Effective and Interpretable Information Aggregation with Capacity Networks

Markus Zopf
NEC Laboratories Europe GmbH
Heidelberg, Germany
markus.zopf@neclab.eu

Abstract—How to aggregate information from multiple instances is a key question in several machine learning problems such as multiple instance learning. Prior neural models implement different variants of the well-known encoder-decoder strategy according to which all input features are encoded to a single, high-dimensional embedding which is then decoded to generate an output. In this work, inspired by Choquet capacities, we propose Capacity networks. Unlike encoder-decoders, Capacity networks generate multiple interpretable intermediate results which can be aggregated in a semantically meaningful space to obtain the final output. Our experiments show that implementing this inductive bias leads to improvements over different encoder-decoder architectures. Moreover, the interpretable intermediate results make Capacity networks interpretable by design, which allows a semantically meaningful inspection, evaluation, and regularization of the network internals.

I. INTRODUCTION

In many important problems, a label is not given for each individual instance but only a set of instances. For instance, in histopathological classification of breast cancer images [1], only one label that indicates if a patient has cancer or not is given for a set of images. Similarly, in drug activity prediction [2], a label is only provided for a set of molecules, but for individual molecule. In sentiment analysis, only a single label may be given for a document that is comprised of multiple sentences, without labeling the sentiment on the sentence level [3], [4].

Problems with this characteristic are at the focus of multiple instance learning (MIL) [2], [5]–[7]. More formally, in MIL, an input consists of a set of instances $X \subseteq \mathcal{G}$, where $\mathcal{G}$ is a ground set of individual instances. The goal is to learn a set function $f$ from training pairs $(X_i, Y_i)$ that maps input sets to set labels. Supervision (i.e. labeling information) for individual instances is not provided.

A key question in MIL is how to aggregate the information from the multiple instances in the input sets. Prior neural approaches for MIL such as [8]–[11] implement different variants of the encoder-decoder strategy. Encoder-decoders use an encoder networks to aggregate the features of all individual instances $x_1, \ldots, x_n \in X$ into a single, high-dimensional set embedding $Z$. The set embedding is then decoded by a decoder network to generate an output $Y$. We provide a high-level illustration of sequential and parallel encoder-decoder networks in Figure 1a.

In this work, we present an alternative approach to aggregate information from multiple instances and demonstrate its effectiveness and interpretability. Inspired by a sequential decomposition of Choquet capacities [13], [14], we propose to decompose multiple instance learning tasks with sets of size $n$ into $n$ smaller sub-problems (i.e. one for each instance in the input) and to produce a meaningful intermediate result for each sub-problem. More specifically, for each instance $i$, an neural module generates an intermediate set embedding which is immediately decoded by another neural module to generate an intermediate result. Each intermediate result models the added value of instance $x_i$ with respect to all already seen instances $x_1, \ldots, x_{i-1}$. Like in Choquet capacities, all intermediate results can be simply summed to obtain the final output. In Figure 1, we illustrate the architectures of the resulting family of neural networks, which we denote as Capacity networks, and show how it differs from encoder-decoder architectures.

Capacity networks have two major advantages over their encoder-decoder counterparts: improved effectiveness and interpretability by design. More specifically, we find that Capacity networks achieve better results than prior works in a wide range of experiments. A potential explanation for the improved performance is that Capacity networks are better able to leverage the compositional nature of MIL problems. Moreover, Capacity networks are interpretable by design. Prior encoder-decoder architectures are end-to-end black boxes. In contrast, Capacity network generate multiple meaningful intermediate results that can easily be interpreted. The improved interpretability allows to evaluate if the trained models are right for the right reasons. Moreover, a quantitative evaluation of the intermediate results is possible. Furthermore, the intermediate results allow to inject semantically meaningful prior knowledge into the training process via regularization terms. To summarize, our contributions are as follows:

1) We propose Capacity networks that implement an alternative information aggregation strategy for multiple instance learning. Our key idea is to generate an interpretable, scalar-valued intermediate result for each instance that represents its added value with respect to all already observed instances and to sum all generated intermediate results to obtain the final prediction.

2) We show that Capacity networks are more effective than prior encoder-decoder architectures at learning challenging set functions. Moreover, we show that Capacity networks perform better for large sets, for varying set...
sizes, with smaller amounts of training data, and in a real-world sentiment analysis dataset.

