# *JWA–PyTorch*: an Open-Source *PyTorch*-based Toolkit for the Joint Weighted Average Operator

Youssef Abdulghani, Christian Wagner
*Lab for Uncertainty in Data and Decision Making (LUCID)*
*School of Computer Science, University of Nottingham*
Nottingham, UK
{youssef.abdulghani, christian.wagner}@nottingham.ac.uk

Stephen B. Broomell, Sean P. Conway, Ethan C. Guthrie
*GRID—Global Risk and Individual Decisions Laboratory*
*Purdue University*
West Lafayette, IN 47907, USA
{broomell, guthriee, spconway}@purdue.edu

***Abstract*—Aggregation operators form the fundamental set of tools used in information aggregation and fusion. The Joint Weighted Average (JWA) was recently proposed as a new aggregation operator that blends two different weighted aggregation approaches; source-based and evidence-based. Specifically, the JWA enables the systematic, joint consideration of the well-known linear weighted average (LWA) and the ordered weighted average (OWA) operators–and their associated weighting strategies. Here, we present a *PyTorch*-based toolkit to facilitate adoption of JWA aggregation in practice as well as to learn JWA weights from data. The latter is critical, not only for machine learning applications, but in particular for leveraging the explanatory power of the underlying weights, for example, in social science research. We integrate the operator learning approach with the established supervised learning framework that *PyTorch* provides, offering access to powerful optimization. We showcase and demonstrate the functionality of the toolkit. The open-source *PyTorch*-based toolkit opens the door for further development and accessible experimentation with the JWA framework and offers the only freely available approach for supervised learning in this setting. The toolkit is lightweight, extensible, and easy to use, allowing researchers and practitioners from other disciplines to leverage and advance JWA, associated techniques and research.**

***Index Terms*—Aggregation, Software, Joint Weighted Average, Learning, Optimization, Python**

## I. Introduction

Information aggregation tools are essential in many disciplines and applications, from AI decision-support and decision-making, to understanding human decision-making and cognition. Carefully designed aggregation tools can offer a bridge that converts obscure data-driven autonomous systems to human-interpretable ones [1]. In recent years, the demand for explainable AI (XAI) has rapidly increased due to the need to build trust in AI systems, verify their safety and make them human-accessible. Additionally, from a computer science standpoint, it is often important to understand how the system works in order to improve it or adapt it into a more robust solution [1]. Another motivation is that XAI can serve as an instrument for modeling real-world processes.

In the context of aggregation operators, previous frameworks have broadly adopted one of two distinct weighting philosophies: assigning worth to each information source (source-based) or assigning worth to the values of the observed data (evidence-based) [2], [3].

Typically, the source-based strategy is applied using a Linear Weighted Average (LWA) operator, while the evidence-based strategy is applied using an Ordered Weighted Average (OWA) operator. In the past three decades, several efforts have been made to find more general operators which can combine both approaches. Those include the Weighted Ordered Weighted Average (WOWA) [4], Hybrid Weighted Average (HWA) [5], Ordered Weighted Average – Weighted Average (OWAWA) [6], and, more recently, the Standard Deviation Weighted Average (SDOWA) [7]. Although all these operators have the LWA or OWA as special cases and allow their strategies to be considered, it was argued recently that they still do not *both* systematically and meaningfully integrate the underlying strategies [8]. To address this discord, the Joint Weighted Average (JWA) [8] was introduced as a novel operator capable of consolidating these approaches by allowing the strategies to be combined simultaneously through the use of compositional geometry [9]. The importance of consolidating both approaches is that it offers a flexible aggregation strategy with a small set of interpretable parameters that can be harnessed for XAI [10], while also offering a route to nuanced interpretable modeling, such as in respect to human judgment [8].

Given the potential of the JWA framework in multi-disciplinary applications, this study presents *JWA-PyTorch*, a software toolkit designed to operationalize it. We bridge the gap between aggregation theory and application by providing a lightweight and extensible implementation.

The specific contributions of this work are as follows:

- Accessible JWA Implementation: We provide an open-source implementation of the JWA operator, enabling researchers to readily deploy joint source- and evidence-based aggregation, and accelerate the development of applications and associated research.
- Optimization Framework: We demonstrate how the toolkit can leverage machine learning (ML) techniques to optimize JWA weights in a supervised setting.
- Explainability Tools: We introduce a visualization module designed to help interpret the learned weights, enhancing the transparency of the aggregation process.

The remainder of this paper is structured as follows; theoretical

background is provided in Section II. In section III, the toolkit design and features are described. Section IV provides a demonstrative application of the software in a supervised learning context. Finally, Section V presents concluding remarks.

## II. Background

### A. Source- and evidence-led operators

*1) The Linear Weighted Average (LWA):* The LWA operator is defined for $n$ information sources by a weighting vector $\mathbf{w} = (w_1, \ldots, w_n)$ such that $w_j \in [0, 1]$ and $\sum_{j=1}^{n} w_j = 1$. The LWA is computed as a linear combination of the evidence provided by each information source $x_j$:

$$\mathrm{LWA}(\mathbf{x}) = \sum_{j=1}^{n} w_j x_j$$

*2) The Ordered Weighted Average (OWA):* The OWA operator [11] is defined for a set of $n$ evidence observations $\mathbf{x} = (x_1, \ldots, x_n)$ by a weighting vector $\mathbf{v} = (v_1, \ldots, v_n)$ such that $v_j \in [0, 1]$ and $\sum_{j=1}^{n} v_j = 1$. The OWA is calculated based on the sorted arguments $\mathbf{x}_\pi$, where, typically, $x_{\pi(j)}$ is the $j^{th}$ largest value $\mathbf{x}$:

$$\mathrm{OWA}(\mathbf{x}) = \sum_{j=1}^{n} v_j x_{\pi(j)}$$

The OWA is a non-linear aggregating operator as a result of the *ordering* step.

