In order to explain the decisions of a prediction we have to introduce a predictive model, which needs to be explained. Any sequence model suffices. Additionally, the model's prediction do not have to be accurate. However, the more accurate the model can capture the dynamics of the process, the better the counterfactual functions as an explanation of these dynamics. This becomes particularly important if the counterfactuals are assessed by a domain expert.

0.0.1 Long Short-Term Memory Models

In this thesis, the predictive model is an Long Short-Term Memory (LSTM) model. LSTMs are well-known models within Deep Learning, that use their structure to process sequences of variable lengths[1]. LSTMs are an extension of Recurrent Neural Networks (RNNs). We choose this model as it is simple to implement and can handle long-term dependencies well.

Generally, RNNs are Neural Networks (NNs) that maintain a state h_{t+1}. The state ist computed and then propagated to act as an additional input alongside the next sequential input of the instance x_{t+1}. The hidden state h is also used to compute the prediction o_t for the current step. The formulas attached to this model are shown in

$$h_{t+1} = \sigma(Vh_t + Ux_t + b) \tag{1}$$
$$o_t = \sigma(Wh_t + b) \tag{2}$$

Here, W, U and V are weight matrices that are multiplied with their respective input vectors h_t, x_t. b is a bias vector and σ is a nonlinearity function. LSTM fundamentally work similarly, but have a more complex structure that allows to handle long-term dependencies better. They manage this behaviour by introducing addional state vectors, that are also propagated to the following step. We omit discussing these specifics in detail, as their explanation is not further relevant for this thesis. For our understanding it is enough to know that h_t holds all the necessary state information. Figure **??** shows a schematic representation of an RNN.

Model Structure

The architecture of the prediction model is shown in Figure 2.

One input consists of an 2-dimensional event tensor containing integers. The second input is a 3-dimensional tensor containing the remaining feature attributes. The first dimension in each layer represents the variable batch size and *None* acts as a placeholder.

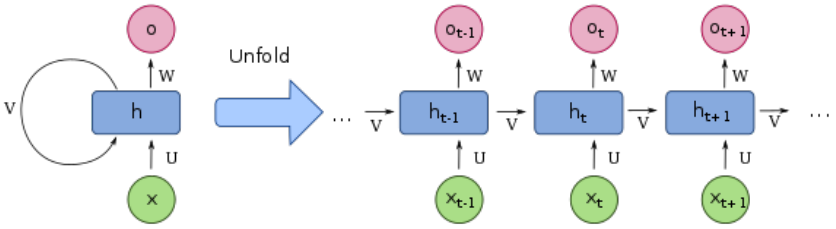

Figure 1: A schematic representation of an RNN viewed in compact and unfolded form**??**.

The next layer is primarily concerned with preparing the full vector representation. We encode each activity in the sequence into a vector-space. We chose a dense-vector representation instead of a one-hot representation. We also create positional embeddings. Then we concat the activity embedding, positional embedding and the event attribute representation to a final vector representation for the event that occured.

Afterwards, we pass the tensor through a LSTM module. We use the output of the last step to predict the outcome of a sequence using a fully connected neural network layer with a sigmoid activation as this is a binary classification task.

0.0.2 Transformer Model

Transformer model are modern sequnetial models within Deep Learning. They have a multitude of advantages over sequential models such as RNNs or LSTMs[2]. First, they do not need to be computed sequentially. Hence, it is possible to parallelise the training and inference substantially using GPUs. Second, they can take the full sequence as an input using the attention mechanism. This mechanism also allows to inspect which inputs have had a significant role into producing the prediction. However, transformer models are more complicated to implement. The overall setup is shown in Figure 3[2].

The transformer model is an Encoder-Decoder model. The encoder takes in the input sequence as a whole and generates a vector which encodes the information. The decoder uses the econding to produce the resulting sequence. The encoder module uses two important concepts. First, in order to preserve the temporal information of the input we encode the position of each sequential input as an additional input vector. We choose to encode every position by jointly learning positional embeddings. The second component is multihead-self-attention. According to Vaswani et al., we can describe

