ulm university universität
uulm

Ulm University
Faculty for Mathematics and Economics

Developing and Evaluating a Marketing Attribution Model: A Time-Gated Approach with Gated Recurrent Units for Conversion Prediction

Masterthesis

in Mathematics and Management

by
Leonie Allgaier
November 29, 2023

Thesis Supervisor
Prof. Dr. Steffen Zimmermann

Contents

List of Figures

1. Introduction

Effectively allocating marketing budgets is a challenging decision for marketers, especially with expanding advertising opportunities. The importance of each individual marketing channel heavily depends on the brand and product one tries to advertise, so there isn't one universally best marketing strategy to maximize return on investment (ROI). Knowing which ads, a customer was exposed to, one could try to derive the importance of each advertising channel.

Traditionally, heuristic approaches are used to assess the impact of marketing channels, for example, last-click attribution where the last ad a customer was exposed to before buying gets all the credits for the conversion. However, with the rise of mobile devices and online advertising, the volume of available data has skyrocketed, which led to the emergence of data-driven methods. This allows for more precise attribution but also introduces complexity.

Mostly explaining the use and importance of MTA, use the source proving MTA effectiveness

Quick overview of the idea

Overview over the chapters

2. Marketing attribution

Using the definition by [Buhalis and Volchek, 2021] marketing attribution describes "a strategy of determining the value of marketing communications and allocating it to identified touchpoints along customer journeys". This chapter aims to give a consistent and concise overview of marketing attribution. To do this, first, the necessary taxonomy, as established by [Buhalis and Volchek, 2021], will be introduced and some possible marketing attribution models will be presented.

To derive the importance of different marketing channels, marketing attribution relies on collecting and evaluating data on customer level.
When talking about this kind of data each interaction between a customer and a brand is referred to as a *touchpoint* that occurs via one *marketing channel* (i.e. visiting a website, seeing a banner ad, receiving a marketing email). The *customer journey* is the collection of all (available) touchpoints between a customer and a brand.
For every customer journey, the fact of whether this customer ended up buying a product/subscribing to a service is documented and called *conversion*.

Marketing attribution started with Single-Touch attribution, which describes the process of attributing conversion only to a single touchpoint per customer journey. The main two approaches for Single-Touch attribution are first- and last-touch attribution with the justification that the first touchpoint made a customer aware of the brand and the last touchpoint is the most recent touchpoint therefore probably the reason for the decision to finally buy.
While those explanations sound reasonable and already provide some insight for a brand, the true decision-making process is often more complex and the whole customer journey influences a customer's decision-making process. Multi-touch attribution (MTA)) recog-

nizes that conversion cannot be attributed to one singular touchpoint and that the whole collection of touchpoints in each customer journey must be taken into account. Inspired by the first- and last-click approach there are some easy heuristic multi-touch attribution models allocating the same impact to each touchpoint in the customer journey or doing some reweighting putting more emphasis on the first and/or last touchpoint while still considering what happens in between.

There are some difficulties that can stand in the way of exact marketing attribution. One thing is that we can't assume complete data, it's very possible to miss touchpoints. This can happen if a customer uses different devices that cannot be linked, or when the customer is confronted with analog advertisement which obviously cannot be tracked as easily. Also, one can't be sure if a customer actually registered an ad when it is presented on screen.

Another thing is that each customer is unique and has a different process of decision-making, and things like personal preferences simply can't be observed.

Still, with the data that is available nowadays marketing attribution has been shown to be effective for developing more efficient marketing strategies and a achieve improved return on investment (ROI) [**?**]

Looking at customer journeys, the influence of a touchpoint on a customer's decision can be either positive or negative and also depends on the timing within the customer journey and the other touchpoints surrounding it. Furthermore, it's not necessary, that the effects of the touchpoints in a customer journey are cumulative. This makes each customer journey unique, which can be difficult.

More complex MTA strategies can reflect those complex decision-making processes even better and due to increasing available data quality and quantity, advancements in Big Data, and the increasing number of potential touchpoints (i.e. new marketing channels) data-driven methods for marketing attribution have emerged to capture how the touchpoints interact and influence each other. Those approaches include Survival Analysis, Markov Chains, and Neural Networks which are constantly extended to become even more accurate.

2.1. Using deep learning for Multi-touch attribution

This thesis focuses specifically on the application of neural networks and deep learning in the context of marketing attribution. Deep learning models are very good at predicting outcomes or classifying but due to their complexity and black-box nature can't be interpreted easily. To get to value attribution we rely on finding to make a deep learning model interpretable.

To overcome this problem a second step in addition to the conversion prediction network is necessary. This second step uses Shapley Values, an important concept from game theory that can be transferred to deep learning and is frequently used for explainable AI tasks and in our case can give us the importance of the different marketing channels. That's why, especially recently, a big focus has been on combining deep learning models with Shapley values for marketing attribution. klingt noch nicht ganz überzeugend

A few examples of existing MTA models that are based on conversion prediction using a neural network:

- **RNN [Du, 2019]:** They implemented a recurrent neural network (RNN) for conversion prediction with "a large set of user features [...] as synthetic control" to prevent confounding bias and make causal interpretations of the Shapley Values of the marketing channels.

- **DARNN [Ren et al., 2018]:** Uses a Dual-attention recurrent neural network that "learns the attribution values [...] directly from the conversion objective" which means they don't need the additional step of Shapley explanations, they observe impressions as well as click events and try to predict/estimate both.

- **DNAMTA [Li, 2018]:** Combine an LSTM network with survival time-decay functions and similar to [Du, 2019] also use some user information as control variables.

- **CAMTA [Kumar et al., 2020]:** For the CAMTA (Casual Attention Model for Multi-touch Attribution) a "causal RNN" is trained to eliminate time-varying confounders by not only observing conversion but also further interaction with the channel (click event).

- **CausalMTA [Yao et al., 2022]:** Another approach to eliminate confounders in order to get an unbiased model which can be interpreted causally. They decompose

the confounding bias into a static and dynamic part and prove the effectiveness of their model.

- **DeepMTA [Yang, 2020]:** For their DeepMTA model they used a phased LSTM network and Shapley explanations and achieved great improvements in conversion predictions with the additional time-gate in their LSTM cell.

Overall the trend is shifting towards using LSTM networks, especially now, that good implementations exist, for example in Pytorch and Tensorflow, that can be applied directly to the MTA. What cannot be found frequently amongst marketing attribution models are gated recurrent unit (GRU) networks even though they often perform just as good as LSTM and contain fewer parameters that need to be trained and therefore require fewer storage space and fewer computations for training and prediction. The goal of this thesis is to implement a similar marketing attribution model as [Yang, 2020] as the added time gate led to substantially improved predictions compared to the base LSTM but with GRU instead of LSTM cells, to combine their advantages.

3. Phased Gated Recurrent Units Network

In this chapter, the goal is to introduce the theoretical structure of the conversion predictor that we are training. Neural networks are very popular as they can be used for regression and classification tasks, achieve reliably good results, and there are several ways to customize them to boost their performance even further.
We start with an introduction of basic neural networks before discussing recurrent neural networks (RNN) to help build an intuition for the GRU network we want to use as the conversion predictor. Finally, the addition and functionality of a so-called time gate in the phased GRU Cell, which we use to handle the non-uniform time intervals between touchpoints is explained.

3.1. Neural Networks

To lay the necessary foundation and introduce a common notation we first consider basic neural networks.
The idea behind neural networks stems from the idea of how information is processed by neurons in the human brain, where neurons collect and process incoming information and distribute their outcome to other neurons.
Mathematically we model *neurons* as follows:
In picture 3.1 an example neuron is depicted, that is taking three input values $in_1, in_2, in_3 \in \mathbb{R}$ and transforming them to an output value $out \in \mathbb{R}$, by performing a linear transformation and then applying an activation function. The linear transformation consists of multiplying the input values using three weights $w_1, w_2, w_3 \in \mathbb{R}$ and adding

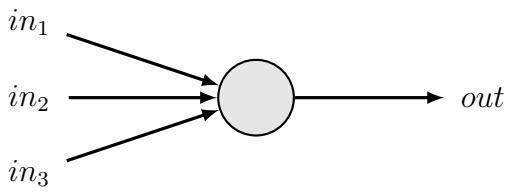


Figure 3.1.: Neuron

a bias value $b \in \mathbb{R}$ and then applying an activation function σ.

$$\begin{aligned} out &= \sigma(w_1 \cdot in_1 + w_2 \cdot in_2 + w_3 \cdot in_3 + b) \\ &= \sigma(w^T in + b), \end{aligned}$$

with $in = (in_1, in_2, in_3)^T \in \mathbb{R}^3$ and $w = (w_1, w_2, w_3)^T \in \mathbb{R}^3$.
More general:

Definition 3.1. A *neuron* of a neural network takes an input vector $x \in \mathbb{R}^p$ of a predetermined size p and performs a linear transformation using fixed weights $w \in \mathbb{R}^p$ and a fixed bias $b \in \mathbb{R}$ and then applies a predetermined activation function σ.

$$h = \sigma(w^T x + b),$$

Example 3.2. There are no specific properties we require from *activation functions* $\sigma : \mathbb{R} \to \mathbb{R}$, but there exist some common choices that are widely used:

- *Sign function*: $\sigma(x) = \mathbb{1}(x \geq 0) - \mathbb{1}(x < 0)$
- *Rectified Linear Unit (ReLU)*: $\sigma(x) = \max\{x, 0\}$
- *Leaky ReLU*: $\sigma(x) = \max\{\alpha x, x\}$, for some small $\alpha > 0$
- *Sigmoid*: $\sigma(x) = \frac{1}{1+e^{-x}}$
- *Hyperbolic tangent*: $\sigma(x) = tanh(x)$

To simplify working with numerous neurons we organize them in *layers*, where all neurons use the same input $x = (x_1, \cdots, x_n) \in \mathbb{R}^n$. Assume a layer with p neurons, then the output of this layer can be collected in a vector $y = (y_1, \cdots, y_p) \in \mathbb{R}^p$ where $y_i = \sigma(w_i^T * x + b_i) \in \mathbb{R}$

is the output of the i-th neuron.