3) We demonstrate the interpretability of Capacity networks, which enable a quantitative evaluation of the network internals and the implementation of a semantically meaningful regularization.

II. CAPACITY NETWORKS

In the following, we discuss an iterative decomposition of Choquet capacities, which motivate the fundamental architecture of Capacity networks.

A. Iterative Decomposition of Choquet Capacities

Let $\mathcal{G} = \{x_1, \ldots, x_m\}$ be a set of instances. A set function $\mu : \mathcal{P}(\mathcal{G}) \rightarrow [0, \infty)$ is called (Choquet) capacity [13], [14] if $\mu(\emptyset) = 0$ and $\mu(A) \leq \mu(B)$ for $A \subseteq B \subseteq \mathcal{G}$. Unlike measures such as the Lebesgue measure, capacities are not additive, which is essential to model non-additive aggregation effects, i.e. problems where $\mu(A \cup B) = \mu(A) + \mu(B)$, $\forall A, B \in \mathcal{G}$ with $A \cap B = \emptyset$ does not hold in general. A capacity $\mu$ is called superadditive if $\mu(A \cup B) \geq \mu(A) + \mu(B)$ and subadditive if $\mu(A \cup B) \leq \mu(A) + \mu(B)$, for $A \cap B = \emptyset$, $A, B \in \mathcal{P}(\mathcal{G})$.

Given a capacity $\mu$, it is natural to ask how large the individual contribution of a single instance $x_i$ is to a set $X = \{x_1, \ldots, x_n\}$, $x_i \notin X$. More formally: Given $X \subset \mathcal{G}$, $x_i \notin X$, how large is $\mu(X \cup \{x_i\}) - \mu(X)$? In the additive case (like in measures), the difference always equals zero $\mu(\{x_i\})$. However, in the non-additive case, the added value of $x_i$ to $X$ depends on $X$. The motivation to identify the individual contribution of an instance to a set utility naturally leads to the following sequential decomposition:

$$
\mu(X) = \sum_{i=1}^{\lfloor \frac{|X|}{2} \rfloor} \nu_i, \text{ with } \nu_i = \mu(C_{i-1} \cup \{x_i\}) - \mu(C_{i-1}), (2)
$$

where $C_{i-1} = \{x_1, \ldots, x_{i-1}\}$, and $C_0 = \emptyset$. The first summand $\nu_1 = \mu(C_0 \cup \{x_1\}) - \mu(C_0)$ equals $\mu(\{x_1\})$. In other words, $\nu_i$ represents the intrinsic value of $x_i$. Similarly, we obtain $\mu(\{x_1, x_2\}) - \mu(\{x_1\})$ for $\nu_2$, which can be interpreted as estimating the added value of $x_2$ with respect to $x_1$. Finally, $\nu_n$ equals to the added value of the last instance $x_n$ with respect to all other instances $x_1, \ldots, x_{n-1}$. The decomposition requires an ordering of the instances in $X$. Otherwise, it is unclear which instance should be considered first, second, etc.

To this end, any arbitrary total ordering can be used. The input in Figure 1, $X = \{\mathcal{S}, \mathcal{S}, \mathcal{Y}\}$, can, for instance, be decomposed as

$$
\mu(X) = \mu(\{\mathcal{S}\}) + \mu(\{\mathcal{S}, \mathcal{S}\}) - \mu(\{\mathcal{S}\}) + \mu(\{\mathcal{S}, \mathcal{S}, \mathcal{Y}\}) - \mu(\{\mathcal{S}, \mathcal{S}\}).
$$

According to this decomposition and the 'unique sum' task described in Figure 1, we first compute the individual contribution of image $\mathcal{S}$ (which is 8), then we compute the contribution of the next image $\mathcal{S}$ with respect to the already considered image $\mathcal{S}$ (which is 8), and finally compute the contribution of last image $\mathcal{Y}$ with respect to $\mathcal{S}$ and $\mathcal{S}$ (which is 0, since class '8' has already appeared once).

