Sketch-based fast and accurate querying of time series using parameter-sharing LSTM networks

Chaoran Fan, Krešimir Matković, and Helwig Hauser

Abstract—Sketching is one common approach to query time series data for patterns of interest. Most existing solutions for matching the data with the interaction are based on an empirically modeled similarity function between the user's sketch and the time series data with limited efficiency and accuracy. In this paper, we introduce a machine learning based solution for fast and accurate querying of time series data based on a swift sketching interaction. We build on existing LSTM technology (long short-term memory) to encode both the sketch and the time series data in a network with shared parameters. We use data from a user study to let the network learn a proper similarity function. We focus our approach on perceived similarities and achieve that the learned model also includes a user-side aspect. To the best of our knowledge, this is the first data-driven solution for querying time series data in visual analytics. Besides evaluating the accuracy and efficiency directly in a quantitative way, we also compare our solution to the recently published Qetch algorithm as well as the commonly used dynamic time warping (DTW) algorithm.

Index Terms—Machine learning, sketch-based interaction, visual analytics, time series data.

1 Introduction

In the emerging era of big data, extensive time series data are common in a large variety of application domains. The visualization of such data is often cluttered, especially when the trend is non-periodic and the data size is large. In the exploration of long time series data, it is often hard for the analysts to visually identify specific patterns efficiently. To overcome this issue, the topic of finding relevant parts of time series data has become popular in recent research.

In general, it is easier to visually describe patterns in time series data than to express them textually or procedurally. Therefore, visual query systems are a convenient user interface with freehand sketching as an efficient means for visual communication. The use of sketching enables the analyst to convey complex free-form patterns of interest, which are matched against the data to identify subsets of interest.

For matching sketches and data, usually a carefully designed, empirical model is adopted to estimating the similarity between the sketch and the time series data. Often, this approach comes with non-optimal efficiency and accuracy, having so far also resulted in a limited deployment of sketch-based visual query systems for real-world visual analytics applications. More specifically in terms of their limited efficiency, most of these empirical methods are based on local characteristics and a sliding window (of the same length as the sketching query) that is used to compute the best match or a similarity ranking, generally leading to a time-consuming comparison procedure that can hamper the interactive exploration. On the other hand, sketches are artistic expressions and due to ambiguity and inaccuracies in sketches, an empirical model is often quite far from robustly representing the underlying ideas and expectations of the user. This can lead to matching algorithms that fail to produce good similarity rankings, especially when "goodness" is evaluated by humans [1].

- *Ch. Fan and H. Hauser are with the Univ. of Bergen, Norway.*
- *Kr. Matković is with the VRVis Research Center, Austria.*

To improve sketch-based querying, we see two main directions. First, in order to secure a fluid data exploration, we aim at a fast computation of the matching procedure. Second, we need a better understanding of the user's intention given her/his sketch—only this way we can make the querying result as close as possible to what the user really needs. Due to great recent success, deep learning in computer vision [2], image classification [3], and natural language processing [4], [5] has attracted a lot of attention. As pattern matching in time series data is somehow similar to detecting patterns in images, we expected that deep learning would boost the performance of matching solutions.

In this paper, we now show a successful exploitation of the long short-term memory (LSTM) architecture [6] to encode the sketch and the time series data respectively in two networks with shared parameters. In principle, two LSTM networks with different parameters could be used to learn the representation of the sketch and the time series data. In our design, the two networks share the same parameters and this parameter sharing helps with accelerating training and limiting overfitting. The networks are trained based on perceived similarities from a user study. This way, we integrate the user's perception into the learned model. As no existing model is capable of fully capturing the complex semantics of a user's sketch, we saw a great potential to improve the situation by learning the matching model directly from users. We demonstrate the effectiveness of our method in comparison with two state-of-the-art matching models—the recently presented Qetch algorithm [1] and the seminal DTW technique (dynamic time warping) [7].

Overall, the main contributions of our paper are:

- **A data-driven method for sketch-based querying of time series data.** To the best of our knowledge, this is the first time that deep learning is used to learn the matching relation between a human sketch and time series data, outperforming two state-of-the-art models in terms of accuracy and efficiency.

- **A sketch-based querying system for time series data.** We present a prototype of a sketch-based querying system for time series data. We offer the user an opportunity to use a freehand sketch to explore the time series data interactively without the need to set any offline parameter.

2 Related work

Sketching is a natural and expressive type of interaction, which has been frequently used in the visualization area, especially as a brushing technique [8], [9], [10], [11], [12] and in visual query systems [1], [13], [14], [15], [16].

In the following, we provide a brief introduction to common time series similarity matching algorithms, followed by a detailed overview of prior work related to visual query systems for time series data and visualization applications based on deep learning knowledge.

2.1 Time series data similarity

Among a variety of similarity measures, the Euclidean distance (ED) and dynamic time warping (DTW) [7] are the most commonly used measures with the squared ED being the sum of the point-wise squared differences of the two time series. The basic ED can be improved by data normalization, often standardization, which considers the variation of similar patterns in amplitude and y-offset [17]. Since ED is computed point-wise and the mapping of a query point to a data point is fixed, it is sensitive to noise and local time misalignments.

