

Fixing Neural Networks by Identifying Top-k Shapley Players

Tim Nielen

September 17, 2023



Department of Mathematics,
Informatics and Statistics
Institute of Informatics



Artificial Intelligence and
Machine Learning

Bachelor's Thesis

Fixing Neural Networks by Identifying Top-k Shapley Players

Tim Nielen

Reviewer Prof. Dr. Eyke Hüllermeier
Institute of Informatics
LMU Munich

Supervisor Patrick Kolpaczki

September 17, 2023



Tim Nielen

Fixing Neural Networks by Identifying Top- k Shapley Players

Bachelor's Thesis, September 17, 2023

Reviewer: Prof. Dr. Eyke Hüllermeier

Supervisor: Patrick Kolpaczki

LMU Munich

Department of Mathematics, Informatics and Statistics

Institute of Informatics

Artificial Intelligence and Machine Learning (AIML)

Akademiestraße 7

80799 Munich

Abstract

The Shapley value is arguably the most popular solution concept in cooperative game theory being utilized in a wide-ranging variety of applications in the field of explainable artificial intelligence. Given a set of players that are able to cooperate by forming groups to perform a certain task, it is used to provide a fair reward distribution representing each player's individual importance for the task. Due to its exponential computational complexity we rely on algorithms approximating the Shapley value given a fixed budget. When applying Shapley values to the structure of a neural network in order to identify neurons responsible for a certain bias for example, we are more interested in the set of the top-k most important players rather than approximating the exact Shapley values of all neurons. Besides evaluating algorithms proposed in the literature, we introduce our own ideas for the top-k Shapley player problem. Using one of our algorithms we are able to show a relevant difference between the approximation and the top-k problem that allows us to achieve better results than other algorithms that are utilizing their budget similarly efficient. We are able to identify highly important neurons in a neural network using a budget several magnitudes lower than the current state of the art.

Contents

1	Introduction	1
2	Fundamentals	5
2.1	Cooperative Games	5
2.2	The Shapley Value	5
2.3	Problem of Identifying Top-k Shapley Players	8
2.4	Neural Networks	10
2.4.1	Feature Importance	12
2.4.2	Neuron Shapley	13
3	Algorithms	15
3.1	ApproShapley	16
3.2	SVARM and Stratified SVARM	18
3.3	Border Uncertainty Sampling	20
3.4	Truncated Multi Armed Bandit Shapley	22
3.5	HybridApproBUS with Caching	24
3.6	HalfBUS	26
4	Evaluation	31
4.1	Synthetic Cooperative Games	31
4.1.1	Experiment setup	31
4.1.2	Results	34
4.2	Measuring Pixel Importance	40
4.2.1	Experiment setup	41
4.2.2	Results	42
4.3	Neuron Shapley	43
4.3.1	Experiment setup	43
4.3.2	Results	45
5	Conclusion and future work	49
Bibliography		51
List of Figures		55

Introduction

Neural networks (NNs), a class of artificial intelligence (AI) models inspired by the human brain, have revolutionized the field of machine learning and data analysis. Their ability to learn from large amounts of data and to make complex predictions is useful in a variety of different applications, such as image detection, healthcare, text generation and even stock market forecasting [KSH12; Mio+18; Bro+20; Sel+17]. Recently, e.g. large language models, such as OpenAI's GPT models [Bro+20] based on Transformers [Vas+17] are receiving a great amount of attention as they are capable of producing text indistinguishable from human written text and are able to act as a useful assistant in almost every domain. Furthermore, Convolutional Neural Networks (CNNs) have become the state-of-the-art in most image processing applications [LBH15; KSH12]. The size and complexity of modern models increases consistently, with some models containing more than six billion parameters [Bro+20].

However, NNs can often appear as black boxes to the human eye, with little to no transparency about their inner mechanisms as the interactions between neurons can seem arbitrary and are hard to track. Even though the mathematics for evaluating a single neuron are rather simple, the enormous number of neurons and connections make it very difficult to comprehend the outcome of a model. Therefore, even though NNs achieve far better results than their specialized competitors in many applications [Bro+20], explaining their decisions poses a quite challenging task. A bank for example, might use a NN to decide whether a client receives a credit based on properties such as income, age, and profession. If the credit gets declined, the bank is required to explain why this decision was made which is challenging in the case of a NN. A neural network's precision and versatility immensely depend on the training data it receives. Similar to how a child primarily acquires knowledge from its parents and often adopts their viewpoints, a neural network trained on biased data is highly likely to produce biased predictions as well. Therefore, it is really important to choose the training data carefully. However, as more sophisticated machine learning models emerge, they often demand an increasing amount of data for effective training. Otherwise, models with a large amount of parameters tend to overfit to the training data, basically memorizing each prediction. This phenomenon

has a bad impact on the versatility of a model.

GPT-3 for example, is trained on a dataset consisting of about 500 billion tokens [Bro+20], where each token represents roughly half a word. For such huge datasets it is practically infeasible to examine all of it. As a result, quantity often outweighs quality in practical applications.

Howard et al. [HB18] present a variety of real examples of biased AI models. In 2015 for example, Google released a face recognition application that mislabeled black people as "gorillas". A reason for that might have been the training data originating from mostly European countries where people of color are underrepresented. Biases in the real world against minorities for example are often transferred directly to the internet. Large language models, such as OpenAI's GPT models [Bro+20], often suffer from this problem as they are directly trained on large chunks of data available on the internet. Brown et al. [Bro+20] state themselves that GPT-3 has certain biases in gender, race and religion. Popular prejudices about the Islam for example are adopted by GPT-3 with words like "violent", "terrorism" and "terrorist" occurring more often with Islam than with other religions.

Even though we are able to observe such biases, discovering how they are created is practically infeasible from inspecting the network architecture with just the human eye. This problem incentivizes research on methods that make the black box problem more transparent.

The field of *Explainable Artificial Intelligence* (XAI) aims to understand and address such behaviors. The Shapley value [Sha+53], a solution concept from cooperative game theory, is a useful tool for identifying biases in ML models. Many XAI problems can be reformulated as so called cooperative games. In these games players form arbitrary coalitions to achieve a collective benefit. The Shapley value of a player represents its importance for the respective game. Shapley's [Sha+53] solution does not require any knowledge about how the value of a coalition is calculated which is ideal for black box ML models. Intuitively this works by adding and removing a player from a coalition and inspecting the change in worth. The worth of all players combined is then distributed among all players according to their average contribution to any coalition. For measuring local feature importance, i.e. the importance of each feature for a single prediction, the input features of a model are treated as players [Roz+22] and coalitions are evaluated by running the model with subsets of features. For example, in the case of declined credit, the Shapley values of the client's properties could be calculated to determine which ones had the most impact on the decision. This way, a previous gambling addiction, for example, could be identified as the main cause of the rejection. In addition, when looking at global feature importance it is possible to determine how much weight a model

gives to a feature in general by computing Shapley values using the whole dataset [Mol20]. This might be interesting to verify if a client's income is the main aspect for decisions made by a credit algorithm.

Ghorbani et al. [GZ20] propose treating the neurons of a NN as players where the worth of a coalition of players is represented by the network's accuracy when only this subset of neurons is used. This approach allows for ranking the importance of all the neural network's neurons by their Shapley values. Neurons with a negative Shapley value would predominantly contribute negatively to the model's performance and could be removed completely. Another use case is when a model's bias is known. By evaluating the Shapley values on a specialized dataset, the neurons responsible for this bias can be identified and eliminated. Ghorbani et al. [GZ20] demonstrate that this method significantly reduces a model's bias.

In practice, we are unable to calculate the exact Shapley value in most usecases, as the number of coalitions that have to be evaluated scales exponentially with the number of players. Therefore we rely on algorithms that approximate the value with as little evaluated coalitions as possible [CGT09; KBH23]. For many applications such as determining neuron importance, it is sufficient to find the top-k players with the highest Shapley values as defined by Kolpaczki et al. [KBH21]. This problem statement has fewer requirements compared to the approximation of the Shapley value for all players. Ghorbani et al. [GZ20] employed a top-k algorithm to identify the most important neurons in a neural network.

The aim of this thesis is to evaluate existing top-k and approximation algorithms on the top-k problem and to discover additional top-k Shapley player algorithms with the aim of applying them in XAI tasks similar to the work of Ghorbani et al. [GZ20].

We begin with fundamental knowledge about cooperative games, the Shapley value and its top-k variant in chapter 2. In addition we give a short overview about Neural Networks and how the Shapley value can be applied in certain XAI tasks. In chapter 3 we describe algorithms by different authors as well as our own ideas which we all compare on the top-k problem later. In chapter 4 we evaluate the algorithms both on synthetic aswell as real data. For each, we provide an experiment setup were we explain how the underlying cooperative games are constructed. We conclude with chapter 5, summing up the most important findings of the experiments and providing directions for possible further research.

Fundamentals

2.1 Cooperative Games

The cooperative game is a concept from game theory where players form arbitrary groups, so called coalitions, to achieve a collective reward.

Definition 2.1.1 (Cooperative game). Given a player set $N = \{1, \dots, n\}$ with $n \in \mathbb{N}$ and a value function $v : \mathcal{P}(N) \rightarrow \mathbb{R}$ with $v(\emptyset) = 0$, the tuple (N, v) denotes an n -player cooperative game.

The coalition formed by all players is often referred to as the grand coalition. Consider the scenario of a class of 20 students that are working jointly in a project. They can form arbitrary groups to complete this task. Here, each student represents a player. The value function is the grade or the number of credits a group of students would receive for their project.

2.2 The Shapley Value

Looking at the classroom setting, students have to be graded separately as it is common that some students contribute more to the task than others which makes it unfair to grade all students equally.

In cooperative games the challenge is to find a fair distribution $(\phi_i)_{i \in N}$ of real values that assigns a numerical payoff to each player i representing their importance to the game without knowing how the value function is calculated. I.e. in our classroom the teacher would have to assess the outcome of the project without monitoring the progress of the group.

The Shapley value is a solution concept for each player's payoff in a cooperative game [Sha+53].

Definition 2.2.1 (Shapley value). Given a cooperative game (N, v) the Shapley value of a player $i \in N$ is defined as:

$$\phi_i = \sum_{S \subseteq N \setminus \{i\}} \frac{1}{n \binom{n-1}{|S|}} [v(S \cup \{i\}) - v(S)]. \quad (2.1)$$

Intuitively, it represents a player's average marginal contribution. The marginal contribution $v(S \cup \{i\}) - v(S)$ of a player i is the difference in worth of a coalition S that does not contain the player compared to when the player is added.

The Shapley value is provably the only solution that guarantees a certain set of axioms which arguably capture fairness [KBH23]. These four axioms are:

- Efficiency: The worth of the grand coalition, the coalition that is formed by all players combined, is shared among all players:

$$\sum_{i \in N} \phi_i = v(N).$$

- Symmetry: The payoff of two players $i, j \in N$ should be the same if they contribute equally to every coalition:

$$(\forall S \subseteq N \setminus \{i, j\} : v(S \cup \{i\}) = v(S \cup \{j\})) \implies \phi_i = \phi_j.$$

- Dummy player: A player $i \in N$ that does not contribute to any coalition has a payoff of zero:

$$(\forall S \subseteq N \setminus \{i\} : v(S \cup \{i\}) = v(S)) \implies \phi_i = 0.$$

- Additivity: If the cooperative game can be split up into subgames, i.e. $v = v_1 + v_2$ with $v, v_1, v_2 : \mathcal{P}(N) \rightarrow \mathbb{R}$, the payoff of each player i in the combined game should be the sum of its payoff in the respective subgames:

$$\phi_i^v = \phi_i^{v_1} + \phi_i^{v_2}.$$

A very basic example of a cooperative game would be a game where each coalition $S \subseteq N = \{1, \dots, n\}$ receives its length as its value i.e. $v(S) = |S|$. Finding the

Shapley values for this game is a trivial task as each player contributes equally to the length of the coalition. As a consequence $\phi_i = \frac{v(N)}{n} = 1$ for all $i \in N$. Cooperative games that are formulated in such a way that the Shapley values are immediately apparent or can be calculated in polynomial time are called synthetic cooperative games.

However, this is not the case for most applications. In fact, from eq. (2.1) it is easy to see that in general in order to calculate the Shapley value for all n players the value function would have to be evaluated for every subset of N resulting in $|\mathcal{P}(N)| = 2^n$ calculations making it an NP-hard problem [KBH23]. This poses an inevitable burden to the application of the Shapley value in XAI. Consider an image detection algorithm, identifying 32×32 pixel images. In order to determine which details of the image the algorithm focused on, one could use the loss function of the model as a value function and consider each pixel as a player and calculate the respective Shapley values. For 32×32 pixel images this would be 1024 players with $2^{1024} \approx 10^{308}$ possible coalitions. The model would have to be run and the loss function calculated for each of these coalitions. Obviously, this is a practically infeasible task.

Because of this drawback, we have to rely on algorithms approximating the Shapley value. In order to estimate a large sum as in eq. (2.1) one can treat the set of all coalitions as possible realizations of a random variable with the respective marginal contributions as values and approximate the expected value of that random variable. Given a player $i \in N$, let X denote a random variable with values $(x_S)_{S \subseteq N \setminus \{i\}}$ where $x_S = v(S \cup \{i\}) - v(S)$ and probability distribution $P(S) = \frac{1}{n \binom{n-1}{|S|}}$. Then the expected value of X exactly matches the player's Shapley value, i.e.:

$$E[X] = \sum_{S \subseteq N \setminus \{i\}} x_S \cdot P(S) = \phi_i. \quad (2.2)$$

Therefore, in order to approximate the Shapley value of a player i we can approximate the expected value of X instead. For a total of m samples $(S_j)_{j \in \{1, \dots, m\}}$ with each $S_j \subseteq N \setminus \{i\}$ we get an estimator for our Shapley value:

$$\hat{\phi}_i = \frac{1}{m} \sum_{j=1}^m x_{S_j}. \quad (2.3)$$

Approximation algorithms are mostly based on this concept of sampling coalitions but often use a transformed version of eq. (2.1) in order to optimize the sampling process and obtain an accurate estimator while calling the value function as little as possible.

2.3 Problem of Identifying Top-k Shapley Players

For some applications it is not necessary to get reasonably precise approximations for every player. It might be interesting enough to know which players are the most important. For example, when trying to reduce the size of a neural network one would want to know the k most important neurons in order to retain them while dropping the rest [GZ20]. Kolpaczki et al. [KBH21] define the Top-k Shapley problem:

Definition 2.3.1 (Top-k Shapley problem). Given a cooperative game (N, v) and $k \in N$ find a set $K := \{i \in N \mid \phi_i \geq \phi_{\pi(k)}\}$ for a permutation $\pi: N \rightarrow N$ such that $\phi_{\pi(1)} \geq \dots \geq \phi_{\pi(n)}$ while evaluating $v(S)$ for any coalition $S \subseteq N$ at most $T \in \mathbb{N}$ times.

K contains the players with the k highest Shapley values.

With this formulation of the problem approximating all Shapley values accurately becomes less important. This allows for a lot more optimization as the sampling procedure can shift the focus to a smaller subset of players that are highly uncertain to be part of the top-k players while players with very high or very low Shapley values can be classified rather quickly [KBH21].

Like Kolpaczki et al. [KBH21] we focus on a fixed budget scenario where the performance of an algorithm is measured by its ability to identify the top-k players given a budget $T \in \mathbb{N}$. T is the number of times the algorithm is allowed to call the value function as in most applications the value function is the bottleneck. In XAI tasks for example, the value function often includes an evaluation of the AI model which is usually quite expensive to compute.