We can summarize the calculations of a layer

$$y = \sigma(W * x + b),$$

with $W \in \mathbb{R}^{p \times n}$ a weight matrix, where each row corresponds to the weights w_i^T of one neuron i and $b = (b_1, \cdots, b_p) \in \mathbb{R}^p$ a vector of biases.

The activation function σ now maps $\mathbb{R}^p \mapsto \mathbb{R}^p$ which is simply defined as applying the one-dimensional activation function elementwise. Even though different activation functions for each neuron in a layer are possible, it's not very useful in application and adds unnecessary difficulty. Therefore we will refrain from considering it.

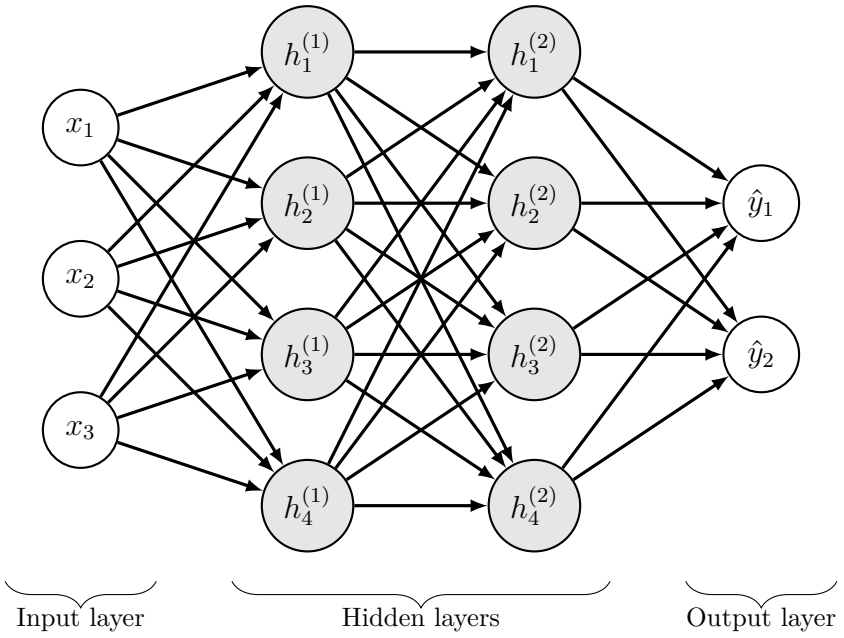


Figure 3.2.: Graph representation of a neural network with two hidden layers

As a layer is constructed to use the same input for each neuron, we can use the output of a layer as input for a new layer which is exemplarily shown in Figure 3.1. A special role has the *input layer*, which is the "first" layer and just contains the input variables, and the *output layer*, the "last" layer which returns the prediction of the neural network. All other layers of the network are called *hidden layers* as in general only input and output are observable for the network user while all the calculations that happen in between remain hidden.

Definition 3.3. A special activation function for the output layer is the softmax function. This is specifically used in classification tasks. Here for each of the output neurons $i = 1$, represents one of the c possible classes. The output of the softmax function for the i-th neuron:

$$softmax_i(x) = \frac{e^{x_i}}{\sum_{j=1}^{c} e^{x_j}},$$

with $x = (x_1, \cdots, x_c) \in \mathbb{R}^c$, then can be interpreted as the probability of the observation belonging to the respective class.

Definition 3.4. We now can define a fully connected neural network f with architecture (L, p), where $L \in \mathbb{N}_0$ is the number of hidden layers and $p = (p_0, p_1, \cdots, p_L, p_{L+1}) \in \mathbb{N}^{L+2}$ contains the layer sizes or widths.
The input layer consists of the input vector $h^{(0)} = x = (x_1, \cdots, x_{p_0})^T \in \mathbb{R}^{p_0}$.
For the L hidden layers $l = 1, \ldots, L$ compute a hidden state

$$h^{(l)} = \sigma^{(l)}(W^{(l)} * h^{(l-1)} + b^{(l)})$$

where $W^{(l)} \in \mathbb{R}^{p_{l-1} \times p_l}$ is the weight matrix, $b^{(l)} \in R^{p_l}$ is the bias vector, and $\sigma^{(l)}$ is the activation function of the l-th layer.
The output of the network is then given by

$$f(x) = \hat{y} = \sigma^{(o)}(W^{(o)} * h^{(L)} + b^{(o)}).$$

with $W^{(o)} \in \mathbb{R}^{p_{L+1} \times p_L}$ being the output weight, $b^{(o)} \in \mathbb{R}$ the output bias and σ_o the output activation function.

3.2. Training a neural network

whole section about training of a nn

training loss/ loss function SGD/batches dropout

3.3. RNN

Because a neural network is defined with fixed layer sizes, especially a fixed input dimension it only allows input $x \in \mathbb{R}^{p_0}$. In the marketing attribution setting, we're working with sequential data that has varying lengths for each individual customer journey. To accommodate different customer journeys and recognize that for each touchpoint the same variables are observed a recurrent approach seems fitting as they are a common approach for this type of sequential data. For now, we do not care about the exact time interval between two observations and rather look at the timestamps as establishing an order. The exact time points will be important for the time gate later.
Let $x = (x_1, \cdots, x_T)$ be one sequential observation (i.e. one customer journey) consisting of T chronologically ordered observations within the sequence, i.e. customer-brand interactions and $x_t \in \mathbb{R}^n$ is a vector that collects all the information of the t-th touchpoint. For all touchpoints, the same n features are observed, with the exact time of the touchpoint possibly being one of them.

RNNs continue basic neural networks to sequential usage, they can make predictions after each time step ($\{\hat{y_1}, \hat{y_2}, \ldots, \hat{y_T}\}$) or only after the last time step ($\{\hat{y_T}\}$) using information from all previous time steps. To do so a neural network f with input size $n + p_{L+1}$ is trained to make calculations for each timestep but instead of only using the observations x_t of this timestep one also uses the output h_{t-1} of the network for the previous timestep as input.

$$h_t = f(x_t, h_{t-1})$$

This output h_t is also called *hidden state* and contains all the information of previous timesteps. The initial hidden state h_0 does not exist as no earlier information before x_1 is known, but the vector is needed for calculations. Therefore in general it simply gets initialized as a zero vector.
The hidden state can be viewed as the output of the network or can be used for further calculations.
[Sherstinsky, 2020] gives an in-depth explanation of how to get and unroll a recurrent neural network which is visually represented in Figure 3.3 where H can be understood as just one single layer or a whole network with multiple layers.

If we take RNNs further we can create even more diverse networks. In Figure 3.3 see several different possibilities that can be realized with a recurrent neural network. Especially several recurrent layers put one after another, RNNs where not all layers are recurrent or recurrent connections that not only connect to the same but different layers.

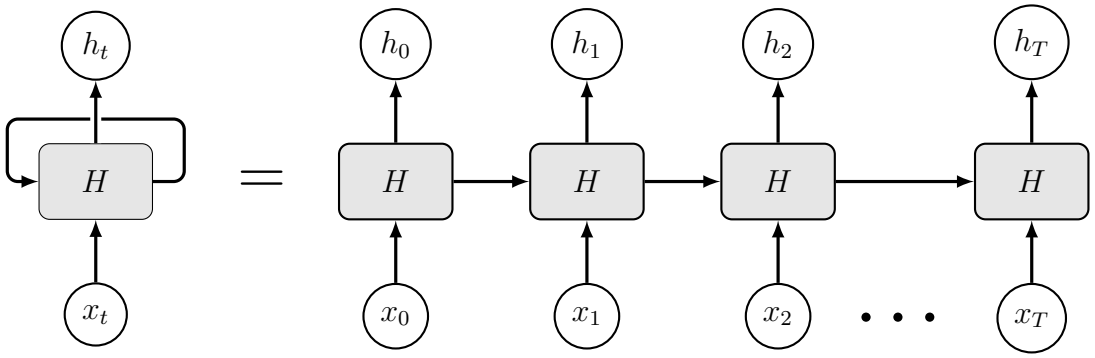
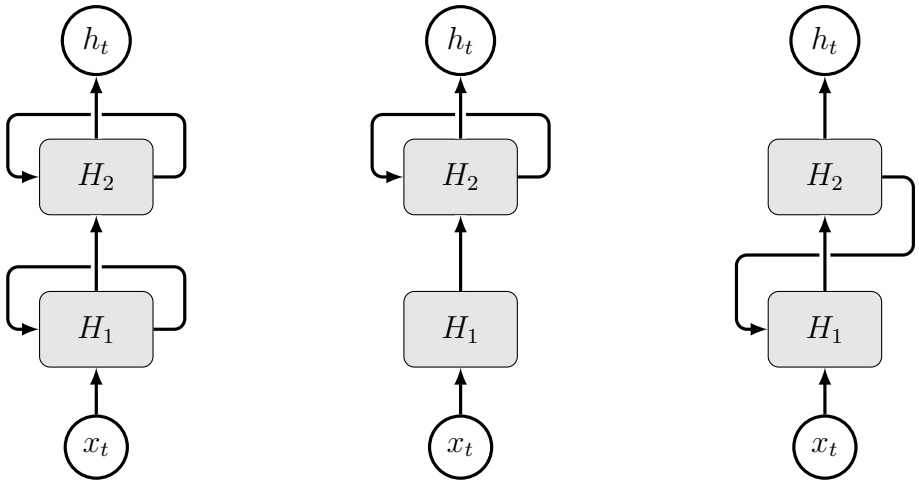