### B. Combining LWA and OWA

Shortly after the introduction of OWA and through the past three decades, several works attempted to combine the two approaches [4], [5], [6], [7].

Most recently, it was argued that those efforts fall short because they are either partial considerations of both, or due to generating outputs not in line with intuitive or mathematical expectations, see [8] for more details. The Joint Weighted Average (JWA) [8] was proposed as a combined operator that can consider both approaches simultaneously while maintaining expected behavior.

Given a source weight vector $\mathbf{w}$ and order weight vector $\mathbf{v}$, the JWA reorders both the evidence $\mathbf{x}$ and the source weights $\mathbf{w}$ by the same permutation function $\pi$. The definition of the JWA involves the perturbation operation ($\oplus$) from compositional geometry [9], which enables the systematic integration and manipulation of components that then forms an overall quantity:

$$\mathbf{w}_\pi \oplus \mathbf{v} = \left( \frac{w_{\pi(1)} v_1}{\sum_{k=1}^{n} w_{\pi(k)} v_k}, \ldots, \frac{w_{\pi(n)} v_n}{\sum_{k=1}^{n} w_{\pi(k)} v_k} \right) \quad (1)$$

Using 1, the JWA operation can be written as:

$$\mathrm{JWA}(\mathbf{x}) = \sum_{j=1}^{n} (\mathbf{w}_\pi \oplus \mathbf{v})_j \; x_{\pi(j)} \quad (2)$$

Unlike other combination methods, the JWA allows the weights to fully interact and systematically integrates both operators strategies [8].

### C. Learning the Operator Weights

A natural question that arises for any weighted aggregation operator in real-world applications is: What should the value of the weights be? One way of answering this question is to estimate weights from *observed* data using supervised learning [2], [12].

The resulting trained (fitted) models can then be used in the future to tackle tasks similar to the established baseline in an autonomous manner. Importantly, however, the weights of the trained model also provide a basis for explaining the aggregation process itself. This in turn is highly valuable from an XAI perspective, as well as critical in other disciplines, such as the social sciences, where the aim commonly is not to develop a predictive model, but to derive a model which explains the behavior of the process underpinning the data, e.g. [13], [14].

Training requires a set of data, each containing an input vector of evidence and an output $(\mathbf{x}_i, y_i)$ for each sample $i$. The optimization problem is canonically formulated to minimize errors between the calculated aggregate $\hat{y}_i$ and the ground truth aggregate value $y_i$. For weighted aggregation operators, the problem is subject to the weight constraints: $\sum w_i = 1$ and $w_i \in [0, 1]$. This is a standard constrained optimization problem, that can be solved through many different approaches. Nonetheless, a recurring theme across the various methods in the literature is iteratively updating the weights using gradient descent [15], [16], [17], [2].

A crucial aspect of the weight optimization problem using data is whether there is a unique solution, i.e. a unique combination of ideal weights [2], [12]. This may not be obvious, as depending on the properties of the information in the input data, there might be multiple or infinite weights that can produce the same output. We are not further exploring this question in this paper which is focused on software. However, we note that it is an important research question in its own right and is currently being explored in another study.

The literature on learning aggregation operator weights focuses on mathematical foundations [15], [18] without publicly available and easy-to-use software interfaces. Nevertheless, a series of works by Beliakov in the early 2000s [19], [20], [21], on the approximation and interpolation of aggregation operators led to the development of a closed-source interface *AOTools* [21]. Although *AOTools* was not maintained after 2010, much of the functionalities of it can still be found as part of the more general *fmtools* library developed by the same researcher. The *fmtools* library contains tools for more generally manipulating discrete fuzzy measures, a generalization of source-based weights which capture the worth of not just individual sources, but also of all their possible combinations, i.e. their power set. The library source code is publicly available and it has *R* and *Python* interfaces, which continue to be in active development. Although the *fmtools* library is well-developed, efficient, and user-friendly, the weight learning functionality is based on a rigid implementation with no native way to easily modify it.

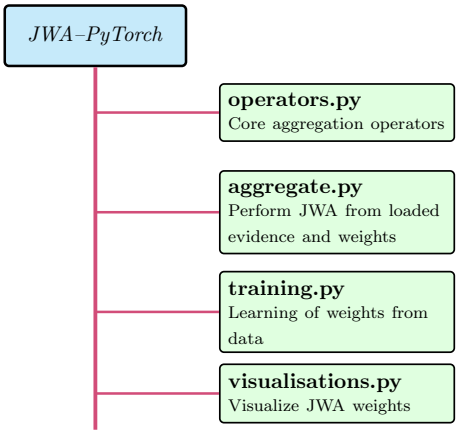


Fig. 1. This diagram shows the main modules in the *JWA–PyTorch* toolkit.

Given the parallels of the weight determination problem for aggregation operators and gradient-based neural network training, it is reasonable to expect that a framework built on established and flexible ML libraries such as *PyTorch* would be advantageous for the aggregation operators research community–which has been the primary motivation for the toolkit presented in this paper.

