Online Appendix: Customer Acquisition via Explainable Deep Reinforcement Learning

Yicheng Song
Carlson School of Management, University of Minnesota, Minneapolis, MN 55455, ycsong@umn.edu

Wenbo Wang
HKUST Business School, Clear Water Bay, Kowloon, Hong Kong, wenbowang@ust.hk

Song Yao
Olin Business School, Washington University in St. Louis, St. Louis, MO 63130, songyao@wustl.edu

Appendix

A. Table of Notations

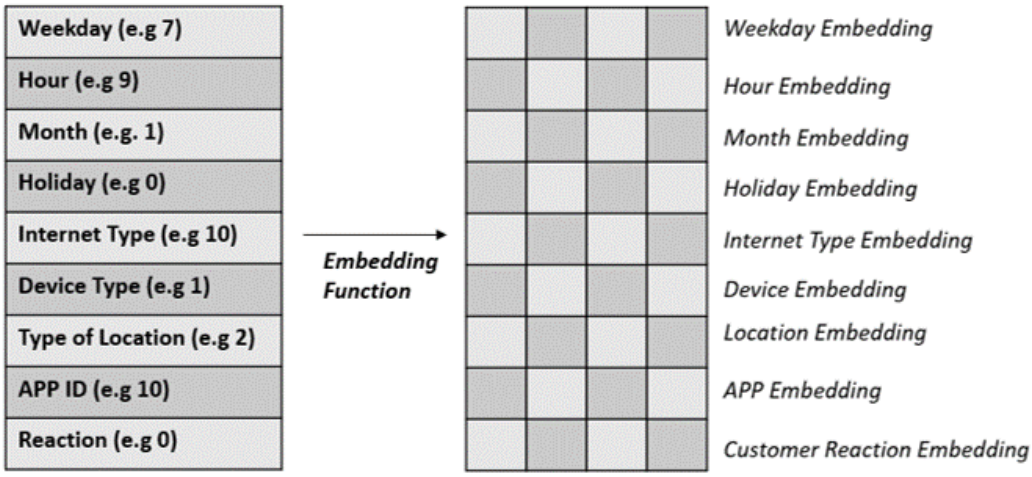
Table 1 Table of Notations

	Notation	Format	Description
Data	i	Scalar	customer index
	t	Scalar	Time index
	j	Scalar	Index of feature in incoming interaction
	k	Scalar	Dimension of query and key
	g	Scalar	Dimension of the state
	h	Scalar	Dimension of the historical interaction
	e	Scalar	Dimension of the embedding for historical interaction
	J	Set	Collection of features in incoming interaction
	$\boldsymbol{hi_{it}}$	$1 \times h$	Historical interaction of prospect i at timestep t
	$\boldsymbol{static_i}$	1×10	Static information of prospect i
	$\boldsymbol{f_{it}}$	$1 \times \lvert J \rvert$	Features of incoming interaction of prospect i at timestep t
RL	$\boldsymbol{s_{it}}$	$1 \times g$	State of prospect i at timestep t
	a_{it}	Scalar	Action of RL model to prospect i at timestep t
	r_{it}	Scalar	Immediate reward of taking action a_{it} under state $\boldsymbol{s_t}$
	$Q(\boldsymbol{s_t}, a_t)$	Scalar	Expected cumulative reward after taking action a_t under state $\boldsymbol{s_t}$
	γ	Scalar	Discount factor in the Bellman Equation
Attention Mechanism	$\boldsymbol{qr_{it}}$	$1 \times k$	Query vector
	$\boldsymbol{K_{it}}$	$k \times \lvert J \rvert$	Key Matrix
	$\boldsymbol{v_{it}}$	$1 \times \lvert J \rvert$	Value vector
	$\boldsymbol{att_{it}}$	$1 \times \lvert J \rvert$	Attention weight vector
	$\widetilde{\boldsymbol{att_{it}}}$	$1 \times \lvert J \rvert$	Pre-Normalized attention weight vector
GRU	$\boldsymbol{m_t}$	$g \times 1$	Hidden state of GRU at time t
	$\boldsymbol{z_t}$	$g \times 1$	Update gate of GRU at time t
	$\boldsymbol{re_t}$	$g \times 1$	Reset gate of GRU at time t
	$\boldsymbol{W_z}$	$g \times e$	Parameter matrix in update gate
	$\boldsymbol{W_r}$	$g \times e$	Parameter matrix in reset gate
	$\boldsymbol{W_m}$	$g \times e$	Parameter matrix in state update function
	$\boldsymbol{U_z}$	$g \times e$	Parameter matrix in update gate
	$\boldsymbol{U_r}$	$g \times e$	Parameter matrix in reset gate
	$\boldsymbol{U_m}$	$g \times e$	Parameter matrix in state update function
	$\boldsymbol{b_z}$	$g \times 1$	Parameter vector in update gate
	$\boldsymbol{b_r}$	$g \times 1$	Parameter vector in reset gate
	$\boldsymbol{b_m}$	$g \times 1$	Parameter vector in state update function