Figure 3.3.: Recurrent Neural Network

When training recurrent neural networks we also want to minimize the training loss and therefore we view the training loss as a function of the weights and biases. To find the the minimal training loss we now use something called backpropagation through time (BPTT) to calculate the gradient. The gradient now reflects the recurrent structure of the network. Because we always use the entire output of a timestep for the next calculations, where it gets multiplied with some weights, when trying to find the derivative we need to go back through all timesteps, and some of the weight matrixes are multiplied several times with themselves. This matrix multiplication can quickly result in either very small or extremely large values, i.e. exploding or vanishing gradients. which means the algorithm used to optimize the parameters can't work well because with large gradients on likely skips over the critical point where loss would be minimized and with small gradients the steps towards this critical point are too insignificant to reach them in a reasonable timeframe. source

[Hochreiter and Schmidhuber, 1997] introduce Long-Short Term Memory (LSTM) Networks to solve this by restricting the information that is remembered from previous time steps appendix LSTM?.



all layers are recurrent	only one recurrent layer	recurrence between different layers

Figure 3.4.: Different possibilities for RNNs

3.4. Gated Recurrent Units

Gated recurrent units (GRU) as proposed by [Cho, 2014] are a simplification of LSTM cells. They have the advantage of using less memory space and fewer calculations while retaining the same benefits of generally leading to improved performance and not having the exploding/vanishing gradient problem that LSTMs were created for.

A gated recurrent unit corresponds to one layer of a RNN. We sometimes will use the term GRU cell to especially refer to a single unit/layer and differentiate it from a whole network.

To define gated recurrent units we need the Hadamard product:

Definition 3.5. The Hadamard Product $\odot$ is defined as an operation between two matrices of the same dimension as elementwise multiplication. Let $A, B \in \mathbb{R}^{n \times m}$ with $n, m \in \mathbb{N}$, then

$$A \odot B = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{pmatrix} \odot \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1m} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nm} \end{pmatrix}$$

$$= \begin{pmatrix} a_{11} * b_{11} & a_{12} * b_{12} & \cdots & a_{1m} * b_{1m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} * b_{n1} & a_{n2} * b_{n2} & \cdots & a_{nm} * b_{nm} \end{pmatrix}$$

Definition 3.6. We define the gated recurrent units as proposed in [Cho, 2014] with two gates. Assume $x_t \in \mathbb{R}^{p_0}$

A reset gate:

$$r_t = \sigma_r \left(W_{rx} x_t + W_{rh} h_{t-1} + b_r \right) \tag{3.1}$$

and an update gate:

$$z_t = \sigma_z \left(W_{zx} x_t + W_{zh} h_{t-1} + b_z \right) \tag{3.2}$$

with σ_r, σ_z the sigmoid activation function $\sigma(x) = \frac{1}{1+\exp -x}$. Both gates return values in $[0, 1]$ in each position, 0 meaning the gate for this variable is fully closed, and 1 that it is open.

We then get a hidden state candidate

$$\tilde{h}_t = \sigma_h (W_{hx} x_t + W_{hh} (r_t \odot h_{t-1} + b_h) \tag{3.3}$$

and finally the hidden state is

$$h_t = z_t \odot h_{t-1} + (\mathbb{1} - z_t) \odot \tilde{h}_t \tag{3.4}$$

which also is the output for this timepoint. 1 sollte $(1, \cdots, 1) \in \mathbb{R}^{p_t}$ Vektor

It is obvious that when the update gate returns a value close to 0, it ignores the previous hidden state on the other hand if it is close to 1 the current information is ignored and only previous information is used for the new hidden state.

3.5. Time gate

In this thesis, an additional time gate which [Neil et al., 2016] introduced for LSTM cells creating a so-called phased-LSTM will be added to the gated recurrent units to form a phased GRU network.

Definition 3.7. The time gate [Neil et al., 2016] proposed uses the timestamp t of the current input and calculates the openness of the time gate k_t which then controls how the output of the layer gets updated. It uses three different parameters for the calculation, which can be learned during the training phase. The time gate opens and closes periodically, so first calculate an auxiliary variable φ_t, to determine the phase of the current cycle.

$$\varphi_t = \frac{(t-s) \mod \tau}{\tau} \in [0,1)$$

It uses τ, which controls the real-time period and s phase shift of the oscillation. Furthermore, there is the parameter r_{on} which determines the ratio of the duration of the open phase to the full period.
Then use this to calculate the time gate

$$k_t = \begin{cases} \frac{2\varphi_t}{r_{on}}, & \text{if } \varphi_t < \frac{1}{2} r_{on} \\ 2 - \frac{2\varphi_t}{r_{on}}, & \text{if } \frac{1}{2} r_{on} < \varphi_t < r_{on} \\ \alpha\varphi_t, & \text{otherwise} \end{cases} \tag{3.5}$$

Explanation 3.7.1. *The time gate 3.5 has two phases, the open and the close phase. In the first half of the open phase ($\varphi_t < \frac{1}{2} r_{on}$) the time gate gradually opens from 0 to 1. and closes back down from 1 to 0 in the second half ($\frac{1}{2} r_{on} < \varphi_t < r_{on}$). While $\varphi_t \geq r_{on}$ the gate is in its closed phase, where instead of setting $k_t = 0$ and cutting off all information only a small portion ($\alpha << 1$) of the current state is let through. This way important information can still make an impact later on and is not completely lost, while the remaining information is reduced enough for the time gate to make sense. A similar thing is done with leaky ReLu instead of a strict ReLu.*

Now a phased GRU cell can be defined by extending the basic GRU cell.

Definition 3.8. We can use 3.1 3.2 and 3.3 from the GRU cell. Instead of the hidden

state

$$\bar{h}_t = z_t \odot h_{t-1} + (\mathbb{1} - z_t) \odot \tilde{h}_t \tag{3.6}$$

is another hidden state candidate and

$$h_t = k_t \odot \bar{h}_t + (\mathbb{1} - k_t) \odot h_{t-1} \tag{3.7}$$

is the actual hidden state

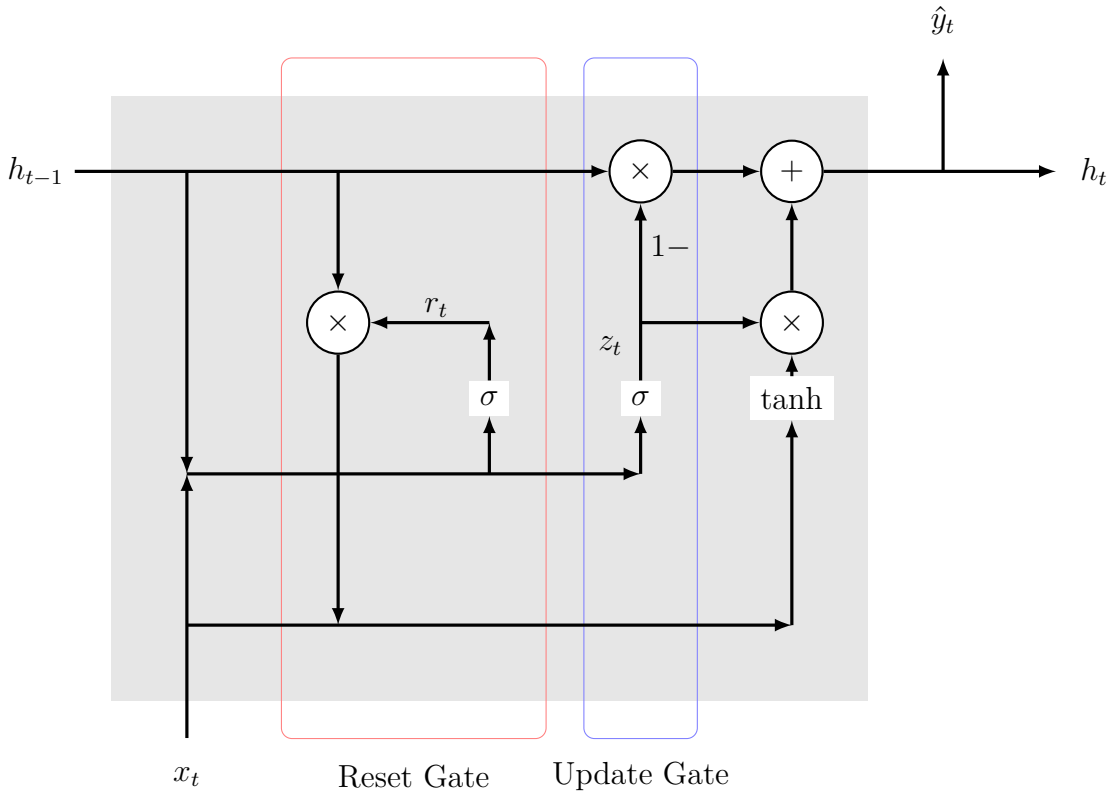


Figure 3.5.: Gated Recurrent Unit

one gru cell corresponds to one layer, multiple layer are possible

prediction does not need to happen each time step.