In addition we need to establish metrics that measure an algorithm's performance. Using synthetic cooperative games that allow us to calculate the exact Shapley values in polynomial time, we can compare the algorithm's estimation with the actual value.

For approximation algorithms a common performance metric is the *mean squared error* (MSE) where the the squared differences between the approximation and the actual Shapley values of all players are averaged:

$$MSE = \frac{1}{n} \sum_{i \in N} (\phi_i - \hat{\phi}_i)^2.$$

However, this metric does not provide any information on the performance on the top-k problem because an algorithm does not need to approximate all Shapley values precisely but find the players with the highest values. In theory an algorithm does not even need to return the approximated Shapley values. But since all of our algorithms internally work with approximations of the values we keep the MSE to compare an algorithm's performance on the top-k problem and the approximation problem.

We define the following performance metrics for the top-k problem. Using a permutation $\pi : N \rightarrow N$ such that $\phi_{\pi(1)} \geq \dots \geq \phi_{\pi(n)}$, $K := \{i \in N \mid \phi_i \geq \phi_{\pi(k)}\}$ denotes the set of players with the k highest Shapley values. We propose two measures:

$$\psi(\hat{K}) = \begin{cases} 1, & \text{if } \hat{K} \subseteq K \\ 0, & \text{otherwise} \end{cases} \quad (2.4)$$

$$\varphi(\hat{K}) = \frac{|\hat{K} \cap K|}{|\hat{K}|} \quad (2.5)$$

Equations (2.4) and (2.5) formulate a nominal scale ψ and a ratio scale φ for an algorithm's estimation \hat{K} of the top-k players with $|\hat{K}| = k$. Similar to Kolpaczki et al.'s evaluation [KBH21] ψ returns one if the algorithm delivered a valid set of k players and zero if it did not. φ returns the ratio of players that the algorithm correctly classified as part of the top-k players.

For most of our evaluations we use the ratio scale as it has a continuous range of values, making it easier to differentiate multiple results.

Example 2.3.1 (The advantage of the ratio scale). Let $K = \{1, 2, 3, 4\}$. Suppose we have two algorithms approximating K with results $\hat{K}_1 = \{1, 2, 3, 5\}$ and $\hat{K}_2 = \{5, 6, 7, 8\}$. Intuitively, the result of algorithm 1 is better than the result of algorithm 2 as it has found most of the top-k players. However, the performance of both algorithms would be measured equal using the nominal scale $\psi(\hat{K}_1) = \psi(\hat{K}_2) = 0$ as they both have not found the correct set. The ratio scale differentiates between the results returning $\varphi(\hat{K}_1) = 3/4$ for algorithm 1 and $\varphi(\hat{K}_2) = 0$ for algorithm 2.

In addition to calculating the performance metric of an algorithm we have to run the experiment multiple times and average the results over all experiments since

the sampling process in most algorithms is randomized and does not always deliver the same result. In order to determine if we conducted enough experiments we calculate the standard error of our averaged result which estimates how confident the result is. Given a vector of $m \in \mathbb{N}$ results $(x_i)_{i=1}^m$, the sample variance [@Wei03] of the algorithm's performance is calculated using

$$\hat{\sigma}^2 = \frac{1}{m-1} \sum_{i=1}^m (x_i - \bar{x})^2, \quad (2.6)$$

where $\bar{x} = \frac{1}{m} \sum_{i=1}^m x_i$ is the average result. In case of the ratio scale we use $x_i = \varphi(K_i)$. The standard error (SE) is then calculated using

$$SE = \frac{\hat{\sigma}}{\sqrt{m}}.$$

2.4 Neural Networks

In the following section we give a short explanation of Neural networks (NNs) and how they work. A more in depth explanation can be found in Charu C. Aggarwal's *Neural Networks and Deep Learning: A Textbook* [Agg+18].

Neural networks are machine learning models that are designed to work similar to the human brain in order to accomplish tasks that are too complex to be explicitly programmed using traditional rule-based methods. Instead, a NN learns patterns and relationships within data through a training process.

Modern NNs achieve astonishing results: Brown et al's language model GPT-3 [Bro+20] in its largest version is able to produce news articles nearly indistinguishable from articles written by humans. The accuracy of humans classifying these articles is close to chance. Image generation models, such as stable diffusion [@AI22] and DALL-E 2 [Ram+22], are capable of producing realistic images from simple text prompts as shown in fig. 2.1.

A NN is structured into layers of nodes, called neurons, a reference to the human brain. Each neuron takes input signals, performs a computation, and produces an output signal that is passed on to the next layer.

The first layer of the network, the input layer, receives the initial data. This data could be anything, ranging from text or sound waves up to image data, as long as it can be serialized into a vector of real numbers. The last layer is the output layer,



(a) a propaganda poster depicting a cat dressed as French Emperor Napoleon holding a piece of cheese
 (b) a teddy bear on a skateboard in Times Square

Fig. 2.1: Samples generated by DALL-E 2 [Ram+22] and the corresponding prompts

which produces the final prediction or output of the network. Between these two layers, there can be one or more hidden layers, which extract and transform features from the input data.

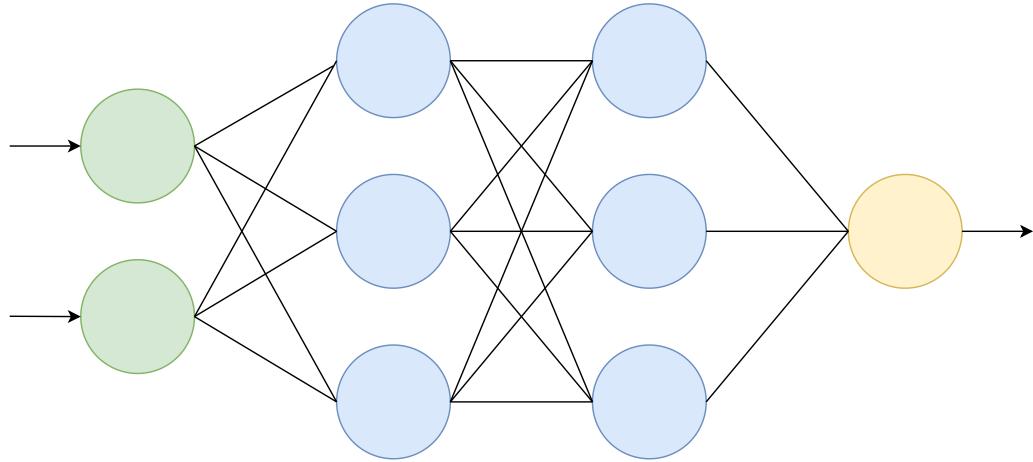


Fig. 2.2: Simplified illustration of a neural network: input layer (green), hidden layers (blue), output layer (yellow)

Figure 2.2 illustrates the structure of a neural network. Each connection between neurons is assigned a weight w_i . A neuron receives the output of all d neurons of the previous layer $(x_i)_{i=1}^d$ in order to produce its output. Different types of NNs or rather NN layers perform different types of operations on its input. CNNs for example pass data through multiple layers of so called filters in order to identify certain patterns,

in images mainly. In a simple linear layer however, the output of a neuron is a linear combination of its inputs and the corresponding weights summed with an invariant bias b :

$$\sum_{i=1}^d w_i x_i + b.$$

An activation function, such as the sign function:

$$sign(x) = \begin{cases} -1 & \text{if } x < 0, \\ 0 & \text{if } x = 0, \\ 1 & \text{if } x > 0 \end{cases}$$

is applied to the output of a layer in order to achieve non-linear results. This is particularly useful for e.g. classification tasks where the data is not linearly separable into two classes [Agg+18].

NNs learn by adjusting the weights of connections between neurons. During training, the network is presented with input data along with the correct outputs. The output of the NN for a single data point based on the current weights is compared to the true label, measuring the loss. The weights are then updated using an optimization algorithm, such as gradient descent, that tries to minimize the loss.

2.4.1 Feature Importance

Going back to the "black box" problem of neural networks we can use Shapley values to calculate the importance of input features for a machine learning model both for a single prediction or for the whole dataset [Mol20]. This can help understand what the major reasons for the model's decision are. For image detection models for example, this enables us to identify the areas of the image that were most influential for the algorithm [LL17]. When applying Shapley values on feature importance, the input features form the set of players, and the performance of the model is used as the value function. Of course, in order to calculate the value function for subsets of the player set, we need to remove input features. Simply removing the input neuron of a feature or fixing its output to zero would negatively influence the model since zero itself could already carry some information. In the image detection example this would be a black pixel. Depending on the image, setting a pixel to zero could completely change the image (if it is mostly light) or not change the image at all (if

it is mostly dark). To address this, removed features are most often replaced by their average or a random value.

2.4.2 Neuron Shapley

Large ML models can be incredibly time-consuming to train and require an extensive amount of training data. Therefore, when biases are detected, the process of cleaning the training data and retraining the model can become impractical. It would be ideal if we could identify and eliminate specific neurons in the model that we consider responsible for these biases. To address this, Ghorbani et al. [GZ20] introduced the concept of *Neuron Shapley*.

Out of the NN and the available data a cooperative game can be constructed, where each neuron represents a player. A performance metric, such as accuracy, is used as the value function v . For image classifications tasks the accuracy of a model is the ratio of correctly labeled images from a sampled batch.

To run the model using only a subset of neurons, the remaining neurons need to be deactivated. However, simply replacing the output of these neurons with zeros would negatively impact the mean statistic of the corresponding layer, which would in turn affect other neurons (similar to removing features in section 2.4.1). Therefore, instead of replacing the output with zeros, Ghorbani et al. [GZ20] propose replacing the output of deactivated neurons with their mean output. This approach ensures that the subsequent neurons still receive inputs that are not completely new values.

Ghorbani et al. [GZ20] employ their method on the Inception-v3 [Sze+16] CNN model, consisting of 17216 filters that are used as players and are able to identify the most important filters of each layer. By inspecting these filters' strongest activations they are able to explain the main workflow of the model intuitively. Early layers tend to detect general concepts of images such as main colors or shapes while deeper layers tend to be more task specific detecting e.g. animal eyes as shown in fig. 2.3.

In addition, they are able to target class specific neurons by calculating Shapley values on certain image classes. For zebras, a filter activated by images containing stripes is critical, while a filter activated by crowded objects is important to detect carousels. They also show that they can strongly reduce the models accuracy in specific classes by removing up to 40 important neurons while retaining most of the models' overall accuracy. A similar method is also used to reduce biases towards

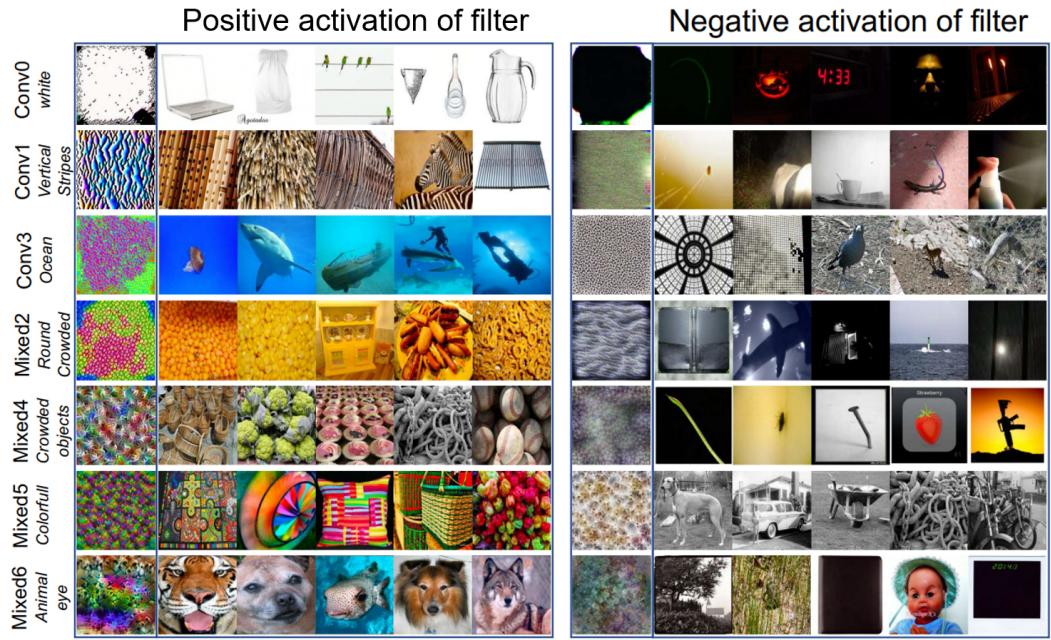


Fig. 2.3: Ghorbani et al.’s [GZ20] visualization of filters with the highest Shapley values for a selection of Inception-v3 layers. For each filter, 5 images with most positive or negative activation of the respective neuron are displayed giving the filter a meaningful interpretation. On the left of both columns they provide an image that is optimized to strongly activate the filter.

certain demographics in a SqueezeNet [Ian+16] model trained to detect gender from faces.

Algorithms

In this thesis, we aim to conduct a comprehensive comparison of algorithms on the Top-k Shapley problem. Our primary focus is on evaluating the core algorithms in order to discover which concepts work the best for the challenge at hand, avoiding any bias towards specific use cases. Furthermore, each algorithm is modified to keep the Shapley value estimator $\hat{\phi}_i$ of each player $i \in N$ up to date. The algorithms are formulated to run indefinitely but are stopped when the budget is depleted and the top-k players can be inferred by sorting the estimated values. Modifications and added details are marked in color.

All the algorithms under evaluation operate by evaluating the value function for sampled coalitions and utilizing the obtained coalition's worth to update the estimated Shapley values of certain players. No matter the coalition, for every player exactly one other coalition exists such that the two coalitions can be used to compute a marginal contribution of that player. Therefore, in theory each sampled coalition's worth could be used to update all players Shapley values if the value of the matching coalition for the player is known. To quantify the effectiveness of an algorithm in updating its Shapley values, we introduce the concept of *budget efficiency*.

Definition 3.0.1 (Budget efficiency). For a cooperative game (N, v) , the budget efficiency β of an algorithm is defined as the ratio between the number of marginal contributions used to update the Shapley value estimate of any player $i \in N$ and the consumed budget (number of value function evaluations).

Example 3.0.1 (Perfect Budget efficiency). Consider an algorithm that calculates the worth $v(S)$ of all 2^n coalition $S \subseteq N$ once and uses these values to calculate the Shapley values of all players. As each Shapley value represents the weighted average of 2^{n-1} marginal contributions, this approach yields a budget efficiency of $\beta = \frac{n2^{n-1}}{2^n} = \frac{n}{2}$.

Example 3.0.2 (Budget efficiency of a simple approximation algorithm). A very basic approximation algorithm would iterate through the players $i = 1, \dots, n$, sample

coalitions $S \subseteq N \setminus \{i\}$ with probability distribution $P(S) = \frac{1}{n^{\binom{n-1}{|S|}}}$ and compute the marginal contribution $v(S \cup \{i\}) - v(S)$ in order to improve the player's Shapley value estimator, according to eq. (2.3). Each marginal contribution update requires two value function evaluations, resulting in a budget efficiency of $\beta = \frac{1}{2}$.

In the following sections we present a selection of algorithms proposed in the literature, the core ideas behind them and our modifications. In addition we introduce our own algorithms for the top-k Shapley problem.