B. Structure of Contextual Vectors for Historical Interaction

The structure of $\boldsymbol{hi_{it}}$ is illustrated in Figure 1. Specifically, $\boldsymbol{hi_{it}}$ describes the agent's tth interaction with prospect i that consists of the contextual features of the interaction (including DateTime, device, location, app, prospect reaction, etc). Different element in the vector represents different information. For example, 7 in the first element means this interaction happened on Sunday. 9 in the second element means this interaction happened at 9am. 10 in the 8th element

Figure 1 Structure of Historical Interaction Vector



means that this interaction happened on the APP with the id of 10. The vector $\boldsymbol{hi_{it}}$ will be processed by an embedding function $\mathbb{E}()$. Specifically, we utilize the embedding function provided by Keras, as detailed in the API documentation [1]. This function is designed to convert positive integers (indexes) into dense vectors of a fixed size. As shown in Figure 1, when applying the embedding function to APP ID, it transforms into a 4-dimensional vector. For instance, APP ID1 will be represented as [0.16, 0.78, 0.13, 0.62], while APP ID 5 results in a different 4-dimensional vector [0.81, 0.32, 0.75, 0.96]. The choice to use the embedding function rather than opting for one-hot encoding for categorical data is motivated by a desire to avoid sparse input, particularly when dealing with a large number of choices in the category, such as 96 Apps in our setting. As explained in Geron's book (Géron 2022) Chapter 13 (pages 466-471), an embedding serves as a trainable dense vector representing a category. For instance, the Keras function $Embedding(7,2)$ indicates there are 7 choices in the category, and each choice is represented by a 2-dimensional vector. The matrices of size 7x2 are parameters that undergo learning in the network training process.

It is crucial to clarify that our model explanation or attention mechanism is applied to the features of incoming interactions $\boldsymbol{f_{it}}$, as opposed to historical interactions $\boldsymbol{hi_{it}}$. Unlike the features of historical interactions, which undergo processing by the embedding function, we have opted for one-hot encoding for the features of incoming interactions. The rationale behind this choice stems from the fact that embedding vectors lack explainability. Applying attention weights directly to black-box embedding vectors could obscure the decision-making process of the model. Therefore, for the features of incoming interactions, we have chosen one-hot encoding to make the visualization of high-weight

[1] https://keras.io/api/layers/core_layers/embedding/

features, as presented in Tables 5-7, feasible. This choice ensures the vector representation of f_{it} remains explainable. For example, if there is a '1' in the 5th element of the one-hot encoding vector, it signifies that the incoming interaction is from App 5. The attention weight is subsequently applied to this one-hot encoding vector. Consequently, the highly weighted values in the attention weight have direct and explainable correspondences. This approach provides a clear understanding of which features in the incoming interaction the attention weight is focusing on.