For the MTA model two GRU layers and a softmax layer prediction only after final timestep $\hat{y} = (\hat{y}_0, \hat{y}_1)^T \in [0,1]^2$ with $\hat{y}_0$ predicted probability for no conversion, $\hat{y}_1$ predicted probability for conversion

This concludes the conversion predictor

4. Value allocation with Shapley

In the machine learning world, one often deals with "black box" models. This refers to the fact that those models take some input and return an output with no explanation or interpretation of what leads to this result. Shapley values have been introduced as a game theoretic approach to determine the value of each player in an n-person game. Nowadays the same approach is applied to machine learning "black box" models to determine the value of the input feature expression.

In the previous chapter, we introduced the structure of a conversion prediction model that fits the black box characterization. While the hope is that the additional time gate lead to even more precise conversion prediction results our main goal of determining the importance of each marketing channel for conversion still remains unanswered by this neural network. We assume that a well-trained prediction network has learned the importance of the individual channels and the influence they have on each other, which then can be extracted using SHAP values.
The goal of this chapter is to give an overview of the origin of the game-theoretic value attribution approach known as Shapley values and its application to machine learning especially in the form of SHapley Additive exPlanatios (SHAP).

4.1. Origin of Shapley

Shapley values were introduced as a solution to assign a value to each player in cooperative game theory. Lloyd Shapley first published his "notes on n-person game" [Shapley, 1951] where he describes how to allocate value to each player, having applications in economy or military in mind. He collected and refined his ideas which he published a year later [Shapley, 1952] titled "The value of n-Person Games".

Definition 4.1. Shapley defines a *game* as a subadditive set-function v from the set of all players U to the real numbers, satisfying:

- $v(\emptyset) = 0$
- $v(S) \geq v(S \cap T) + v(S \setminus T)$, for all $S, T \subseteq U$

Definition 4.2. A *carrier* of v is any set $N \subseteq U$ with:

$$v(S) = v(N \cap S) \text{ for all } S \subseteq U$$

Definition 4.3. permutation

Definition 4.4. The value of a game $\phi(v)$ is defined as a function that maps each player $i \in U$ to a real number $\phi_i(v)$ and satisfies the following three axioms he calls it axioms but they are actually properties imo

Axiom 1: For each $\pi \in \Pi(U)$,

$$\phi_{\pi i}(\pi v) = \phi_i(v), \text{ for all } i \in U$$

Axiom 2: For each carrier N of v,

$$\sum_N \phi_i(v) = v(N)$$

Axiom 3: For any two games v and w,

$$\phi(v + w) = \phi(v) + \phi(w)$$

explanation what axiom does

Theorem 4.5. *There exists a unique value function ϕ satisfying axioms 1-3, for games v with finite carriers N.*

$$\phi_i(v) = \sum_{S \subseteq N} \gamma_n(s)[v(S) - v(S \setminus \{i\})], \textit{ for all } i \in U$$

with $\gamma_n(s) := \frac{(s-1)!(n-s)!}{n!}$.

The Axioms can be easily verified, for proof see [Shapley, 1952]. The resulting values of these functions are nowadays known as Shapley values(of a n-person game).

4.2. Shapley in ML

The problem in explaining machine learning models is that they are too complicated to directly interpret as is possible for example for the parameters in linear regression. Therefore it is necessary to implement a simplified explanation model that can be interpreted. One class of possible explanation models are the additive feature attribution methods. The following section will summarise the application of explanations for machine learning models as presented in detail by [Lundberg and Lee, 2017].
First of all, assume a model f and input x that is to be explained.
We then consider simplified inputs x' that map the original inputs $x = h_x(x')$. A local explanation model g tries to ensure that $g(z') \approx f(h_x(z'))$, where $z' \approx x'$. This model g is simple enough to give explanations of the features values.

Definition 4.6. The explanation model of additive feature attribution methods is a linear function of binary variables:

$$g(z') = \phi_0 + \sum_{i=1}^{M} \phi_i z'_i,$$

where $z' \in \{0,1\}^M$, M is the number of simplified input features, and $\phi_i \in \mathbb{R}$.

Some additive feature attribution methods are LIME (Locally Interpretable Model-agnostic Explanations) citeMarco Tulio Ribeiro why should I trust you, DeepLift citeAvanti Shrikumar learning important features or Layer-wise relevance propagation citeSebastian bach On pixel-wise explanations. When transferring the Shapley values to a machine learning setting the input features (or their expression) are equivalent to the players in game theory the question remains how to define a game value for a subset of features.
There are some ways to estimate the classic Shapley value.
Probably the directest way is using Shapley regression values which require refitting the model on all possible subsets S of features F:

$$\phi_i = \sum_{S \subseteq F} \frac{|S|!(|F| - |S| - 1)!}{|F|!} \left[f_{S \cup \{i\}}(x_{S \cup \{i\}}) - f_S(x_S) \right], \tag{4.1}$$

where f_S is the model trained on the feature subset S and x_S is the input vector restricted to the features in S. With $\phi_0 = f_\emptyset(\emptyset)$ this model belongs to the additive feature attribution methods. Also, two methods called Shapley sampling values and Quantitative input influence use the same approach as in (4.1), but add some approximation methods to reduce the amount of necessary computations.

One often instead resorts to SHAP, short for Shapley Additive exPlanations, which belongs to the additive feature attribution methods and is closely related to Shapley values and often used synonymously. SHAP values have been proposed by [Lundberg and Lee, 2017]. they, introduce three desired properties for explanation models.

Assume a model f with input x for which we want a local explanation using an explanation model g and simplified input x'. Then this model should hold the following properties:

Property 4.7 (Local accuracy)**.**

$$f(x) = g(x') = \phi_0 + \sum_{i=1}^{M} \phi_i x'_i$$

The explanation model should match the output of f at least for the original input x.

Property 4.8 (Missingness)**.**

$$x'_i = 0 \Longrightarrow \phi_i = 0$$

When the simplified input indicates a feature as missing, it shouldn't have any attributed impact.

Property 4.9 (Consistency)**.** *Let $f_x(z') = f(h_x(z'))$ and $z' \setminus i$ denote setting $z'_i = 0$. For any two models f and f', if*

$$f'_x(z') - f'_x(z' \setminus i) \geq f_x(z') - f_x(z' \setminus i), \text{ for all } z' \in \{0,1\}^M,$$

then $\phi_i(f', x) \geq \phi_i(f, x)$.

Theorem 4.10. *With the condition of fulfilling property 1-3 and also definition 4.6 there is only one possible explanation model:*

$$\phi_i(f, x) = \sum_{z' \subseteq x'} \frac{|z'|!(M - |z'| - 1)!}{M!} \left[f_x(z') - f_x(z' \setminus i) \right] \tag{4.2}$$

For the SHAP Shapley Additive eXplanations we use this explanation model and $f_x(z') = f(h_x(z')) = \mathbb{E}[f(z) \mid z_S]$ with S being the set of the non-zero indexes of z'. The resulting SHAP values "closely align with the Shapley regression, Shapley sampling, and quantitative input influence feature attributions, while also allowing for connections with LIME, DeepLIFT, and layer-wise relevance propagation"([Lundberg and Lee, 2017]). They also note that concrete calculation is very complicated but estimation

5. The Data

A proof of concept for the proposed marketing attribution model involves its actual implementation. But in order to train and evaluate it, we first need relevant data, which is introduced in this section.
Beginning with an exploration of the available data, we subsequently outline necessary transformations essential for the adaptation to our conversion prediction model. The dataset used in this thesis is an anonymized data set from 'GetYourGuide' a German-based online travel agency.