## III. *JWA–PyTorch* Toolkit and its Features

The toolkit leverages *PyTorch*'s ML framework to facilitate the efficient adoption of a variety of parameter learning approaches. In particular, the core component of the toolkit, which is the JWA aggregation process, is done by implementing it as an ML model using the `torch.nn.Module` as a base class. An overview of the toolkit structure is shown in Fig. 1. In this section, we first detail how the core JWA operation is implemented in the toolkit. We then discuss the `aggregate` and `training` modules, which can be used to quickly perform a JWA aggregation and learn JWA weights, respectively. Finally, we introduce the `visualisations` module, which enables the rapid production of convenient plots for visualizing the JWA weights, aiding in the explanation of the aggregation with respect to the given context.

### A. The JWA Aggregation Operation

The central concept of the JWA toolkit is the JWA aggregation operation. In order to perform this aggregation operation in a general and efficient manner, we first generalize the evidence vector from the one sample framework presented in Section II to a multi-sample dataset framework. We use a $k \times n$ evidence matrix $\mathbf{X}$, where $k$ is the number of samples, and $n$ is the number of information sources.

The ordered evidence matrix, is the matrix $\mathbf{X}_\pi$, where each sample $i$ in $\mathbf{X}$ is ordered such that $x_{\pi(i1)} \geq \ldots \geq x_{\pi(in)}$.

Since the order of the source weights will change according to the evidence values in each sample, we also generalize the $\mathbf{w}_\pi$ vector to the two dimensional matrix, $\mathbf{W}_\pi$. To obtain the JWA weights in this setting, we define the perturbation operation $\mathbf{A} \oplus_{\text{row}} \mathbf{b}$:

$$\mathbf{A} \oplus_{\text{row}} \mathbf{b} = \begin{bmatrix} \frac{a_{11}b_1}{\sum_{j=1}^{n} a_{1j}b_j} & \cdots & \frac{a_{1n}b_n}{\sum_{j=1}^{n} a_{1j}b_j} \\ \vdots & \ddots & \vdots \\ \frac{a_{k1}b_1}{\sum_{j=1}^{n} a_{jk}b_k} & \cdots & \frac{a_{kn}b_n}{\sum_{j=1}^{n} a_{jk}b_k} \end{bmatrix} \tag{3}$$

which performs the usual perturbation operation, as in (1), between each row in $\mathbf{A}$ and the vector $\mathbf{b}$. Operationally, in *JWA–PyTorch*, this is done by broadcasting the $\mathbf{b}$ vector to a matrix that has the same dimensions as $\mathbf{A}$, then calculating the Hadamard product (element-wise multiplication). In particular, $\mathbf{b}$ vector entries are replicated such that we obtain a matrix with the same number of rows as $\mathbf{A}$. Lastly, each row of the resulting matrix is normalized separately such that each row sums to unity.

Using the matrices and the perturbation operator as defined in the previous paragraph, the multi-sample JWA aggregation operation can be expressed as the following:

$$\text{JWA}_i(\mathbf{X}) = \sum_{j=1}^{n} (\mathbf{W}_\pi \oplus_{\text{row}} \mathbf{v})_{ij} x_{\pi(ji)} \tag{4}$$

Note that the output $\text{JWA}_i(\mathbf{X})$ is a column vector with $k$ entries, each entry in this vector represents the JWA aggregate for the $i$-th sample.

Interestingly, the operation in (4), can be done in an efficient and succinct way using *PyTorch*'s `einsum` function:

```
torch.einsum('ij,ji->i', weight_JWA, x_transpose)
```

The JWA model utilizes this in its forward method which is used when computing the aggregates.

The *operators* module contains the full implementation of the `JWA` *PyTorch* model in the conventional way that custom *PyTorch* models are implemented. The *operators* module also includes an explicit LWA model, the `LWA` class, for easy access. Additionally, the module includes a constrained version of JWA, the `constrained_JWA` class. The constrained version allows the learning of weights of either the LWA or OWA while the other is fixed to a certain weighting vector. For example, consider a set of source weight which are given, e.g. the accuracies of a set of sensors (LWA weights), we may be interested to still obtain a set of evidence weights (OWA weights) that when combined through the JWA framework yield an observed set of aggregates.

### B. The Aggregate Module

To provide a convenient way to perform a JWA aggregation operation using existing data, the toolkit provides the `aggregate` module. It is a simple wrapper that uses a command line interface to load a dataset in the form of tabulated data in a CSV file. This dataset reflects the *evidence* and should contain the values that need to be aggregated. The rows represent the different samples and the columns represent the different estimates of the various information sources. The CSV file should not contain any headers and should just consist of numerical data. Additionally, the *aggregate* module

requires user-specified weights in the form of CSV data. This file should contain two rows representing the LWA and OWA weights, while the columns represent the information source weight and order weight, for the LWA and OWA, respectively.

After obtaining the required input, the module initiates a JWA model instance with the weights equal to the user-specified weights. Lastly, it calls the JWA instance with the evidence dataset as input, which will compute the aggregates of the dataset and store them in an output CSV file, where each row is the aggregated value for the corresponding sample.

### C. Training a JWA Model

As mentioned before, there are several situations where learning the JWA operator weights from existing data is needed. The purpose of the `training` module is to load the evidence and existing aggregates, and then run an optimization routine to learn the JWA weights. The JWA learning problem can be defined in the following way; find an optimal set of LWA and OWA weights that combine to JWA weights such as to minimize a loss criterion in respect to the JWA output. Thus, when the routine converges to a final weight set, the individual LWA and OWA weights obtained can be utilized to semantically explain how the final JWA aggregates were obtained, shedding light on both the LWA and OWA components and associated strategies.

