COMP0078 Coursework 1

Student Number: 19002505

Date: 16/11/2022

1 PART I

1.1 Linear Regression

Question 1:

(a) Polynomial basis functions with dimensions k = 1,2,3,4 fitted to the data:

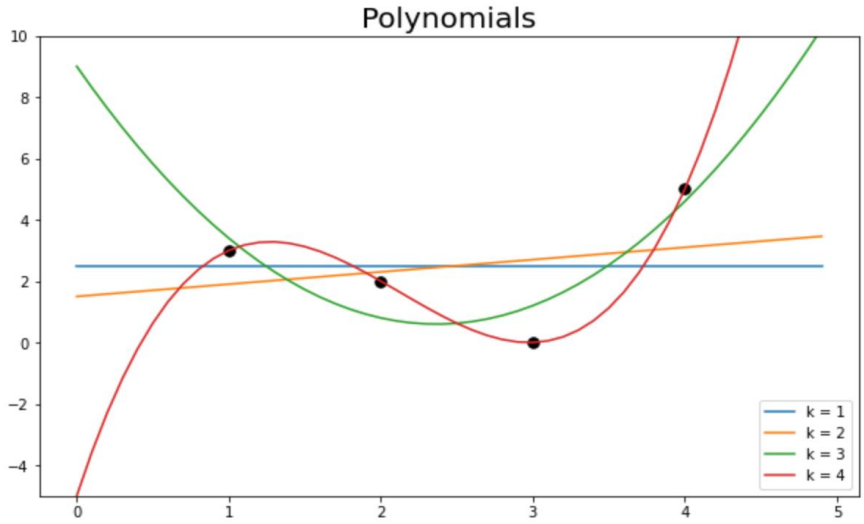

Figure 1. Dataset fitted with k-dimensional polynomial basis functions

(b) when $k = 1$, the equations is: $y = 2.5$
when $k = 2$, the equations is: $y = 1.5 + 0.4x$
when $k = 3$, the equations is: $y = 9 - 7.1x + 1.5x^2$

(c) when $k = 1$, the mean square error is: $MSE = 3.250$
when $k = 2$, the mean square error is: $MSE = 3.050$
when $k = 3$, the mean square error is: $MSE = 0.800$
when $k = 4$, the mean square error is: $MSE = 0.000$

Question 2:

(a) The function $sin^2(2\pi x)$ and the k-dimensional fitted polynomials superimposed over data:

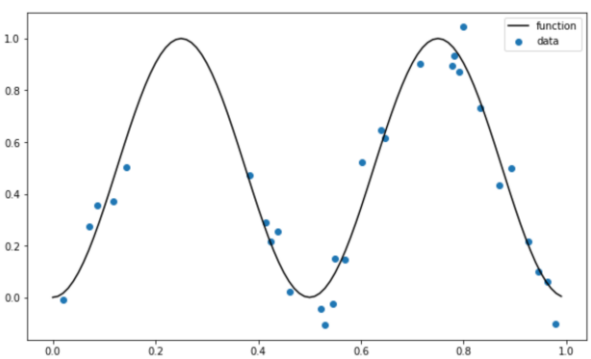

Figure 2 (left). The function $sin^2(2\pi x)$ and 30 noisy data points

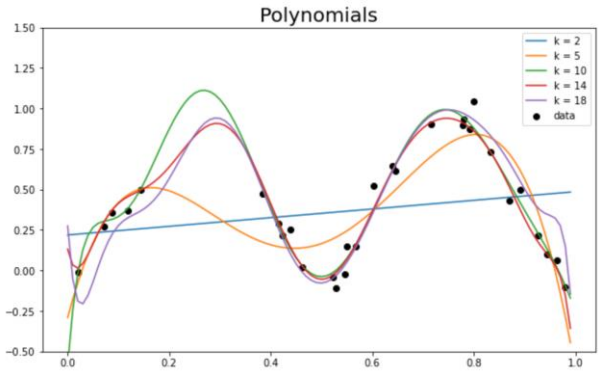

Figure 3 (right). Polynomial bases of dimensions k = 2,5,10,14,18 fitted to the data