B. Network Architecture

Inspired by the sequential decomposition of capacities, we now define a new family of neural networks for multiple instance learning. Key idea is to introduce an inductive bias such that neural networks mimic the presented sequential decomposition of capacities. To this end, we propose to design networks such that they generate an intermediate result $y_i$ after reading instance $x_i$, which models the added value of instance $x_i$ to the set of all already seen instances $\{x_1, \ldots, x_{i-1}\}$. As in Equation 2, the output of the network for a set $X$ can be obtained by simply computing the sum of all intermediate
In terms of encoder and decoder functions, we define Capacity networks as

\[ z_i = \text{encoder}(x_i, z_{i-1}) \]
\[ y_i = \text{abs}(\text{decoder}(z_i)) \]
\[ Y = \sum_{i=1}^{\vert X \vert} y_i, \tag{6} \]

where \( z_0 \) is an initial state and \( \text{abs}(\cdot) \) denotes the absolute value of the decoder output. We compute \( \text{abs}(\cdot) \) to model the fact that each \( y_i \geq 0 \), since \( \mu(C_{i-1} \cup \{x_i\}) \geq \mu(C_{i-1}) \) due to the monotonicity of \( \mu \). In principal, this additional inductive bias is not essential for the networks architecture and can be removed if monotonicity should not be enforced. Furthermore, we do not explicitly model subtrahend and minuend explicitly but estimate the difference directly. Again, this is a non-essential design choice that can be modified in future work.

Capacity networks are closely connected the sequential decomposition of capacities presented in Equation 2. Each application of the decoder in Equation 5 corresponds to one \( y_i \) in Equation 2, which represents the added value of the \( i \)-th instance with respect to all already observed instances. Similar to the sequential decomposition of capacities in Equation 2, Capacity networks require a sequential ordering of the inputs. Otherwise, it is not possible to model the added value of an instance with respect to already seen instances. Any sequential network such as RNNs, LSTMs, and GRUs can be used as basis to implement the proposed inductive bias. Hence, we do not present a specific neural networks but rather a new family of neural networks. Capacity networks are not guaranteed to be permutation invariant. This is, however, not necessarily a limitation of the presented idea since sequential approaches have been demonstrated strong performance for MIL in prior works. Furthermore, several methods exist to mitigate or completely remove the permutation sensitivity of permutation sensitive networks as described in the next section. Permutation sensitivity can even be viewed as advantage since our networks can also be used if the output depends on the order of the input instances (e.g. in ordered sets or sequences).

A key distinction between prior encoder-decoder architectures and the newly presented idea is that prior encoder-decoders methods do not produce intermediate results, whereas Capacity networks generate \( n \) intermediate results, each of which modeling the added value of a specific instance. Another way to describe this distinction is to note that standard encoder-decoders only produce a single latent set embedding which is fed only once into a decoder to produce a scalar-valued output. In contrast, Capacity networks apply a decoder several times (once for each produced latent state \( z_i \)) to produce many latent scalar-valued intermediate results. As a consequence, encoder-decoders aggregate information from different instances in an uninterpretable high-dimensional feature space, whereas Capacity networks aggregate interpretable scalar-valued intermediate results. The inductive bias to generate scalar-valued intermediate results can also be viewed as introducing multiple information bottlenecks into the network architectures since the Capacity networks need to compress the feature representation to a meaningful, one-dimensional value multiple times.

### III. Related Work

Capacities and Choquet integration have already inspired other works in machine learning before. [15] fit values of the discrete Choquet integral with linear programming techniques. [16] use the Choquet integral to model monotone nonlinear aggregations for binary classification. [17] use the Choquet integral in a pairwise preference learning scenario. [18] replace pooling layers in convolutional neural networks with Choquet integration. In contrast to our work, they consider a classification setup and do not consider multiple instance learning. [19] and [20] use the Choquet integral to build an ensemble of classifiers for face recognition. Similarly, [21] and [22] build an ensemble of CNNs.

Many works on multiple instance classification [2], [5] and regression [6], [23] make strong task-specific assumption [7]. For instance, [2] assume that the label of a set is positive if at least one instance in the set is positive (commonly known as the standard MIL assumption [7]). Similarly, [6] assume that the target variable depends only on a single primary instance. TrueSkill [23] assumes additivity. These independence assumptions simplify the problem substantially, since models do not have to consider interdependencies between instances. In contrast, we consider more challenging non-additive problems. Prior neural approaches for MIL such as [8]–[10], [12] implement different variants of the encoder-decoder strategy. The approaches can be grouped into two groups, depending on whether the encoder network reads all input instances in parallel or sequentially (see Figure 1a).