For ease of use, the pipeline was designed with a default optimizer and hyperparameters (e.g. the number of training epochs/iterations, and learning rate) so that it requires only the evidence and their aggregates in CSV files to search for a set of optimal JWA weights. The default training optimizer uses the mean squared error (MSE) as a loss function (criterion) and runs a stochastic gradient descent (SGD) routine to find a set of OWA and LWA weights that minimize the loss. Note that those defaults can be changed to any other loss function or optimizer that *PyTorch* supports. As an example, the default optimizer can be changed from

```
optimizer = torch.optim.SGD(model.parameters(), lr
    =10**(-5))
```

to an *Adam* [22] optimizer

```
optimizer = torch.optim.Adam(model.parameters(), lr
    =0.0001)
```

Formally, the learning problem for $k$ samples and $n$ sources in the default implementation can be written as:

$$\begin{aligned} \min_{\hat{\mathbf{w}},\hat{\mathbf{v}}} \quad & MSE = \frac{1}{k}\sum_{i=1}^{k}\left(\mathrm{JWA}_i(\hat{\mathbf{w}},\hat{\mathbf{v}},\mathbf{x}_i) - y_i\right)^2 \\ \text{s.t.} \quad & \sum_{j=1}^{n}\hat{w}_j = 1, \quad \sum_{j=1}^{n}\hat{v}_j = 1 \\ & \hat{w}_j \geq 0, \quad \hat{v}_j \geq 0 \end{aligned}$$

The $\mathbf{x}_i$ vector corresponds to a row in the evidence matrix $\mathbf{X}$, so the squared error is averaged across the samples. The above formulation is the constrained optimization version, it can be made such that it is unconstrained by using a transformation of the weights of form $w_i = e^{\lambda_i}/\sum_{i=1}^{n} e^{\lambda_i}$ , e.g. [15], [17].

In the toolkit, the training module learning rate for SGD is set at $10^{-5}$ by default, but can be optionally changed by invoking a CLI flag before training. Similarly, the default number of epochs is 2000 but can be changed using a CLI flag. For example, the following CLI flags `--epochs 4000 --lr 1e-3` runs the training process using 4000 epochs and a learning rate of $10^{-3}$.

**Algorithm 1** Training JWA using an evidence/aggregate pair

**Require:** evidence $\mathbf{X} \in \mathbb{R}^{k\times n}$, aggregates $\boldsymbol{y} \in \mathbb{R}^k$

**Defaults:** epochs $N \leftarrow 2000$, Learning rate $\eta \leftarrow 10^{-5}$
**warn** risk of degeneracy in learned weights
Initialize parameters: $\hat{\mathbf{w}} \sim \mathrm{Uniform}(0,1)$, $\hat{\mathbf{v}} \sim \mathrm{Uniform}(0,1)$
**for** $t = 1$ to $m$ **do**
  $\hat{\boldsymbol{y}}_{\boldsymbol{i}} \leftarrow \mathrm{JWA}_i(\hat{\mathbf{w}},\hat{\mathbf{v}},\mathbf{x}_i)$ *forward*
  $\mathcal{L} \leftarrow \frac{1}{k}\sum_{i=1}^{k}\left(\hat{\boldsymbol{y}}_i - \boldsymbol{y}\right)^2$ *MSE*
  Update $(\hat{\mathbf{w}},\hat{\mathbf{v}})$ with SGD step at rate $\eta$ *backprop*
**end for**
**return** $\hat{\mathbf{w}},\hat{\mathbf{v}}$

The learning process underlying the training module is summarized in Algorithm 1. As alluded to in Section II, depending on the nature of the input evidence, the weights might not be uniquely identifiable, so the training algorithm provides a warning to ensure that the user is aware of this issue.

A detailed characterization of the weight learning solution space would allow the learning algorithm to better determine whether optimal solution degeneracy is present or not, prior to training. However, this is out of the scope of this paper, but associated research is under way and will be presented in other research articles.

We acknowledge that the training algorithm does not contain more elaborate mechanisms such as early stopping or hyperparameter optimization. However, this is intentional to maintain simplicity. Furthermore, the *PyTorch* backend of the training pipeline means that there is great flexibility in changing the learning algorithm (e.g. adding early stopping), as previously demonstrated. It is worth to note that the *PyTorch* integration enables taking advantage of GPUs in problems where there is a large number of JWA weights and evidence, i.e. high number of information sources and samples.

### D. Visualizing the Weights

It is widely recognized that visualization can greatly help with human interpretability and decision making, e.g. [23], [24]. A key objective is to visualize JWA weights and, when possible, their decomposition into LWA and OWA weights, so that the respective strategies being integrated by JWA become explicit. LWA reflects the widely used linear/compensatory aggregation paradigm and is central in modeling human judgment [13], [14], whereas OWA captures order-based aggregation commonly used in decision-making and

multi-sensor fusion [11], [25]. Presenting these complementary views (JWA as a whole and its LWA/OWA components) supports explainability by revealing the balance and role of each strategy in the learned or user-defined aggregation.

The toolkit focuses on generating familiar and scalable visualizations. This ranges from bar charts to heatmap plots. It also includes ternary plots, which are a staple in the visualization of compositions with $n = 3$.

The toolkit also implements a novel scalable heatmap plot that shows color-mapped strength of the weights in either a 1D or a 2D representation, the former being designed for the direct visualization of weights independently from evidence. Conversely, the latter leverages permutations arising from a set of provided evidence in order to provide visualization that links the given evidence to how JWA performs its aggregation.