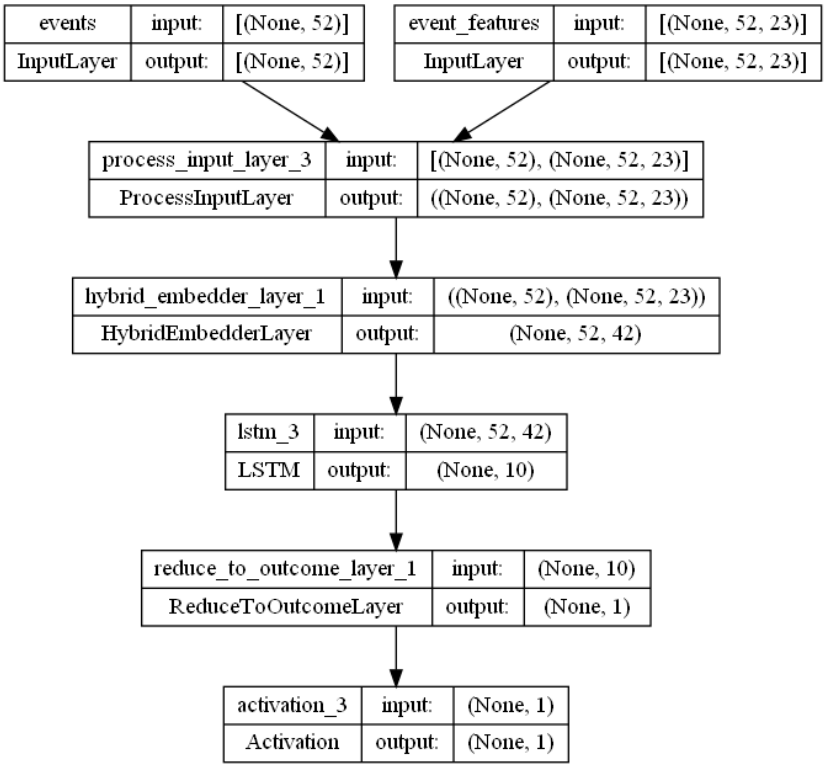

Figure 2: Shows the different components of the LSTM architecture. Each elements contains information about the input and output of a layer. None is a placeholder for the batch size.

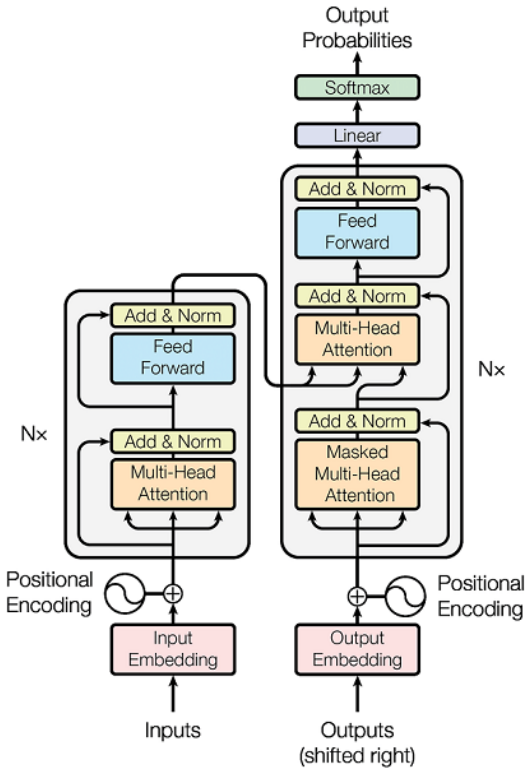

Figure 3: A schematic representation of a Transformer model.

attention as a function which maps a query and a set of key-value pairs to an output. More specifically, self attention allows us to relate every input element in the sequence to be related to any other sequence in the input. The output is a weighted sum of the values. It is possible to stack multiple self-attention modules. This procedure is called multihead-attention. Figure 4 shows how to compute self-attention accoring to Equation 3[2].

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{3}$$

Q, K, V are all the same input sequence. d_k refers to the dimension of an input vector. Note, that T is the transpose operation of matrix computations and does not relate to the time step of the final sequence element.

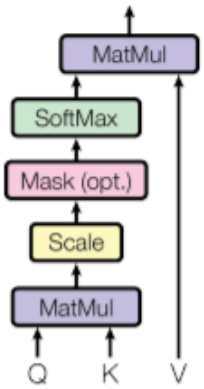

Figure 4: Shows the computational graph of self-attention. Q, K and V are all the same input sequence. Q stands for query, K for key and V for value.