(b) The variation of the training error with the dimension of the polynomial basis functions:

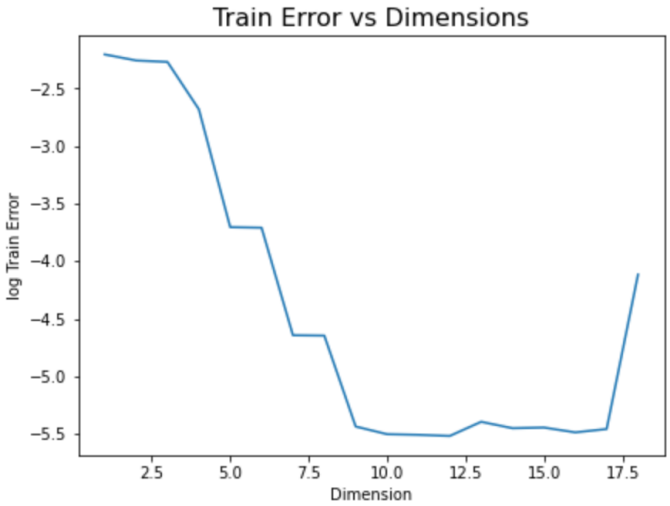

Figure 4. Decreasing log training error as the polynomial dimension increases from 1 to 18

(c) The variation of the test error with the dimension of the polynomial basis functions:

Figure 5. The phenomena of overfitting observed on the resultant test errors

(d) The smoothed training and test curves over 100 runs:

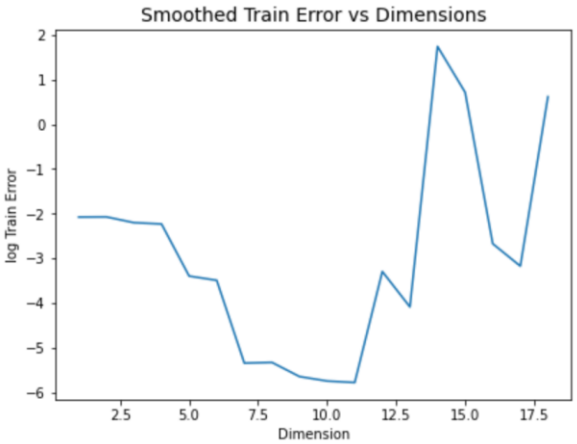

Figure 6. Smoothed training curves over 100 runs

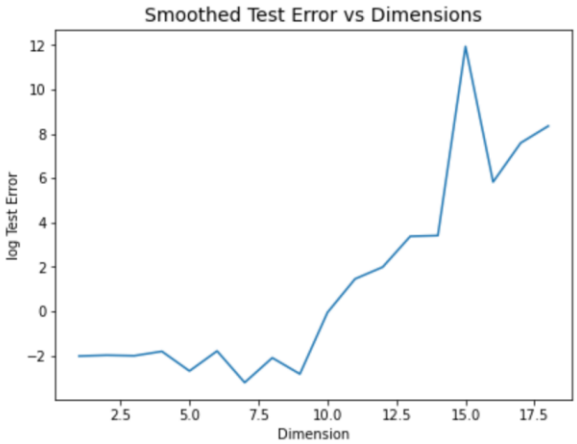

Figure 7. Smoothed test curves over 100 runs

Question 3:

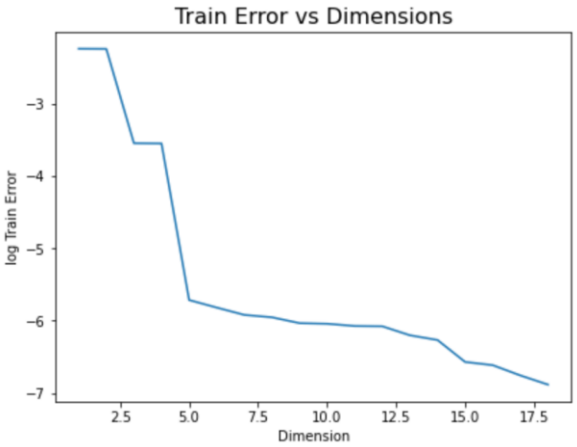

Figure 8. Training curves with the sin ($k\pi x$) basis function

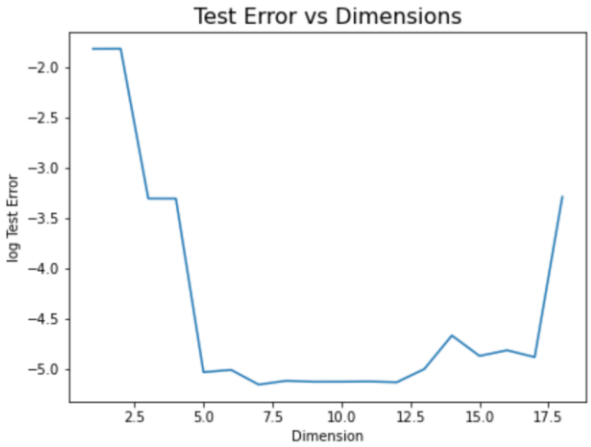

Figure 9. Test curve with the sin ($k\pi x$) basis function

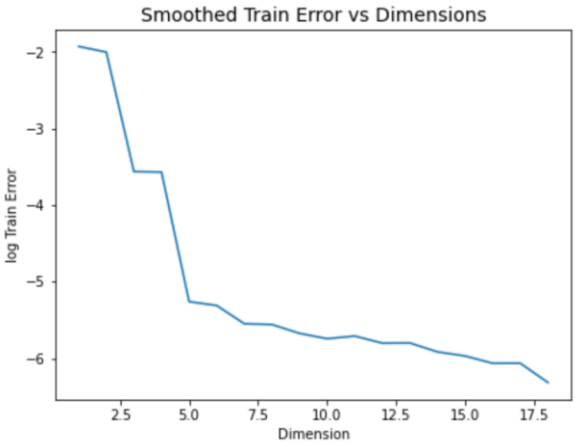

Figure 10. Smoothed training curves over 100 runs

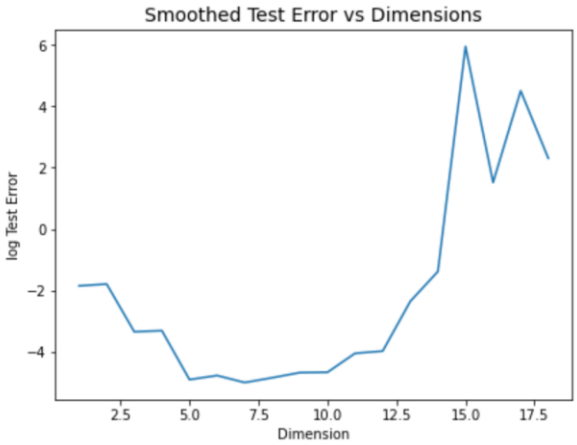

Figure 11. Smoothed test curves over 100 runs

1.2 Filtered Boston housing and kernels

Question 4:

a. For the Naïve Regression, the averaged MSE over 20 runs:
   - Training set: 84.11767
   - Test set: 85.28986

b. The constant function of ones used in the Naïve Regression is equating the predictions for all entries to the input feature, which is the mean y-value of the training data.