C. Recurrent Neural Network to Learn State Representation

A Recurrent Neural Network (RNN) is a type of neural network that aims to process sequential data or time series data. These deep learning algorithms are commonly used for temporal problems, such as language translation, natural language processing, and speech recognition. In our setting, we use RNN to process historical interactions to learn state representation. Specifically, we choose the many-to-one RNN structure [2], where the input is the historical interactions and the only output at the end of the RNN is the state representation. We have explored three options for RNN: 1) Vanilla RNN (Werbos 1990), 2) Long Short Term Memory (LSTM) (Hochreiter and Schmidhuber 1997), and 3) Gated Recurrent Unit (GRU) (Cho et al. 2014). GRU is selected due to its superior model performance.

There are two gates in a GRU at a given time step t: the update gate and reset gate:

$$\boldsymbol{z_{it}} = \sigma(\boldsymbol{W_z}\mathbb{E}(\boldsymbol{hi_{it}}) + \boldsymbol{U_z} m_{i(t-1)} + \boldsymbol{b_z}) \tag{1}$$

$$\boldsymbol{re_{it}} = \sigma(\boldsymbol{W_r}\mathbb{E}(\boldsymbol{hi_{it}}) + \boldsymbol{U_r} m_{i(t-1)} + \boldsymbol{b_r}) \tag{2}$$

where $\boldsymbol{hi_{it}}$ is the historical interaction for prospect i at time t, $\mathbb{E}()$ is the embedding function, $m_{i(t-1)}$ is the memory from the previous time step, and σ is the sigmiod function. Addtionally, $\boldsymbol{W_z}$, $\boldsymbol{U_z}$, and $\boldsymbol{b_z}$ are parameters associated with the update gate $\boldsymbol{z_{it}}$. $\boldsymbol{W_r}$, $\boldsymbol{U_r}$, and $\boldsymbol{b_r}$ are parameters related to the reset gate. With two gates ready, we can update the candidate's hidden state $\hat{m}_{it}$ as:

$$\hat{m}_{it} = \phi(\boldsymbol{W_m}\mathbb{E}(\boldsymbol{hi_{it}}) + \boldsymbol{U_m}(\boldsymbol{re_{it}} \otimes m_{i(t-1)}) + \boldsymbol{b_m}) \tag{3}$$

where $\otimes$ is Hadamard product operator and ϕ is tanh activation function. $\boldsymbol{W_m}$, $\boldsymbol{U_m}$, and $\boldsymbol{b_m}$ are parameters utilized for updating the candidate hidden state. The updated memory at time step t is defined as:

[2] https://stanford.edu/ shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks

$$\boldsymbol{m_{it}} = \boldsymbol{z_{it}} \otimes m_{i(t-1)} + (1 - \boldsymbol{z_{it}}) \otimes \hat{m}_{it} \tag{4}$$

Within this updating process, $\boldsymbol{m_{io}}$ denotes the initial state at step 0. Adhering to the standard setting in GRU and other RNN models, $\boldsymbol{m_{io}}$ is randomly generated. All the follow-up states in GRU will be updated based on EQ 4. By feeding GRU with historical interaction $\{\boldsymbol{hi_{i1}}, ..., \boldsymbol{hi_{it}}\}$, the output of GRU at the last step is $\boldsymbol{m_{it}}$, which is used to represent state $\boldsymbol{s_{it}}$ in DRQN model. However, for the DRQN-Attention model, the GRU generates $\boldsymbol{he_{it}}$ (historical embedding) by process input sequence $\{\boldsymbol{hi_{i1}}, ..., \boldsymbol{hi_{i(t-1)}}\}$, as depicted in Section 4.2 of the manuscript.

D. Off-Policy Evaluation

Many RL algorithms assume that an agent actively interacts with an online environment to learn from its own collected experience and evaluate the learned model in the same setting. So, the performance of these algorithms is evaluated via on-policy interaction with the environment. Traditionally, algorithms have been evaluated via on-policy format on simple hand-designed problems, often with a small number of states. Recently, many works adopted simulators (e.g., Atari video games environment (Mnih et al. 2015)) as a testbed, or directly played with human experts on those well-designed games (Silver et al. 2016), to evaluate RL algorithms. However, it is challenging to evaluate RL algorithms on real-world problems (e.g., business decisions) via on-policy, as it can be extremely expensive and risky to collect extensive data using an unjustified RL system to interact with the real-world environment. On the other hand, those applications in simulation settings require high-fidelity simulators that are challenging to build. Fortunately, for many applications, there exist pre-collected data which can be utilized to make RL training and evaluation feasible, and enable better generalization by incorporating diverse prior experiences.