3.1 ApproShapley

Castro et al. [CGT09] proposed a simple yet powerful approximation algorithm called *ApproShapley*. This algorithm is based on the idea of sampling permutations $O : N \rightarrow N$ of the player set $N = \{1, \dots, n\}$ instead of sampling coalitions directly. $\pi(N)$ denotes the set of all $n!$ possible permutations. For simplicity, we often define a permutation $O \in \pi(N)$ as a vector $O = (O(1), \dots, O(n))$ of players. Using this notation a permutation is interpreted as an ordering of the players where $O(j) = i$ implies that player i is at the j th position in the sequence. Castro et al. [CGT09] approximate the Shapley value using an equation equivalent to eq. (2.1):

$$\phi_i = \sum_{O \in \pi(N)} \frac{1}{n!} [v(Pre^i(O) \cup \{i\}) - v(Pre^i(O))], \quad (3.1)$$

where $Pre^i(O) = \{O(1), \dots, O(i-1)\}$ is the predecessor set of player i in a permutation $O \in \pi(N)$, with $i = O(k)$. The algorithm functions by uniformly sampling a random permutation with probability $\frac{1}{n!}$ in each iteration. The sampled permutation O is then used to compute a marginal contribution for each player. This is achieved by iterating through the permutation and updating the current player i using the values of the predecessor set $Pre^i(O)$ with and without the player added.

Although the paper does not explicitly demonstrate it, ApproShapley can be optimized to utilize only one value function call per update. This optimization is possible because the value of the predecessor set combined with the current player $i = O(k)$ can be reused to calculate the marginal contribution of the next player $O(k+1)$, since $Pre^{O(k)}(O) \cup \{O(k)\} = Pre^{O(k+1)}(O)$.

Example 3.1.1. Consider a player set $N = \{1, 2, 3\}$, and assume we sample the permutation $(2, 3, 1)$. The respective marginal contributions are calculated as follows:

1. Player 2:

$$\begin{aligned} v(Pre^2((2, 3, 1)) \cup \{2\}) - v(Pre^2((2, 3, 1))) \\ = \textcolor{orange}{v}(\{2\}) - v(\emptyset) \end{aligned}$$

2. Player 3:

$$\begin{aligned} v(Pre^3((2, 3, 1)) \cup \{3\}) - v(Pre^3((2, 3, 1))) \\ = \textcolor{teal}{v}(\{2, 3\}) - \textcolor{orange}{v}(\{2\}) \end{aligned}$$

3. Player 1:

$$\begin{aligned} v(Pre^1((2, 3, 1)) \cup \{1\}) - v(Pre^1((2, 3, 1))) \\ = v(\{2, 3, 1\}) - \textcolor{teal}{v}(\{2, 3\}) \end{aligned}$$

With this approach, we can reuse the values for each player except for the values of the empty set, which is always zero, and the grand coalition. Consequently, updating every player once requires only $n = 3$ value function calls.

Due to this behavior, ApproShapley (Algorithm 1) achieves a budget efficiency of $\beta = \frac{n}{n} = 1$.

Algorithm 1 optimized ApproShapley

```

Require: ( $N = \{1, \dots, n\}, v$ )
 $\hat{\phi}_i \leftarrow 0, t_i \leftarrow 0$ , for all  $i \in N$ 
while true do
    Sample  $O \in \pi(N)$  uniformly with probability  $\frac{1}{n!}$ 
     $pre \leftarrow 0$ 
    for  $k \in N$  do
         $i \leftarrow O(k)$ 
         $t_i \leftarrow t_i + 1$ 
         $\delta_{i,t_i} \leftarrow v(Pre^i(O) \cup \{i\}) - pre$ 
         $\hat{\phi}_i \leftarrow \frac{(t_i-1)\hat{\phi}_i + \delta_{i,t_i}}{t_i}$ 
         $pre \leftarrow \phi_{i,t_i} + pre$ 
    end for
end while

```

3.2 SVARM and Stratified SVARM

When inspecting eq. (2.1), we can observe that for each player i , the value $v(S)$ of every coalition $S \subseteq N$ will be added to the sum if $i \in S$, or subtracted if $i \notin S$. Kolpaczki et al. [KBH23] propose an approach to efficiently update multiple players' marginal contributions simultaneously by approximating two separate sums. By transforming eq. (2.1) we get:

$$\phi_i = \sum_{S \subseteq N \setminus \{i\}} \frac{v(S \cup \{i\})}{n \binom{n-1}{|S|}} - \sum_{S \subseteq N \setminus \{i\}} \frac{v(S)}{n \binom{n-1}{|S|}}. \quad (3.2)$$

When sampling a coalition though, we have to keep in mind that two coalitions forming a marginal contribution have different sizes. Kolpaczki et al. [KBH23] demonstrate that to avoid introducing a bias, the probability distributions for sampled coalitions S^+ (used for positive updates) and S^- (used for negative updates) need to satisfy the following condition for all $i \in N$ and $S \subseteq N \setminus \{i\}$:

$$P(S^+ = S \cup \{i\} \mid i \in S^+) = P(S^- = S \mid i \notin S^-) = \frac{1}{n \binom{n-1}{|S|}}.$$

By using separate distributions for S^+ and S^- that satisfy this condition, the authors introduce the algorithm *Shapley Value Approximation without Requesting Marginals* (SVARM) [KBH23], which updates the positive sum for all players $i \in S^+$ and the negative sum for all players $i \notin S^-$ by sampling coalitions S^+ and S^- in alternating fashion and evaluating $v(S^+)$ and $v(S^-)$ respectively.

Kolpaczki et al. [KBH23] further enhance SVARM, eliminating the need for separate probability distributions, by employing stratification: The Shapley value's sum is rearranged into multiple sums that can be approximated separately with each one summing over all coalitions of a specific length. Equation (2.1) is reformulated as:

$$\phi_i = \frac{1}{n} \sum_{l=0}^{n-1} \phi_{i,l}^+ - \phi_{i,l}^-, \quad (3.3)$$

with:

$$\phi_{i,l}^+ = \sum_{S \subseteq N \setminus \{i\}, |S|=l} \frac{v(S \cup \{i\})}{\binom{n-1}{l}} \quad (3.4)$$

$$\phi_{i,l}^- = \sum_{S \subseteq N \setminus \{i\}, |S|=l} \frac{v(S)}{\binom{n-1}{l}}. \quad (3.5)$$

The stratas $\phi_{i,l}^+$ and $\phi_{i,l}^-$ can be approximated in a similar fashion as $\hat{\phi}_i$ in eq. 2.3. Kolpaczki et al.'s *Stratified SVARM* (StratSVARM) [KBH23] functions by sampling a coalition $S \subseteq N$ and calculating $v(S)$. For every player i , it updates $\hat{\phi}_{i,|S|-1}^+$ if $i \in S$ and $\hat{\phi}_{i,|S|}^-$ if $i \notin S$. This enables an update of every player's Shapley value with each call of the value function. As in theory each sampled coalition resembles half of a marginal contribution, StratSVARM has a budget efficiency of $\beta = \frac{n/2}{1} = \frac{n}{2}$ basically the best possible budget efficiency, regarding example 3.0.1.

Remark 3.2.1. It is important to note that StratSVARM does not directly update marginal contributions. Updating $\hat{\phi}_{i,l}^+$ and $\hat{\phi}_{i,l}^-$ once each, using samples $S_1, S_2 \subseteq N$, might not be as valuable as updating $\hat{\phi}_i$ using a complete marginal contribution $v(S \cup \{i\}) - v(S)$.

The sampling strategy for the length of a coalition is arbitrary. Each $\hat{\phi}_{i,l}^+$ and $\hat{\phi}_{i,l}^-$ is approximated separately, enabling Kolpaczki et al. [KBH23] to choose a probability distribution for l with desirable mathematical properties.

For our evaluation, we remove the algorithm's warmup phase as it requires a substantial amount of budget and is needed mostly for theoretical guarantees.

In addition to StratSVARM we propose *Truncated Stratified SVARM* (algorithm 2) for Neuron Shapley applications. This simplified version of StratSVARM includes the feature of truncation. With enough neurons removed, the accuracy of most neural networks drops to zero. Consequently $\phi_{i,l}^+, \phi_{i,l}^- \approx 0$ for $l < \delta$ with a certain size threshold $0 \leq \delta < n$ which makes TruncStratSVARM even more memory efficient as these values do not have to be stored. We can abstain from evaluating coalitions smaller than δ and sample l uniformly at random from the remaining sizes. Like Stratified SVARM, TruncStratSVARM updates all $\hat{\phi}_{i,l-1}^+$ for $i \in S$ and $\hat{\phi}_{i,l}^-$ for $i \notin S$ using a uniformly sampled coalition $S \subseteq N$ with $|S| = l$. Additionally, the exact calculation of StratSVARM, which involves computing stratas of certain sizes exactly, is minimized since calculating the value for all coalitions of size 1 and $n - 1$ would require $2n$ value function calls which is a substantial portion of the budget we have available in our evaluation. Also $1 \ll \delta$ most of the time.

Algorithm 2 TruncStratSVARM

Require: $(N = \{1, \dots, n\}, v), 0 \leq \delta < n$

$$\hat{\phi}_{i,l}^+, \hat{\phi}_{i,l}^- \leftarrow 0, \text{ for all } i \in N, l \in \{\delta, \dots, n-1\}$$

$$c_{i,l}^+, c_{i,l}^- \leftarrow 0, \text{ for all } i \in N, l \in \{\delta, \dots, n-1\}$$

$$\hat{\phi}_i \leftarrow 0, \text{ for all } i \in N$$

$$\hat{\phi}_{i,n-1}^+ \leftarrow v(N), c_{i,n-1}^+ \leftarrow 1, \text{ for all } i \in N$$

while *true* **do**

- Draw $l \in \{\delta + 1, \dots, n - 1\}$ uniformly at random
- Draw $S \in \{S \subseteq N \mid |S| = l\}$ uniformly at random
- $\gamma \leftarrow v(S)$
- for** $i \in S$ **do**
- $\hat{\phi}_{i,l-1}^+ \leftarrow \frac{c_{i,l-1} \hat{\phi}_{i,l-1}^+ + \gamma}{c_{i,l-1}^+ + 1}$
- $c_{i,l-1}^+ \leftarrow c_{i,l-1}^+ + 1$
- end for**
- for** $i \notin S$ **do**
- $\hat{\phi}_{i,l}^- \leftarrow \frac{c_{i,l}^- \hat{\phi}_{i,l}^- + \gamma}{c_{i,l}^- + 1}$
- $c_{i,l}^- \leftarrow c_{i,l}^- + 1$
- end for**
- $\hat{\phi}_i \leftarrow \frac{1}{n-\delta} \sum_{l=\delta}^{n-1} \hat{\phi}_{i,l}^+ - \hat{\phi}_{i,l}^-, \forall i \in N$

end while

3.3 Border Uncertainty Sampling

Kolpaczki et al. [KBH21] formally introduced the problem of identifying the Top-k Shapley players and proposed Border Uncertainty Sampling (BUS) (algorithm 3), an algorithm that focusses on players that are uncertain to belong to the top-k by calculating their current estimated Shapley value's distance to the border separating the top-k players from the rest. Although they did not provide mathematical proofs for its approximation quality, this method demonstrated promising empirical results.

In each iteration, the ordering $\hat{\pi} : N \rightarrow N$ is computed such that $\hat{\phi}_{\hat{\pi}(1)} \geq \dots \geq \hat{\phi}_{\hat{\pi}(n)}$. The next player i to be updated is determined based on the lowest value of $\Delta_i \cdot t_i$, where Δ_i represents the difference between the player's current estimator and the border (mean of the kth and k+1th largest values). t_i is the number of times the player's average marginal contribution has been updated. Intuitively the product represents the certainty of this player being classified correctly as part of the top-k or not. A player whose estimator is far away from the border and has been updated many times is improbable to change partition while a player close to the border with rare updates is more likely to be misclassified as illustrated in fig. 3.1.

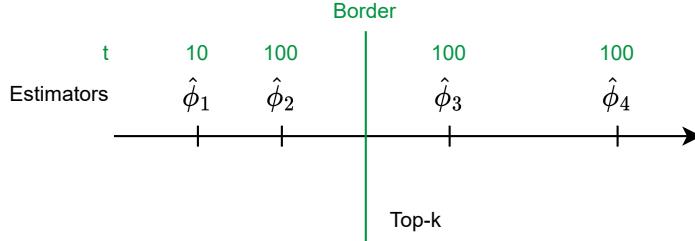


Fig. 3.1: Illustration of the Shapley value estimators of four players on the number line and their respective update count t . Player 1 is the most uncertain as its estimator has been updated 10 times less than the others' while it is only twice the distance from the border than $\hat{\phi}_2$ and $\hat{\phi}_3$.

This method fully capitalizes on the simplified problem statement by abstaining from wasting budget on players that are already confidently classified, instead improving the estimator of more uncertain players.

However, it still requires two value function calls per update, resulting in a budget efficiency of $\beta = \frac{1}{2}$. Unfortunately, easily adapting the concept of permutation sampling from ApproShapley, which allows for only one call per update, is not feasible as it relies on updating all players' Shapley values once per permutation.

Algorithm 3 Border Uncertainty Sampling

```

Require:  $(N = \{1, \dots, n\}, v), k \in \mathbb{N}$ 
 $\hat{\phi}_i \leftarrow 0, t_i \leftarrow 0$ , for all  $i \in N$ 
for  $i \in N$  do
    Draw  $S \subseteq N \setminus \{i\}$  with probability  $\frac{1}{n \binom{n-1}{|S|}}$ 
     $\phi_i \leftarrow v(S \cup \{i\}) - v(S)$ 
end for
while true do
    Compute  $\hat{\pi} : N \rightarrow N$  with  $\hat{\phi}_{\hat{\pi}(1)} \geq \dots \geq \hat{\phi}_{\hat{\pi}(n)}$ 
     $\hat{\phi}^* \leftarrow \frac{\hat{\phi}_{\hat{\pi}(k)} + \hat{\phi}_{\hat{\pi}(k+1)}}{2}$ 
     $\Delta_i \leftarrow |\hat{\phi}_i - \hat{\phi}^*|, \forall i \in N$ 
     $p \leftarrow \arg \min_{i \in N} \Delta_i \cdot t_i$ 
     $t_p \leftarrow t_p + 1$ 
    Draw  $S \subseteq N \setminus \{i\}$  with probability  $\frac{1}{n \binom{n-1}{|S|}}$ 
     $\phi_{p,t_p} \leftarrow v(S \cup \{p\}) - v(S)$ 
     $\hat{\phi}_p \leftarrow \frac{(t_p-1)\hat{\phi}_p + \phi_{p,t_p}}{t_p}$ 
end while

```

3.4 Truncated Multi Armed Bandit Shapley

Ghorbani et al. [GZ20] propose the *Truncated Multi-Armed Bandit* (TMAB) Shapley algorithm, which builds upon the optimized version of Castro et al.'s ApproShapley [CGT09], also sampling permutations $O \in \pi(N)$. The TMAB algorithm enhances the approach by incorporating players' variances to compute confidence bounds for the estimators. Players whose confidence bounds do not cover the k th largest player's estimator are labeled as sufficiently approximated and are disregarded for the next iteration.

Example 3.4.1. Suppose we randomly sampled the permutation $O = (3, 4, 2, 1, 0)$ and players 2 and 4 have already been labeled as sufficiently approximated. When iterating through the permutation as in ApproShapley, there is no need to calculate $v(2, 1, 0)$ since the only players whose marginal contribution would require this value are the players 2 and 4.