c. The averaged MSE over 20 runs for linear regression with single attributes:
   - For attribute = “CRIM”, training MSE = 70.82028, test MSE = 74.36369
   - For attribute = “ZN”, training MSE = 71.96865, test MSE = 76.75674
   - For attribute = “INDUS”, training MSE = 65.31261, test MSE = 63.76977
   - For attribute = “CHAS”, training MSE = 81.35577, test MSE = 83.20067
   - For attribute = “NOX”, training MSE = 69.26962, test MSE = 68.79069
   - For attribute = “RM”, training MSE = 44.16679, test MSE = 43.02052

- For attribute = “AGE”, training MSE = 71.09966, test MSE = 75.54005
- For attribute = “DIS”, training MSE = 79.76835, test MSE = 78.23432
- For attribute = “RAD”, training MSE = 72.94221, test MSE = 71.03977
- For attribute = “TAX”, training MSE = 66.05708, test MSE = 65.95092
- For attribute = “PTRATIO”, training MSE = 62.05360, test MSE = 64.30679
- For attribute = “LASTA”, training MSE = 37.98397, test MSE = 39.84628

d. For linear regression with all the attributes, the averaged MSE over 20 runs:
   - Training set: 21.66333
   - Test set: 25.50301

1.3 Kernelised Ridge Regression

Question 5:

a. The optimal values are: $\gamma = 2^{-29}$, $\sigma = 2^{9.5}$

b. A 3D plot of the cross-validation MSE as a function of γ and σ:

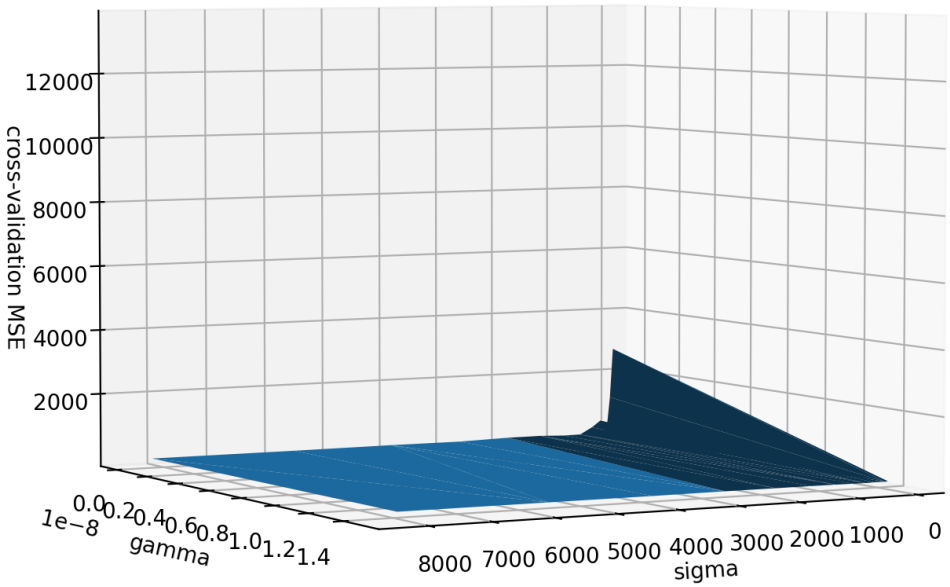

Figure 12. Cross-validation error varying with γ and σ

c. Using the best γ and σ, the MSEs are:
   - Training set: 8.87473
   - Test set: 11.50232

d. Table of the training and test errors of different models:

Method	MSE train	MSE test
Naive Regression	84.12±5.52	85.29±10.86
Linear Regression (attribute 1)	70.82±4.43	74.36±9.00
Linear Regression (attribute 2)	71.97±4.04	76.76±8.05
Linear Regression (attribute 3)	65.31±4.65	63.77±9.40
Linear Regression (attribute 4)	81.36±5.67	83.20±11.41
Linear Regression (attribute 5)	69.27±4.76	68.79±9.50
Linear Regression (attribute 6)	44.17±4.04	43.02±8.10
Linear Regression (attribute 7)	71.10±4.95	75.54±10.04
Linear Regression (attribute 8)	79.77±4.22	78.23±8.33
Linear Regression (attribute 9)	72.94±5.18	71.04±10.43
Linear Regression (attribute 10)	66.06±3.35	65.95±6.77
Linear Regression (attribute 11)	62.05±4.70	64.31±9.50
Linear Regression (attribute 12)	37.98±2.54	39.85±5.22
Linear Regression (all attributes)	21.66±2.50	25.50±5.40
Kernel Ridge Regression	8.94±1.62	12.65±1.79

