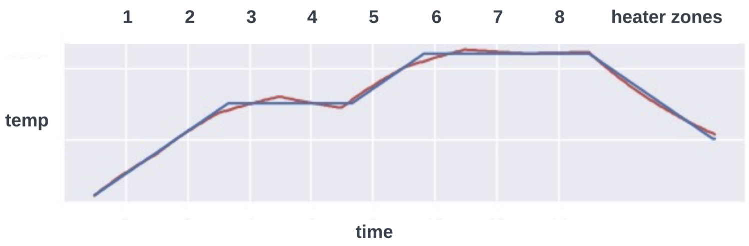

Figure 12: **Test 2** Blue trace: target profile. Red trace: actual product profile.

4.3 Test 3

Test 3 increases the size of the FR4 product and has a different profile from the baseline of test 1. The oven parameter values are the same as used in the baseline of test 1, except that both the top and bottom heating elements are active. The following tables reflect the profile and product parameters used for this test (changes from the baseline training parameters are **bold**).

Target Profile Parameters			
zone	**start temp (degC)**	**slope (degC/sec)**	**duration (sec)**
1	20	**1.5**	**60**
2	**110**	0	60
3	**110**	2	**30**
4	**200**	0	**75**
5	**200**	-2	**50**

Table 5: Test 3 profile parameters

Product Parameters	
material	FR4 circuit board
density	2000 kg/m3
specific heat	1300 J/kg-degC
thermal conductivity	0.8 w/m-degK
length	0.2 m
width	0.15 m
thickness	0.0032 m

Table 6: Product parameters (test 3)

Following are the test 3 errors, the optimal heat zone settings, and the target profile vs. actual temperature-time plot.

Test 3	
Average Error	3.2 %
Maximum Error	9.4 %

Optimal Heater Settings (°C)							
zone 1	zone2	zone3	zone 4	zone 5	zone 6	zone 7	zone 8
173	179	142	83	376	232	216	220

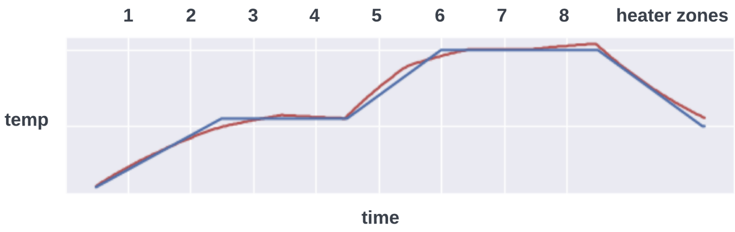

Figure 13: **Test 3** Blue trace: target profile. Red trace: actual product profile.

4.4 Test 4

Test 4 changes the product from FR4 to aluminum oxide (alumina 99%). The oven and profile parameter values are the same as used in the baseline of test 1. The following table reflects the product parameters used for this test (changes from the baseline training parameters are **bold**).

Product Parameters	
material	alumina 99% circuit board
density	3900 kg/m3
specific heat	800 J/kg-degC
thermal conductivity	27.5 w/m-degK
length	0.1 m
width	0.1 m
thickness	0.0016 m

Table 7: Product parameters (test 4)

Following are the test 4 errors, the optimal heat zone settings, and the target profile vs. actual temperature-time plot.

Test 4	
Average Error	1.8 %
Maximum Error	6.6 %

Optimal Heater Settings (°C)							
zone 1	zone2	zone3	zone 4	zone 5	zone 6	zone 7	zone 8
236	306	232	97	369	268	224	227

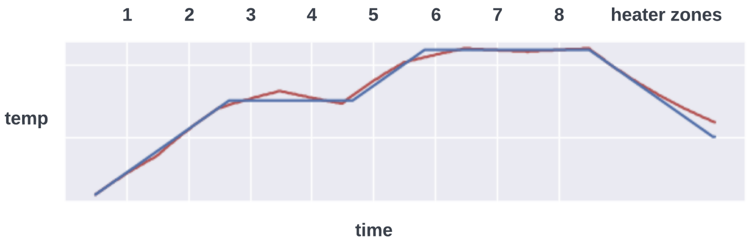

Figure 14: **Test 4** Blue trace: target profile. Red trace: actual product profile.

4.5 Test 5

Test 5 changes the product to aluminum oxide (alumina 99%) and increases the size of the product. The oven and profile parameter values are the same as used in the baseline of test 1, except that both the top and bottom heating elements are active. The following table reflects the product parameters used for this test (changes from the baseline training parameters are **bold**).

Product Parameters	
material	alumina 99% circuit board
density	3900 kg/m3
specific heat	800 J/kg-degC
thermal conductivity	27.5 w/m-degK
length	0.2 m
width	0.15 m
thickness	0.0032 m

Table 8: Product parameters (test 5)

Following are the test 5 errors, the optimal heat zone settings, and the target profile vs. actual temperature-time plot.