For this to work we need to keep the estimators' variances up to date. This is done by reformulating the sample variance eq. (2.6) to

$$\hat{\sigma}^2 = \frac{1}{m-1} \left(\sum_{i=1}^m x_i^2 - m\bar{x}^2 \right). \quad (3.6)$$

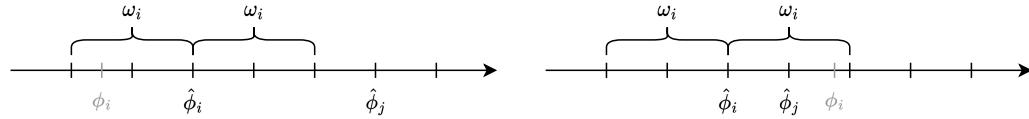
For this use case $(x_i)_{i=1}^m$ denote the marginal contributions used to update a player $j \in N$ where m is the total number of updates. Consequently, $\bar{x} = \hat{\phi}_j$ since the Shapley value estimator is equal to the player's average marginal contribution. By introducing a separate variable $S_j := \sum_{i=1}^m x_i^2$ that sums over all squared marginal contributions, we are able to calculate the sample variance of the player's estimator in each iteration.

To determine the confidence bounds, the algorithm employs the empirical Bernstein method [MP09]. This ensures that with a probability of $1 - \delta$ (where $\delta \in (0; 1]$), the inequality

$$\phi_i - \hat{\phi}_i \leq \underbrace{\sqrt{\frac{2\hat{\sigma}_i^2 \ln 2/\delta}{m}} + \frac{7 \ln 2/\delta}{3(m-1)}}_{:=\omega_i}$$

holds, where m denotes the number of samples used for the estimator $\hat{\phi}_i$ as in equation 2.3 and $\hat{\sigma}_i^2$ represents the estimator's variance. This means that the player's estimator is at most ω_i off of its true value. Therefore, if the absolute difference

between the estimors of this player and the k th largest player is larger than ω_i , the true Shapley value of the player is likely to be on the same side of the k th largest player as the estimator (see fig. 3.2) which indicates that the player is classified correctly and can be disregarded for now.



- (a) $\hat{\phi}_j$ is outside the confidence bounds of player i . This suggests, that ϕ_i is also smaller than $\hat{\phi}_j$.
- (b) $\hat{\phi}_j$ is inside the confidence bounds of player i . Therefore, ϕ_i might be greater than $\hat{\phi}_j$ even though $\hat{\phi}_i < \hat{\phi}_j$.

Fig. 3.2: Illustration of how the confidence bounds ω_i indicate whether the Shapley value of player i is certain to be smaller or greater than the Shapley value of player j .

In practice however, the algorithm tends to perform a large number of updates before players are excluded, as the confidence bounds can be quite high.

Example 3.4.2. Let's assume that all Shapley values fall between 0.01 and 0.02, and let $\delta = 0.2$. Even with a variance of zero, it would take at the very least 538 updates before a player is removed. This is because

$$\frac{7 \ln 2 / 0.2}{3(538 - 1)} \geq 0.01 = |0.01 - 0.02|$$

After 538 updates, the confidence bounds of each player still encompass all other players, including the k th largest one.

Initially, TMAB's budget efficiency is similar to ApproShapley, as all players are included. As the first players start to get excluded, the budget efficiency slightly decreases when the algorithm jumps over a set of players.

As TMAB was developed for the Neuron Shapley application it employs a form of truncation aswell. Contrary to TruncStratSVARM, it does not use a fixed maximum coalition length but stops iterating through a permutation when the coalitions' values become lower than a certain threshhold v_T .

Algorithm 4 describes the TMAB Shapley algorithm by Ghorbani et al. [GZ20] with additional details compared to the original paper.

Remark 3.4.1. The authors did not evaluate their algorithm on a fixed budget scenario but rather allowed it to run until all players were excluded. To adapt the algorithm to our setting, we stop it when the budget is exhausted.

Algorithm 4 Truncated Multi Armed Bandit Shapley

```

Require:  $(N = \{1, \dots, n\}, v), k \in \mathbb{N}, 0 < \delta \leq 1, v_T \in \mathbb{R}$ 
 $\hat{\phi}_i \leftarrow 0, \sigma_i^2 \leftarrow 0, S_i \leftarrow 0, \omega_i \leftarrow 0$ , for all  $i \in N$ 
 $t \leftarrow 0$ 
 $\mathcal{U} \leftarrow N$ 
while  $\mathcal{U} \neq \emptyset$  do
     $t \leftarrow t + 1$ 
    Draw  $O \in \pi(N)$  uniformly with probability  $\frac{1}{n!}$ 
     $v_0^t \leftarrow v(N)$ 
    for  $k \in N$  do
         $i \leftarrow O(k)$ 
        if  $i \in \mathcal{U}$  then
            if  $v_{k-1}^t = \text{null}$  then
                 $v_{k-1}^t \leftarrow v(\{O(1), \dots, O(n)\})$ 
            end if
            if  $v_{k-1}^t < v_T$  then
                 $v_k^t \leftarrow v_{k-1}^t$ 
            else
                 $v_k^t \leftarrow v(\{O(k+1), \dots, O(n)\})$ 
            end if
             $S_i \leftarrow S_i + (v_{k-1}^t - v_k^t)^2$ 
             $\hat{\phi}_i \leftarrow \frac{(t-1)\hat{\phi}_i + v_{k-1}^t - v_k^t}{t}$ 
             $\sigma_i^2 \leftarrow \frac{S_i - t\hat{\phi}_i^2}{t-1}$ 
             $\omega_i \leftarrow \sqrt{\frac{2\sigma_i^2 \ln 2/\delta}{t}} + \frac{7 \ln 2/\delta}{3(t-1)}$ 
        else
             $v_k^t \leftarrow \text{null}$ 
        end if
    end for
     $\mathcal{U} \leftarrow \{i \mid \hat{\phi}_i - \omega_i < k\text{'th largest } \{\hat{\phi}_i\}_{i=1}^n < \hat{\phi}_i + \omega_i\}$ 
end while

```

3.5 HybridApproBUS with Caching

In the first phase of BUS, a single marginal contribution is sampled for each player in order to establish a foundation for uncertainty sampling in the next phase. However, we believe that obtaining a better approximation for all Shapley values initially

would be more reasonable before focusing on updating only the uncertain players. In the initial iterations of uncertainty sampling, most players are equally uncertain, making it less significant to prioritize sampling the most uncertain player. Updating this player alone always requires two value function calls, which can be quite resource-intensive, especially considering that we do not benefit as much from BUS's strength in the beginning.

To address these issues, we propose a new algorithm called *HybridApproBUS*, where the initial warm-up phase is replaced by ApproShapley. Due to ApproShapley's better budget efficiency of $\beta = 1$ versus BUS's $\beta = \frac{1}{2}$ it is well-suited to create a first estimation for all Shapley values. We are repeating the more efficient sampling of permutations and specialize later with the uncertainty sampling as more and more players become irrelevant, rendering the permutation sampling less efficient. In order to determine when to transition from ApproShapley to BUS, we introduce a constant $s \in \mathbb{N}$ that defines the total amount of permutations that are iterated by ApproShapley. In our evaluation we experiment with different values for s in order to find an optimal value.

Furthermore, we introduce *SmartHybridApproBUS* (algorithm 5), which incorporates a form of caching. Instead of sampling a coalition $S \subseteq N$, we sample permutations $O \in \pi(N)$ similar to ApproShapley. However, instead of iterating through the permutation, we determine the most uncertain player p as done in BUS. Like in ApproShapley we want to utilize permutation sampling to reuse coalition values. In order to do so, we use $E = (\epsilon_i)_{i=1}^n$ with $\epsilon_i \in \mathbb{R} \cup \{\text{null}\}$ to cache coalition values for the current permutation. When calculating the marginal contribution of the selected player, we look up E to check if the required values are already cached and can be reused. If a coalition value is not cached, we evaluate it and store it in the cache. Additionally, we maintain a binary indicator $w_i \in \mathbb{B}$ with $i \in N$ to track whether we have already calculated the marginal contribution of a player for the current permutation. If this is the case, the player's estimator has already been updated using this marginal contribution and a new permutation O has to be sampled.

Example 3.5.1. Consider the player set $N = \{1, 2, 3, 4\}$. Suppose we freshly sampled the permutation $(1, 2, 3, 4)$ and the cache $E = (\text{null}, \text{null}, \text{null}, \text{null})$ is empty. Let's assume player 2 is currently the most uncertain player. The marginal contribution for player 2 can be calculated using $v(\{1, 2\}) - v(\{1\})$. After this calculation, the cache is updated, becoming $E = (v(1), v(1, 2), \text{null}, \text{null})$. Now, if the next most uncertain player is player 1, for its marginal contribution $v(1) - 0$, we can reuse the value ϵ_1 from the cache. However, if the next most uncertain player is player 2 again,

we must sample a new permutation since we already updated player 2 using the marginal contribution from this permutation.

3.6 HalfBUS

We propose a new algorithm, called HalfBUS (Algorithm 6), as an approximation method for identifying the top-k Shapley players. Our algorithm is inspired by the works of Kolpaczki et al., specifically BUS [KBH21] and StratSVARM [KBH23]. While retaining the focus on players who are uncertain to belong to the top-k, HalfBUS improves budget efficiency by reusing one sample to update the Shapley values of multiple players, similar to StratSVARM.

Unlike BUS, which concentrates on the most uncertain player, HalfBUS identifies the set $P \subseteq N$ of the w most uncertain players, with w ranging from 1 to n . Similar to StratSVARM, we draw a sample $S \subseteq N$ with probability distribution $P(S) = \frac{1}{(n+1)\binom{n}{|S|}}$ and evaluate its worth $v(S)$. However, instead of updating all players using this value, we evaluate an additional coalition for each player $i \in P$. This coalition is selected in a way that its value forms a marginal contribution with $v(S)$ for player i . Specifically, if $i \in S$, we evaluate $v(S \setminus \{i\})$ and $v(S \cup \{i\})$ if $i \notin S$. Now, each player can be updated using $v(S) - v(S \setminus \{i\})$ or $v(S \cup \{i\}) - v(S)$ depending whether it is in S or not.

Example 3.6.1. Suppose we identified the players $P = \{1, 2\}$ as the $w = 2$ most uncertain players and sampled the coalition $S = \{2, 3\}$. We evaluate $v(S)$ and form the additional coalitions $S \cup \{1\} = \{1, 2, 3\}$ and $S \setminus \{2\} = \{3\}$ for player 1 and 2 respectively. By evaluating these additional coalitions, we are able to update player 1 using $v(\{1, 2, 3\}) - v(S)$ and player 2 using $v(S) - v(\{3\})$, evaluating the value function only three times in total.

Our approach does not introduce a bias. This can be proven by reformulating eq. (2.1). Let $w_l = \frac{1}{n\binom{n-1}{l}}$ for $0 \leq l < n$. It is irrelevant if we sum over all coalitions that do not contain player i or if we sum over all coalitions that do contain player i and remove this player afterwards. Therefore:

$$\sum_{S \subseteq N, i \notin S} w_{|S|} \cdot [v(S \cup \{i\}) - v(S)] = \sum_{S \subseteq N, i \in S} w_{|S \setminus \{i\}|} \cdot [v(S) - v(S \setminus \{i\})].$$

Algorithm 5 SmartHybridApproBUS

Require: $(N = \{1, \dots, n\}, v), k \in \mathbb{N}, s \in \mathbb{N}$

$$\hat{\phi}_i \leftarrow 0, t_i \leftarrow 0, \forall i \in N$$

for $r \leftarrow 1, \dots, s$ **do**

- Take $O \in \pi(N)$ with probability $\frac{1}{n!}$
- $pre \leftarrow 0$
- for** $k \in N$ **do**

 - $i \leftarrow O(k)$
 - $t_i \leftarrow t_i + 1$
 - $\phi_{i,t_i} \leftarrow v(Pre^i(O) \cup \{i\}) - pre$
 - $\hat{\phi}_i \leftarrow \frac{(t_i-1)\hat{\phi}_i + \phi_{i,t_i}}{t_i}$
 - $pre \leftarrow \phi_{i,t_i} + pre$

- end for**
- end for**
- Take $O \in \pi(N)$ with probability $\frac{1}{n!}$
- $\omega_i = false, \epsilon_i = null, \forall i \in \mathbb{N}$
- while** $true$ **do**

 - Compute $\hat{\pi} : N \rightarrow N$ with $\hat{\phi}_{\hat{\pi}(1)} \geq \dots \geq \hat{\phi}_{\hat{\pi}(n)}$
 - $\hat{\phi}^* \leftarrow \frac{\hat{\phi}_{\hat{\pi}(k)} + \hat{\phi}_{\hat{\pi}(k+1)}}{2}$
 - $\Delta_i \leftarrow |\hat{\phi}_i - \hat{\phi}^*|, \forall i \in N$
 - $p \leftarrow \arg \min_{i \in N} \Delta_i \cdot t_i$
 - if** $\omega_p = true$ **then**

 - Take $O \in \pi(N)$ with probability $\frac{1}{n!}$
 - $\omega_i = false, \epsilon_i = null, \forall i \in \mathbb{N}$

 - end if**
 - $\omega_p \leftarrow true$
 - get i with $O(i) = p$
 - if** $i = 0$ **then**

 - $pre \leftarrow 0$

 - else**

 - $pre \leftarrow \epsilon_{O(i-1)}$

 - end if**
 - if** $pre = null$ **then**

 - $pre \leftarrow v(Pre^p(O))$
 - $\epsilon_{O(i-1)} = pre$

 - end if**
 - $curr \leftarrow \epsilon_p$
 - if** $curr = null$ **then**

 - $curr \leftarrow v(Pre^p(O) \cup \{p\})$
 - $\epsilon_p \leftarrow curr$

 - end if**
 - $t_p \leftarrow t_p + 1$
 - $\phi_{p,t_p} \leftarrow curr - pre$
 - $\hat{\phi}_p \leftarrow \frac{(t_p-1)\hat{\phi}_p + \phi_{p,t_p}}{t_p}$

- end while**

Furthermore eq. (2.1) can be reformulated as:

$$\phi_i = \sum_{S \subseteq N} \begin{cases} \frac{1}{2} w_{|S|-1} \cdot [v(S) - v(S \setminus \{i\})] & \text{if } i \in S, \\ \frac{1}{2} w_{|S|} \cdot [v(S \cup \{i\}) - v(S)] & \text{otherwise.} \end{cases} \quad (3.7)$$

Lemma 3.6.1. Given $i \in N$, the probability that a coalition $S \subseteq N$ drawn with distribution $P(S) = \frac{1}{(n+1)\binom{n}{|S|}}$ contains player i is $\frac{1}{2}$.

Proof:

$$\begin{aligned} P(i \in S) &= \sum_{l=0}^n P(i \in S, |S| = l) \\ &= \sum_{l=0}^n P(i \in S \mid |S| = l) \cdot P(|S| = l) \\ &= \sum_{l=0}^n \frac{\binom{1}{1} \binom{n-1}{l-1}}{\binom{n}{l}} \cdot \frac{1}{n+1} \\ &= \sum_{l=0}^n \frac{l}{n(n+1)} \\ &= \frac{1}{2} \end{aligned}$$

Theorem 3.6.1. By using eq. (3.7) to approximate the Shapley value of any player $i \in N$ we do not introduce a bias, i.e. $E(\hat{\phi}_i) = \phi_i$. Proof:

Given $i \in N$, let X denote a random variable with values $(x_S)_{S \subseteq N}$ where

$$x_S = \begin{cases} v(S) - v(S \setminus \{i\}) & \text{if } i \in S, \\ v(S \cup \{i\}) - v(S) & \text{if } i \notin S \end{cases}$$

and probability distribution $P(X = x_S) = \frac{1}{(n+1)\binom{n}{|S|}}$. Then the expected value of X is:

$$\begin{aligned} E[X] &= \sum_{S \subseteq N} P(X = x_S) \cdot x_S \\ &= \sum_{S \subseteq N} \begin{cases} P(X = x_S, i \in S) \cdot [v(S) - v(S \setminus \{i\})] & \text{if } i \in S, \\ P(X = x_S, i \notin S) \cdot [v(S \cup \{i\}) - v(S)] & \text{if } i \notin S. \end{cases} \end{aligned}$$

Using eq. (3.7) and:

$$\begin{aligned}
P(X = x_S, i \in S) &= P(S \mid i \in S) \cdot P(i \in S) \\
&= \frac{1}{n \binom{n-1}{|S|-1}} \cdot \frac{1}{2} \\
&= \frac{1}{2} w_{|S|-1} \\
P(X = x_S, i \notin S) &= P(S \mid i \notin S) \cdot (1 - P(i \in S)) \\
&= \frac{1}{n \binom{n-1}{|S|}} \cdot \frac{1}{2} \\
&= \frac{1}{2} w_{|S|}
\end{aligned}$$

we can show that $E[X] = \phi_i$.