DTW overcomes ED's inability to handle local time misalignments (or warps) by allowing horizontal stretching (or compression) of a time series when searching for similar data subsets. Therefore, DTW is considered to yield better fits for shape matching, especially when the similar shapes are not aligned along time.

For matching a sketched query and time series data, both ED and DTW require a sliding window of size equal to the query length to compute the similarities over time. In their survey, Ding et al. [18] conclude that there is no distance measure that is systematically better than DTW, while the relatively simple and straight-forward ED can be computationally competitive with DTW, when the size of the data increases.

2.2 Visual query systems

In visual query systems, visual interface components are used to formulate the user's queries. TimeSearcher [19] was a pioneering information visualization tool using timeboxes to query time series data. The analyst draws a rectangular region to indicate time points of interest on the time axis and the range of interesting values on the value axis. Time series data is then highlighted while passing through the timeboxes. Later, extended versions have been proposed to improve the basic timeboxes by incorporating the variable (fuzziness in the boundaries) [20], angular queries and slopes to search ranges of differentials [21] and supporting more flexibility with options to adjust the query [22]. Overall, timeboxes are powerful value-based widgets and they are used in several visual query systems. Still, it is far from straight-forward to specify a shape-based query with timeboxes, for example, a head-and-shoulders pattern.

The Querylines system [23] realizes a filter-based approach to visual querying. It offers the user the opportunity to specify constraints by using line segments. The analyst can qualify these line segments as hard or soft constraints based on their preference. If the query gets over-constrained, feedback from the system enables the users to refine the query specification.

An alternative technique for constructing visual queries is to first identify common shapes such as a spike, sink, rise, drop, plateau, and valley, and then build queries using these basic shapes as pattern templates [24].

The concept of a sketch-based visual query system was first proposed by Wattenberg [13]. In his approach, the analyst sketches an approximate pattern on the same display where also the data is visualized for searching similar patterns. The similarity to the time series data is calculated as simple ED. The system is straight-forward to use, but the quality of matching relies strongly on details and well-defined time and amplitude ranges of the sketch, which is in general not easy for the user to handle.

To improve the flexibility and tolerance in their sketch-based visual query system, Holz and Feiner [14] provided a relaxed selection technique which allows the user to implicitly indicate a level of similarity that can vary across search patterns during sketching. Specifically, the mouse speed is used to inform the system about the spatial and temporal tolerance of points in the sketched query.

In order to study the human perception of correspondence between sketches and time series patterns, Eichmann and Zgraggen presented a comparison of rankings of computed pattern matches with human-annotated results [25]. They found that human-annotated rankings can differ drastically from algorithmically generated rankings and concluded that the meaning of sketching is too diverse to be captured in one algorithm or metric.

As a multitude of queries can be targeted by the same sketch, Correll and Gleicher [15] investigate the ambiguities of sketch-based query systems in time series data and define a set of "invariants", enabling the user to choose the properties of data to ignore while sketching. In addition, they adapt different matching algorithms to support different invariants. The main drawback of this approach is that it is not easy and straight-forward for the user to think about the invariants while doing data exploration.

Muthumanickam et al. [16] outline important perceptual features for effective shape matching and define a grammar to express time series data approximately by considering the data as a combination of basic elementary shapes positioned across different amplitudes. These basic shapes are represented by using a ratio value and then a symbolic approximation can be achieved by performing binning on ratio values. The major problem of this method is the limited query expressiveness, along with the black-box nature of query execution with each shape often having its own processing or matching steps.

Research on visual perception suggests that we mentally decompose complex shapes into salient parts such as piecewise upward or downward lines, peaks and troughs [26], [27], [28]. Based on this research, Mannino and Abouzied

present Qetch [1], a tool where users freely sketch patterns on a scale-less canvas to query time series data and get rid of specifying query length or amplitude. This method claims its advantage (dealing with the scale-less sketch) over the traditional matching algorithms—ED and DTW. However, in our observation, the query result is very sensitive to the smoothing level of the time series data as the query length is based on the salient parts (constructed by extrema and inflection points) of the data.

2.3 Deep learning for visualization

Traditional machine learning has been used, for example, in automated visualization design [29], [30], [31], [32], [33], [34]. In recent years, deep learning has become popular due to its successful application to a wide range of fields, especially in image processing and natural language processing. In the visualization area, according research focuses on helping with the design, training, diagnosis and refinement of deep learning models [35], [36], [37]. Using deep learning for solving visualization tasks, however, is still rare.

Han et al. [38] presented FlowNet, an approach based on an autoencoder, for improving clustering and the selection of streamlines and stream surfaces. Kim and Gunther [39] extract a robust reference frame based on a convolutional neural network (CNN) that is able to yield a steady reference frame for a given unsteady 2D vector field. Hu et al. [40] introduced VizML that predicts visualization design choices from a large corpus of datasets using neural networks. Data2Vis [41] makes use of recurrent neural networks to generate Vega-lite visualization specifications from JSON-encoded datasets.

