Exercises 07 and 08

Kevin Heubacher (15329079), Akshay Chikhalkar (15489036),
John Paul Semakula (15487087)
Group: LATE

6.12.2022

Exercise 7.3

The aim of this exercise is to benchmark Multiple - Choice Knapsack Problem with different items with their values and objects for multiple iterations.
The multiple-choice knapsack problem is a generalization of the ordinary knapsack problem, where the set of items is partitioned into classes. The binary choice of taking an item is replaced by the selection of exactly one item out of each class of items.

Mathematical formulation:

$$\text{maximize} \sum_{i=1}^{k} \sum_{j \in N_i} v_{ij} x_{ij} \text{ subject to } \sum_{i=1}^{k} \sum_{j \in N_i} w_{ij} x_{ij} \leq limit$$

$$\text{where} \sum_{j \in N_i} x_{ij} = 1, i = 1, ..., k, x_{ij} \in \{0, 1\}, i = 1, ...k, j \in N_i$$

v - Item value, w - Item weight, k - No. of items, $limit$ - Weight limit
x_i - The number of instances of item i to include in the knapsack, $j \in N_i$

Steps:

1. Items generator: This function generates a given number of items with a random value between 1 to 20 and a weight between 1 to 10.
2. Recording execution time for the multiple iterations.
3. Calculating the mean of a recorded time set (i.e. Set of n).
4. Calculating variance of a recorded time set. (i.e. Set of n).
5. Visual representation of time vs iteration.

The following table represents the benchmarking results:

Sr.	**Items**	**Iteration**	**Mean**	**std**
1	50	50	0.18	1.00
2	50	60	0.18	2.57
3	50	70	0.18	5.21
4	50	80	0.18	6.05
5	50	90	0.18	4.13
6	100	50	0.72	3.69
7	100	60	0.72	2.51
8	100	70	0.77	4.27
9	100	80	0.76	9.77
10	100	90	0.75	1.94
11	150	50	1.68	2.23
12	150	60	1.67	2.24
13	150	70	1.66	2.79
14	150	80	1.66	2.32
15	150	90	1.65	2.48
16	200	50	2.97	2.53
17	200	60	2.96	2.98
18	200	70	2.99	4.74
19	200	80	2.96	2.33
20	200	90	2.97	2.69
21	250	50	4.69	3.17
22	250	60	4.66	4.89
23	250	70	4.66	6.04
24	250	80	4.67	1.53
25	250	90	4.65	7.23

Table 1: Benchmarking results

From the results, we can infer that the execution time depends on the number of items. In the graph plot of Mean and Variance, the mean execution time increases as the number of items increases and the variance is almost the same all over the result.