5.1. Data Description

The dataset consists of 1,738,080 observations documenting 7 features, with several of these touchpoint observations belonging to one customer journey. The features, their range and description can be found in the following list:

transaction: 1 - if customer journey lead to conversion; 0 - otherwise

platform: mobileWeb, desktop, android or ios

country_name: country where the touchpoint occurred

journey_id: Identifying all the observations belonging to one customer/customer journey

channel_id: Identifying the 24 different marketing channels (Channel_1, $\cdots$, Channel_24)

timestamp: time when the touchpoint occurred

timestamp_conversion: If a conversion occurred, time of this conversion, else NaN

A key value is given by **journey_id** as it uniquely identifies a touchpoint with a customer which can be used to form the individual customer journeys.
In total there are 748,468 customer journeys, a majority only consisting of a single touchpoint, an average length of 2.322 and a maximum length of 3,143. As customer journeys of this length seem unreasonable and are impractical for implementation reasons we limit the data to the customer journeys with a length of at most 16 touchpoints which is the 0.99 percentile of customer journey lengths. Following this approach, 6,820 customer journeys with at least 17 touchpoints are eliminated leaving us with 1,514,307 observations.
The target value for the phased GRU network to predict is the variable **transaction**. For all customer journeys with conversion (*transaction==1*), the timestamp of the conversion is recorded and can be used to identify touchpoints that happen after conversion. As those touchpoints cannot influence the previous conversion we remove another 26 touchpoints, so we effectively work with 1,514,281 touchpoints and 741.646 customer journeys of which a total of 13,506 (1.822 % of all) customer journeys lead to conversion. This means that the data set is heavily unbalanced with over 98% of customer journeys not having an observed transaction. Therefore, it's important to be careful as a classifier labeling all data as "no transaction" would be highly accurate without learning anything valuable from the data. How this problem is actually dealt with is further elaborated in Chapter 6 when discussing the implementation of the predictor.
As the value of **timestamp_conversion** depends on the outcome it can't be used for conversion prediction and **journey_id** shouldn't be used for prediction beyond grouping the customer journeys together, since it only serves as an identifier and contains no actual information. Therefore the next section does not consider those variables and focuses on **channels**, **country_name**, and **platform** instead.

Channels:

The data set contains a total of 24 distinct marketing channels that serve as potential touchpoint avenues, which are the focal points in the MTA model. In the following paragraph a more in-depth analysis for the channels is performed.

Upon examining the distribution of these channels across customer journeys, it becomes apparent that their occurrence does not follow a uniform pattern. Figure 5.1 illustrates this observation as it visualizes the number of channel occurrences across all customer journeys (not counting duplicates per customer journey) in a histogram. It's evident that

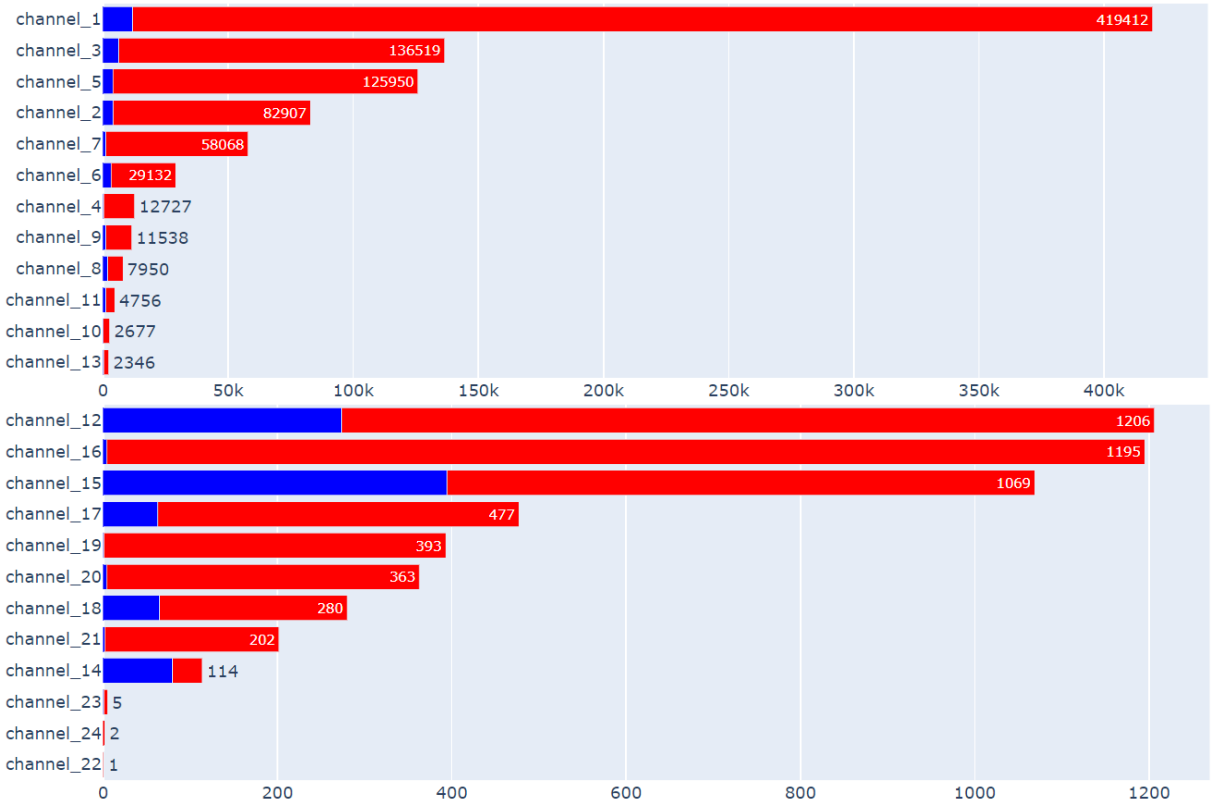


Figure 5.1.: Channel Occurences

some channels are encountered in only a handful of customer journeys, like channel_22 which is only present in a single customer journey, while some other channels are remarkably prevalent, being observed several hundred thousand times.

Furthermore, Figure 5.1 divides the bars into a blue part, representing those customer journeys that ended in a transaction, and a red part representing the remaining customer journeys for which no conversion was documented. It is noteworthy that across nearly all instances, the red portion of the bars surpasses the blue counterpart with channel_14 being the only exception there. However, it is important to highlight that for all channels the relative proportions between these two categories vary.

To further explore these proportions, in Figure 5.2 the ratio of customer journeys containing a channel that actually led to conversion is visualized. For example of all customer journeys that contain channel_14 around 70% led to conversion, while of the customer journeys that contained channel_10 only about 5% ended with conversion.

As a result of the multi-touch attribution model one would expect the obtained channel importances to reflect this distribution to some extent, while also containing some

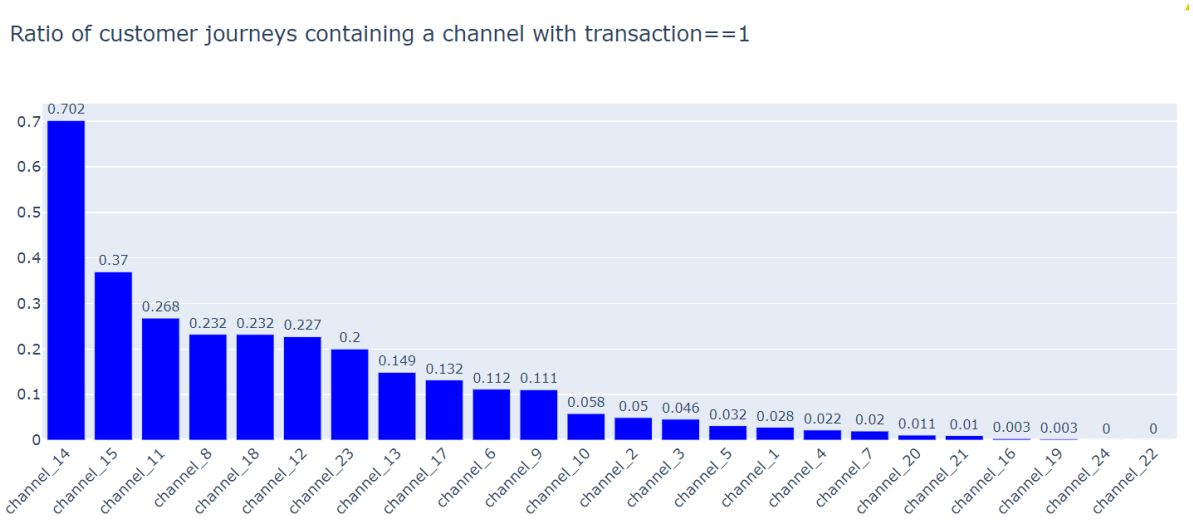


Figure 5.2.: Ratios

differences.

Countries:

Another variable that is recorded is the country, where the customer is located. A total of 233 different countries were observed in the dataset. The location is observed for each touchpoint separately and thus does not have to stay constant within a customer journey.

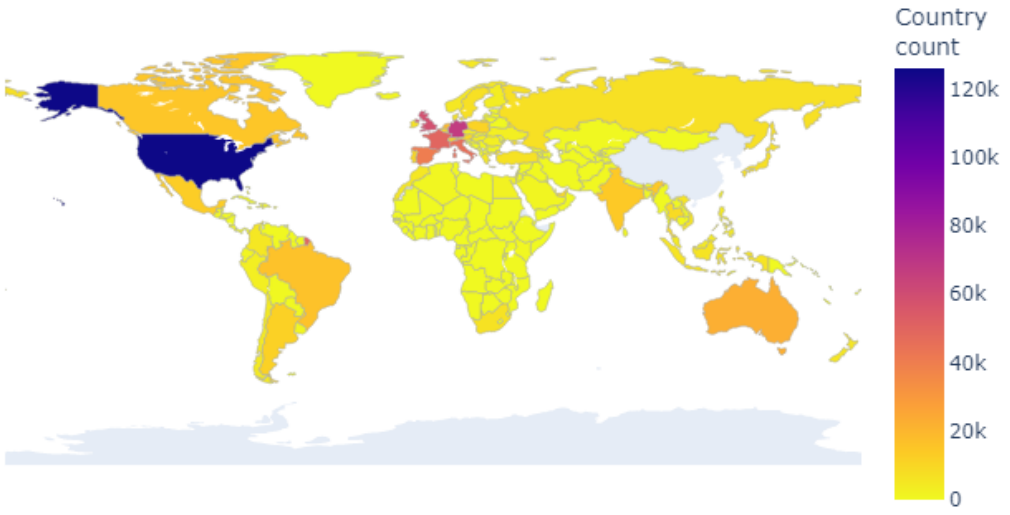


Figure 5.3.: Touchpoint location

In Figure 5.3, a visual representation of the global distribution of touchpoints is displayed. It's visible that touchpoints are dispersed worldwide, spanning from the United States to

the Vatican, while China is a notable exception. This is surprising as there are offers for China on the GetYourGuide website but is probably caused by the censorship of the Internet in China and the common usage of VPN while traveling there and therefore masking China as the actual location. The United States boasts the highest number of touchpoints, closely followed by Western European countries, while the fewest touchpoints are found across Asia and Africa.
This does not represent the customer's nationality. As mentioned in the chapter introduction the data stems from an online travel agency therefore it's very possible that customers are traveling when exposed to a touchpoint.