Further, we see visualization solutions that leverage deep learning to improve techniques in visual analytics, for example interaction techniques. Fan and Hauser [10] exploited a CNN and modeled sketch-based brushing in scatterplots to predict the selected points. This method achieves state-of-art accuracy while providing a fast interaction. The model is trained on data from different users in a user study, leading to a general model that is thus not optimized to every single user. To address this issue, they presented a personalized CNN-based brushing technique that is able to iteratively refine the brushing model for a single user with additional data that he/she provides while using the brushing technique [11].

More recently, Chen et al. [42] developed a learning-based approach to realize a lasso selection of 3D points by modelling the selection as a latent mapping from viewpoint and lasso to point cloud regions.

3 The Principal Approach

The overall goal of our research was to design a visual query system for time series data with a fast interaction and an accurate query result in order to solve efficiency and accuracy problems of existing solutions. Also, the system was expected to be friendly to the non-expert and easy to use with limited training. Figure 1 shows an illustration of the principal approach. To achieve a swift interaction, we use freehand sketching as the querying input and a similarity function S that is capable of interpreting the relation between the human sketch I and the matching goal in the time series data V as accurately as possible. Further, we aimed at a real-time system, meaning that the computational cost should be minimal, as well.

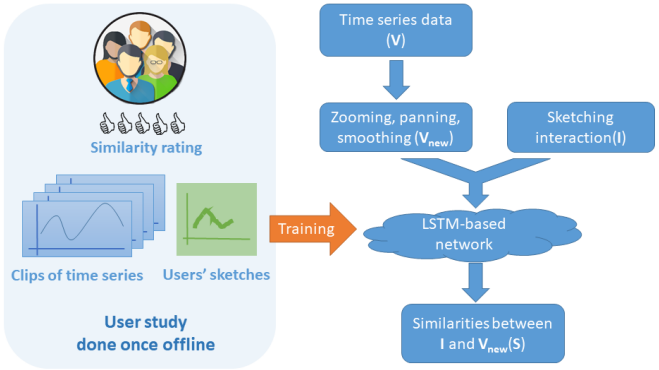

Fig. 1. Illustration of our principal approach: users specify the targeted scale of the time series data by zooming, panning or smoothing, then freely sketch an approximate pattern on the sketching panel. Then a similarity rank between the user sketch and the processed time series data is computed by the proposed parameter sharing LSTM networks. The network is trained only once offline based on user study data.

Based on our understanding that all empirical models have their limitation at estimating the intended meaning of a human sketch, we found it promising to exploit learn the needed similarity measure directly from users. Recurrent neural networks (RNNs) are a straight-forward solution for encoding time series data and to do the matching for two reasons: 1., RNNs have a memory which allows the model to keep information about its past computations. This enables RNNs to have dynamic temporal behavior, which naturally fits to sequential data like time series data. 2., An advanced version of RNNs, LSTM networks (long short-term memory), can be trained to remember the information from a specific length of past times steps. This mechanism can be used to mimic a sliding window while doing the matching computation along the time series data. At the same time, it avoids reading the same data repeatedly, leading to a relatively low computation cost.

To construct the network structure, we used a pair of LSTM networks with shared parameters to encode the sketch and the time series data, respectively. The sharing of the network parameters was beneficial because of the high similarity between the sketch and the time series. The thereby reduced overall number of parameters accelerates the training procedure and helps with preventing overfitting. This design is inspired by the "Siamese" network-based solution for sentence similarity [5]. Detailed description of our network is given in section 4.3.

To train this pair of networks, we collected data from two user studies. For the first user study, we gathered the ground truth about how different users sketch patterns and how they rate similarities based on their visual perception of correspondences between their sketches and several clips of the time series data that we offered. In the second user study, we examined the variation of the user's sketches in order to use this for modeling an extension of the training data for a more stable training.