Table 1. Training and test errors of different models

2 PART II

2.1 k-Nearest Neighbors

2.1.1 Generating the data

Question 6:

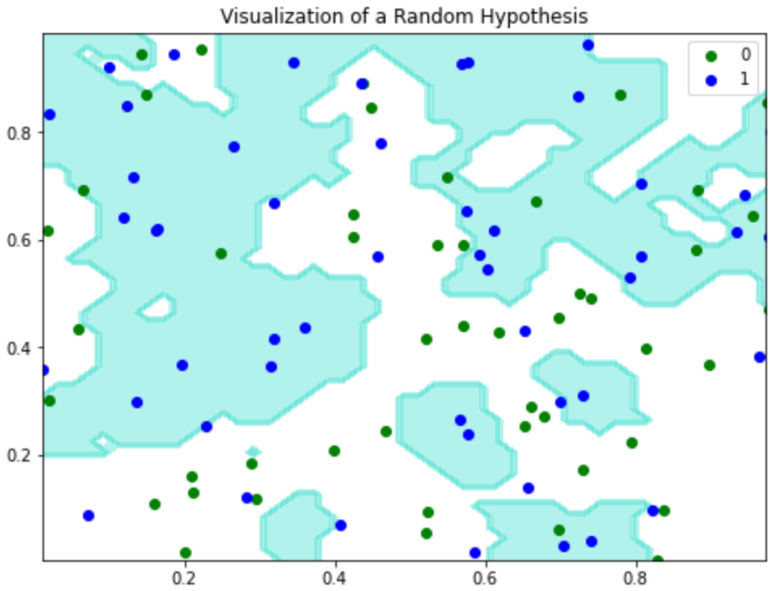

Figure 13. Visualization of a randomly sampled hypothesis

Question 7:

(a) Estimated generalization error as a function of k:

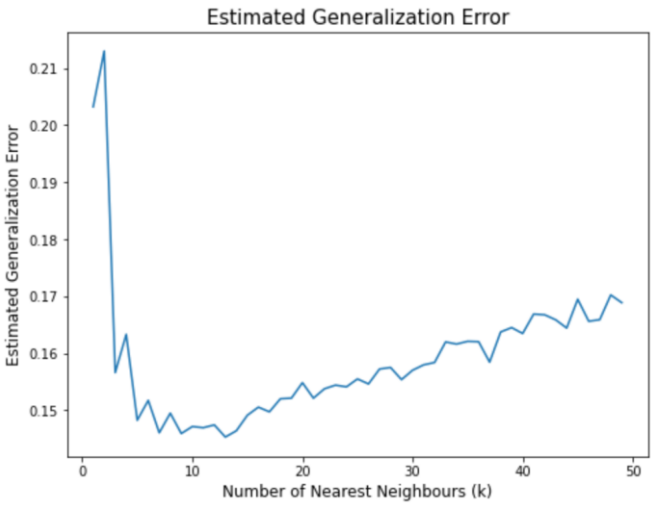

Figure 14. Estimated generalization error as a function of k

(b) The curve above is the approximate shape because both the hypothesis and the training and test sets are randomly sampled. There are infinitely many possible ways to do the sampling. Therefore, the curve can never exactly represents the generalization error of the model with different k values. Moreover, the performance of KNN models are highly dependent on the training data and the training data used in the 100 repeat trials to obtain the above curve were different. However, as the value of n (the number of repeat trails) becomes very large, the curve will be closer to the truth.
It can observed that the above curve roughly follows a U-shape where it decreases to reach the minimum error at some k value and then the error increases as k gets larger than this threshold value. This indicates that the KNN model starts to overfit the training data as too much nearest neighbours are being considered to make predictions.