Platform:

The variable **platform** holds information on the devices the touchpoint recorded on. The four possible expressions for this feature are *mobileWeb*, *desktop*, *android*, and *ios*.

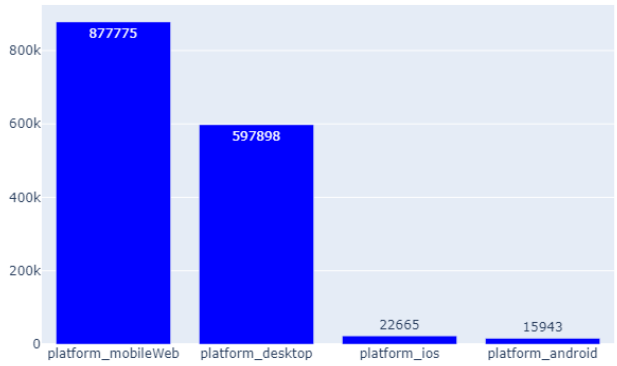


Figure 5.4.: Platform

In this Histogram 5.4 one can observe two groups forming. *mobileWeb* and *desktop* are both in a similar range and together represent the majority of all touchpoints. On the other hand *ios* and *android* are also in a similar range but together only represent around 2.5% .

With the available features we have no information on the customers or any other brand interactions such as click events, therefore no causal model could be implemented for this data.

5.2. Data Transformation

The data set previously has been used for a different MTA approach, and thus some transformation is necessary to make it compatible with the phased GRU network. In the previous section already some changes to the data set were established with which we continue to work. Even though we said to remove the *journey_id* we will still use it for the transformation as an identifier in this section. The first transformation is to create dummy variables for the categorical variables *channel, platform* and *country_name*. A dummy variable is created by replacing the variable column with columns for each possible expression of this variable. Each of these newly created columns is assigned '1' if a given observation expressed the respective categorical value and '0' else.
This greatly expands the size of the data set but is the best way to handle categorical data, that can not easily be transformed in an ordinal way.
The next transformation is to group together all touchpoints belonging to the same customer in order to form individual customer journeys. Previously the format of the dataset was in a table format *touchpoint* $\times$ *features* where each row corresponds to a touchpoint observation with the observed features in the columns and the customer journeys aren't apparent. For the GRU network, we want to use customer journeys grouped together as one observation, which adds another dimension *customer* $\times$ *touchpoints* $\times$ *features*. So for each customer c with a customer journey of length L, we get $x_c = (f_l^T, t_l)_{l=1\cdots,L}$, where f_l collects the recorded features of the l-th touchpoint, t_l the timepoint of this touchpoint. We deliberately put the timestamp in the last position of the created tensor in order to extract the time by simply accessing the last position using the index "*-1*". These timestamps are then used for the time gate. A problem that emerges with this approach are the varying customer journey lengths. While ragged tensors exist and could be used here to create tensors with an unspecified dimension size, ragged tensors are not very practical and cannot be used efficiently as of today. In an article [Patton, 2022] presented an example that illustrates, that while using ragged tensors he was able to achieve marginally improved results but the computations for each training epoch took around 60 times more time compared to the same model trained with padded input. Therefore we also use padded input data, meaning all customer journeys get filled up with zeros to achieve the same length of 16. Here the decision to exclude long customer journeys proves to be helpful, as we don't have to artificially inflate the data set in order

to keep all 3,143 touchpoints of the longest customer journey.

As target value, we use $y_c \in \{0, 1\}$, the **transaction** for each customer which identifies two classes the successful and unsuccessful customer journeys. Therefore our prediction task is in fact a classification problem and we need to predict the class once per customer journey, at the end, when all touchpoints are considered.

The whole data is then split into a training and a test data set to appropriately fit and evaluate the model. 75 percent of customer journeys are randomly selected as training data in order to allow the model to learn well. The remaining 25 percent of the data is used to evaluate the predictions of the model after training is done.

Finally the data is split into batches of a fixed size to not overstrain storage space and achieve a faster training speed.

6. The Model

The proposed marketing attribution model is composed of two steps a phased GRU network for conversion prediction and Shapley explanations for value attribution, which previously have been explained in detail. To test out the actual capabilities of this proposed multi-touch attribution model it has been realized in Python.
This chapter summarizes the actual realization of the phased GRU model in Python and the subsequent application of Shapley explanations. Before getting to the evaluation of the trained models a few metrics on which the model comparison is done are introduced. Finally, the trained phased GRU model will be compared to its counterpart without time gate as well as a basic and phased LSTM network.

6.1. Implementation of phased GRU

The implementation uses the Python TensorFlow library as the basis for the model, one of the main machine learning packages that are available for Python, that contains the necessary functions to create and train neural networks.

While a tensorflow.keras.Sequential() model allows stacking several layers on top of

```
modelphasedGRU = tf.keras.Sequential()

modelphasedGRU.add(
    tf.keras.layers.RNN(
        phased_GRUCell(...)
    )
modelphasedGRU.add(
    tf.keras.layers.RNN(
        phased_GRUCell(...)
    )
modelphasedGRU.add(tf.keras.layers.Dense(num_classes, activation='
    softmax'))
```

Figure 6.1.: Network Structure

each other the tensorflow.keras.layers.RNN() layer structure can be used to achieve a recurrent network layer that can process time series data and use the output of the current time step as input for the next time step. The actual phased GRU Cell that does all the computations in each time step, including the time gate calculations wasn't already available to use in this package and had to be separately implemented.

We use the source code of the basic GRU cell that can be found in the TensorFlow library [TensorFlow, 2023] as a foundation from which the phased GRU cell class could be built by adding the time gate.

The phased GRU cell process one time step $l \in \{1, \cdots, 16\}$ of a customer journey $c \in C$ at a time $x_{c,l} = (f_{c,l}, t_{c,l})$. In order to implement the time gate, the time variable has to be extracted from the rest of the input. To access it, it has been placed as the last feature which can be referenced with the index -1 and stored separately from the other input variables. Then the necessary weights of the time gate are initialized as $\tau \sim \mathcal{N}^{p_l}(0, 300),\ s \sim \mathcal{N}(0, 3000),\ r_{on} \sim \mathcal{N}(0, 0.5),\ \tau \sim \mathcal{N}^{p_l}(0, 300),\ \alpha \sim \mathcal{N}(0, 0.5)$ as trainable parameters to get them in a range where they will end up.

The actual computations for the time gate are implemented as seen in 6.2 with one function determining the phase of the current time gate φ and the other function returning the corresponding openness of the gate, and then using it to compute the hidden state. At the end of all the calculations within the phased GRU cell, if another phased GRUCell follows the current cell, the time variable has to be added to the output of this cell again, so the time gate in the next layer works properly. If it's the last phased layer, we have to make sure that it's not attached and returned to the next layer. In order to make sure everything is functioning properly the phased GRU Cell object class is created with two additional boolean parameters:

time_gate *True*: to use the time gate; *False*: to use the Cell as a basic GRU Cell without a time gate

last_layer only needed if **timegate == True**; *True*: if next layer uses a time gate and the time needs to be passed onto the next layer; *False*: if the next layer has no time gate

As seen in 6.1 the actual model uses two phased GRU layers, followed by a Dense layer with softmax activation, returning a two-dimensional output $\hat{y}_c \in [0, 1]^2$ with the first

```python
def phi_fast(time, s, tau):
    x = time - s
    x = tf.math.floormod(x, tau)
    x = tf.math.divide(x, tau)
    return x

def time_gate_fast(phi, ron, alpha):
    cond_1 = tf.cast(tf.less_equal(phi, 0.5 * ron), dtype='float32')
    cond_2 = tf.cast(tf.logical_and(tf.less(0.5 * ron, phi), tf.less(phi
    , ron)), dtype='float32')
    cond_3 = tf.cast(tf.greater_equal(phi, ron), dtype='float32')

    term_1 = tf.math.multiply(cond_1, 2.0 * phi / ron)
    term_2 = tf.math.multiply(cond_2, 2.0 - 2.0 * phi / ron)
    term_3 = tf.math.multiply(cond_3, alpha * phi)

    return term_1 + term_2 + term_3
```

Figure 6.2.: Time gate

component $\hat{y}_{c,0}$ representing the predicted probability for no transaction and the second component $\hat{y}_{c,1} = 1 - \hat{y}_{c,0}$ the probability for transaction.