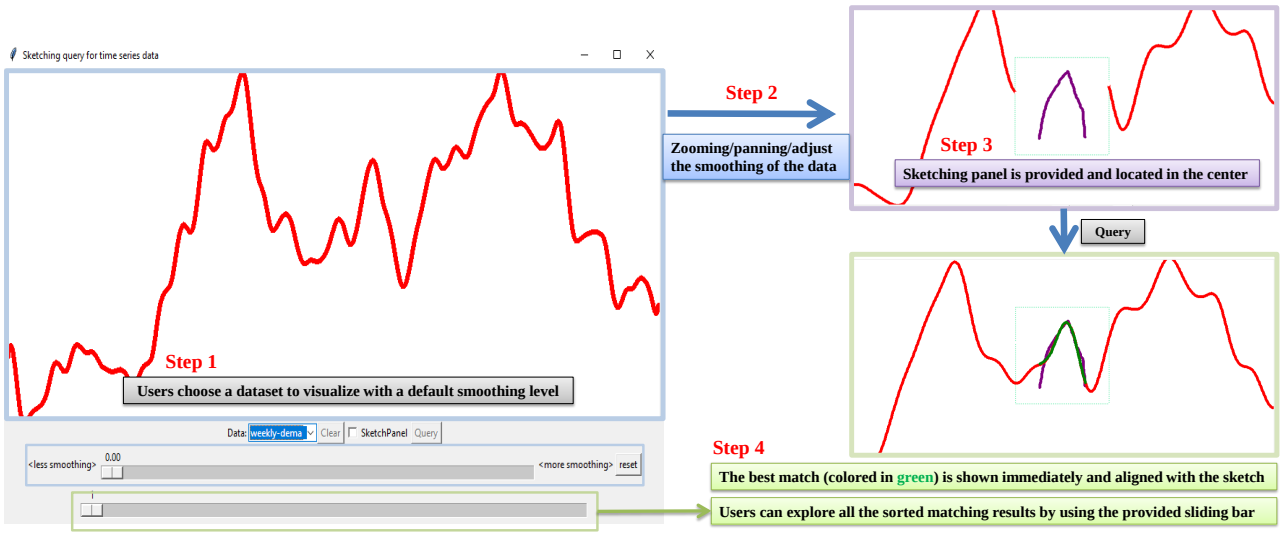

Fig. 2. The interface of our sketching system for time series data. To explore, users choose data first and interact with the data then to specify the scale of interest by zooming and panning. Then the user sketches a pattern of interest. A matching rank is computed and results are explored.

In the following, we introduce the four basic steps of our data exploration workflow (technical details are provided in section 4.1). Figure 2 illustrates the user interface of our proposed sketching system and the four steps to data exploration.

Step 1: Data Preprocessing. For most time series data, some smoothing is necessary to capture the key patterns of interest, leaving out noise and patterns on other scales. In our design, cubic splines are used to smooth the data. Instead of asking the user to specify the smoothing level, we offer a default smoothing level after loading the dataset.

Step 2: Interactive Scaling and Smoothing. Choosing a scale (and a smoothing level) for data exploration is a crucial user-side task – meaningful questions may be asked about time series data at multiple scales, depending on the user task. Instead of iterating through all possible scales and smoothing levels while matching, we allow the user to interact with the data via zooming and panning (and/or adjusting a slider to specify the targeted scale and smoothing level), after initially estimating a proper scale automatically.

Step 3: Sketching. Once the scale and smoothing is determined, a sketch panel is provided for the user to do free-hand sketching. The empty sketch panel is located at the center of the canvas to let the user sketch at the targeted scale, visually referring to the scaled time series data in the background. Sketches are then slightly smoothed in order to remove hand jitter and the query length is determined by the sketch length.

Step 4: Query and explore the matching results. The two parameter-sharing LSTM networks are then executed to obtain an ordered set of similarities between the sketch and subsets of the time series data. The best match is immediately highlighted in green after the computation and the corresponding time series data is shifted by aligning the best matching part with the sketch. Moreover, a slider is provided to explore all the other results from the ranked list of matching results.

4 Technique in Detail

In the following, we first go through the details of scaling and smoothing before we then describe the specification of the used RNN and the design of the proposed network.

4.1 Scaling and smoothing the data

As we mentioned, a default smoothing level (denoted by k_0) is computed for the data after loading, based on the number of salient parts (denoted by N_s). We count the salient parts by segmenting the time series data at extrema and inflection points. To obtain the default smoothing level for each dataset, we adjust the smoothing level until we are satisfied with the number of salient parts that were enough to represent the time series data in advance and this smoothing level is then chosen as the default smoothing setting in the beginning.

To represent the scale of the data, we use z with $z = 1$ in the beginning. We assume that the user wants to see more details when zooming in (and vice versa when zooming out). To automatically adjust the smoothing level during zooming, a linear function is used to adapt the smoothing according to the (logarithm of the) scale: $k(z) = k_0 - a \cdot \ln z$, where a is a coefficient that we obtained via a simple regression procedure. Specifically, we used that the levels of details are related to the number of salient parts. A correctly chosen a should lead to a stable number of salient parts while zooming in or out. To achieve this, we randomly choose 10 points in the time series data for a specific value of a and then compute the number of salient parts with 10 different scales. The results were used to fit a linear function with h as the coefficient ($N_s = h \cdot \ln z + d$). This procedure was repeated several times by trying different values of a. This way, we identified a well-working a by finding h which was closest to 0. In our work (10 different datasets), the value of a varied from 50 to 1371, while k_0 ranged from 9054 to 549923. Doing this offline in advance (finding a and k_0 for each dataset), we can minimize the operations that

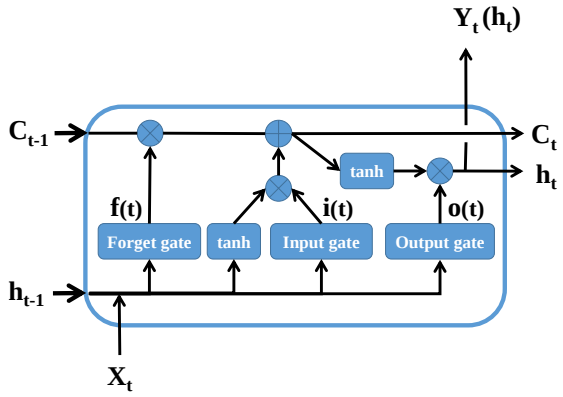

Fig. 3. Complete structure of the LSTM cell, which can process data sequentially and keep its hidden state through time.