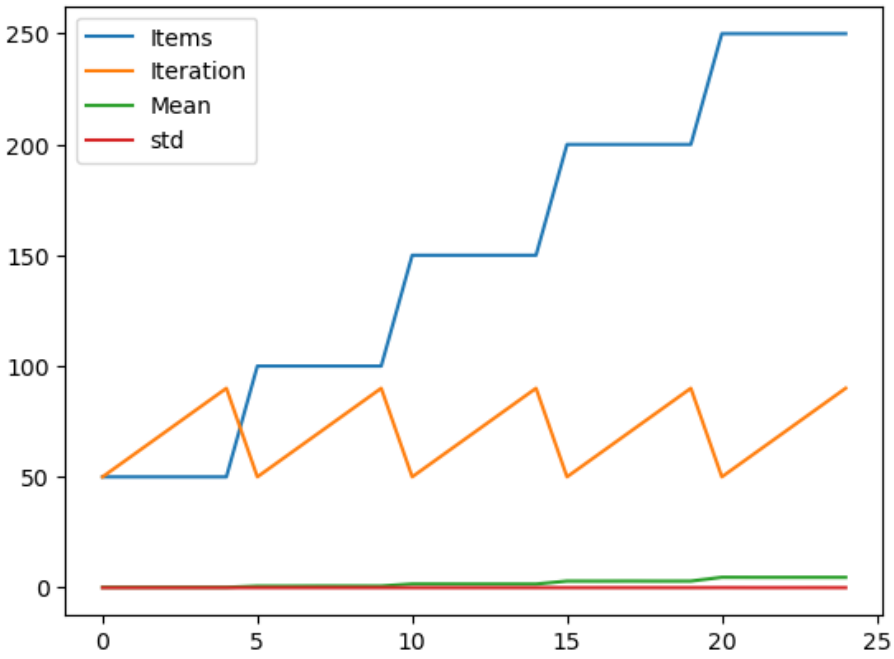

Figure 1: Items, Iteration, Mean and Variance Plot

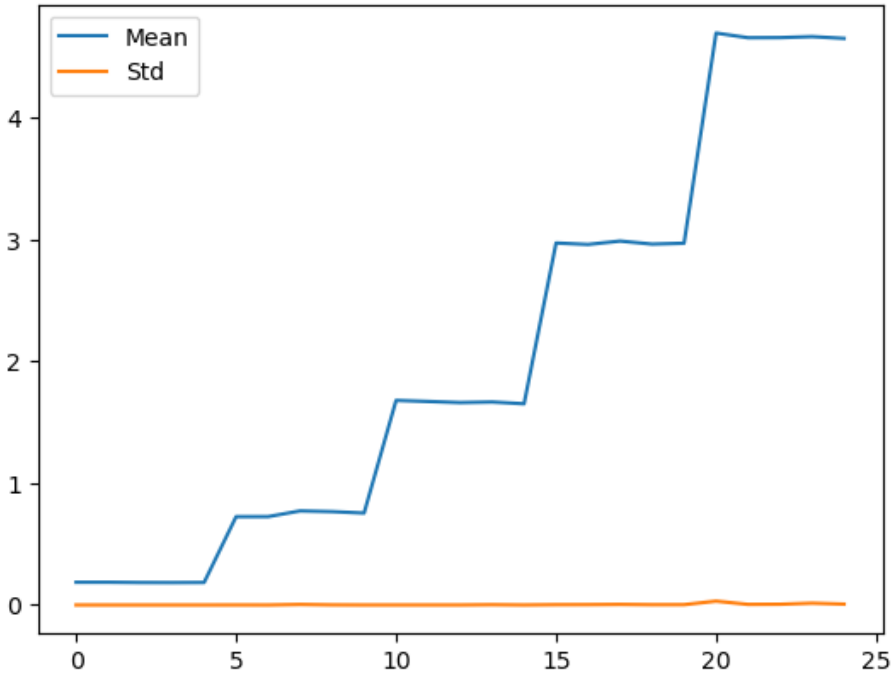

Figure 2: Mean and Variance Plot

Exercise 8.1

Implementation of Super Simple using Keras

Keras is an open-source deep-learning API written in python. It supports multiple backend machine learning platforms such as TensorFlow and Theano

Since it supports many machine learning platforms it is easy to implement neural networks.

Super simple is basically a binary classification algorithm that uses neural networks.
Below is the comparison of Keras's super simple implementation model using Keras with Professor's implemented model using Mean Squared Error Values using the same epochs and batch Size values.
2

Results		
	Keras implementation	Professor's Implementation
MSE(batch-size =100,epochs = 1500)	0.000072	0.00083306

Table 2: Shows the two Mean Squared Error Values.

Conclusion.
Super simple Keras implementation shows a low mean squared error value on the data set provided compared to Professor's super simple implementation but this all depends on the number of layers and the number of neurons used per layer.

Exercise 8.2

KerasTuner is a hyperparameter optimization framework. It includes Bayesian Optimization, Hyperband, and Random Search optimization algorithms. In this exercise, we use KerasTuner to determine the optimal number feature space dimensions. We are using the Bayesian Optimization algorithm. The optimizer searches between 32 and 512 feature space dimensions in steps of 32 for the highest validation accuracy. The optimization results are shown in table 3.

Version	**Feature Space Dimensions**	**Accuracy**
Original	256	0.8822
KerasTuner	512	0.8918

Table 3: Comparison of original against KerasTuner implementation

While the accuracy improved on paper, when comparing the classification result images it appears to be worse. In the bottom left of figure 4 you can see that the misclassification rate of the sneaker is much higher. The classification of all other images only changed marginally, if at all.