Test 5	
Average Error	2.2 %
Maximum Error	10.0 %

Optimal Heater Settings (°C)							
zone 1	zone2	zone3	zone 4	zone 5	zone 6	zone 7	zone 8
220	311	233	86	390	267	220	240

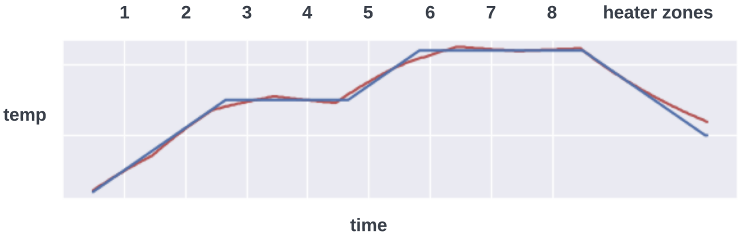

Figure 15: **Test 5** Blue trace: target profile. Red trace: actual product profile.

4.6 Test 6

Test 6 changes the product from FR4 to aluminum oxide (alumina 99%), changes the size of the product, and changes the profile. The oven parameter values are the same as used in the baseline of test 1, except that both the top and bottom heating elements are active. The following tables reflect the profile and product parameters used for this test (changes from the baseline training parameters are **bold**).

Target Profile Parameters			
zone	**start temp (degC)**	**slope (degC/sec)**	**duration (sec)**
1	20	**3**	**60**
2	**200**	0	60
3	**200**	2	**40**
4	**280**	0	**80**
5	**280**	**-3**	60

Table 9: Test 6 profile parameters

Product Parameters	
material	alumina 99% circuit board
density	3900 kg/m3
specific heat	800 J/kg-degC
thermal conductivity	27.5 w/m-degK
length	0.2 m
width	0.15 m
thickness	0.0032 m

Table 10: Product parameters (test 6)

Following are the test 6 errors, the optimal heat zone settings, and the target profile vs. actual temperature-time plot.

Test 6	
Average Error	2.8 %
Maximum Error	15.0 %

Optimal Heater Settings (°C)							
zone 1	zone2	zone3	zone 4	zone 5	zone 6	zone 7	zone 8
314	478	245	120	524	286	314	284

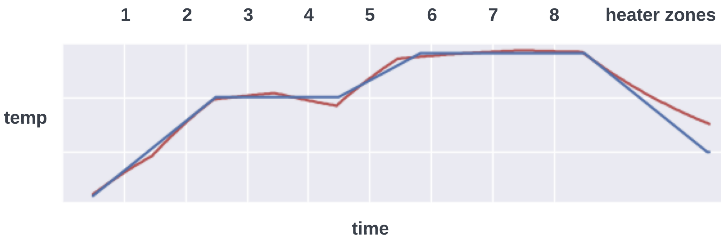

Figure 16: **Test 6** Blue trace: target profile. Red trace: actual product profile.

5 Conclusion

This project demonstrates that a reinforcement learning system can provide solutions to control a complex industrial process. Specifically, a reinforcement learning system successfully learns the optimal control settings of a reflow oven used to solder electronic components to a circuit board. Further, once trained, the system can generalize to produce acceptable results in environments with different requirements from those used during training.

6 Appendix A: Mellowmax

This project uses a SARSA model [Sutton & Barto, 2018, p.129] [6] with a mellowmax policy [Asadi and Littman, 2017][1]. Mellowmax is a variant of the Boltzmann softmax operator updated to avoid potential stability issues. The mellowmax operator produces a value derived from all of the action values of a state.

$$mm\_value(s) = \frac{1}{mm\_temp} \cdot \log\left(\frac{1}{n} \cdot \sum_{i=1}^{n} e^{(x_i - max_i(x_i))}\right) + max_i\,(x_i) \tag{14}$$

where x_i are the action values of a state, s.

This operator was chosen for its ability to provide a good balance between exploitation and exploration. Other properties of the mellowmax operator are discussed in the following references: [Asadi and Littman, 2017][1] and [Kim, 2019][2].

7 Appendix B: Experiences per episode

The model was trained with episodes that start from a random state followed by a sequence of experiences generated by following an epsilon-greedy policy. Each experience is added to the experience memory as it is generated. Two model configurations were investigated: sixteen experiences per episode and one experience per episode.

With sixteen-experience episodes, the experience samples in each episode are in relatively tight clusters in state space. This contrasts with one-experience episodes where the samples are more randomly distributed (Figure 17). For a given number of experiences, the one-experience model gives better state space coverage (upper plots). The figure also shows that the one-experience model's average return during training stabilizes with fewer total experiences (lower plots). It is stable well before 48,000 experiences as compared to the sixteen-experience model, which has still not reached stability at 160,000 experiences.