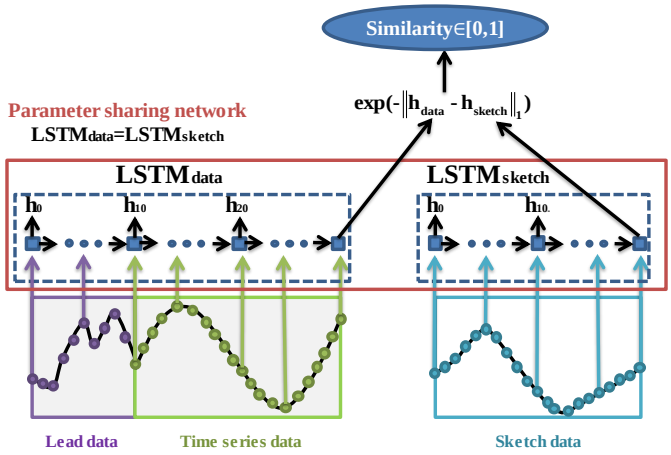

Fig. 4. The structure of our proposed double network: the time series data (green) with lead data (purple) on the left and the sketch (blue) are encoded by the two parameter-sharing LSTM networks, which are trained against a distance metric based on the Manhattan distance.

the users have to do and help them focus on the pattern searching in the time series data. To increase the flexibility, the user can also use a slider to adjust the smoothing level, if they are not satisfied with the suggested smoothing level.

4.2 Recurrent neural network (RNN)

An RNN is an extension of the traditional feed-forward neural network which is able to store relevant parts of the input and use this information to predict future outputs. More formally, at time step t, the memory cell's current hidden state $\mathbf{h}_t$, preserved by the RNN structure, is a function of the input at the current time step ($\mathbf{X}_t$) and the hidden state at the last time step ($\mathbf{h}_{t-1}$). The RNN updates its current state by computing $\mathbf{h}_t = \phi(\mathbf{h}_{t-1}, \mathbf{X}_t)$, where ϕ is a nonlinear function such as the composition of a logistic sigmoid with an affine transformation. Optionally, the output at time step t, denoted by $\mathbf{Y}_t$ is a function of the previous state and the current input, and it is the same as the hidden state $\mathbf{h}_t$ for basic cells.

Although a basic RNN performs well in capturing nonlinearity in time series data, it was observed that back-propagation dynamics caused the gradients in an RNN to either vanish or explode while training to capture the long-term dependencies [6].

To overcome this disadvantage, the LSTM (long short-term memory) architecture [43] was proposed by Hochreiter and Schmidhuber. As shown in Figure 3, in addition to the hidden state vector $\mathbf{h}_t$, LSTMs also maintain a memory cell $\mathbf{c}_t$ at time t. At each time step, the LSTM can choose to read from, write to, or reset the cell using explicit gating mechanisms. The memory cell $\mathbf{c}_t$ is updated by partially forgetting the existing memory and adding new memory content:

$$\mathbf{c}_t = f(t) \otimes \mathbf{c}_{t-1} + i(t) \otimes \tanh(\mathbf{W}_c \mathbf{X}_t + \mathbf{U}_c \mathbf{h}_{t-1} + \mathbf{b}_c) \quad (1)$$

where the forget gate $f(t)$ controls the extent to which the existing memory should be erased while the input gate $i(t)$ is used to decide the degree to which the new memory content is added. The two gates are computed respectively by

$$f(t) = \sigma(\mathbf{W}_f \mathbf{X}_t + \mathbf{U}_f \mathbf{h}_{t-1} + \mathbf{b}_f) \quad (2)$$

$$i(t) = \sigma(\mathbf{W}_i \mathbf{X}_t + \mathbf{U}_i \mathbf{h}_{t-1} + \mathbf{b}_i) \quad (3)$$

Moreover, the output gate $o(t)$ controls the exposure of the memory content and it is computed by

$$o(t) = \sigma(\mathbf{W}_o \mathbf{X}_t + \mathbf{U}_o \mathbf{h}_{t-1} + \mathbf{b}_o) \quad (4)$$

As a last step, the output of the LSTM unit ($\mathbf{Y}_t (= \mathbf{h}_t)$) at time step t is be obtained by

$$\mathbf{h}_t = o(t) \tanh(\mathbf{c}_t) \quad (5)$$

In all of the above, operator $\otimes$ is the Hadamard product (entry-wise product). $\mathbf{X}_t \in R^d, f(t), i(t), o(t), \mathbf{h}_t, \mathbf{c}_t \in R^h$. The weight matrices $\mathbf{W} \in R^{h \times d}$ and $\mathbf{U} \in R^{h \times h}$ and the bias vector $\mathbf{b} \in R^h$ are learned during training. The dimensions d and h correspond to the number of input features and the number of hidden units, respectively.

Instead of overwriting its content at each time step, an LSTM unit is able to decide whether to store or retrieve the existing memory via the introduced gates. The activations of these gates are based on the sigmoid function and hence range smoothly from 0 to 1 (not at the least to keep the model differentiable). Intuitively, if the LSTM unit detects an important feature from an input sequence at an early stage, it carries this information (the existence of the feature) over multiple steps, capturing potential long-distance dependencies.