As loss function we choose "sparse categorical crossentropy loss" as the true target values $y_c \in \{0, 1\}$ are given as integer and the predicted labels as floating point values per possible output class $\hat{y}_c \in [0, 1]^2$. In this case of binary classification the loss function returns:

$$\begin{aligned} l(y_c, \hat{y}_c) &= -y_c \cdot \ln(\hat{y}_{c,1}) - (1 - y_c) \cdot \ln(\hat{y}_{c,0}) \\ &= \begin{cases} -\ln(\hat{y}_{c,1}) \ , & \text{if } y_c = 1 \\ -\ln(\hat{y}_{c,0}) \ , & \text{if } y_c = 0 \end{cases} \end{aligned}$$

This loss function penalizes wrong predictions very harshly, as the function grows rapidly, if the model predicts the wrong class with high probability $-\ln(x) \xrightarrow{x \to 0} \infty$.

So the overall loss is computed as the average individual loss

$$L = \frac{1}{C} \sum_{c=1}^{C} l(y_c, \hat{y}_c) = -\frac{1}{C} \sum_{c=1}^{C} y_c \cdot \ln(\hat{y}_{c,1}) + (1 - y_c) \cdot \ln(\hat{y}_{c,0}).$$

The model will then be fitted on the training data set minimizing this loss function by using the adam optimizer to. explain adam optimizer

6.2. SHAP

With the just created prediction model we now want to generate feature importances for the different marketing channels in the form of Shapley Additive exPlanations by using the shap library.

The shap package is the most commonly used Python library for XAI tasks as it unifies several different estimation approaches, for example *TreeExplainer, DeepExplainer, PermutationExplainer, GradientExplainer,...*, that are optimized for a variety of different machine learning algorithms. The different explainer options provide global as well as local explanations and additionally offer easy comprehensible visualization options for the explanation model.

As we work with a deep learning model a DeepExplainer is the right explainer to use for the conversion prediction model, which is also used as the example explainer for an LSTM model on their official website [1] and therefore should work for our model. But when applying it to the trained model one quickly realizes that this doesn't work as expected and throws errors left and right. Looking through the problem reports for shap on GitHub, a few issues regarding the compatibility of shap and TensorFlow can be found. There are two frequently suggested solutions, to downgrade TensorFlow to an earlier version, especially versions <2.0.0, where the shap package had no problem with a sequential model. However, the model is built on the newer contents of Tensorflow and is not easily transferable to an older outdated version. Another suggested solution is to flatten the input, which would mean all timesteps of a customer journey are combined into one long array. In this approach all timesteps would be processed at the same time, which would contradict the purpose of using a sequential model in the first place, and especially the time gate would be useless.

The most recent similar problem report is issue 3344 [2] from October 16th where a user encountered the same problem of not being able to generate shap explanations with the DeepExplainer for his LSTM network. A contributor of the package acknowledged the problem with the package and said he would look into solutions and update the problem report when a fix is found, which as of now (17.11.2023) is not the case.

As the other explainers in the package don't work as well, the only way to get shapley ex-

[1] https://shap.readthedocs.io/en/latest/example_notebooks/text_examples/sentiment_analysis/Keras%20LSTM%20for%20IMDB%20Sentiment%20Classification.html

[2] https://github.com/shap/shap/issues/3344

planations for our model would be to implement something ourselves, which would exceed the scope of this thesis.

6.3. Metrics

To evaluate and compare the models we use a few different metrics, which will briefly be introduced so that we can concentrate on comparing the models in the next section.
We compute the loss function on the training and the test dataset separately. During training the the model parameters are adjusted so that the model fits the training data set best
and the test loss or test error is a better estimate for the actual !!! WAS??
We can lable a customer journey a transaction at the end of a customer journey a positive result and consequently no transaction a negative result. Comparing the predicted and real labels of the test data set we can form a *Confusion Matrix* (6.3):
The True Positive and True Negative predictions combined form all correct classifications

	predicted: transaction (PP)	predicted: no transaction (PN)
actual: transaction (P)	True Positive (TP)	False Negative (FN)
actual: no transaction (N)	False Positive (FP)	True Negative (TN)

Figure 6.3.: Confusion Matrix

of the model. Calculating the percentage of correct predictions we obtain the accuracy of the model:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}.$$

As in the last chapter discussed accuracy alone isn't expressive, especially with this very unbalanced dataset, a few other metrics we can calculate from this confusion matrix are

$$Precision = \frac{TP}{PP} = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{P} = \frac{TP}{TP + FN} = TruePositiveRate = TPR$$

and

$$FPR = FalsePositiveRate = \frac{FP}{N} = \frac{FP}{FP + TN}.$$

The precision returns the fraction of correct predictions when looking only on positive predicted. The recall is also called true positive rate, returns which fraction of actually positive observations have been identified as such by the model. The data set contains only very few actually positive observations therefore having a good recall is especially important to ensure the model learned something and not only predicts the negative class. On the other hand the false positive rate

The predictions we obtain using the phased GRU and the three comparison models don't simply classify the observations as zero or one but return probabilities for each class. From those probabilities, a classification is made choosing the class for which the probability is higher. This requires testing if $\hat{y}_{c,1} > \hat{y}_{c,0}$ and as we only have two classes and the probabilities sum up to 1, that's equivalent to checking $\hat{y}_{c,1} > \frac{1}{2}$.

In case we want to allow more or less positive predictions we can move this threshold. When we lower the threshold more observations will be put in the positive class, when raising it more will be put in the negative class. As we want the model to be very confident in the predictions observing the metrics for different thresholds makes sense.

In the receiver operating characteristic (ROC) curve we plot the TPR against the FPR. A perfect classifier's ROC curve would stick to the top left corner of the $[0, 1]^2$ square. While a model that classifies purely random would stick to the diagonal. Therefore a model whose ROC is below the diagonal should be scraped, as it performs worse than a random guess, and any model further in the top left corner should be preferred. To simplify the comparison of ROC curves we can calculate the area under the curve (AUC) which takes values between 0 and 1. The better a model is the closer the AUC will be to 1.

Another similar metric that we will evaluate is the precision-recall (PR) curve. In a similar fashion to the ROC curve, precision and recall for different thresholds are plotted against each other. Here a perfect model would stick to the top right corner and a random model's PR-curve would be constant ???

We compute those metrics for the phased GRU and the three comparison models with our test data set in order to have a good basis on which we can compare the models.

6.4. Evaluation

In order to evaluate the prediction of the phased GRU model we compare it to other models with a similar structure to see how well it performs. The models used in the comparison are a phased LSTM and a basic GRU and LSTM network to see the effect of the time gate and see if LSTM networks, which are the predecessor of GRU networks and in general are expected to outperform GRU networks, actually work better in this application.

They all use two model specific layers (i.e. phased/basic GRU/LSTM layers), followed by a dense output layer with softmax activation which returns a two-dimensional output vector for each customer journey $\hat{y}_c = (\hat{y}_{0,c}, \hat{y}_{1,c}) \in [0,1]^2$ predicting the probability for each class.

The models are fitted using the same set of hyperparameters seen in 6.4 on the training data set.

Inputsize	(?, 16, 262)
First Layer size	64
Second Layer size	64
Output size	2
Batchsize	5000
Epochs	30
Learning Rate	0.01
Loss function	sparse_categorical_crossentropy

During the model fitting process, the weights, biases, and time gate parameters are updated successively to minimize the training loss. Each epoch involves using all batches once, representing a subset of whole the training dataset, batch job . Once all batches have been utilized, the epoch concludes, and both training and test loss are recorded before the next training epoch begins. In 6.4 we see how the training loss changes over the epochs for all four models.

The training loss of the two models without time gate converges quickly after a few epochs the training error of the basic GRU model isn't changing significantly anymore, the training loss of the basic LSTM model does another significant jump down around

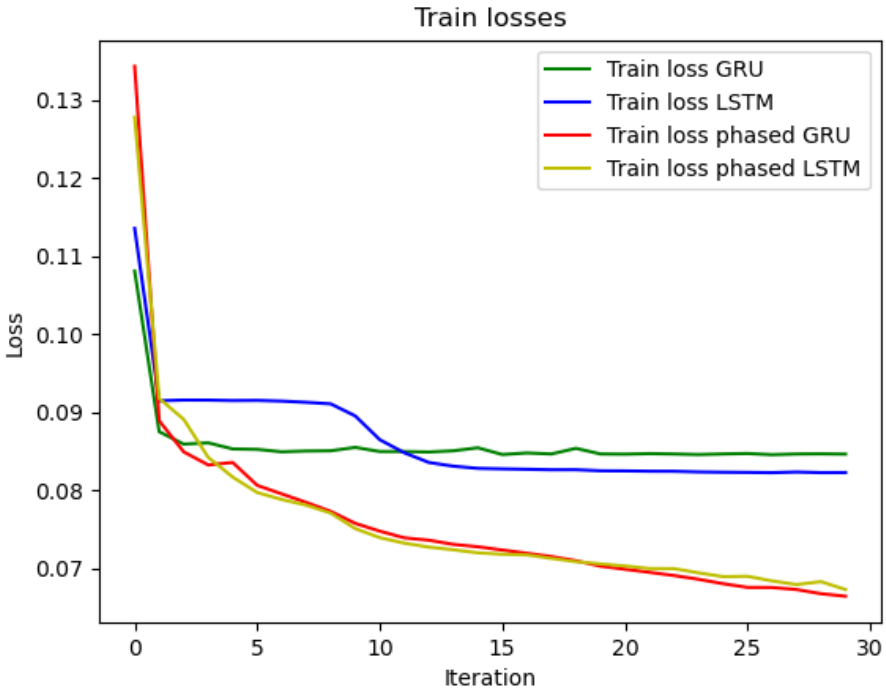


Figure 6.4.: Training losses

the 10th epoch whereafter it remains pretty much constant and ends with a little lower loss than the GRU model. The time-gated models perform substantially better regarding the training loss. Not only do they end with a lower training loss, but after 30 epochs the loss still is decreasing for both models.