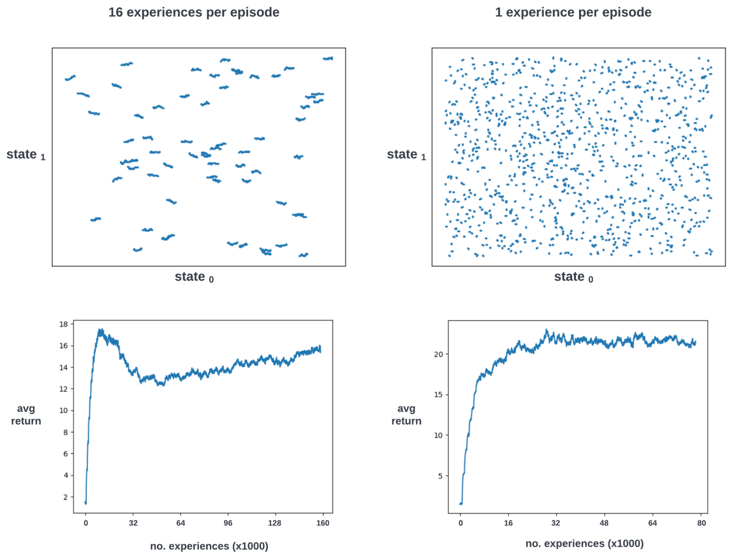

Figure 17: **Experiences per episode** (upper plots) 2-dimension state-space simulation of 1000 experience samples. $state_0$ and $state_1$ are the state parameters of the space. (lower plots) The average return plots are from training sessions of the 8-dimension state-space model.

It is believed that this result is due to the model neural network serving as a complex function generator that learns the function, $action\_value = f(state\_parameters)$, by training on update batches generated from experience memory samples. Since the model then uses the learned function to deliver an action value when presented with an arbitrary input of state parameters, the more evenly distributed the training samples in state space, the more accurately the learned function can interpolate outputs for arbitrary inputs.

8 Algorithms

Algorithm 1 learning algorithm

```
 $model(\cdot)$ model network
 $target(\cdot)$ target network
algorithm parameters
$n\_exp$  total experiences
$n\_train$  experiences per model network update
$n\_update$  experiences per target network update
procedure learn
    $state \leftarrow random\_state$                                   ▷ generate random state
    $n = 0$
    while $n \neq n\_exp$ do
        $action\_values \leftarrow model(state)$
        $action \leftarrow get\_action(action\_values)$                 ▷ epsilon-greedy selection
        $htr\_settings \leftarrow get\_htr\_settings(action)$
        $oven\_pts \leftarrow simulator(htr\_settings)$                ▷ run pass with oven simulator
        $reward \leftarrow get\_reward(oven\_pts)$
        $new\_state \leftarrow get\_nxt\_state(oven\_pts)$
        $exp\_memory \leftarrow state, action, reward, new\_state$     ▷ save experience to exp_memory
        $state \leftarrow random\_state$
        $n \leftarrow n + 1$
        if $(n \bmod n\_train) = 0$ then                              ▷ train model every n_train experiences
            sample random batch from exp_memory
            for each experience in batch do
                $state, action, reward, new\_state \leftarrow exp\_sample$
                $target\_action\_values \leftarrow target(new\_state)$
                $target\_value \leftarrow mellowmax(target\_action\_values)$
                $update(state, action) \leftarrow reward + (\gamma \cdot target\_value)$
                add this update to batch of updates
            end for
            train model with batch of updates
        end if
        if $(n \bmod n\_update) = 0$ then                             ▷ update target every n_update experiences
            $target \leftarrow model$                                   ▷ target updated by cloning model
        end if
    end while
end procedure
```

Algorithm 2 planning algorithm

```
 $model(\cdot)$ model network
procedure plan
    $oven\_pts \leftarrow simulator(initial\_htr\_settings)$        ▷ run pass with oven simulator
    $current\_error \leftarrow get\_error(oven\_pts)$
    $state \leftarrow get\_nxt\_state(oven\_pts)$        ▷ initial state
    while $error > current\_error$ do
        $action\_values \leftarrow model(state)$
        $action \leftarrow get\_action(action\_values)$        ▷ greedy selection
        $htr\_settings \leftarrow get\_htr\_settings(action)$
        $oven\_pts \leftarrow simulator(htr\_settings)$
        $error \leftarrow get\_error(oven\_pts)$
        $new\_state \leftarrow get\_nxt\_state(oven\_pts)$
        $state \leftarrow new\_state$
        if $error < current\_error$ then
            $current\_error \leftarrow error$
        end if
    end while
    output: $htr\_settings$        ▷ final heater settings to program oven
end procedure
```

References

[1] K. Asadi and M. L. Littman, “An alternative softmax operator for reinforcement learning,” 2016. [Online]. Available: https://arxiv.org/abs/1612.05628

[2] S. Kim, K. Asadi, M. Littman, and G. Konidaris, “Deepmellow: Removing the need for a target network in deep q-learning,” in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, 7 2019, pp. 2733–2739, also available as https://www.ijcai.org/proceedings/2019/0379.pdf.

[3] L. Lin, “Reinforcement learning for robots using neural networks,” 1992, also available as https://apps.dtic.mil/sti/pdfs/ADA261434.pdf.

[4] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, pp. 529–533, 2015, also available as https://www.deepmind.com/publications/human-level-control-through-deep-reinforcement-learning.

[5] J. Morrow, “Reinforcement learning: Training environment simulator,” 2022. [Online]. Available: https://morrowconsultants.com/rl-environment-simulator-paper

[6] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book, 2018, also available as http://www.incompleteideas.net/book/the-book-2nd.html.