Off-policy RL using an offline dataset of logged interactions is an important tool for real-world applications. Off-policy RL can help (1) train an RL model, and (2) empirically evaluate the RL model using existing data, while the existing data is generated by an executing agent that is different from the model to be evaluated.[3] From the model training perspective, the Q-learning framework that DRQN-Attention builds on is a well-known off-policy RL algorithm (Sutton and Barto 2018).

We aim to sample episodes with different lengths that match the learned policy of the proposed model. Here an episode represents an ordered trajectory of states, actions, and rewards. For example, episode

[3] There is no requirement that the executing agent must be a purely random policy. Instead, we only need to know the probabilities of generating different actions under different states of the executing agent.

$\{s_1, a_1, r_1, s_2, a_2, r_2, ..., s_H, a_H, r_H\}$ is obtained by executing the agent (i.e., contextual multi-armed bandit in our setting) H times in the environment. However, there is a mismatch of distributions: we need episodes sampled to follow the distribution of the proposed model (i.e., DRQN-Attention); but we have data drawn from the distribution of the executing agent (i.e., contextual MAB). Importance sampling is a technique for handling such a mismatch and there is a wide range of off-policy evaluators in the literature (Precup 2000, Theocharous et al. 2015). Unfortunately, most of these off-policy evaluators assumed the environment is MDP, which is inappropriate in our setting. Therefore, we adopt Fixed-M per-episode rejection sampling (Fixed-M PERS, Mandel et al. 2016) to evaluate the model performance, as this evaluator naturally accommodates POMDP. Intuitively, Fixed-M PERS aims to sample from hold-out data and select those episodes more favorably when they match the learned action policy of the RL model to be evaluated. Fixed-M PERS is a well-established method with the following nice properties: 1) When applying Fixed-M PERS, given the current history and that the algorithm accepts episode samples of observations and rewards, the observations and rewards are drawn from a distribution identical to the distribution the algorithm would have encountered if it was to run online. This is known as true sample property [4]. 2) Based on the episode samples accepted by Fixed-M PERS, we can derive an unbiased estimate of the reward obtained by the RL algorithm in the episode if it was to run online. This is known as unbiased estimation of the average reward of episode property. 3) The dynamics of the environment in our empirical setting depend not only on the most recent observation (i.e., MDP) but also on the full history of interactions (i.e., POMDP). Fixed-M PERS is a representation-agnostic evaluator, which does not require Markov assumption.

When applying Fixed-M PERS, we need to decide the length of episodes to sample. Denote the length of the episode as H and we evaluate the performance of different models with $H = 1, 3, 6$.[5] For example, when $H = 3$, we sample candidate episodes as $\{s_1, a_1, r_1, s_2, a_2, r_2, s_3, a_3, r_3\}$. For all the sampled episodes, we calculate the average reward per episode to evaluate model performance. The detailed algorithm is listed below:

Fixed-M PERS (Mandel et al. 2016) aims to sample from the dataset D that contains episodes, where each episode d represents an ordered trajectory of actions, rewards, and states, $(s_0, a_0, r_0, s_1, a_1, r_1, ..., r_H)$ obtained by executing existing policy e for H steps in the environment. Fixed-M PERS performs rejection sampling at the episode level. It

[4] The requirement for sample true (Mandel et al. (2016) Theorem 6.1) is met in our empirical setting. Please refer to Appendix E for more details.

[5] Mandel et al. (2016) show Fixed-M PERS is less efficient for longer episodes as the acceptance rate in the rejection sampling fell sharply, leading to few or zero accepted episodes. We also found that the average number of ad exposure for customer acquisition is 3.6296, and 86% of the customer acquisitions get less than or equal to 6 ad exposures. Thus, we set maximal $H = 6$.