Similar development is displayed in the test loss. The networks with time gates perform notably better than the two models without. We see that the small jump in training error we observed in 6.4 for the LSTM network around the 10th epoch corresponds to a huge decrease in test loss.
Therefore we conclude a big effect of the time gate on the performance of the networks.
All other metrics are calculated on the test data

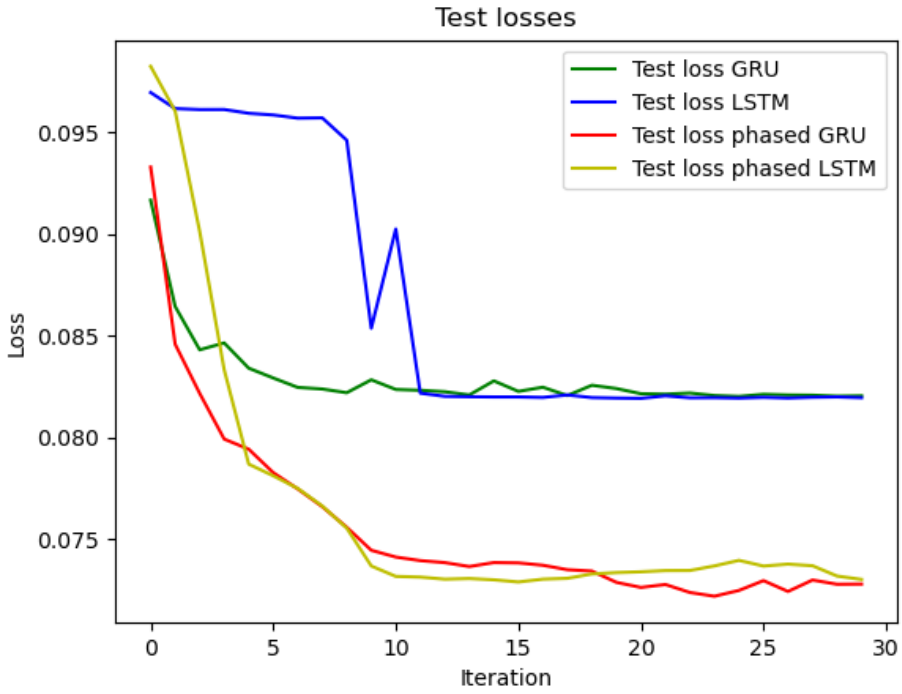


Figure 6.5.: Test losses

	phased GRU	**GRU**	**phased LSTM**	**LSTM**
trainable Parameters	**87,624**	87,682	116,744	116,866
Fitting time	5011s	**3910s**	11808s	5666s
Accuracy	**0.98406**	0.98178	0.98372	0.98178
Precision	**0.78162**	-	0.73281	-
Recall	0.17377	-	0.16726	-
ROC-AUC	**0.81**	0.74	**0.81**	0.74
train loss	0.06646	0.08467	0.06731	0.08230
test loss	**0.07277**	0.08203	0.07300	0.08194
minimal test loss	**0.07218**	0.08200	0.07287	0.08191

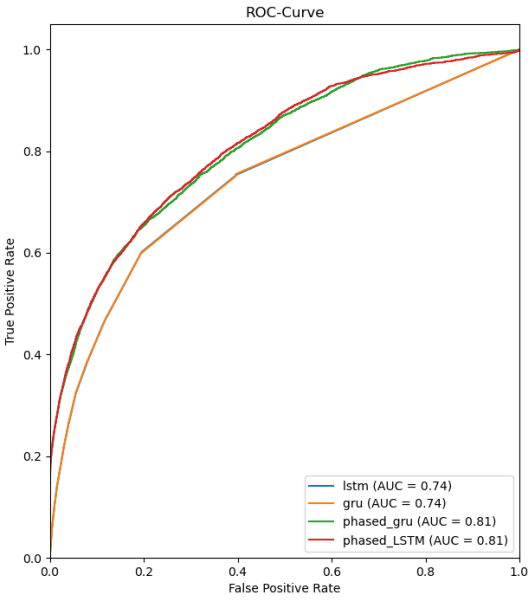


Figure 6.6.: ROC/AUC

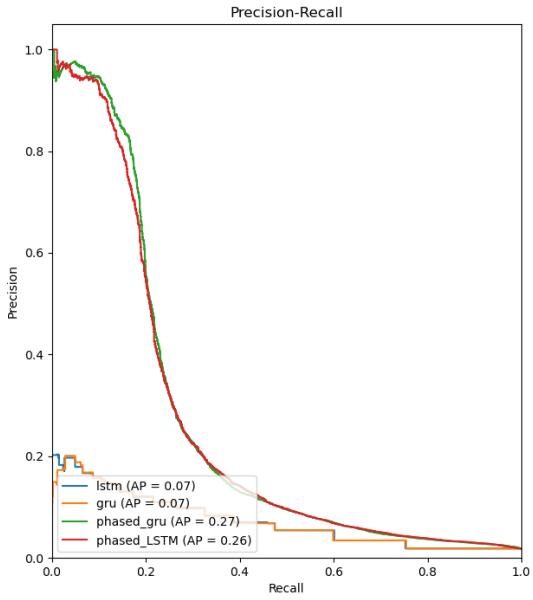


Figure 6.7.: Precision Recall

7. Conclusion/Outlook

einordnung wie ob man das Model so verwenden soll

next steps

As one can see, a phased GRU can predict conversion given a customer journey more reliably than the base GRU and LSTM models while using fewer parameters.

Obviously, the next step would be to add explanations to the model, whether it is using the shap package as soon as they are compatible with sequential data or to programming an own version that is compatible with the phased GRU model.
Even though the predictions might be more accurate using neural networks, especially the phased GRU network, question still remains how much more and better information does such a model provide to a company compared to other simpler or easier interpretable models.

Bibliography

[Buhalis and Volchek, 2021] Buhalis, D. and Volchek, K. (2021). Bridging marketing theory and big data analytics: The taxonomy of marketing attribution. *International Journal of Information Management*, 56:102253.

[Cho, 2014] Cho, K. (2014). Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation.

[Du, 2019] Du, R. (2019). Causally Driven Incremental Multi Touch Attribution Using a Recurrent Neural Network.

[Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780.

[Kumar et al., 2020] Kumar, S., Gupta, G., Prasad, R., Chatterjee, A., Vig, L., and Shroff, G. R. (2020). CAMTA: Causal Attention Model for Multi-touch Attribution.

[Li, 2018] Li, N. (2018). Deep Neural Net with Attention for Multi-channel Multi-touch Attribution.

[Lundberg and Lee, 2017] Lundberg, S. M. and Lee, S.-I. (2017). A unified approach to interpreting model predictions. volume 30, pages 4768–4777.

[Neil et al., 2016] Neil, D., Pfeiffer, M., and Liu, S.-C. (2016). Phased LSTM: Accelerating Recurrent Network Training for Long or Event-based Sequences. volume 29, pages 3882–3890.

[Patton, 2022] Patton, PhD, D. (2022). Using TensorFlow ragged tensors - towards data science.

[Ren et al., 2018] Ren, K., Fang, Y., Zhang, W., Liu, S., Li, J., Zhang, Y., Yu, Y., and Wang, J. (2018). Learning Multi-touch Conversion Attribution with Dual-attention Mechanisms for Online Advertising.

[Shapley, 1951] Shapley, L. S. (1951). Notes on the N-Person Game — II: The Value of an N-Person Game.

[Shapley, 1952] Shapley, L. S. (1952). A value for N-Person Games.

[Sherstinsky, 2020] Sherstinsky, A. (2020). Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network. *Physica D: Nonlinear Phenomena*.

[TensorFlow, 2023] TensorFlow (2023). GRUCell. *https://www.tensorflow.org/api_docs/python/tf/k*

[Yang, 2020] Yang, D. (2020). Interpretable Deep Learning Model for Online Multi-touch Attribution.

[Yao et al., 2022] Yao, D., Gong, C.-D., Zhang, L., Chen, S., and Bi, J. (2022). CausalMTA: Eliminating the User Confounding Bias for Causal Multi-touch Attribution.

A. Appendix

Collection of everything needed in this thesis, which would have disrupted the information presented.

LSTM?

ADAM?

Abkürzungsverzeichnis

KI-Tabelle

Sworn Declaration

I hereby declare, that I have independently created the present work with the title

[Title of the work]

any thoughts taken directly or indirectly from external sources have been identified as such. The work has not been submitted to any other examining authority or published before.
I am aware that a false declaration may have legal consequences.

Ulm, den November 29, 2023 _____

(Leonie Allgaier)