4.3 Network design

Figure 4 provides an overview of our proposed network structure for estimating the similarity between the user's sketch and the time series data, composed of two LSTM networks: $LSTM_{\text{data}}$ and $LSTM_{\text{sketch}}$, sharing their parameters.

In each time step, the hidden state (h) has to be carried as an input to the next time step. For the similarity computation, we only consider the final representation of both the time series data and the sketch, encoded as h_{data} and h_{sketch}, respectively. To compute the similarity between these two vectors, we use a metric based on the Manhattan distance,

which can be defined as $\exp(-\| h_{\text{data}} - h_{\text{sketch}} \|_1) \in [0, 1]$. The reason to choose an L1 norm for the similarity computation is that an L2 norm can lead to undesirable plateaus in the overall objective function due to vanishing gradients of the Euclidean distance [44]. During the training, the network learns how the predicted similarity between h_{data} and h_{sketch} deviates from the user-annotated ground truth.

In order to compute the similarities along the time series data, we mimic a sliding window by making use of the special feature of the LSTM network that in each time step it can choose to forget a part of the information extracted from the previous time steps. In the training, which we explain in more detail further below, the network is trained to force the output of each time step to represent the information only for a specific number (L) of previous time steps. Figure 4 shows a typical example, where the sketch data is sampled as 21 "blue" points, determining the size of the sliding window ($L = 21$). Therefore, after a proper training, h_{data} only contains the information of the 21 "green" points and the influence of the previous 10 "purple" points are forgotten. As the output of each time step can be trained to contain the information of a certain previous time steps, our method only needs to iterate the data once and then interpret the output of each time step for the matching computation, which is much more efficient than the traditional sliding window, which needs to access a data point several times while moving.

4.4 Training the network

We define the training data as $([L_i, R_i] \in T_{in}, y_i \in T_{out})_{i=1}^{N}$ as pairs of input and expected output (N is the number of training samples). L_i contains the time series data and its synthesized left lead data while R_i is the corresponding user sketch. The reference output y_i is the human annotated similarity between L_i and R_i, which the model is trained against. We optimize the parameters of the network based on the training data using the mean-squared error as a loss function.

During training, no hyper-parameter explicitly "tells" the network to learn the information from a certain length of previous time steps in each time step. The mechanism for the LSTM to forget earlier data is when the network finds that the information carried by certain previous time steps is important, while the information before that is not. To teach the network to "understand" this, we add some synthesized data of half the length of the query to the left of the time series data (for example the purple points in Figure 4). The synthesized data is chosen from other parts of the same data, where the clip of the time series data is extracted from and then smoothly connected to the left of the time series data. For one pair of time series data and a sketch, we add 10 different synthesized lead data to the left of the time series data, which has been tested to make sure that the network can recognize the real data and force itself to forget the influence of the synthesized data. In this way, the network is trained to only remember a specific length of earlier data in each time step. Based on this, we can compute the similarity rank along the time series data while reading the time series only once.

High accuracy cannot be achieved without providing enough training data. It is labor-intensive and time-consuming to invite a large number of users to provide a large amount of user data. Instead, we follow a common strategy and synthesize additional training data from the already acquired training set by modeling the natural variation of user sketches. In the following, we describe how we synthesize the left lead data of the time series data and the variation of user sketches in detail.

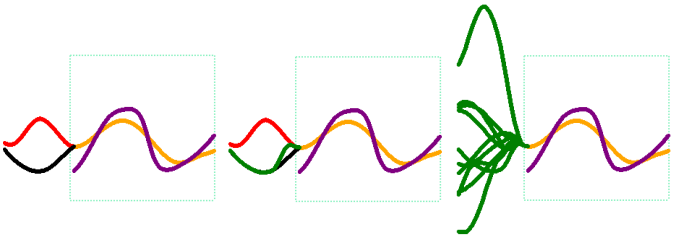

Fig. 5. Lead data synthesis. Left: time series data (orange) with actual lead data (red), and the user's sketch (purple). New lead data (black) is randomly chosen. Middle: the new lead data is smoothly attached (green) to the time series data (orange) by interpolation. Right: ten instances of synthesized lead data.

4.5 Training data augmentation

In addition to the training data that we acquired by a user study, we employ two strategies to augment the training data: First, we synthesize lead data ahead of the actual time series training data in order to teach the network the actual query length. Second, we generate variants of the sketch, based on a second user study that informed us about the natural variation of the user's sketching interaction.

4.5.1 Lead data synthesis

To preserve the character of the time series data, we initialize the newly synthesized lead data with randomly chosen snippets from the original time series data connected them to the left of the time series training data. Figure 5 shows the whole procedure of lead data synthesis: The user's sketch is shown in purple and the corresponding time series data clip in orange with its actual lead data in red (denoted as $r(t)$). New lead data (black, denoted as $b(t)$) is chosen randomly from the time series data, forming the basis of the newly synthesized lead data.

To start, we randomly choose another part of the original time series data (with the targeted length). We do so to maintain the overall character of the time series data when synthesizing new lead data. Obviously, this usually leads to a non-smooth connection with the actual time series data (illustrated on the left in Figure 5).