Algorithm 1 Fixed-M Per-Episode Rejection Sampling Evaluator

1: Input: Executing policy e, evaluated policy b, state space S, binary transition matrix T denoting whether a nonzero transition probability from one state to another, maximum horizon H, and Dataset of episodes D with each episode length of H.
2: Output: AER, where $AER(i)$ is the ith Accepted Episode cumulated Reward
3: Initialize $M_s = 1.0$ for all $s \in S$
4: **for** h=1 to H **do**
5: **for** $s \in \mathcal{S}$ **do**
6: Update $M_s^{'} = max_a \frac{\pi_b(a|s)}{\pi_e(a|s)} max_{s'}(T(s,a,s')M_{s'})$
7: **end for**
8: $M = M^{'}$
9: **end for**
10: $i = 1$
11: **for** $d \in \mathcal{D}$ **do**
12: $p = 1.0, R = 0, s = []$
13: Get start state st of the episode d
14: **for** $(o,a,r) \in d$ **do**
15: $s = (s,o)$
16: $p = p\frac{\pi_b(a|s)}{\pi_e(a|s)}$
17: $R = r + \gamma R$
18: **end for**
19: Sample $\mu \sim Uniform(0,1)$
20: **if** $\mu \leq \frac{p}{M_{st}}$ **then**
21: $AER(i) = R$
22: $i = i + 1$
23: **end if**
24: **end for**
25: **return** AER

will first compare evaluated policy b (RL model needs to evaluate) against the executing policy e (MAB) to get the ratio of the probability of executing an action under a state between two policies, then accept or reject the episode according to whether a random variable sampled from the uniform distribution is lower than the computed ratio. In order to ensure that rejection sampling returns a sample from the candidate distribution that represents the distribution as applying evaluated policy b online, it is critical to set ratio normalization constant M correctly. Define the probability of executing action a under state s via policy e as $\pi_e(a|s)$, the probability of executing action a under state s via policy b as $\pi_b(a|s)$. The ratio $\frac{\pi_b(a|s)}{\pi_e(a|s)}$ can grow extremely large, we need an M such that $\frac{\pi_b(a|s)}{M\pi_e(a|s)}$ is a probability between 0 and 1. Therefore, M should be assigned as a constant that represents the maximum possible ratio. The detailed implementation of Fixed-M PERS is shown in Algorithm 1, where the for-loop between lines 4-10 is used to construct M, and the for-loop in lines 11-24 for episode-level rejection sampling.

We note that the learned intervention policy may only be optimal within the explored space given the training data but may be sub-optimal globally. This is mainly caused by the off-policy training using the archived data, where the model can only learn from the limited action combinations within the explored space, and there may be unexplored action combinations that could result in better performance. The DRQN-Attention model, when applied in an

on-policy setting, can benefit from RL exploration strategies to enhance intervention policies. RL models inherently balance exploitation and exploration, taking advantage of learned policies for stable revenue while also exploring new intervention opportunities for policy improvement. Such exploration could be a new advertisement, a new advertisement channel, or existing channels but under-explored. This can be achieved through classical exploration methods such as ε-greedy (Sutton and Barto 2018) and Boltzmann exploration (Bertsekas and Tsitsiklis 1995). The key idea of ε-greedy and Boltzmann exploration is not only utilizing the known promising action (e.g. the action leads to the highest Q-Value), but there are chances to explore other actions which might have the potential to improve the future reward. Additionally, the model can be further optimized using the Upper Confidence Bound (UCB) (Annasamy and Sycara 2019) to balance the exploration and exploitation.