Parallel architectures usually use a permutation invariant pooling operation such as sum, mean, or max to generate a set embedding [11], [8] show that permutation invariant architectures can be described in a fairly simple framework consisting of a network \( \phi \) that maps instance into a latent feature space, a sum pooling operation, and a decoder network \( \rho \). In this case, the encoder can be written as \( \text{encoder}(X) = \sum_{x \in X} \phi(x) = Z \) and the decoder is simply \( \text{decoder}(Z) = \rho(Z) \). Even though using a sum is sufficiently expressive from a theoretical point of view, [24] show that the networks can be highly sensitive to the choice of the aggregation function in practice. Similarly, PointNets [25] and PointNetST [26] are universal approximations of invariant and equivariant set functions, respectively. [24] propose a trainable recurrent aggregation function based on the read-process-write architecture [27]. Similarly, [9] and [28] use an attention-based weighted aggregation. The encoder can be written as \( \text{encoder}(X) = \sum_{x \in X} a_i \cdot \phi(x_i) \) where \( a_i = \frac{\exp h(x_i)}{\sum_{j=1}^{\vert X \vert} \exp h(x_j)} \) and \( h \) is an implementation-specific function to transform the input features. [11] present the self-attention-based [29] Set Transformer and describe how
our architecture fits into the encoder-decoder design pattern. Similarly, Deep Message Passing on Sets [30] can also model interactions between different instances.

Sequential architectures generate the embedding \( Z \) with a sequential encoder network. For instance, [27] compute attention weights based on the hidden state of an LSTM and the input instances. Hence, the encoder can be formulated as \( \text{encoder}(X) = \sum a_i x_i \), where \( a_i \) are the last layer’s attention weights. Since sequential encoder-decoders consume one instance at each timestep \( t \) [31], [32], the output may change when the input is permuted [27]. Several approaches exist to use permutation-sensitive models as basis for creating permutation-invariant models. A very simple yet effective approach is to order the objects in the input set according to an arbitrary total ordering \( \pi \) before they are fed into the permutation-sensitive model [33], [34] present a pooling method for sets of feature vectors which can be used to construct permutation-equivariant models. Janossy pooling [10] aggregates the outputs of permutation sensitive models to obtain a permutation-invariant model or an approximation thereof. [35] extend Janossy pooling and use an LSTM to construct a hierarchical feature aggregation network for set-of-sets problems. [36] learn how to order the instances in sets before they are fed into an LSTM. Similarly, [37] learn to reconstruct scrambled objects with Gumbel-Sinkhorn networks. Neural networks with external memories [38]–[40] such as RNNSearch [41], Memory Networks [42], and Neural Turing Machines [43] are alternative permutation sensitive approaches. Similarly, AMRL [44] is an permutation invariant external memory for reinforcement learning. Unlike the intermediate results in Capacity networks, the external memory is an uninterpretable accumulator of knowledge.

### IV. Effectiveness Evaluation

In the following, we evaluate the effectiveness of Capacity networks and compare it with their non-capacity counterparts and other prior architectures. To this end, we implement three Capacity networks based on RNNs, LSTMs [45], and GRUs [31]. We refer to the resulting Capacity networks as \( C \)-RNN, \( C \)-LSTM, and \( C \)-GRU, respectively. Table II shows the number of trainable parameters for each model. Most important, all sequential models (i.e. RNN, LSTM, and GRU) and their Capacity Network counterparts (i.e. \( C \)-RNN, \( C \)-LSTM, and \( C \)-GRU) have the same number of parameters. Hence, the performance differences observed in the experiments cannot be explained by different model sizes. Following [8], we use 3 fully connected layers for decoder and encoder. In all experiments, we feed the instances in random order into the networks. Additionally, we use a DeepSet [8], two Set Transformers [11] with different sizes\(^1\) and an attention-based network [32] as additional reference models. Mean squared error (MSE) is used as loss and evaluation metric. In all experiments, we report the average of three runs with different random seeds. Adam [46] is used as optimizer as it has been used by prior works [10] and shows good performance across several tasks and setups [47]. We perform the same hyperparameter optimization for all models.