To achieve a natural, smooth concatenation, we designed a simple, fifth-order polynomial weighting function $w(t)$ to merge $r(t)$ and $b(t)$ by convex combination. We set up $w(t)$ to fulfill six constraints: $w(0) = 0$, $w(1) = 1$, $w'(0) = w'(1) = 0$, and $w''(0) = w''(1) = 0$, leading to

$$w(t) = 6\,t^5 - 15\,t^4 + 10\,t^3\,. \tag{6}$$

Given that we wish to adapt the last $n+1$ values of the randomly chosen new lead data $b(t)$ (of length m) to smoothly connect to the following time series data, we compute

$$b_{\text{new}}[i] = (1 - w(t))\,b[i] + w(t)\,r[i] \tag{7}$$

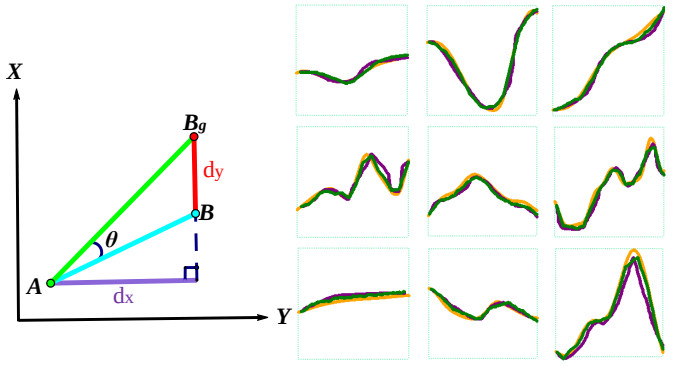

Fig. 6. Sketch synthesis. Left: illustration of the modeling base between two consecutive points A and B in the sketch and the corresponding goal point B_g. Right: 9 modeled interactions (colored in green) according to the specific querying target (orange curve) and the original user sketch (purple curve).

for $t = \frac{i-(m-n)}{n}$ and $i \in [m-n, m]$, keeping $b_{\text{new}}[i] = b[i]$ for $i < m-n$. Adapted $b_{\text{new}}[i]$ then smoothly connects to the following time series data in $i = m$.

In our experiment, the randomly chosen part is sampled into $m = 20$ points and the last $n = 8$ points are merged with the real lead data. In the middle of Figure 5, r is shown in red, b in black, and b_{new} in green. On the right of the Figure 5, 10 different pieces of lead data are generated for one pair of sketch and time series data, all smoothly connected to the time series data (orange). This approach leads to a good variation of synthesized lead data, mimicking plausible cases for all time series data that we worked with. Synthesizing ten instances of substantially varying lead data allows the LSTM to learn that only the actual time series data (orange) is to be taken into account when matching with the sketch.

4.5.2 Sketch synthesis

Clearly, there is a certain amount of natural variation in the users' sketching interaction (even if intended, they would not repeat the same sketch twice – at least not exactly). In order to achieve a stable training result and to reduce overfitting as much as possible, we synthesize additional sketches based on the natural variation of human sketches. To collect information about natural sketch variation, we organized a user study in which we asked the users to repeat the same sketch several times for a specific matching goal. The details of this user study are presented in section 5.2.

Figure 6 is an illustration of how we consider the variation of sketches based on the user study, also showing nine sample synthesis results according to the fitted model. As the user sketch is recorded as discrete points, the variation is modeled point-wise and consists of two parts that are meaningfully modeled separately: the horizontal displacement d_x and the vertical displacement d_y. A and B are two consecutive points of the user sketch, while B_g is the corresponding goal point which is located in the time series data (the point which the user aimed for). The distance d_y is the vertical point-wise displacement between the user sketch and the goal. θ is the angle between AB_g and AB.

By examining the user study data, we found that θ is strongly correlated with d_y, meaning that larger angles lead also to larger vertical displacements. Based on this observation, we use a cubic polynomial model $m_1(\theta)$ to fit the relation between θ and d_y. As m_1 represents the central tendency, we can compute the absolute difference between $m_1(\theta)$ and d_y to model deviation information. Also here, we use a cubic polynomial to fit the relation between this difference and θ as $m_2(\theta)$. Eventually, we can sample d_y from a normal distribution $N(m_1(\theta), m_2(\theta))$ with $m_1(\theta)$ as the mean and $m_2(\theta)$ as the standard deviation. Additionally, we compute the horizontal difference d_x between two consecutive points for distribution fitting from the user study data. We found that the average of d_x is around 1.5 pixels, which is too small and detailed to observe any variation. Thus, we consider a larger interval by taking every 10 points into account instead of every point, which we think is more reasonable for distribution fitting. Based on the statistical data we gathered, we used the statistical tool EasyFit [45] to analyze which distribution fits our variation data best. As a result, the variance of d_x (denoted as $var(d_x)$) follows a logarithmic distribution $f(x) = \frac{-\alpha^x}{x \ln(1-\alpha)}$ with $\alpha = 0.8525$.