E. Action Randomness of the Collected Data

In off-policy RL, action policy π_b of the proposed model (e.g. DRQN-Attention) aims to learn from the data that is drawn from a different action policy π_e of the executing agent (e.g. contextual MAB). If $\pi_e(a|s) = 0$ for certain state-action pair, the proposed policy π_b will never get a chance to explore executing a under state s, which might lead to learning an inferior targeting policy due to under-exploration. To ensure the proposed model could learn optimal targeting policies under different states, it would be better to minimize the cases of $\pi_e(a|s) = 0$. As the contextual MAB adopts ε-greedy to balance exploration and exploitation, there is a randomness to guide contextual MAB to try different actions under different states. To show the randomness, we first check how many unique actions have been executed when a customer lands an APP. The average number of unique actions is 190.5 (theoretical maximal is 196) per APP, indicating that the executing agent does explore almost all the actions on different APPs. While APP is just a proportion of features being used to construct state representation and other features (historical interactions, time, location) also matter. Thus, we further check the unique number of actions being explored under different states. As the states are represented by 20-dimensional vectors, we first run K-means clustering on all the state representations. Based on the elbow method, we set the number of clusters to 54. Then we check how many unique actions have been executed under different state clusters. The average number of unique actions is 190.1 (theoretical maximal is 196) per state cluster, this also indicates that the executing agent does explore almost all the actions under different states. Therefore, we can conclude that the executing agent does comprehensive state-action explorations, which will help the proposed model learn optimal policy.

For model evaluation, Fixed-M PERS requires $\pi_b(ep) > 0 \rightarrow \pi_e(ep) > 0$ for all possible episodes ep (Mandel et al. 2016). We have checked the proposed model π_b and executing agent π_e for all possible episodes ep with episode length of $H = 1, 3, 6$, the requirement is met in our empirical setting.

F. Validation of Attention Weight

Serrano and Smith (2019) tested the reliability of attention mechanisms in Natural Language Processing (NLP) settings. They used a bidirectional LSTM to process word tokens and connected it with an attention mechanism to predict the target variable (such as a review rating or the sentiment of a text). To test the validity of the attention weights, they zeroed out the attention weights from highest to lowest and re-normalized the remaining weights. If the attention weights were reliable, a change in the model prediction should occur quickly. However, the authors found that they had to zero out 90% of the weights on average to observe a change in the model's decision, suggesting that the attention weights were not explainable. This highlights the need for further validation of explainability through attention weights in NLP tasks.

We have adopted the same validation strategy in Serrano and Smith (2019) to the proposed DRQN-attention model. We zero out the attention weights from highest to lowest and re-normalized the remaining weights. Unlike the NLP tasks, where 90% of the weights had to be zeroed out to change the model decision, in our setting, on average, only 7.1% of the highest weights had to be zeroed out (standard deviation 0.013). This sensitivity of the attention weights in our setting can be attributed to the way the attention weights are applied. In our model, the attention weights are applied to the incoming interaction features, which are represented through one-hot encoding (96 dimensions for APPs, 7 dimensions for location types, 5 dimensions for devices, etc.). These features are generally independent of each other. However, in NLP tasks, the attention weights are applied to word embeddings, and there are strong contextual dependencies between words in any text. As a result, if a few keywords are erased, the meaning of the text can still be inferred and the overall tone will not change. On the other hand, in our setting, changing any feature might mean: 1) the user is using APP1 rather APP2, 2) the user is on a mobile phone rather than Pad, 3) the incoming interaction happens on Monday rather than Friday. Thus, a minor change in the feature vector of the incoming interaction can dramatically change the context of the interaction, leading to the sensitivity of the attention weights. As noted by Wiegreffe and Pinter (2019), the validity of attention weights is highly dependent on the task, and attention weights work well in uncontextualized settings. From our analysis, we have tested the sensitivity and quality of the attention weights in the proposed model, and the results show that the highly weighted features are highly responsible for the model's decisions.

G. Model Explainability Comparison

The proposed model, as an intrinsic explainable model, can be compared with a post-hoc explainable model to highlight the differences in explainability between these two representative explainable models. The closest black box

model to the proposed model is DRQN, and we applied TimeSHAP (Bento et al. 2021) on top of this model. Bento et al. (2021) underscores that blindly applying SHAP (Lundberg and Lee 2017) to RNNs may overlook the importance of past events and features throughout the sequence. It tends to attribute significance predominantly to features of the current input. In response to these limitations, we have explored TimeSHAP (Bento et al. 2021), a model-agnostic recurrent explainer. TimeSHAP builds upon KernelSHAP and extends its capabilities to the sequential domain, offering a more comprehensive solution to address the challenges posed by applying SHAP to RNN-style models.