Overall, the *visualisation* interface includes the following plotting methods:

- `plot_ternary()` that produces ternary plots for systems with $n = 3$ (Fig. 3).
- `plot_bar_lwa()` and `plot_bar_owa()`, producing individual bar charts of source and order weights. `plot_bars()`, a side-by-side comparison of LWA and OWA weights (Fig. 4).
- `heatmap_weights_1D()`, which renders 1D horizontal heatmaps for LWA and OWA, regardless of the sample size (Fig. 5).
- `heatmap_weights_2D()`, a layout-adaptive routine that first displays the OWA row and then a lower block containing an ordered (based on data) LWA heatmap and a corresponding JWA weight heatmap (Fig. 6). The 2D routine automatically falls back to the 1D variant if the joint weights are absent, i.e., no samples are provided).

The 2D heatmap can be read one sample at a time, i.e., row by row. Each row corresponds to one input sample after the evidence has been ranked from highest to lowest according to the permutation induced by that sample. The top panel shows the OWA weights, which indicate how much importance is assigned to the rank positions themselves (e.g., largest, middle, smallest value). The lower LWA block shows the source weights after they have been reordered to match the same ranking for the given sample. The JWA block then shows the final combined weights used in the aggregation for that sample. In these heatmaps, the lighter color indicates a stronger contribution of that ranked value to the final aggregate, whereas the darker color indicates a weaker contribution. Thus, for any specific sample, the reader can follow a single row/column from the ordered evidence ranking to the reordered source weights and finally to the JWA weights to see which ranked observation contributed most strongly to the aggregate and why (their combined JWA weight strength).

The visualizations are produced with the `matplotlib` package [26] and `mpltern` [27], thus ensuring broad compatibility and customization flexibility.

To produce any visualization, the user simply instantiates a `visualisations` object using a JWA model from the `operators` module, for example:

```
from visualisations import jwa_vis
vis = jwa_vis(jwa_model)
```

From there, producing a weight visualization is achieved by calling the desired plotting method for that object. For instance, to produce side-by-side bar charts, we call:

```
vis.plot_bars()
```

This command produces an informative bar chart (an example is shown in Fig. 4) for the LWA and OWA weights that highlights the relative strengths of their associated evidence in the aggregation process.

## IV. Demonstration

In this section, a demonstration of the toolkit using a synthetic example problem is presented. We note that, in addition to the following demonstration, the toolkit contains an interactive *Jupyter* notebook that showcases the toolkit modules and functions as a quick-start guide.

### A. Generating Synthetic Data

For illustrative purposes, we consider a simple synthetic problem, where we fuse multi-sensor data with $n = 3$ information sources. This can represent, for example, a system monitoring temperature using three distinct sensors. Real-world data aggregation tasks typically involve more complex configurations, often with more than three sources. However, the example problem here is deliberately kept simple to enable accessible demonstration of the toolkit.

In this example, we assume that sensor 1 is the most reliable with a normalized reliability score of 0.6, sensor 2 is less reliable with a score of 0.3, and sensor 3 is the least reliable with a score of 0.1. The reliability scores will act as the LWA weights. To set up the OWA weights, we use a skewed median-like filter that puts the most weight on the middle temperature value.

Furthermore, it places the second highest weight on the maximum value and the least weight on the minimum value. Namely, for this example problem, we let the LWA weight vector $\mathbf{w}$ and the OWA weight vector $\mathbf{v}$ be.

$$\mathbf{w} = (0.6, 0.3, 0.1), \quad \mathbf{v} = (0.1, 0.85, 0.05) \tag{5}$$

Note that here, the value of $\mathrm{w}_i$ is a weighting for the source/sensor $i$. We simulate a synthetic sensor reading dataset with a sample size of $k = 6$. Here, each temperature reading $x_{ij}$ is generated by randomly sampling a real-valued number from a continuous uniform distribution over the interval $[5, 55]$. The simulated samples are shown in Table I.

### B. Obtaining JWA aggregates using Pre-defined Weights

To compute the JWA aggregates for this multi-sensor example, we simply save the simulated evidence (described in the previous paragraph) and the example weights as CSV files and run the *aggregate* module via the command line interface, e.g., if the evidence dataset is saved in `X_s3_t6.csv`, and weights are saved in `example_weights.csv`:

```
python aggregate.py --evidence X_s3_t6.csv --weights
    example_weights.csv
```

TABLE I
SYNTHETIC EVIDENCE $X \in \mathbb{R}^{6\times 3}$ AND CORRESPONDING AGGREGATES $\mathbf{y}$ FOR THE MULTI-SENSOR EXAMPLE PROBLEM.

| | Original Evidence | | | | Permuted evidence | | |
|---|---|---|---|---|---|---|---|
| Sample | $x_1$ | $x_2$ | $x_3$ | $y_i$ | $x_{\pi(1)}$ | $x_{\pi(2)}$ | $x_{\pi(3)}$ |
| 1 | 19.81 | 30.83 | 17.58 | 20.40 | 30.83 | 19.81 | 17.58 |
| 2 | 39.43 | 8.70 | 48.33 | 38.73 | 48.33 | 39.43 | 8.70 |
| 3 | 11.83 | 10.12 | 14.20 | 11.83 | 14.20 | 11.83 | 10.12 |
| 4 | 41.32 | 20.76 | 39.36 | 38.35 | 41.32 | 39.36 | 20.76 |
| 5 | 8.78 | 14.83 | 20.82 | 14.42 | 20.82 | 14.83 | 8.78 |
| 6 | 25.09 | 10.93 | 46.37 | 25.09 | 46.37 | 25.09 | 10.93 |

This, by default, will produce an `aggregates.csv` file inside a `results` folder that contains the computed JWA aggregates $y_i$, in 6 rows for this instance. Note that the JWA aggregates $y_i$ (as calculated by the aggregate module) are shown in Table I. Table I also makes explicit the sample-wise permutation $\pi(i)$ induced by the evidence $(x_1, x_2, x_3)$ in each sample. The permuted $\mathbf{x}$ sorted in descending order to produce the rank-indexed vector $(x_{\pi(1)}, x_{\pi(2)}, x_{\pi(3)})$, where $x_{\pi(1)}$ is the largest value, $x_{\pi(2)}$ the second-largest, and so on. This is precisely the permutation referenced in the JWA definition: the same $\pi$ that reorders the evidence also reorders the source weights into $w_{\pi(j)}$, ensuring that the source-based weights align with the rank positions before they are combined with the order weights $v$ to form the (sample-specific) JWA weights and compute the final aggregate $y_i$.