Question 8:

(a) Optimal k as a function of the number of training points (m):

Figure 15. Optimal k as a function of the number of training points

(b) The curve above is the approximate shape because both the hypothesis and the training and test sets are randomly sampled. There are infinitely many possible ways to do the sampling, therefore, the curve can never exactly represents optimal k value for different amount of training data. In other words, it is impossible to determine the real optimal k value for a specific amount of training data (so we use 100 repeat trials to make estimations). Moreover, the performance of KNN models are highly dependent on the training data and the training data used in the 100 repeat trials to obtain the above curve were different, that is, we are varying the size of the training data when doing the experiment but we were unable to control other variables, for example, the training data are very likely to be not only different in size but also in other aspects as they are selected randomly.

3 PART III

3.1 Questions

Question 9:

(a) The function K_c can be written as: $K_c(x,z) := c + \sum_{i=1}^{n} x_i z_i = c + x^T z$. It is a polynomial kernel of the form $K(x,z) = (a + x^T z)^r$ with $r = 1$ and $a = c$ under the condition that $c \geq 0$. Therefore, K_c is a positive semidefinite kernel when $c \geq 0$. To prove this, we can show that the kernel matrix is semidefinite. For any vector $t \in R^n$, we have:

$$t^T K_c t = \sum_{i=1}^{n}\sum_{j=1}^{n} [K_c]_{ij} t_i\, t_j$$

$$= \sum_{i=1}^{n}\sum_{j=1}^{n} K_c(x_i, x_j) t_i\, t_j$$

$$= \sum_{i=1}^{n}\sum_{j=1}^{n} (\sum_{k=1}^{n} c + x_{ik} x_{jk}) t_i\, t_j$$

$$= \sum_{i=1}^{n}\sum_{j=1}^{n} \left(\sum_{k=1}^{n} x_{ik} x_{jk}\right) t_i\, t_j + c \sum_{i=1}^{n}\sum_{j=1}^{n} t_i\, t_j$$

$$= {K_c}^2 \left|\left|t\right|\right|^2 + c\left|\left|t\right|\right|^2 \geq 0 \quad if\ c \geq 0$$

Since the kernel matrix is positive semidefinite when $c \geq 0$, the function K_c is a positive semidefinite kernel when $c \geq 0$.

(b) When using K_c as a kernel function with $c \geq 0$, we have:

$$K_c(x,z) = < \emptyset(x), \emptyset(z) >$$

$$= \emptyset(x)^T \emptyset(z)$$

$$= c + \sum_{i=1}^{n} x_i z_i$$

where the basis function $\emptyset(x) = (\sqrt{c},\ x_1,\ x_2, \dots ,\ x_m)^T$.

The linear regression is given as: $y = w^T\ \emptyset(x) = w_0\sqrt{c} + \ w_1\ x_1 + \ \ w_2\ x_2 + \cdots + \ w_m\ x_m$

As we can see, the parameter c controls the size of the bias term of the model. Therefore, larger the value of c, larger the bias (prediction error) of the linear model.

Question 10:

Traditional KNN models combine the y-values of the k-nearest neighbours (to the input data points) with equal weights to make predictions. Kernel regression models, on the other hand, can be considered as using all the neighbours (all the training data) instead of selecting the k-nearest neighbours but with different weights. These weights are given by the kernel used in the model which measures the similarity (as opposed to distance) between data points. For example, in the Gaussian kernel:

$$K(x, t) = e^{-\frac{||x-t||^2}{\sigma^2}}$$

where $||x - t||^2$ is the distance between data points x and t, and σ is known as the width which determines how quickly the weights of neighbours drop as distances get larger.