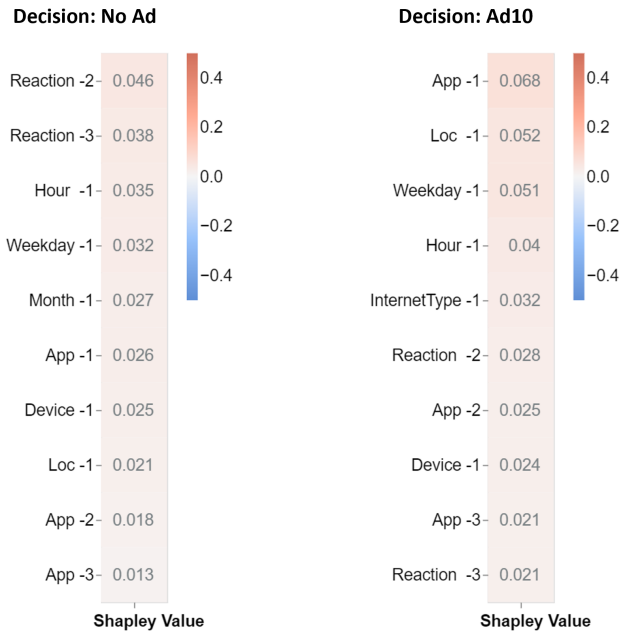
TimeSHAP provides a variety of methods, each serving specific purposes based on the desired explanations[6]. Local methods offer a detailed view of a model decision corresponding to a specific sequence being explained. On the other hand, global methods aggregate local explanations of a given dataset to present a holistic view of the model's behavior. Within the global methods, TimeSHAP enables the analysis of the Shapley value of a time lag (e.g., the effect of all entities in lag 2) or a specific input entity (e.g., the effect of historical customer reaction). However, it's noteworthy that there isn't a dedicated tool for the analysis of time lag $\times$ entity interactions (e.g., the Shapley value of customer reaction in lag 2). Considering our specific analysis needs, we have opted for the local event analysis tool in TimeSHAP. This choice facilitates the examination of randomly selected cases, offering detailed insights into model decisions at the level of individual sequences.

We present the Shapley values for two distinct cases: one with no-ad decision and the other involving the decision to show AD10. Figure 2 visually represents the SHAP values for these cases. In the plot, the -1 index corresponds to features of the incoming interaction, while the -2 index is associated with features in the latest historical interaction. Observing the plot, we discern that the customer's reactions in the previous interactions (-2, -3) and the temporal information of the incoming interaction predominantly influence the decision not to show the ad. This observation suggests that the model learns from the customer reactions in preceding interactions, determining that targeting this particular customer may not be financially advantageous. Contrastingly, the decision to display AD10 in the second case is primarily influenced by the information pertaining to the incoming interaction, such as APP, location, weekday, and hour. This signals that the DRQN model adaptively relies on different types of information to make the decision.

The analyses of TimeSHAP explanations reveal distinct differences with DRQN-Attention: 1) TimeSHAP explanations are intricately tied to specific decisions, providing insights into which features in historical and incoming interactions contributed to a particular model decision. DRQN-Attention offers dynamic explanations based on static

[6] https://github.com/feedzai/timeshap/tree/main

Figure 2 **Top Sharpley values for two cases with a decision of not showing AD, verse showing AD10**



characteristics, historical interactions, and contextual information. 2) The explanations from TimeSHAP aim to identify which features in the historical interaction as well as incoming interaction are more responsible for the model decision. However, the proposed model looks forward to identifying which features in the incoming interaction signal a more profitable targeting. These differences highlight the varied applications of the two models.