### C. Fitting a JWA model to Existing Data

To illustrate how the training module works, the (in our case very small number of) evidence/aggregate pairs in Table I can be used as its input:

```
python training.py --aggregate results/aggregates.
    csv --evidence X_s3_t6.csv
```

This will run the default training procedure as described in Algorithm 1. At the end of the run, our final MSE loss was 0.812 and the estimated weight vectors are:

$$\hat{\mathbf{w}} = (0.6040, 0.1979, 0.1981), \quad \hat{\mathbf{v}} = (0.2872, 0.5228, 0.1900)$$

This indicates that the algorithm is converging (see also Fig. 2), however, the default learning rate seems to be too small. Increasing the learning rate $\eta$ to $10^{-3}$ (using the `--lr` flag) leads to a final MSE of $1.36 \times 10^{-6}$ and the following estimated weight vectors, at the end of the 2000 epochs run:

$$\hat{\mathbf{w}} = (0.5998, 0.2999, 0.1003), \quad \hat{\mathbf{v}} = (0.0100, 0.8500, 0.0500)$$

Comparing these fitted weights to (5), reveals that the training module–in this case–was able to almost perfectly recover the ground truth LWA and OWA weights that form the JWA aggregates. The learning curves for these two training runs are shown in Fig. 2 for comparison.

It is worth emphasizing that this initial implementation of the training module should be treated as a starting point and the default settings should not be relied upon as ideal parameters. Conversely, the simplicity and flexibility of the module should enable users to readily adapt the implementation to their specific scenario. Further development of the training module may include setting the defaults to values that perform well more generally.

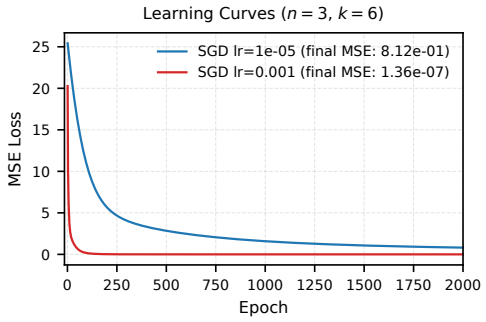


Fig. 2. This is a plot of the learning curves for the two example runs of the training module on the example problem in Table I.

### D. Weight Visualizations: Explainable JWA

Once a JWA model is defined or learned, the LWA, OWA, and, if applicable, the combined JWA weights can be visualized using the `visualisation` module and its various plotting methods, as detailed in Section III-D. Fig. 3 shows the OWA and LWA weights from our example in a ternary plot. Importantly, the relationship between *rank* and *source* can be seen as sample-dependent: for each sample $i$ (Table I), the evidence vector is sorted to obtain a permutation $\pi$, and the LWA weights are also permuted accordingly; the green JWA weights are then computed by the interaction of the OWA with permuted LWA weights.

Note how the points being far from the $v_3$ ($w_3$) vertex highlights, for example, that rank 3 (and whichever source falls into that rank for a given sample) receives a much lower weight than the other two ranks/sources. Additionally, the position of the LWA (OWA) weights closer to the source 1 (rank 2) vertex illustrates the strength of the weights relative to the other vertex of source 2 (rank 1).

Side-by-side bar charts of the LWA and OWA weights are shown in Fig. 4. Beyond quickly conveying the relative strength of individual weights (via easily comparable bar heights/areas), these plots provide a practical interpretability advantage: the implied aggregation strategy is typically not known a priori. By inspecting the LWA and OWA profiles, one can infer whether the aggregation is behaving more linearly (LWA-like) or more order-driven (OWA-like), and thus better understand the behavior of the overall aggregation process.

Lastly, the toolkit provides a heatmap plot that also shows visually how the weights compare to each other, for the LWA and OWA, in the 1D case. More crucially, the heatmap can be extended to a 2D representation, when a sample evidence matrix is loaded. The second dimension represents samples and the JWA weighting based on the inherent order. The 1D version is shown in Fig. 5, while the 2D heatmap leveraging evidence ordering information is plotted in Fig. 6.

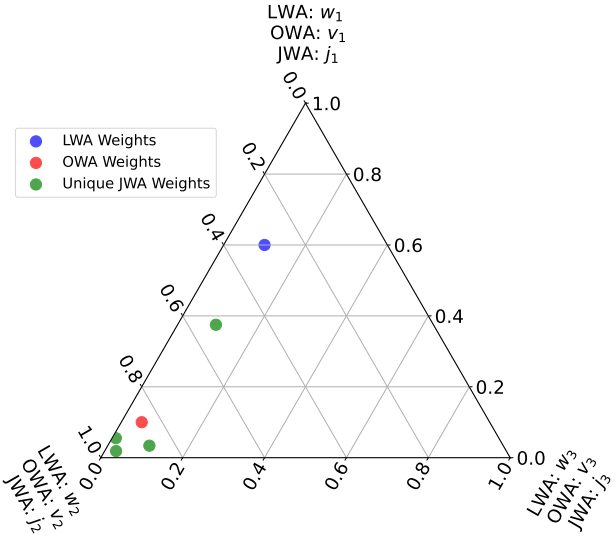


Fig. 3. Three integrated ternary plots (overlaid in respect to order permutations to conserve space) visualizing of the LWA, OWA and JWA weights in (5) and Table I.