By approximating this expected value using sampled coalitions $S \subseteq N$ like in eq. (2.3), we obtain an unbiased estimator for the Shapley value of player i .

The algorithm uses $w + 1$ value function calls for updating w players. Therefore the budget efficiency of HalfBUS is $\beta = \frac{w}{w+1}$. The choice of w depends on the application. When $w = 1$, HalfBUS essentially becomes BUS, as P contains only one player, and we require $1 + 1 = 2$ value function calls to update this player. On the other extreme, when $w = n$, we do not benefit from the uncertainty behavior of BUS, but we achieve the highest budget efficiency with n updates for $n + 1$ value function calls, similar to ApproShapley, which requires n calls for n updates.

We evaluate multiple versions of HalfBUS in order to find the ideal value for w .

Algorithm 6 HalfBUS

Require: $(N = \{1, \dots, n\}, v), k \in \mathbb{N}, w \in N$

$\hat{\phi}_i \leftarrow 0, t_i \leftarrow 0, \forall i \in N$

$\gamma_n \leftarrow v(N)$

while *true* **do**

Compute $\hat{\pi} : N \rightarrow N$ with $\hat{\phi}_{\hat{\pi}(1)} \geq \dots \geq \hat{\phi}_{\hat{\pi}(n)}$

$\hat{\phi}^* \leftarrow \frac{\hat{\phi}_{\hat{\pi}(k)} + \hat{\phi}_{\hat{\pi}(k+1)}}{2}$

$\Delta_i \leftarrow |\hat{\phi}_i - \hat{\phi}^*|, \forall i \in N$

Compute $\hat{\pi}' : N \rightarrow N$ with $\Delta_{\hat{\pi}'(1)} \cdot t_{\hat{\pi}'(1)} \leq \dots \leq \Delta_{\hat{\pi}'(n)} \cdot t_{\hat{\pi}'(n)}$

$P \leftarrow \{\hat{\pi}'(1), \dots, \hat{\pi}'(w)\}$

Take $l \in \{0, \dots, n\}$ uniformly at random

Take $S \subseteq N$ with $|S| = l$ uniformly at random

if $l = n$ **then**

$\gamma \leftarrow \gamma_n$

else if $l = 0$ **then**

$\gamma \leftarrow 0$

else

$\gamma \leftarrow v(S)$

end if

for $i \in P$ **do**

$t_i \leftarrow t_i + 1$

if $i \in S$ **then**

$\phi_{i,t_i} \leftarrow \gamma - v(S \setminus \{i\})$

else

$\phi_{i,t_i} \leftarrow v(S \cup \{i\}) - \gamma$

end if

$\hat{\phi}_i \leftarrow \frac{(t_i-1)\hat{\phi}_i + \phi_{i,t_i}}{t_i}$

end for

end while

Evaluation

4.1 Synthetic Cooperative Games

4.1.1 Experiment setup

We evaluate the different algorithms on synthetic cooperative games. These are cooperative games that are formulated in such a manner that the precise Shapley values can be computed within polynomial time. Synthetic games in which all players share identical Shapley values, such as the simple example in section 2.2, are not insightful for comparing algorithms on the top-k problem. This limitation arises because all players fall within the top-k category, leading to a scenario where all algorithms exhibit 100% accuracy. To circumvent this issue, we employ the algorithms on synthetic games where players can be categorized meaningfully as either top-k players or not. In the following sections we present the chosen synthetic games for examination.

Sum of Unanimity Games

Definition 4.1.1 (Unanimity game). Given a playerset $N = \{1, \dots, n\}$ and a subgroup of players $R \subseteq N$, the cooperative game (N, v_R) using

$$v_R(S) = \begin{cases} 1 & \text{if } R \subseteq S, \\ 0 & \text{otherwise} \end{cases}$$

is called a *unanimity game*.

The unanimity game is a sort of voting game where a certain set of players has a right to veto, i.e. all of these players have to cooperate in order to achieve the reward. All players that do not belong to this subgroup are unimportant for the value

function making them dummy players. Therefore the total reward is distributed equally amongst the players in the veto group $i \in R$ each receiving a Shapley value of $\phi_i = \frac{1}{|R|}$.

Definition 4.1.2 (Sum of unanimity games). Given a set of unanimity games $\{(N, v_{R_k})\}_{k=1}^m$, $m \in \mathbb{N}$ the linear combination of these games using weights $\{w_k\}_{k=1}^m$ is called a *sum of unanimity games* (SOUg) game. The value function is defined as:

$$v(S) = \sum_{k=1}^m w_k \cdot v_{R_k}(S)$$

The Shapley values of a SOUG game can be calculated by a linear combination of the Shapley values of the subgames, i.e.

$$\phi_i = \sum_{k=1}^m \begin{cases} \frac{w_k}{|R_k|} & \text{if } i \in R_k, \\ 0 & \text{otherwise.} \end{cases}$$

For our evaluation we generate random SOUG games. The parameters are drawn uniformly at random from the same ranges used by Kolpaczki et al. [KBH21]:

- $\{5, 6, \dots, 50\}$ for the number of combined unanimity games m
- $\{0, 1, \dots, n\}$ for the size of each R_k
- N for the members of each R_k
- $[0, \frac{1}{m}]$ for the weights w_k

Airport game

The *airport game* is the problem of distributing the cost of an airport's runway amongst the airplanes using this runway. Depending on the size of a plane i it requires a shorter or longer runway, represented by a cost c_i . The cost of a coalition of planes is the cost of the largest plane in this coalition as all smaller planes can use

the runway aswell. For a player set of planes $N = \{1, \dots, n\}$ and costs $c_1, \dots, c_n \in \mathbb{R}^+$ the value function of an airport games is defined as

$$v(S) = \max\{c_i \mid i \in S\}.$$

G. Owen [Owe13] provides a proof that the Shapley value of all planes can be calculated in polynomial time.

For our evaluation we use $N = \{1, \dots, 100\}$ with costs

$$\begin{aligned} c_1, \dots, c_{100} = & \underbrace{1, \dots, 1}_{8 \text{ times}}, \underbrace{2, \dots, 2}_{12 \text{ times}}, \underbrace{3, \dots, 3}_{6 \text{ times}}, \underbrace{4, \dots, 4}_{14 \text{ times}}, \underbrace{5, \dots, 5}_{8 \text{ times}}, \\ & \underbrace{6, \dots, 6}_{9 \text{ times}}, \underbrace{7, \dots, 7}_{13 \text{ times}}, \underbrace{8, \dots, 8}_{10 \text{ times}}, \underbrace{9, \dots, 9}_{10 \text{ times}}, \underbrace{10, \dots, 10}_{10 \text{ times}}. \end{aligned}$$

Castro et al. [CGT09] provide a list of precalculated Shapley values for this setting which we use in our evaluation.

Non-symmetric voting game

A *non-symmetric voting game* is a game where votes of different players i can have different weights w_i . In order for a coalition to win the reward of 1 it has to outweigh to opposing coalition of players. I.e. for a set of players $N = \{1, \dots, n\}$ and weights w_1, \dots, w_n the value function is defined as

$$v(S) = \begin{cases} 1 & \text{if } \sum_{i \in S} w_i > \sum_{j \in N} w_j / 2, \\ 0 & \text{otherwise.} \end{cases}$$

For our evaluation we use a non-symmetric voting game by Castro et al. [CGT09] that mimics the voting process in the USA using $N = \{1, \dots, 51\}$ and weights

$$\begin{aligned} w_1, \dots, w_{51} = & 45, 41, 27, 26, 26, 25, 21, 17, 17, 17, 14, 13, 13, 13, 12, 12, 12, 11, \\ & \underbrace{10, \dots, 10}_{4 \text{ times}}, \underbrace{9, \dots, 9}_{4 \text{ times}}, 8, 8, \underbrace{7, \dots, 7}_{4 \text{ times}}, \underbrace{6, \dots, 6}_{4 \text{ times}}, 5, \underbrace{4, \dots, 4}_{9 \text{ times}}, \underbrace{3, \dots, 3}_{7 \text{ times}} \end{aligned}$$

They also provide a list of exact Shapley values which we reuse in our evaluation aswell.

4.1.2 Results

In this section we present the results of evaluating both the established and new algorithms on synthetic games. Like in fig. 4.1 we plot the average performance of an algorithm over a certain number of experiments (as explained in section 2.3) on the y-axis. The SE is visualized using a light shadow behind the line. The x-axis represents the budget that has been used up.

SmartHybridApproBUS implements the conjecture that the performance of BUS improves, if the Shapley values of the players are already reasonably approximated beforehand, which would make sorting for the most uncertain player more meaningful.

To test this conjecture we run the algorithm using various different values for the switch parameter s indicating after how many iterations of ApproShapley we switch to BUS.

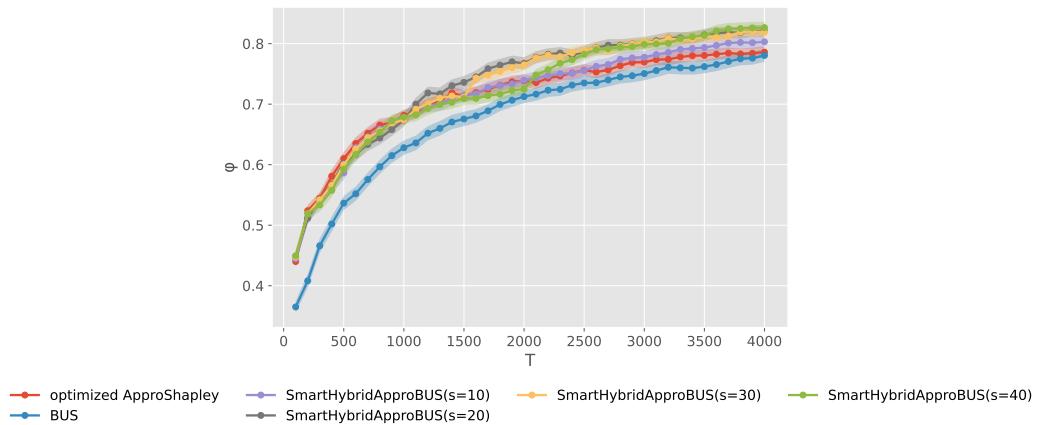


Fig. 4.1: Comparison of SmartHybridApproBUS with different values for s on SOUG games using $n = 50, k = 10$, averaged over 200 experiments

The results are depicted in fig. 4.1. We can see a significant improvement over BUS and ApproShapley for $s \geq 20$. For $s = 20, 30, 40$ we can observe a jump in performance at the point in time the algorithm switches to BUS (at $T = 1000, 1500, 2000$ respectively) which supports the theory. However, this effect of performance jump seems to be limited since all three versions end up on a similar line after $T = 3000$. This suggests that it helps BUS to have a proper estimation of the Shapley values first, to get a quick performance boost out of the uncertainty sampling, but from that point on the slope is similar to the original BUS algorithm.

For smaller values of s the effect is less significant. This suggests that the approximation of the values was not good enough to acquire the performance boost. We

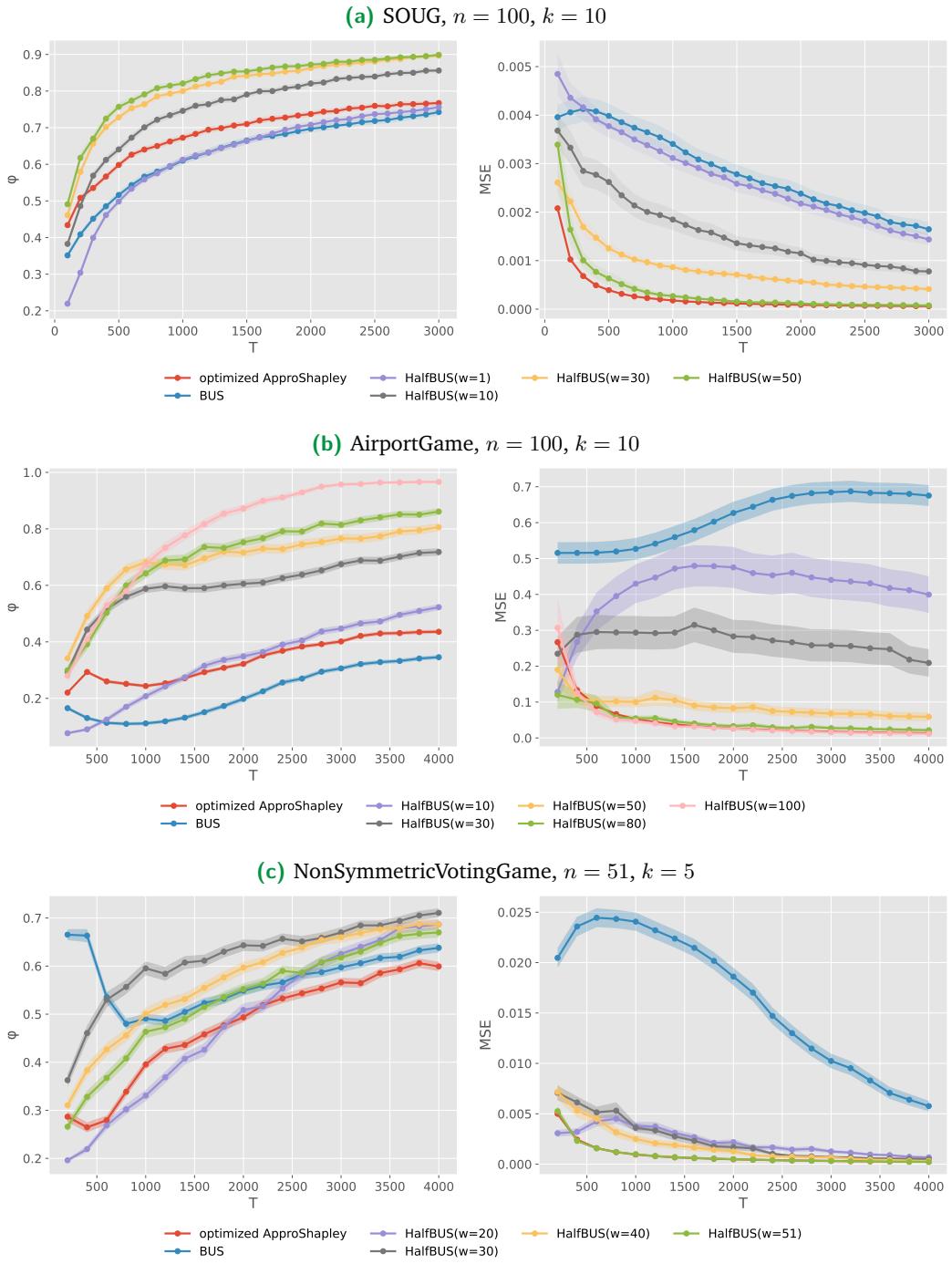


Fig. 4.2: Comparison of HalfBUS with different values for w on diffent games, averaged over 300 experiments

still see a slight improvement over BUS and ApproShapley which is probably due to the caching. The caching itself does not result in a strong increase in performance which is not surprising since the cached values are rarely used. Since BUS mostly

focusses on a small set of uncertain players, a sample cannot be reused very often.