H. Federal Learning and Attacks

To ensure data security and privacy in FL, it is crucial to consider potential attacks. Two relevant types of attacks are: passive attacks, involving the observation of information or learning from a system without altering it, and active attacks, aiming to manipulate the system's resources or operations (Yin et al. 2021). In the context of FL, passive attackers typically observe computations, including weights, gradients, and the final model during the training and inference phases. On the other hand, active attackers may seek to influence the FL system by manipulating model parameters to achieve adversarial goals. For the proposed model, we can enhance privacy protection by employing commonly adopted cryptographic techniques (Xu et al. 2019) and perturbation techniques (Yang et al. 2022) within FL. Cryptographic techniques widely used in privacy-preserving machine learning include homomorphic encryption, secret sharing, and secure multi-party computation. Perturbation techniques involve adding noise to the original data, ensuring that statistical information calculated from the perturbed data remains statistically indistinguishable from the original data. These methods could enhance the security and privacy of FL models in our context.

References

Annasamy RM, Sycara K (2019) Towards better interpretability in deep q-networks. *Proceedings of the AAAI conference on artificial intelligence* 33(01):4561–4569.

Bento Ja, Saleiro P, Cruz AF, Figueiredo MA, Bizarro P (2021) Timeshap: Explaining recurrent models through sequence perturbations. *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery Data Mining*, 2565–2573 (New York, NY, USA: Association for Computing Machinery), ISBN 9781450383325, URL `http://dx.doi.org/10.1145/3447548.3467166`.

Bertsekas DP, Tsitsiklis JN (1995) Neuro-dynamic programming: an overview. *Proceedings of 1995 34th IEEE conference on decision and control*, volume 1, 560–564 (IEEE).

Cho K, Van Merriënboer B, Bahdanau D, Bengio Y (2014) On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259* .

Géron A (2022) *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow* (" O'Reilly Media, Inc.").

Hochreiter S, Schmidhuber J (1997) Long short-term memory. *Neural computation* 9(8):1735–1780.

Lundberg SM, Lee SI (2017) A unified approach to interpreting model predictions. Guyon I, Luxburg UV, Bengio S, Wallach H, Fergus R, Vishwanathan S, Garnett R, eds., *Advances in Neural Information Processing Systems 30*, 4765–4774 (Curran Associates, Inc.), URL `http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf`.

Mandel T, Liu YE, Brunskill E, Popović Z (2016) Offline evaluation of online reinforcement learning algorithms. *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30.

Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, Graves A, Riedmiller M, Fidjeland AK, Ostrovski G, et al. (2015) Human-level control through deep reinforcement learning. *Nature* 518(7540).

Precup D (2000) Eligibility traces for off-policy policy evaluation. *Computer Science Department Faculty Publication Series* 80.

Serrano S, Smith NA (2019) Is attention interpretable? *arXiv preprint arXiv:1906.03731* .

Silver D, Huang A, Maddison CJ, Guez A, Sifre L, Van Den Driessche G, Schrittwieser J, Antonoglou I, Panneershelvam V, Lanctot M, et al. (2016) Mastering the game of go with deep neural networks and tree search. *Nature* 529(7587):484–489.

Sutton RS, Barto AG (2018) *Reinforcement learning: An introduction* (MIT press).

Theocharous G, Thomas PS, Ghavamzadeh M (2015) Personalized ad recommendation systems for life-time value optimization with guarantees. *Twenty-Fourth International Joint Conference on Artificial Intelligence*.

Werbos PJ (1990) Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE* 78(10):1550–1560.

Wiegreffe S, Pinter Y (2019) Attention is not not explanation. *arXiv preprint arXiv:1908.04626* .

Xu G, Li H, Liu S, Yang K, Lin X (2019) Verifynet: Secure and verifiable federated learning. *IEEE Transactions on Information Forensics and Security* 15:911–926.

Yang X, Feng Y, Fang W, Shao J, Tang X, Xia ST, Lu R (2022) An accuracy-lossless perturbation method for defending privacy attacks in federated learning. *Proceedings of the ACM Web Conference 2022*, 732–742.

Yin X, Zhu Y, Hu J (2021) A comprehensive survey of privacy-preserving federated learning: A taxonomy, review, and future directions. *ACM Computing Surveys (CSUR)* 54(6):1–36.