Intuition:

The problem aims to calculate the averages of subarrays within a given array, where each subarray has a fixed radius of k (i.e., a subarray of length 2k + 1). The goal is to efficiently compute these averages for each subarray.

Approach 1: Prefix Sum

1. First, it checks if the size of the input list nums is too small to form a valid subarray. If it is, the code returns a list of -1s as there are not enough numbers to calculate the averages.
2. If the input list is large enough, the code proceeds to calculate the averages. It creates a new list called ans with the same size as nums and fills it with -1s. This list will store the calculated averages.
3. The code then creates another list called prefixSum to help efficiently calculate the sum of subarrays. This list will store the cumulative sums of the numbers in nums.
4. It goes through each number in nums and calculates the cumulative sum up to that point. It does this by adding the current number to the sum of all previous numbers. The cumulative sums are stored in the prefixSum list.
5. Next, the code enters a loop that goes through the indices of nums, starting from k and ending at n - k - 1, where n is the size of nums. These indices represent the starting positions of the subarrays for which we want to calculate the averages.
6. For each index i, the code calculates the sum of the subarray by subtracting the prefix sum at index i - k from the prefix sum at index i + k + 1. This gives us the sum of all the numbers within the subarray.
7. After obtaining the sum of the subarray, the code divides it by the size of the subarray (2k + 1) to calculate the average. This average represents the average value of the numbers within the subarray.
8. The code stores the calculated average in the ans list at the corresponding index i. This way, each element in the ans list represents the average value of the subarray starting at that index.
9. Once the loop completes, the code has calculated the averages for all valid subarrays. It returns the ans list, which contains the computed averages.

Approach 2: Sliding Window

1. It starts by initializing variables, such as the size of the input list nums (n), the window size (2k + 1), a variable windowSum to track the sum of the current window, and a list called result to store the calculated averages.
2. It checks if the size of the input list nums is too small to form a valid subarray. If it is, the code returns the initialized result list filled with -1s.
3. The code enters a loop that iterates over each index i in the list nums.
4. Inside the loop, it updates the windowSum by adding the current number nums[i] to it. This step represents adding the new number to the current window.
5. The code checks if the current index i minus the window size (i - windowSize) is greater than or equal to zero. If this condition is met, it means the current window has moved beyond the first window (starting at index 0). In this case, the code subtracts the number at nums[i - windowSize] from the windowSum to remove it from the window. This step represents removing the number that is no longer part of the window.
6. The code checks if the current index i is greater than or equal to windowSize - 1. If this condition is met, it means the current window has reached the desired size. In this case, the code calculates the average of the window by dividing the windowSum by the window size (windowSum / windowSize). It then stores the calculated average in the result list at the corresponding index i - k, where k represents half the window size. This step represents storing the average value in the result list.
7. Once the loop completes, the code has calculated the averages for all valid subarrays. It returns the result list, which contains the computed averages.