HalfBUS is supposed to improve the budget efficiency of BUS by updating the marginal contribution of a set of uncertain players instead of a single one. This way the value of a single sample can be reused multiple times. As shown in fig. 4.2, we tested HalfBUS on multiple synthetic games using varying values for w , which specifies the length of the selected set of players to be updated in each iteration. As expected, visible in fig. 4.2a, using $w = 1$ HalfBUS achieves a very similar result to BUS as it becomes basically the same algorithm. All other variants perform far better than both BUS and ApproShapley. Interestingly though, in most experiments it seems that choosing $w = n$ achieves the best results. This is surprising, because using $w = n$ practically removes the uncertainty sampling of the algorithm as all players are updated in each iteration. It also has a very similar budget efficiency to ApproShapley, infact even a slightly worse one of $\beta = \frac{w}{w+1}$ versus ApproShapley's $\beta = 1$ which is why we would expect it to perform similar to ApproShapley. As the plots for the MSE show, this is exactly what is happening for the approximation problem. ApproShapley and HalfBUS have a very similar curve for $w = n$ with the one of HalfBUS being slightly worse. The decrease in budget efficiency should be negligible, when using a value of w only slightly smaller than n . However, in fig. 4.2b for example, *HalfBUS(w=100)* performs significantly better than *HalfBUS(w=80)*. Both have similar budget efficiency of $\frac{100}{101} \approx 0.99 \approx \frac{80}{81}$. This suggests, that another factor apart from budget efficiency or uncertainty sampling plays a part here.

We suppose, the main reason for HalfBUS's performance increase over ApproShapley and BUS is the fact that in each iteration all players of the selected set receive an update using coalitions with similar sizes and consisting of mostly the same players, which makes comparing their values much more reasonable in contrast to updating each player with a random coalition. For the approximation problem this does not matter because each player's estimated value is solely compared to its own exact value and not the other players.

Assuming that this is actually the reason for HalfBUS's performance increase in the top-k problem, this would also explain why choosing smaller values than n for w and taking advantage of uncertainty sampling does not yield even better results. This is because only the players in the selected set are updated using similar coalitions, while the rest receives no update, which negatively influences the ability to compare players of these separate sets. When using $w = n$, all players always receive an update using a similar coalition in each iteration, making all values comparable.

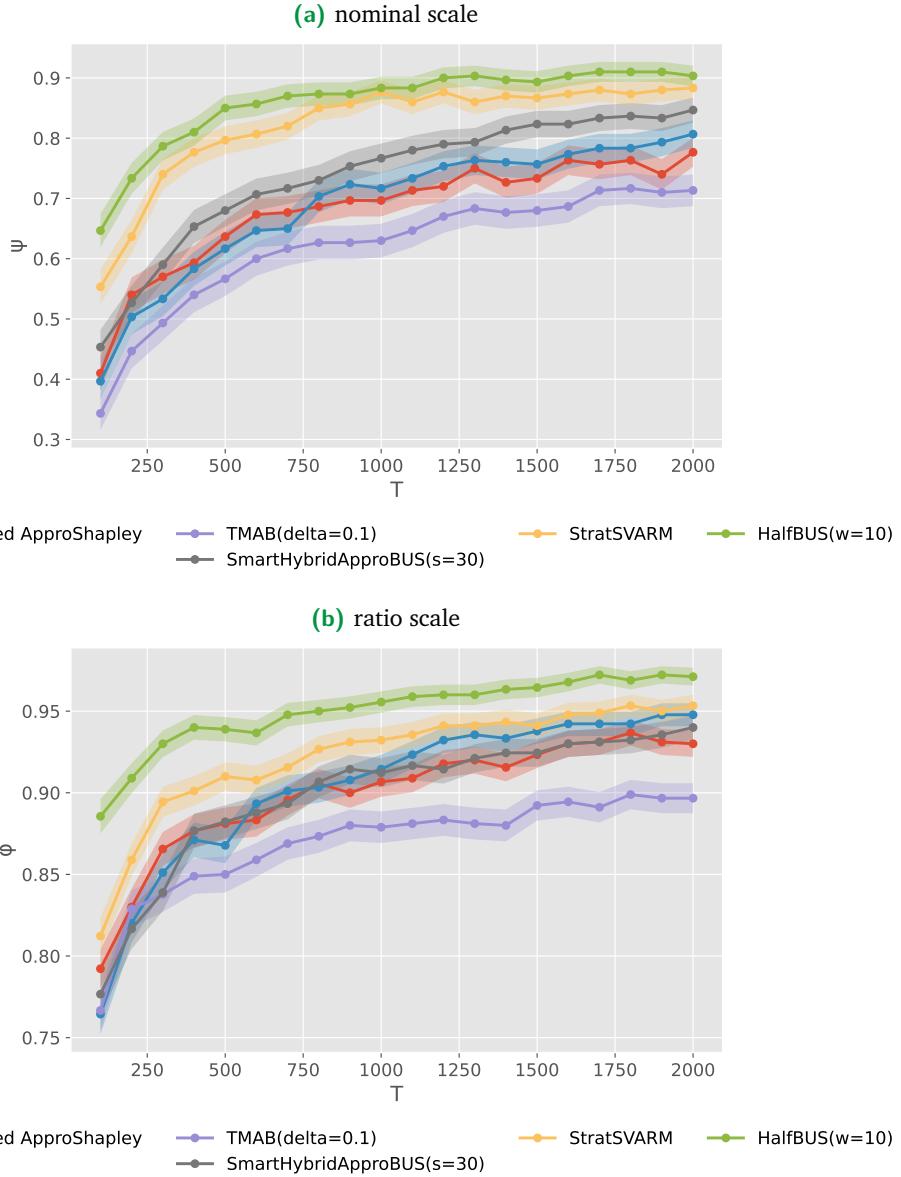


Fig. 4.3: Comparison of all algorithms on SOUG games using $n = 10, k = 3$, averaged over 300 experiments

Next, we compare our algorithms using the optimal parameters with the remaining algorithms of chapter 3. Figures 4.3 to 4.5 show the experiments we conducted. Figure 4.3a mimics the SOUG experiment of Kolpaczki et al. [KBH21] as they use a nominal scale as well. The results of BUS are slightly better in our experiment, which we suppose is due to the fact that their problem statement assumes the set of top-k players to be unique.

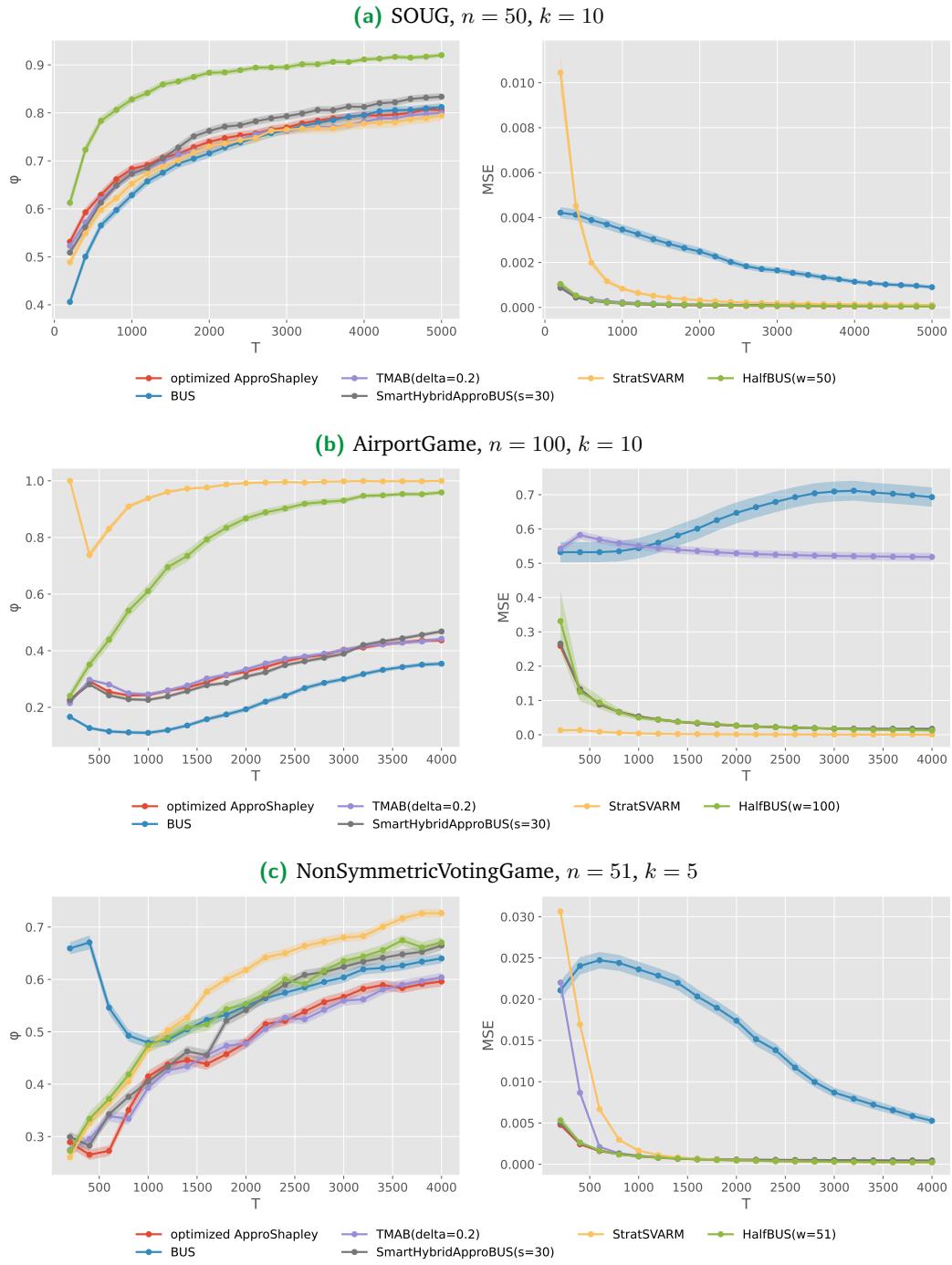


Fig. 4.4: Comparison of all algorithms on different games, averaged over 300 experiments

In fig. 4.4 we evaluate all algorithms in the different synthetic games using medium sized player sets. SmartHybridApproBUS seems to perform slightly better than ApproShapley and BUS but it does not achieve groundbreaking improvements. Ghorbani et al.’s TMAB algorithm [GZ20] performs almost identically to ApproShapley.

As explained in section 3.4, this is due to the fact that the confidence bounds used in the algorithm do not come into play for such a low budget. Since the value of synthetic games does not necessarily approach zero for small coalitions, we disabled truncation for TMAB. With these limitations, TMAB differs only slightly from ApproShapley. HalfBUS on the other hand yields far better results than most algorithms, especially in SOUG and the AirportGame. The only other algorithm that achieves similar or even better results is Kolpaczki et al.’s Stratified SVARM [KBH23] which is not surprising considering it has by far the best budget efficiency out of all algorithms. At least for fig. 4.4a the sampling strategy of HalfBUS, updating all players using similar coalitions seems to outweigh StratSVARM’s budget efficiency. However, StratSVARM’s budget efficiency increases proportional to the size of the player set which is why we conducted the SOUG experiment using larger values of n visible in fig. 4.5.

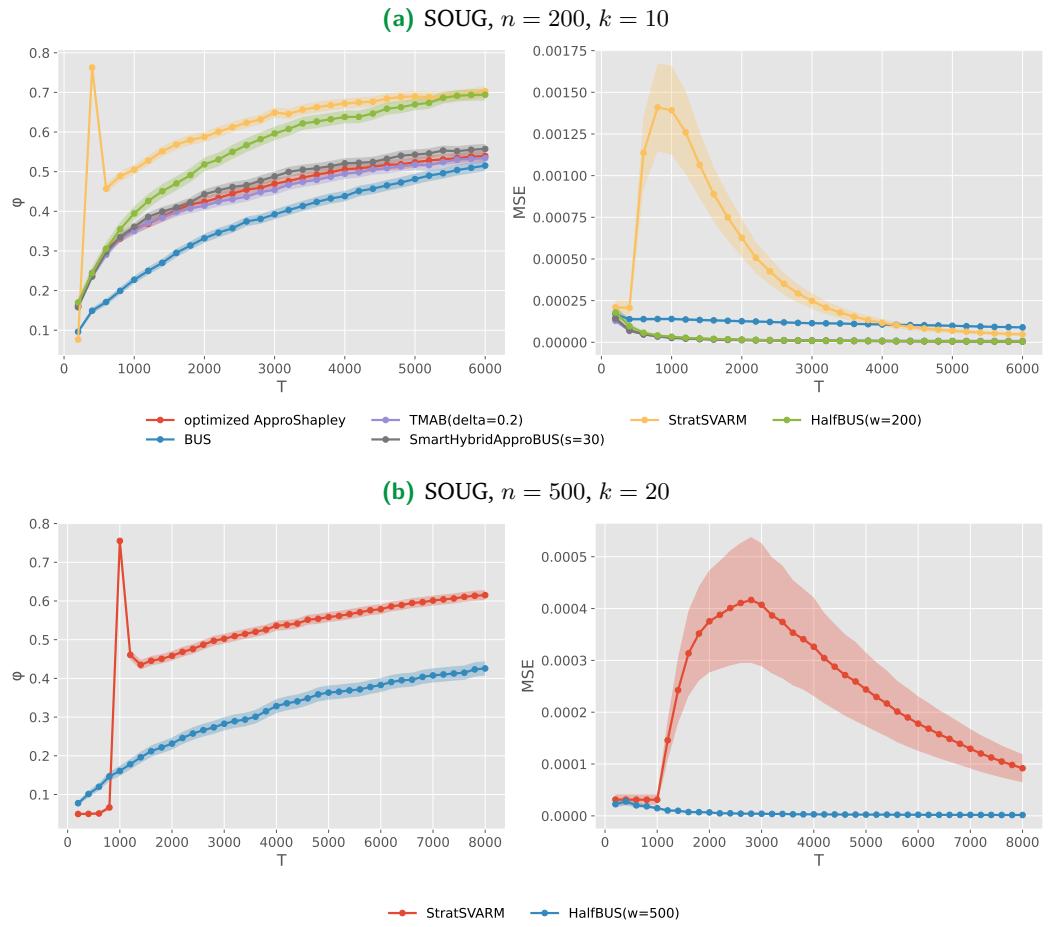


Fig. 4.5: Evaluation on larger player sets, averaged over 300 experiments

As expected, compared to the other algorithms the performance of StratSVARM increases with the size of the player set. For $n = 200$, it performs slightly better than HalfBUS, where HalfBUS is able to catch up in the end. However, for $n = 500$ StratSVARM clearly outperforms HalfBUS.

The spike of StratSVARM's curve at the beginning comes from the algorithm's exact calculation phase where the exact values for the stratas of length $0, 1, n - 1$ and n are calculated which apparently already results in an accurate ordering of the players in SOUG games. The drop in performance afterwards is due to the fact, that StratSVARM does not update whole marginal contributions, which makes the approximation inaccurate at the beginning as the coalitions used to update $\hat{\phi}^+$ might not be easily comparable to the coalitions used to update $\hat{\phi}^-$. This effect is also tracked by the MSE in fig. 4.5. However, the comparison between φ and the MSE also shows that the performance of an algorithm on the top-k problem is not necessarily strongly correlated to its approximation accuracy.