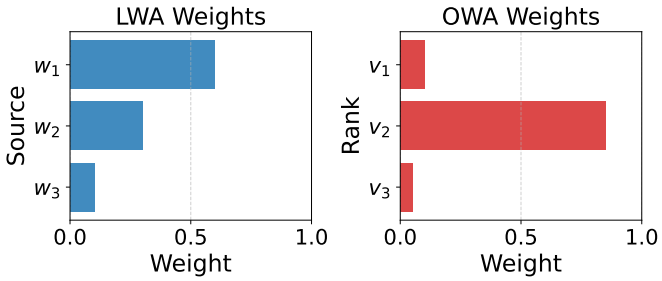


Fig. 4. Bar chart visualization of the LWA and OWA weights (5) which jointly produce the JWA weights in Fig. 3 and Fig. 6.

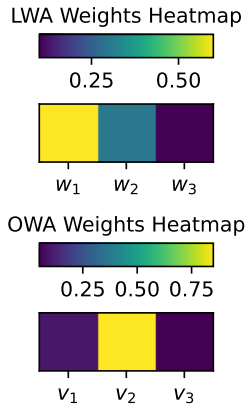


Fig. 5. A 1D heatmap visualization of the weights in (5).

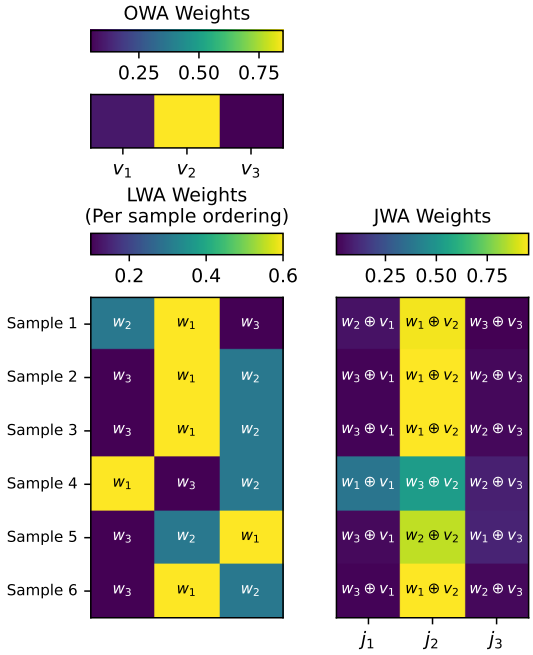


Fig. 6. A two-dimensional heatmap visualization of the weights in (5), evidence in Table I, and the corresponding JWA weights. The rows correspond to samples, and column correspond to information source. Lighter colors indicate greater weighting strength in JWA aggregation. The plot shows OWA weights (top-right), ordered LWA weights (bottom-left), and the combined JWA weights (bottom-right).

Inspecting the 2D color-mapped representation gives an intuitive approach to explain how the LWA and OWA weights interact to form the corresponding JWA weights for each sample. Moreover, using the heatmap and examining Table I, one can clearly explain the JWA aggregates ($y_i$). For instance, the JWA weights for sample 2 have the most weight (yellow color) for the middle value, which is the reading from sensor 1. This visually communicates the reason behind why the sample 2 aggregate $y_2 = 38.73$ is nearly equal to the reading from sensor 1 in sample 2, $X_{21} = 39.43$.

## V. Conclusions

We presented *JWA-PyTorch*, an open-source toolkit designed to bridge the gap between the theoretical formulation of the JWA operator and its practical application. By implementing the operator within the established and flexible *PyTorch* framework, we provide an efficient and extensible toolkit that enables researchers to perform aggregations and learn the JWA weights directly from observed data. Furthermore, the inclusion of a dedicated visualization module contributes towards the need for explainability in aggregation and AI. In particular, the visualizations offer intuitive graphical representations of how source-weighting and evidence-based strategies interact to form a final aggregate.

The toolkit provides a robust foundation for experimenting with the JWA, however, certain limitations exist. For example, there is no mechanism to handle order rank ties in the evidence dataset when two values are exactly equal. In practice, this

can arise if two sources provide the exact same evidence value, then when ordering the evidence, there is no preference between them in terms of the position that they should take in the ordered evidence vector. This might then lead to unexpected outcomes from the aggregation process, as the OWA weights might differ significantly between those two positions.

Moreover, as mentioned before, the training module relies on the user to change optimization hyperparameters and routines. The current implementation also assumes that there is no uncertainty in either the evidence or the aggregates. Future development of the toolkit should be well-positioned to address these limitations and more, especially given the open-source nature of the toolkit. The toolkit is available on GitHub (https://github.com/LUCIDresearch/JWA-pytorch), and we very much invite the community to try, use and contribute to its development.

Given the toolkit's compatibility with the extensive *PyTorch* ecosystem, we anticipate future work will explore the integration of JWA across other operators, including within ensemble methods and deep neural network architectures. This has the potential to extend the operator utility from solely tabular data fusion to complex learning tasks. Finally, we are hopeful that JWA-PyTorch implementation makes JWA accessible to other disciplines, such as the social sciences.

## ACKNOWLEDGMENT

This work has been supported by the joint *NSF* (SES-2343580) and *UKRI* (ES/Z000084/1) co-funded project: A Novel Theory of Ordered Judgment Processes.