A 1-Neareast Neighbour Classifier only considers the closest neighbour when making predictions. Therefore, in order to simulate a 1-Neareast Neighbour Classifier with the Gaussian kernel regression model, we want the to the weights of other neighbours to be minimal and ideally zero. That is, we want to have the width σ as close to zero as possible so that the Gaussian kernel will give the closest neighbour a huge weight compare to others.

However, the width σ cannot be zero exactly as it will lead to the problem of underflow (negative overflow of the exponential Gaussian kernel value). Therefore, we aim to have $\sigma \rightarrow 0$ under the condition that underflow does not occur. This is equivalent of having $\beta \rightarrow \infty$ under the condition that underflow does not occur. In other words, we maximize β.

Question 11:

We can transform the problem into solving linear equations.
Let's consider playing the game on a 2×2 board with any random initial configuration, for example:

$$\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$$

where a 1 represents a hole with a risen model and a 0 represents a hole with a hidden model. This can be think of as a state matrix of 0s and 1s that can represent the configuration of the game. A more generalized state matrix for the above configuration is given as:

$$S = \begin{bmatrix} S_1 & S_2 \\ S_3 & S_4 \end{bmatrix} \text{ with } \begin{cases} S_1 = 1 \\ S_2 = 0 \\ S_3 = 1 \\ S_4 = 1 \end{cases}$$

Now, we whack the holes. We can think of whacking the holes even times as not changing the states of the hole and its immediate 4 adjacent holes and whacking the holes odd times is equivalent to whacking it only once. This is because a repeat whack will just restore whatever it was before the last whack. Therefore, we know that we will either only whack a hole once or not whack it at all. Let's use an action matrix to represent our whacking strategy:

$$X = \begin{bmatrix} x_1 & x_2 \\ x_3 & x_4 \end{bmatrix}$$

where each element will take the value of either 1 (whack once) or 0 (not at all). For example, to clear out the above configuration, the action matrix will be:

$$\begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}$$

In the above case:

- Actions x_1, x_2, x_3 will affect the state S_1
- Actions x_1, x_2, x_4 will affect the state S_2
- Actions x_1, x_3, x_4 will affect the state S_3
- Actions x_2, x_3, x_4 will affect the state S_4

We can store this information in an effect matrix A of dimensions $(n^2 \times n^2)$ where its i^{th} row indicates the actions that affects the state S_i. The effect matrix for the above configuration is:

$$A = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}$$

To clear out the state $S_1 = 1$, we want the actions x_1, x_2, x_3 to fulfill $x_1 + x_2 + x_3 = 1$. Similarly, we want $x_1 + x_2 + x_4 = 0$ to leave the state $S_2 = 0$ unchange. This applies to all the states. Therefore, we transforms the problem into solving the following linear equations:

$$\begin{cases} x_1 + x_2 + x_3 = S_1 = 1 \\ x_1 + x_2 + x_4 = S_2 = 0 \\ x_1 + x_3 + x_4 = S_3 = 1 \\ x_2 + x_3 + x_4 = S_4 = 1 \end{cases}$$

This is equivalent to solving the matrix equation:

$$Ax = S$$

where A is the effect matrix, x is the action vector $(x_1, x_2, \dots x_{n^2})^T$ and S is the state vector $(S_1, S_2, \dots S_{n^2})^T$.

To do so, we need an invertible matrix A. Since a game of *Whack-A-Mole!* always have a solution, we can assume that the matrix A is always invertible. Now we have the solution to the problem, we need to consider the time complexity of this algorithm.

Firstly, we need to compute the effect matrix A which has n^4 elements and therefore takes time $O(n^4)$. Then, we perform Gaussian elimination to solve the linear system which takes time $O(n^6)$. This step dominates the runtime and therefore the total time complexity of the algorithm is $O(n^6)$ which is polynomial.