Remark 4.1.1 (Computational expense of StratSVARM). Since StratSVARM updates all players with each sample, it requires way more computational power compared to the other algorithms. Even though we tried to optimize the code for updating the players as much as possible it requires roughly five times the amount of time to finish compared to the rest. The experiment in fig. 4.5b took over five hours for StratSVARM while HalfBUS finished in just over an hour. The increased memory requirements of StratSVARM are also significant for large player sets.

4.2 Measuring Pixel Importance

In XAI, it is desirable to understand how a model arrived at a certain conclusion. In the case of image detection we are interested in which pixels influenced the model the most in making a specific decision.

To identify these influential pixels, we can create a local feature importance game by treating each pixel as a player and using the model's performance as the value function.

4.2.1 Experiment setup

We trained a simple linear neural network on the Fashion-MNIST dataset [XRV17] to classify images of clothing into 10 categories: 'T-Shirt', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', and 'Ankle Boot'. The model consists of an input layer with 784 neurons (one for each pixel in the 28x28 image), one hidden layer with 512 neurons, and an output layer with 10 neurons (one for each class). We achieved an overall test accuracy of 71.9%.

When running the model on a sampled image, such as in Fig. 4.6, it can correctly classify the item as a coat.

We create the cooperative game where each of the 784 pixels represents a player.

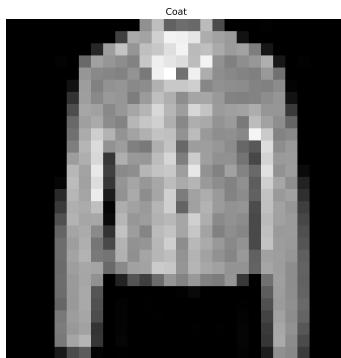


Fig. 4.6: Sample from the Fashion MNIST test dataset: Coat

The value function takes a coalition of players and runs the model with the remaining pixels being zeroed out. However, merely setting these values to zero (resulting in black pixels) would not be sufficient since black pixels can still carry information just like white pixels. To address this, we replace the zeroed-out pixels with their average value in the dataset, as illustrated in fig. 4.7. This kills the information

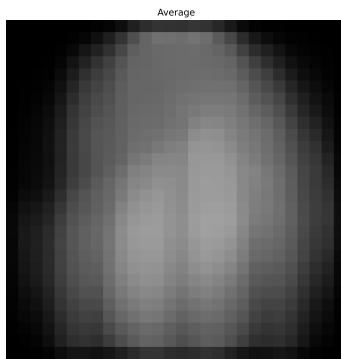


Fig. 4.7: Average pixel values of Fashion MNIST test dataset

of this pixel. In order to quantify the model's performance on the sampled image

with partially zeroed out pixels we calculate the cross entropy loss using the model’s output and the actual value. Since the loss decreases as the model’s performance improves, the value function returns the inverse of the loss. To ensure that $v(\emptyset) = 0$ when no pixels are included, we subtract the value of running the model with all pixels zeroed out. In practice this would not be necessary since we do not care about the exact Shapley values, only the top-k players.

4.2.2 Results

We evaluate the feature selection game on HalfBUS using $w = 100$, along with the optimized version of ApproShapley and Stratified SVARM, as shown in figs. 4.8 to 4.10. Unlike in section 4.1.2 we cannot plot the algorithms’ performances as we do not know the actual top-k players. Correctly identifying them would require the evaluation of all 2^{784} coalitions which we are unable to carry out. However, by running the algorithms for longer and comparing these results to outputs with smaller budgets, we get an indication of how fast the model is able to identify the top-k.

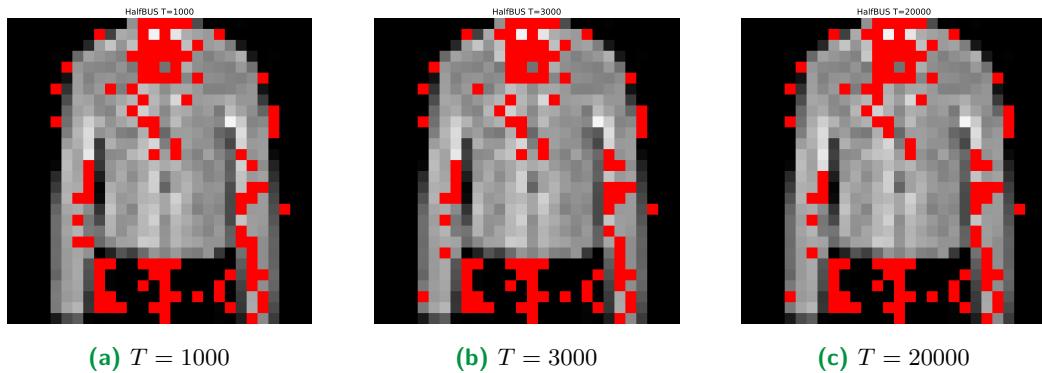


Fig. 4.8: Resulting top-100 pixels (red) of HalfBUS with Budget T using $w = 100$

The red pixels indicate the top 100 pixels identified by the algorithms for the sampled image of a coat.

Upon inspecting the results, it appears that the model focuses primarily on the area around the neck, the contours of the coat, and the black area at the bottom. The latter may be crucial in distinguishing it from, for example, a dress, which is usually longer.

Notably, HalfBUS delivered the best results, as after only 1000 value function calls, it seems to provide a good estimation of the top-100 pixels as its selection only changes slightly when running it up to a budget of 20000.

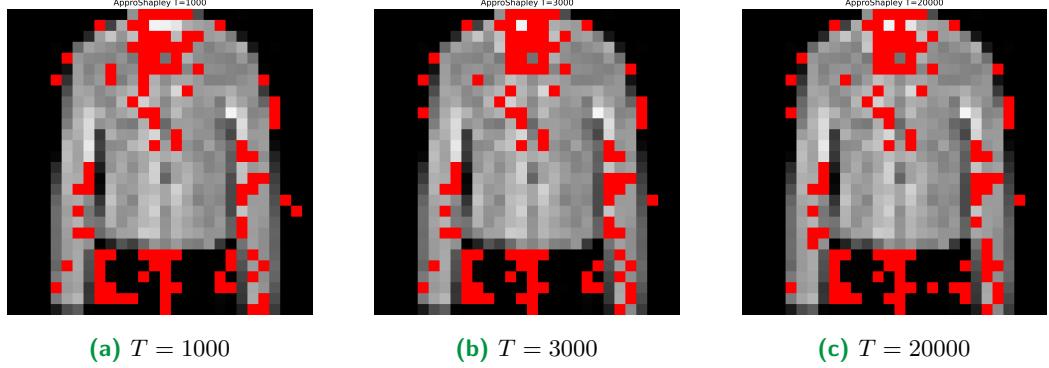


Fig. 4.9: Resulting top-100 pixels (red) of optimized ApproShapley with Budget T

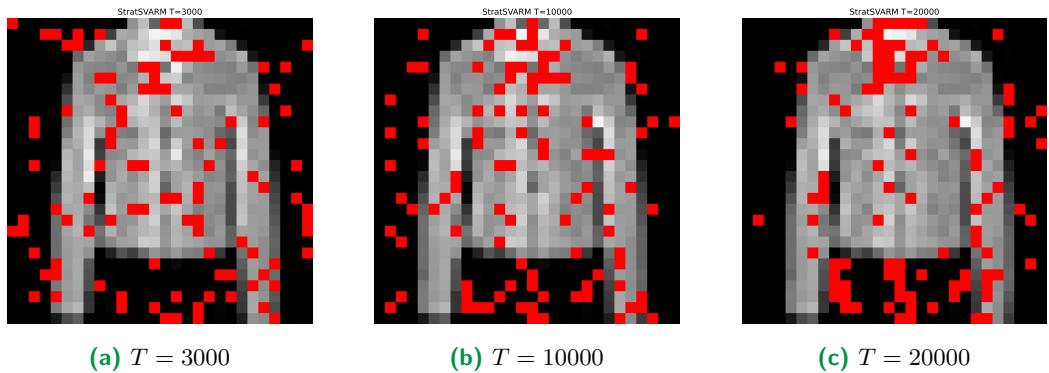


Fig. 4.10: Resulting top-100 pixels (red) of StratSVARM with Budget T

ApproShapley achieved a very similar result to HalfBUS after 20000 calls, which strongly suggests that these results accurately describe the actual top-100 pixels. However, it seems to perform slightly worse than HalfBUS, as its selection varies slightly up to a budget of 3000.

On the other hand, Stratified SVARM does not seem to perform well, as its selection appears to be quite random. Only after around 20000 calls do the selected pixels start to group together similarly to the other results.

4.3 Neuron Shapley

4.3.1 Experiment setup

We try to replicate some of the experiments conducted by Ghorbani et al. [GZ20]. The authors focused on CNNs applied on image detection.

In the first experiment we try to identify a set of critical neurons in the Inception-v3 [Sze+16] model using different algorithms. The used model is trained on the ImageNet dataset ILSVRC2012 [Rus+15]. Like Ghorbani et al. [GZ20] we split the ImageNet test dataset in a validation set and a test set where the validation set is used to calculate Shapley values. The model achieves an accuracy of 76.8% on our test set. Similar to Ghorbani et al. [GZ20] we create a cooperative game where the 17216 convolutional filters of the model (excluding the last layer, also called logit layer) represent the player set. The accuracy of the model, evaluated on a batch of 64 images, is chosen as an approximation for the value function. In theory, to get a well defined result for the value function the accuracy of the model on the whole validation dataset would have to be calculated. This however, would dramatically increase the runtime which is not feasable for our hardware constraints. Ghorbani et al. approximate the value aswell using a batch size of 128. Coalitions of players are formed by "zeroing" the remaining players. Players are zeroed by fixing their output to their average output as explained in section 2.4.2. We run the algorithms on this cooperative game and evaluate their performance by zeroing the identified top-k players and evaluating the model's accuracy on the test dataset. If the algorithm successfully identified highly important neurons, the accuracy of the model should drop. Additionally, we evaluate how strongly the model is impacted when the players with the lowest Shapley values are removed.

Ghorbani et al. [GZ20] test whether it is possible to fix a NN's bias by removing the responsible neurons. They trained the SqueezeNet [Ian+16] architecture on the CelebA [Liu+18] dataset to detect the gender from face images and achieve a test accuracy of 98%. We tried to recreate this setting but were only able to achieve a test accuracy of 97.1%, possibly due to minimal deviating model constructions which is a result of missing documentation by the authors. The model is tested on a different dataset to test whether it has a certain bias towards certain ethnicities. Similar to the Inception model we use the 2976 convolutional filters as players and the model's accuracy on a batch of 256 images as the value function. As we were not able to get access to the PPB dataset [BG18] as used by Ghorbani et al. we use the FairFace dataset [KJ21] as it has an equal representation of different ethnicities aswell. We try to recreate Ghorbani et al.'s experiment [GZ20], where the Shapley values are calculated using the dataset with fair representations of all ethnicities and test if the model's performance increases or biases are removed when the harmfull players are removed.

Additionally we calculate the Shapley values on a single ethnicity in order to target a certain bias explicitly by removing the worst players again.

Ghorbani et al. [GZ20] run their experiments using a cluster of 100 computing machines each with 12 CPU cores taking a total of 21 hours per experiment. In our setting this is equivalent to a budget of $T \approx 4.5 \cdot 10^7$. Due to hardware limitations we are only able to allow for a budget of $T = 50000$ per experiment.

4.3.2 Results

We remove the top-10 and top-100 Shapley players of the Inception model calculated using the algorithms HalfBUS, TMAB, and TruncStratSvrm. For HalfBUS we implement a simple form of truncation aswell, similar to TruncStratSvrm. For both algorithms, we ignore coalitions of sizes smaller than 16000. Removing up to 100 neurons from the model only barely impacts its accuracy if chosen randomly. The results are shown in fig. 4.11.

Not surprisingly, HalfBUS and TMAB are not able to identify highly important neurons using a budget of only $T = 50000$. Due to their budget efficiency and the total number of players of 17216 both are only able to update each player a few times. TruncStratSvrm however, is able to impact the model's performance already using a budget of $T = 20000$ and completely diminishes its accuracy after removing the top-10 players after 50000 calls. This is a major improvement over the results of Ghorbani et al. [GZ20]. Using TMAB they where only able to drop the model's accuracy on their test set from 74% to 38% and used a budget of roughly $T \approx 4.5 \cdot 10^7$. TruncStratSvrm is able to drop the model's accuracy on our test set from 76.8% to 15.7% using $T = 50000$ which is just a fraction of the author's budget. This can be explained using TruncStratSvrm's budget efficiency being proportional to the number of players. After 50000 calls, each player is updated 25000 times while Ghorbani et al. reportedly ran the algorithm for 3000 iterations which roughly equals 3000 updates per player considering TMAB has a similar budget efficiency to ApproShapley. This way we are able to produce better results than Ghorbani et al. [GZ20], even with a way lower budget. Figure 4.12 shows the results of removing the worst-k players of the model. Again TruncStratSvrm perfoms the best, retaining a lot of the model's accuracy, even after removing 1000 players.

Our SqueezeNet model, trained on the CelebA dataset, is able to achieve an accuracy of 74.2%. Unlike Ghorbani et al. [GZ20] we are not able to increase the model's performance by removing the worst neurons identified by any of the algorithms using our fair dataset FairFace [KJ21]. Neither did the accuracy increase for a

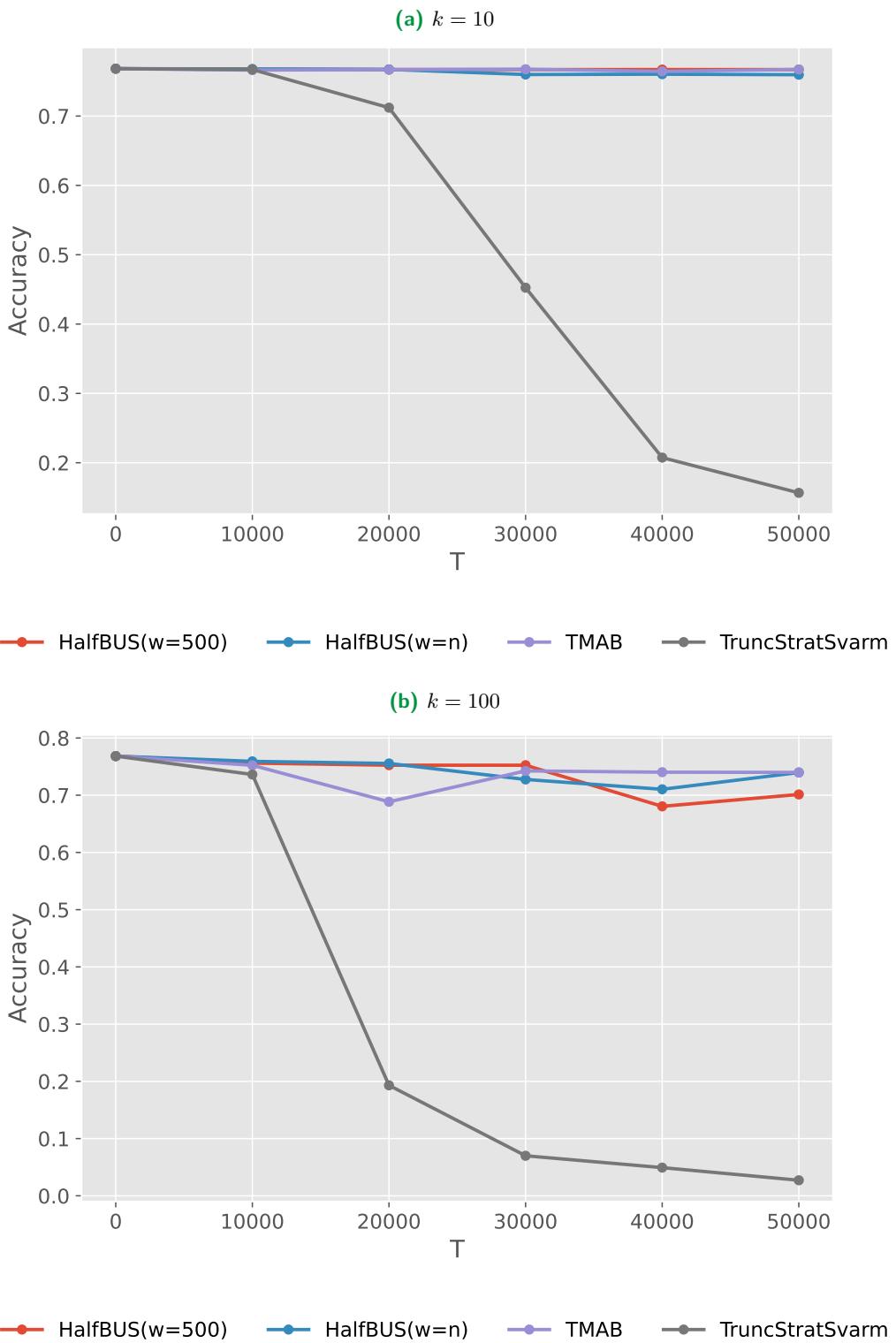


Fig. 4.11: Performance comparison of Inception model after removing the top- k players identified by different algorithms.