AI Large Language Models (LLMs)—*Google-Gemini 3*, *OpenAI-ChatGPT 5*, and *Writfull* was used to refine language in some instances and to draft early versions of a few short excerpts. All text was then carefully reviewed and extensively edited by the authors, so no substantial portion can be considered solely AI-generated.

## REFERENCES

[1] B. J. Murray, M. A. Islam, A. J. Pinar, D. T. Anderson, G. J. Scott, T. C. Havens, and J. M. Keller, "Explainable ai for the choquet integral," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 5, no. 4, pp. 520–529, 2021.

[2] V. Torra and Y. Narukawa, *Modeling Decisions: Information Fusion and Aggregation Operators (Cognitive Technologies)*. Berlin, Heidelberg: Springer-Verlag, 2006.

[3] B. Llamazares, "On generalizations of weighted means and owa operators," in *Proceedings of the 7th conference of the European Society for Fuzzy Logic and Technology (EUSFLAT-11)*, pp. 9–14, Atlantis Press, 2011/08.

[4] V. Torra, "Weighted owa operators for synthesis of information," in *Proceedings of IEEE 5th International Fuzzy Systems*, vol. 2, pp. 966–971 vol.2, 1996.

[5] Z. S. Xu and Q. L. Da, "An overview of operators for aggregating information," *International Journal of Intelligent Systems*, vol. 18, no. 9, pp. 953–969, 2003.

[6] J. M. Merigó, A. M. Gil-Lafuente, and O. Martorell, "On the use of the uncertain induced owa operator and the uncertain weighted average and its application in tourism management," in *2009 Ninth International Conference on Intelligent Systems Design and Applications*, pp. 856–861, 2009.

[7] M. Cardin and S. Giove, *SDOWA: A New OWA Operator for Decision Making*, pp. 305–315. Singapore: Springer Singapore, 2021.

[8] S. B. Broomell and C. Wagner, "The joint weighted average (jwa) operator," in *Combining, Modelling and Analyzing Imprecision, Randomness and Dependence* (J. Ansari, S. Fuchs, W. Trutschnig, M. A. Lubiano, M. Á. Gil, P. Grzegorzewski, and O. Hryniewicz, eds.), (Cham), pp. 44–58, Springer Nature Switzerland, 2024.

[9] J. Aitchison, "The statistical analysis of compositional data," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 44, no. 2, pp. 139–177, 1982.

[10] D. Buffum, S. B. Broomell, C. Wagner, and D. T. Anderson, "The compositional nature of weights," in *Information Processing and Management of Uncertainty in Knowledge-Based Systems* (M.-J. Lesot, S. Vieira, M. Z. Reformat, J. P. Carvalho, F. Batista, B. Bouchon-Meunier, and R. R. Yager, eds.), (Cham), pp. 171–182, Springer Nature Switzerland, 2025.

[11] R. Yager, "On ordered weighted averaging aggregation operators in multicriteria decisionmaking," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 18, no. 1, pp. 183–190, 1988.

[12] M. Grabisch, J.-L. Marichal, R. Mesiar, and E. Pap, *Aggregation Functions*. Encyclopedia of Mathematics and its Applications, Cambridge University Press, 2009.

[13] R. M. Dawes and B. Corrigan, "Linear models in decision making," *Psychological Bulletin*, vol. 81, no. 2, pp. 95–106, 1974.

[14] N. Karelaia and R. M. Hogarth, "Determinants of linear judgment: A meta-analysis of lens model studies," *Psychological Bulletin*, vol. 134, no. 3, pp. 404–426, 2008.

[15] D. Filev and R. Yager, "Learning owa operator weights from data," in *Proceedings of 1994 IEEE 3rd International Fuzzy Systems Conference*, pp. 468–473 vol.1, 1994.

[16] R. Yager and D. Filev, "Induced ordered weighted averaging operators," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 29, no. 2, pp. 141–150, 1999.

[17] V. Torra, "Learning weights for weighted owa operators," in *2000 26th Annual Conference of the IEEE Industrial Electronics Society. IECON 2000. 2000 IEEE International Conference on Industrial Electronics, Control and Instrumentation. 21st Century Technologies*, vol. 4, pp. 2530–2535 vol.4, 2000.

[18] V. Torra, "On the learning of weights in some aggregation operators: the weighted mean and owa operators," *Mathware & Soft Computing*, vol. 6, no. 2-3, pp. 119–129, 1999.

[19] G. Beliakov, "Three new techniques of approximating aggregation operators from empirical data," 1 2002.

[20] G. Beliakov, "How to build aggregation operators from data," *International Journal of Intelligent Systems*, vol. 18, no. 8, pp. 903–923, 2003.

[21] G. Beliakov, "Learning weights in the generalized owa operators," *Fuzzy Optimization and Decision Making*, vol. 4, no. 2, pp. 119–130, 2005.

[22] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, 2014.

[23] L. M. Padilla, S. H. Creem-Regehr, M. Hegarty, and J. K. Stefanucci, "Decision making with visualizations: a cognitive framework across disciplines," *Cognitive Research: Principles and Implications*, vol. 3, no. 1, p. 29, 2018.

[24] K. Eberhard, "The effects of visualization on judgment and decision-making: a systematic literature review," *Management Review Quarterly*, vol. 73, no. 1, pp. 167–214, 2023.

[25] J. Wang and Q. Yu, "A dynamic multi-sensor data fusion approach based on evidence theory and wowa operator," *Applied Intelligence*, vol. 50, pp. 3837–3851, 2020.

[26] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[27] Y. Ikeda, "yuzie007/mpltern: 1.0.4," Apr. 2024.