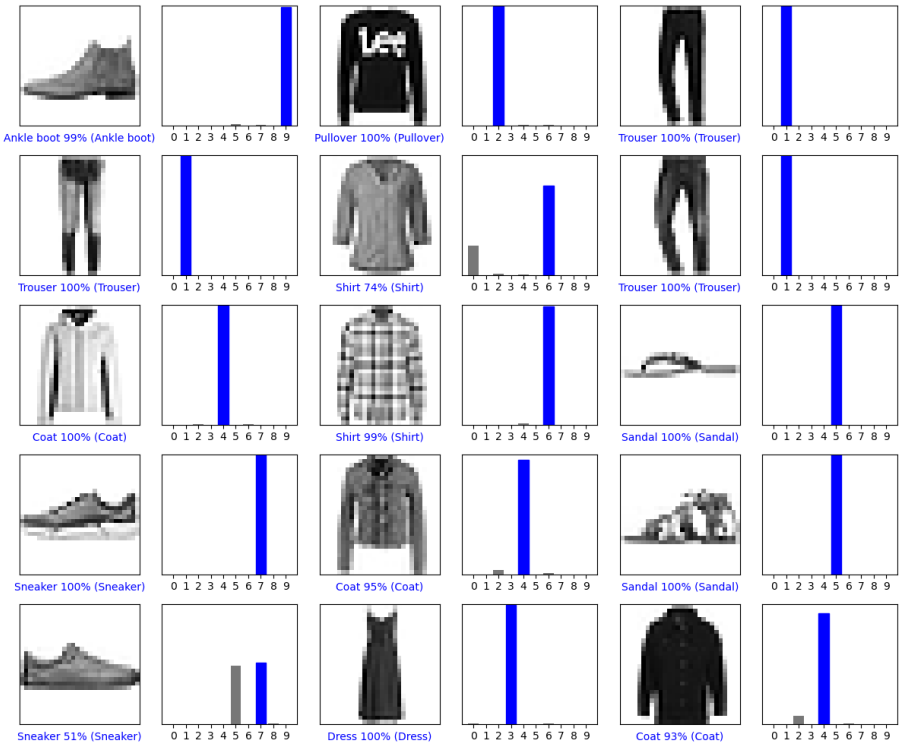

Figure 3: Original Results

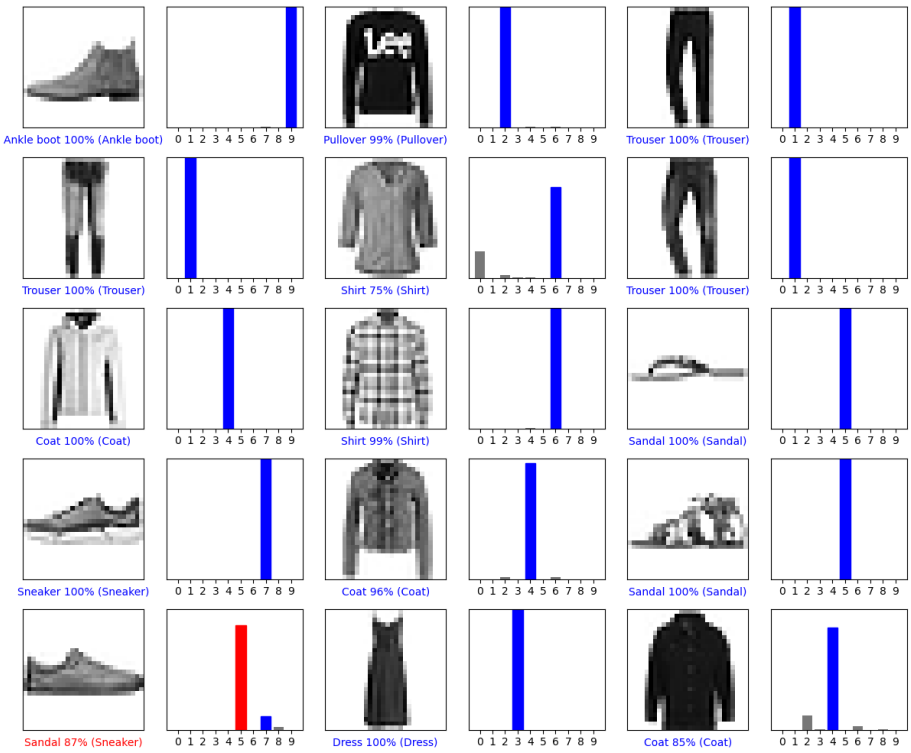

Figure 4: KerasTuner Results