#### A. Datasets

We follow prior works [8]–[10], [12] and generate input sets \( X_i \) and corresponding labels \( Y_i \) based on the MNIST [48] dataset. Similar to [10] and [12], we generate sets with 10 instances and use 100k input sets for training, and 10k sets for validation and test. We generate five challenging datasets:

In the subadditive MNIST-based Unique Sum (US) task [10], the goal is to learn the set function \( f_{US}(X) = \sum_{c=0}^{9} c \cdot I_X(c) \), where \( I_X(c) \) denotes the indicator function. In principle, the mapping from MNIST classes to indexes \( c \) is arbitrary. We map each class to the index corresponding to its numerical value, i.e. class ‘1’ is mapped to index \( c = 1 \), etc. Weighted Triangular (WTri) models superadditive effects according to \( f_{WTri}(X) = \sum_{c=0}^{9} c \cdot T_{countX(c)} \), where \( \text{count}_X(c) \) denotes the number of times MNIST image class \( c \) appears in set \( X \) and \( T_m \) is the \( m \)-th triangular number, i.e. \( T_m = \frac{m(m+1)}{2} \). Unique Sum + Synergy Bonus (US+S) combines both subadditive and superadditive effects in a single dataset by computing the label according to

\([^1\)We use the code provided by the authors [11] at github.com/juho-lee/settransformer.\]
results provide clear evidence that the introduced inductive capacity counterparts (i.e. RNNs, LSTMs, and GRUs). These almost all cases better than their directly comparable non-capacity counterparts in almost all cases (indicated by green bias has a systematic positive effect on the network performance. Third, datasets US and WTri are easier to learn than US+S and TriC. More difficult tasks (US+S) and more difficult images (TriC) can be explanations for this. Hence, using these datasets in future works will contribute to a better evaluation of neural MIL works. Furthermore, we find that permutation invariant networks do not have a systematic advantage over permutation sensitive models albeit the set functions to learn are permutation invariant. This is an surprising additional insight that can be investigated further in future work.

C. Experiments with Larger Set Sizes

In Figure 2, we report the performance of all architectures for the WTri task with larger set sizes. These experiments are more challenging and go beyond the scope of prior works such as [10] and [12] which only consider sets up to 10 instances. Again, we observe that Capacity networks achieve smaller errors that their non-capacity counterpart (dashed vs. solid lines). It should be noted that we use a log scale for the MSE, which means that the performance improvement when using Capacity networks is not approximately constant, but increases substantially with increasing set sizes. We also report the performance of other prior architectures and can confirm that they usually perform worse for larger set sizes.

D. Training with Smaller Amounts of Training Data

In Table III, we report the performance for different amounts of training data. Similar to the previous experiments, we observe that Capacity networks perform better than their non-capacity counterparts in almost all cases (indicated by green negative numbers).

E. Permutation Sensitivity Evaluation

We perform additional experiments to evaluate how sensitive the sequential architectures are with respect to the order of the input instances. In Figure 3, we plot the results of five different runs with randomly permuted instances. The experiments show that the permutation sensitivity of all models is rather low, except for the RNN. In addition to the results in Table I, the results show that permutation sensitivity seems to be a minor important issue in these datasets. Moreover, we

\begin{equation}
    f_{US+S}(X) = \sum_{i=0}^{9} c_i \cdot \mathbb{1}_X(c_i) + \sum_{i=0, j=1 \atop i < j}^{10} 10 \cdot \mathbb{1}_{\{c_i, c_j\}} \in P, \quad (7)
\end{equation}

where \( \mathbb{1}_{\{c_i, c_j\}} \in P \) indicates whether set \( \{c_i, c_j\} \) with MNIST classes \( c_i \) and \( c_j \) appear in a randomly generated set of pairs \( P \).

In addition to previously used tasks, we also perform experiments on a real-world Sentiment Analysis (Sent) dataset [3]. The dataset is based on work by [4] and available online\(^2\). A set in this dataset represents a document containing multiple sentences. The goal is to predict the overall sentiment of the documents. We use 300-dimensional feature representation to encode each sentence with a standard library\(^3\) and use the overall sentiment as described in [3] as label.

B. Main Results

We report the results for all datasets and all architectures in Table I and make several key observations. First, Capacity networks achieve overall the best results. \( C \)-LSTM and \( C \)-GRU perform better than \( C \)-RNN, indicating that Capacity networks benefit from a more complex hidden state update approach. Second, and more importantly, Capacity networks perform in almost all cases better than their directly comparable non-capacity counterparts (i.e. RNNs, LSTMs, and GRUs). These results provide clear evidence that the introduced inductive

\(^2\)Available at https://www.idiap.ch/paper/hatdoc.
\(^3\)Available at github.com/UKPLab/sentence-transformers.
find that Capacity networks can further reduce the permutation sensitivity for all three base architectures.