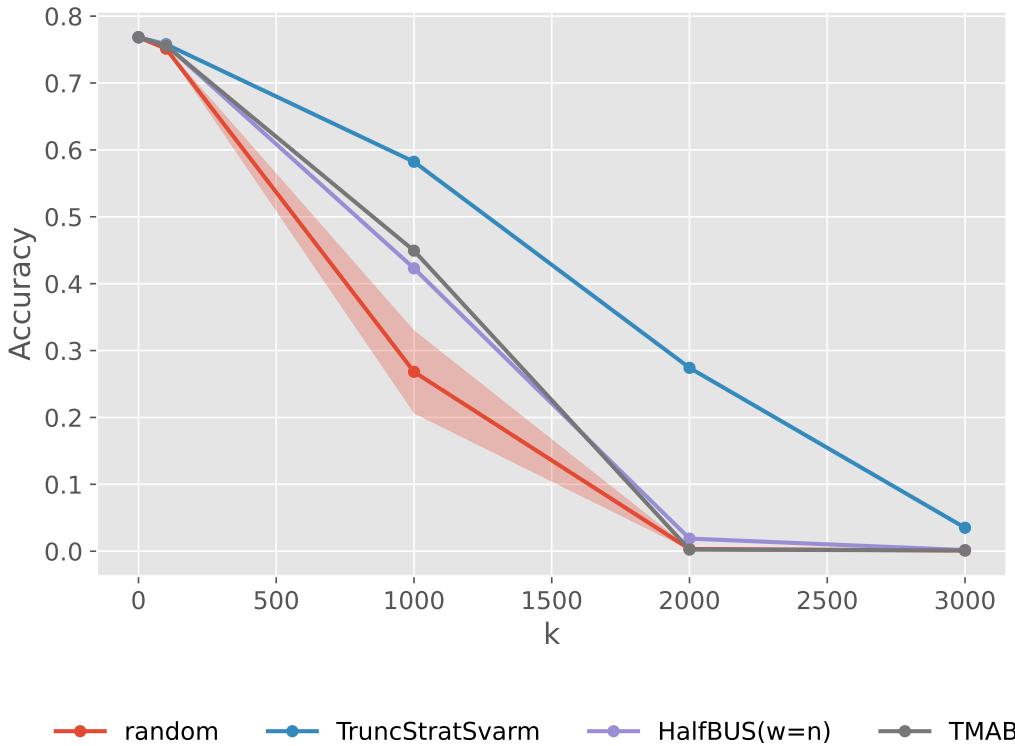


Fig. 4.12: Performance comparison of Inception model after removing the worst- k players identified by different algorithms using a budget of $T = 50000$.

minority group.

However, we conducted a second experiment. The model performed the worst for faces of people from southeast asia with an accuracy of 70.0%, probably due to an underrepresentation of this ethnicity in the CelebA dataset. We tried to specifically target this bias by calculating the Shapley values only using faces of this ethnicity. We were able to increase the model’s accuracy on that minority group. Removing the worst 3 neurons, results in an accuracy of 71.3% for TruncStratSvarm, 71.4% for HalfBUS and 71.1% using TMAB on this minority while the accuracy on the whole dataset dropped to 71.7% for TruncStratSvarm and 73.7% for HalfBUS. Using TMAB the accuracy on the whole dataset even slightly increased to 74.8%.

All these results however, are not as significant as those by Ghorbani et al. [GZ20]. Since the authors used a different dataset it is hard to compare the results. Possibly, the data of FairFace is too different to the CelebA dataset to even apply the model meaningfully.

Conclusion and future work

In this thesis we gave an overview of Shapley values and the more specific Top-k Shapley player problem and its applications in XAI. We tested state-of-the-art approximation algorithms on the top-k problem and investigated new ideas by introducing the algorithms SmartHybridApproBUS and HalfBUS. We evaluated all algorithms thoroughly in order to find core mechanisms that have a strong impact on an algorithm's performance for the top-k problem.

Although there is no dominant algorithm that performs best across all considered experiments, we observed that StratSVARM and HalfBUS tend to yield superior results. Particularly, when it comes to applying Shapley values to identify highly significant neurons within a neural network, StratSVARM emerges as the clear top performer. The reason is its budget efficiency that increases proportional to the amount of players. For smaller player sets however, HalfBUS achieved the best results. Even though StratSVARM still has a better budget efficiency, HalfBUS highlights the significance of comparability between the players' estimators. For the approximation problem all Shapley values can be approximated individually as they are solely compared to their real value. This is not the case for the top-k problem as the players need to be partitioned by their size. Thus, approximating all players using similar coalitions as done in HalfBUS has a strong positive effect on comparability even with a lower budget efficiency.

By applying TruncStratSVARM on a neural network in order to identify the most important neurons we were able to achieve even better results than TMAB with a budget several magnitudes lower. The reason again is Stratified SVARM's budget efficiency. With our set budget of 50000 in each experiment TruncStratSVARM was able to update every player's average marginal contribution 25000 times. TMAB, on the other hand, even with a budget of roughly $4.5 \cdot 10^7$ per experiment [GZ20], was only able to update each player about 3000 times since its budget efficiency is similar to optimized ApproShapley's.

In addition we were able to conduct our experiments on a single GPU in comparison to Ghorbani et al. [GZ20] who used a cluster of 100 computing machines with 12 CPU cores each.

For future work, it might be reasonable to investigate on the behaviour of HalfBUS further, as it demonstrates that for the top-k problem, the factor of comparability between the players' estimators plays an important role. This aspect has not drawn much attention yet as it is irrelevant with regards to the approximation problem alone. Therefore, we strongly suggest taking this factor into account for future top-k algorithm research.

Bibliography

- [Agg+18] Charu C Aggarwal et al. “Neural networks and deep learning”. In: *Springer* 10.978 (2018), p. 3 (cit. on pp. 10, 12).
- [Bro+20] Tom Brown, Benjamin Mann, Nick Ryder, et al. “Language models are few-shot learners”. In: *Advances in neural information processing systems* 33 (2020), pp. 1877–1901 (cit. on pp. 1, 2, 10).
- [BG18] Joy Buolamwini and Timnit Gebru. “Gender shades: Intersectional accuracy disparities in commercial gender classification”. In: *Conference on fairness, accountability and transparency*. PMLR. 2018, pp. 77–91 (cit. on p. 44).
- [CGT09] Javier Castro, Daniel Gómez, and Juan Tejada. “Polynomial calculation of the Shapley value based on sampling”. In: *Computers & Operations Research* 36.5 (2009), pp. 1726–1730 (cit. on pp. 3, 16, 22, 33).
- [GZ20] Amirata Ghorbani and James Y Zou. “Neuron shapley: Discovering the responsible neurons”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 5922–5932 (cit. on pp. 3, 8, 13, 14, 22, 23, 38, 43–45, 47, 49).
- [HB18] Ayanna Howard and Jason Borenstein. “The ugly truth about ourselves and our robot creations: the problem of bias and social inequity”. In: *Science and engineering ethics* 24 (2018), pp. 1521–1536 (cit. on p. 2).
- [Ian+16] Forrest N Iandola, Song Han, Matthew W Moskewicz, et al. “SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and < 0.5 MB model size”. In: *arXiv preprint arXiv:1602.07360* (2016) (cit. on pp. 14, 44).
- [KJ21] Kimmo Karkainen and Jungseock Joo. “FairFace: Face Attribute Dataset for Balanced Race, Gender, and Age for Bias Measurement and Mitigation”. In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 2021, pp. 1548–1558 (cit. on pp. 44, 45).
- [KBH23] Patrick Kolpaczki, Viktor Bengs, and Eyke Hüllermeier. “Approximating the shapley value without marginal contributions”. In: *arXiv preprint arXiv:2302.00736* (2023) (cit. on pp. 3, 6, 7, 18, 19, 26, 39).
- [KBH21] Patrick Kolpaczki, Viktor Bengs, and Eyke Hüllermeier. “Identifying Top-k Players in Cooperative Games via Shapley Bandits.” In: *LWDA*. 2021, pp. 133–144 (cit. on pp. 3, 8, 9, 20, 26, 32, 37).
- [KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks”. In: *Advances in neural information processing systems* 25 (2012) (cit. on p. 1).

- [LBH15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep learning”. In: *nature* 521.7553 (2015), pp. 436–444 (cit. on p. 1).
- [Liu+18] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. “Large-scale celebfaces attributes (celeba) dataset”. In: *Retrieved August 15.2018* (2018), p. 11 (cit. on p. 44).
- [LL17] Scott M Lundberg and Su-In Lee. “A unified approach to interpreting model predictions”. In: *Advances in neural information processing systems* 30 (2017) (cit. on p. 12).
- [MP09] Andreas Maurer and Massimiliano Pontil. “Empirical bernstein bounds and sample variance penalization”. In: *arXiv preprint arXiv:0907.3740* (2009) (cit. on p. 22).
- [Mio+18] Riccardo Miotto, Fei Wang, Shuang Wang, Xiaoqian Jiang, and Joel T Dudley. “Deep learning for healthcare: review, opportunities and challenges”. In: *Briefings in bioinformatics* 19.6 (2018), pp. 1236–1246 (cit. on p. 1).
- [Mol20] Christoph Molnar. *Interpretable machine learning*. Lulu. com, 2020 (cit. on pp. 3, 12).
- [Owe13] Guillermo Owen. *Game theory*. Emerald Group Publishing, 2013 (cit. on p. 33).
- [Ram+22] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. “Hierarchical text-conditional image generation with clip latents”. In: *arXiv preprint arXiv:2204.06125* (2022) (cit. on pp. 10, 11).
- [Roz+22] Benedek Rozemberczki, Lauren Watson, Péter Bayer, et al. “The shapley value in machine learning”. In: *arXiv preprint arXiv:2202.05594* (2022) (cit. on p. 2).
- [Rus+15] Olga Russakovsky, Jia Deng, Hao Su, et al. “Imagenet large scale visual recognition challenge”. In: *International journal of computer vision* 115 (2015), pp. 211–252 (cit. on p. 44).
- [Sel+17] Sreelekshmy Selvin, R Vinayakumar, EA Gopalakrishnan, Vijay Krishna Menon, and KP Soman. “Stock price prediction using LSTM, RNN and CNN-sliding window model”. In: *2017 international conference on advances in computing, communications and informatics (icacci)*. IEEE. 2017, pp. 1643–1647 (cit. on p. 1).
- [Sha+53] Lloyd S Shapley et al. “A value for n-person games”. In: *Princeton University Press Princeton* (1953), pp. 307–317 (cit. on pp. 2, 5).
- [Sze+16] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. “Rethinking the inception architecture for computer vision”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2818–2826 (cit. on pp. 13, 44).
- [Vas+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, et al. “Attention is all you need”. In: *Advances in neural information processing systems* 30 (2017) (cit. on p. 1).

- [XRV17] Han Xiao, Kashif Rasul, and Roland Vollgraf. “Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms”. In: *arXiv preprint arXiv:1708.07747* (2017) (cit. on p. 41).

Webpages

- [@AI22] Stability AI. *Stable Diffusion Launch Announcement*. 2022. URL: <https://stability.ai/blog/stable-diffusion-announcement> (visited on Aug. 11, 2023) (cit. on p. 10).
- [@Wei03] Eric W Weisstein. *Sample variance*. 2003. URL: <https://mathworld.wolfram.com/> (visited on July 20, 2023) (cit. on p. 10).

List of Figures

2.1	Samples generated by DALL-E 2 [Ram+22] and the corresponding prompts	11
2.2	Simplified illustration of a neural network: input layer (green), hidden layers (blue), output layer (yellow)	11
2.3	Ghorbani et al.'s [GZ20] visualization of filters with the highest Shapley values for a selection of Inception-v3 layers. For each filter, 5 images with most positive or negative activation of the respective neuron are displayed giving the filter a meaningful interpretation. On the left of both columns they provide an image that is optimized to strongly activate the filter.	14
3.1	Illustration of the Shapley value estimators of four players on the number line and their respective update count t . Player 1 is the most uncertain as its estimator has been updated 10 times less than the others' while it is only twice the distance from the border than $\hat{\phi}_2$ and $\hat{\phi}_3$	21
3.2	Illustration of how the confidence bounds ω_i indicate whether the Shapley value of player i is certain to be smaller or greater than the Shapley value of player j	23
4.1	Comparison of SmartHybridApproBUS with different values for s on SOUG games using $n = 50, k = 10$, averaged over 200 experiments . . .	34
4.2	Comparison of HalfBUS with different values for w on different games, averaged over 300 experiments	35
4.3	Comparison of all algorithms on SOUG games using $n = 10, k = 3$, averaged over 300 experiments	37
4.4	Comparison of all algorithms on different games, averaged over 300 experiments	38
4.5	Evaluation on larger player sets, averaged over 300 experiments	39
4.6	Sample from the Fashion MNIST test dataset: Coat	41
4.7	Average pixel values of Fashion MNIST test dataset	41
4.8	Resulting top-100 pixels (red) of HalfBUS with Budget T using $w = 100$.	42
4.9	Resulting top-100 pixels (red) of optimized ApproShapley with Budget T .	43
4.10	Resulting top-100 pixels (red) of StratSVARM with Budget T	43

4.11 Performance comparison of Inception model after removing the top-k players identified by different algorithms.	46
4.12 Performance comparison of Inception model after removing the worst-k players identified by different algorithms using a budget of $T = 50000$. .	47

Colophon

This thesis was typeset with \LaTeX . It uses the *Clean Thesis* style developed by Ricardo Langner. The design of the *Clean Thesis* style is inspired by user guide documents from Apple Inc.

Download the *Clean Thesis* style at <http://cleanthesis.der-ric.de/>.

Declaration

I hereby declare that I have completed this bachelor's thesis solely and only with the help of the references I have mentioned.

Karlstad, September 17, 2023

Tim Nielsen