F. Results without Non-negative Intermediate Value Constraint

In Table IV, we report results without enforcing non-negative intermediate results by removing $abs(\cdot)$ in Equation 5 as already mentioned in Section II-B. We observe that the accuracy can further improve in several cases and that we achieve even new best results in three datasets (US, US+S, TriC). These results indicate that removing the monotonicity constraint can be helpful to achieve better results.

V. INTERPRETABILITY EXPERIMENTS

In the following, we demonstrate the improved interpretability of Capacity networks and further advantages of Capacity networks that are related to the improved interpretability.

A. Demonstration of the Improved Interpretability

To evaluate the interpretability of the intermediate results, we extract the intermediate values generated by Capacity networks from the trained models and show which intermediate values are expected. Table V illustrates the improved interpretability of Capacity networks in three different datasets. It can be seen that the networks learned the underlying set functions. For instance, in the first dataset, the $C$-RNN generates very good intermediate results while the intermediate results generated by the $C$-GRU are less precise - an insight that cannot be gained by investigating the set-level predictions alone.

B. Interpretability in Multiplication-based Problems

In general, Capacities and Capacity networks can be applied to any (monotone) non-additive problem, which also includes non-additive problems that have less obvious added values such as multiplication-based problems. To illustrate this, we created a multiplication-based dataset according to $f_{\text{mult}}(X) = \prod_{i=1}^{n} c_i$, where $c_i$ indicates the class index of instance $i$ and define the empty set to have a utility of 1. Table VI shows an illustration of the learned intermediate results to demonstrate that the Capacity networks are able to learn added values as expected.

C. Evaluation Beyond Input-Output Testing

Capacity networks also allow in-depth evaluation beyond mere input-output testing. More specifically, for Capacity net-
not possible for their non-Capacity counterparts and other prior architectures, which do not produce intermediate results, in a wide range of setups. Furthermore, we demonstrate the improved interpretability of Capacity networks that allows a detailed inspection of the network internals, an evaluation beyond mere input-output testing, and incorporation of prior knowledge via intermediate result regularization.

VII. FUTURE WORK

It is noteworthy that inductive bias presented in this work is not limited to MIL. Aggregating information from multiple sources is also a key problem in areas such as multimodal learning and geometric deep learning. Hence, exploring more application areas beyond MIL for Capacity networks is promising. Furthermore, extending Capacity networks to classification problems is a promising future research direction, especially if the monotonicity constrained is not enforced. Moreover, prior works have shown that learning to order instances can further improve network accuracy [27]. In future research, it would be interesting to see if and to which extent this also applies to Capacity networks.

REFERENCES

[1] P. J. Sudharshana, C. Petitjean, F. Spanhol, L. Oliveira, P.HONEINE, P. J. Sudharshana, C. Petitjean, F. Spanhol, L. Oliveira, and L. Heutte, “Multiple Instance Learning for Histopathological Breast Cancer Images,” Expert Systems with Applications, vol. 117, pp. 103–111, 2019.

[2] T. G. Dietterich, R. H. Lathrop, and T. Lozano-Pérez, “Solving the multiple instance problem with axis-parallel rectangles,” Artificial Intelligence, vol. 89, no. 1-2, pp. 31–71, 1997.

[3] N. Pappas and A. Popescu-Belis, “Explicit document modeling through weighted multiple-instance learning,” Journal of Artificial Intelligence Research, vol. 58, pp. 591–626, 2017.

[4] J. McAuley, J. Leskovec, and D. Jurafsky, “Learning attitudes and attributes from multi-aspect reviews,” Proceedings of the 12th IEEE International Conference on Data Mining, pp. 1020–1025, 2012.

[5] O. Maron and T. Lozano-Pérez, “A framework for multiple-instance learning,” Advances in Neural Information Processing Systems, pp. 570–576, 1998.

[6] S. Ray and D. Page, “Multiple Instance Regression,” in Proceedings of the International Conference on Machine Learning, 2001, pp. 425 –432.

[7] M. A. Carbonneau, V. Cheplygina, E. Granger, and G. Gagnon, “Multiple instance learning: A survey of problem characteristics and applications,” Pattern Recognition, vol. 77, pp. 329–353, 2018.

[8] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. Salakhutdinov, and A. Smola, “Deep Sets,” in Proceeding of the 31st Conference on Neural Information Processing Systems, 2017, pp. 1–11.