For synthesizing a new sketch, given a user sketch I and the querying target G, we compute a new user interaction I' based on random samples from the fitted PDFs. We start from the first point in I, denoted by I_1. We then have $I'_1 = I_1$ and θ_1 as the angle between I_1I_2 and $I_1I'_2$. Based on this, we can synthesize the next point I'_2 with $I'_{2x} = I'_{1x} + d_{x_1}$ and for I'_{2y} we have $I'_{2y} = G_{2y} - d_{y_1}$ when $G_{2y} > I_{2y}$ while $I'_{2y} = G_{2y} + d_{y_1}$ when $G_{2y} < I_{2y}$. For sampling, d_{x_i} is sampled from the logarithmic distribution $f(x)$ and d_{y_i} is sampled from the normal distribution $N((m_1(\theta_1), m_2(\theta_1)))$ respectively. All subsequent points of the new sketch I' are estimated in the same way.

On the right of Figure 6, nine user interactions (purple) from the second user study are shown that we used for studying the variation of sketches and the correspondingly modelled synthetic variations are shown in green, confirming that our model is able to generate meaningful and realistically looking sketching variations.

4.6 Training details

The original training data collected from the first user study consists of 2200 pairs of sketches and time series data with similarity ratings by the users. We extended this data by synthesizing ten pieces of lead data for each time series data and adding four modeled sketches for each actually recorded sketch, resulting in 110 000 pairs of $([L_i, R_i], y_i)$ as the training data. As the queries were supposed to match independently of their horizontal and vertical position, we preprocessed the input data $([L_i, R_i])$ by removing the x-coordinate and replacing the y-coordinate by its first derivative.

For our research, we implemented the network and executed the training in Keras with Tensorflow as the backend, providing powerful GPU acceleration and convenient coding flexibility. For training and testing, we used a PC with an Intel Xeon E5-1650 CPU and an NVIDIA GeForce GTX 1080 GPU. There are two main hyper-parameters of the network to be set: time step (i.e., sliding window length) and cell size (i.e., output dimension). As the sketching panel in our experiment is 200×200 pixels, we experimented with

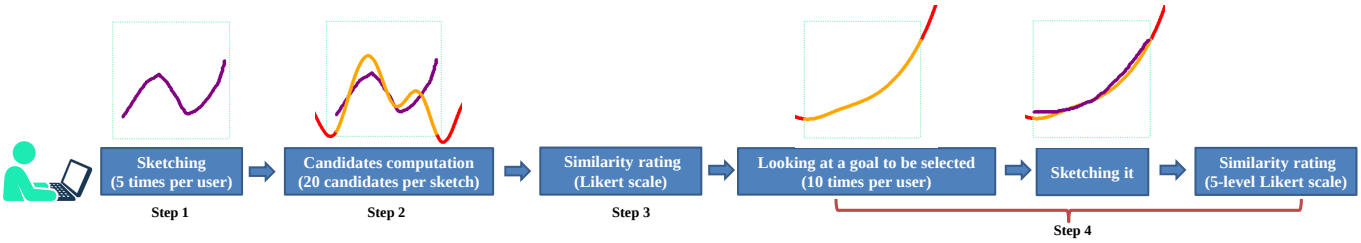

Fig. 7. Pipeline of the first user study. After sketching a pattern, we offer 20 candidates (computed by ED) per sketch and ask the user to rate similarity between them. Then, in order to enforce some high similarity data, we offer 10 target time series clips and ask the user to do the sketching aiming for them. In total, we can get 110 pairs of user sketches and time series data per user with corresponding similarity ratings.

different values of the sampling frequency and found that 40 (sampling interval up to 5 pixels) is sufficient to represent the details of a sketch. Therefore, we set the time steps of the $LSTM_{\text{data}}$ to 60 and the one of $LSTM_{\text{sketch}}$ to 40, which means that the length of the sliding window that we mimic is 40 (L=40). The hidden feature of the LSTM cell was set to be of size 20. Besides, a dropout function with a drop rate of 0.2 (common value) was used to limit overfitting. We optimized the parameters (1760 in total) of the network based on the training data using the mean-squared error as a loss function. The Adam optimizer was used during the optimization and the learning rate was set to 10^{-3}. In total, we ran 2000 epochs for the training with a full batch.

5 User Studies

Since the user plays a key role in visual query systems and to meet the user's expectation as close as possible regarding the matching, we conducted a user study to investigate how users sketch patterns and how they rate the similarities between their sketches and corresponding time series data. We then used this information to train our double network. Further, we also conducted a follow-up user study to explore how the user uses our sketching query in practice and analyzed the natural variation of their sketching interactions in order to prepare for the synthesis of additional data for a more stable training. Eventually, to evaluate our model, we also organized a third user study, in which we asked the users to rate the results from our model in comparison to the Qetch algorithm as well as to the DTW algorithm. Below, we provide more details about the three user studies.

5.1 User study for base training data

For the first user study, 10 different time series data with length ranges from 91 to 4774 were prepared as a basis. All of these data were carefully chosen in advance from datasets in finance, industry, medicine, and the labor market, with a healthy spread of characteristics (covering important and common patterns: head-and-shoulder, sharp rises/dips, upward or downward slopes, peaks and troughs, etc.). For each dataset, 5 new variants were produced based on 5 different scales and these 50 new time series data were used for the user study. The procedure of the first user study consists of four steps, illustrated in Figure 7:

1. First, an empty canvas (900×400 pixels) was shown to the user and at the same time the user was asked to think about a pattern he/she wanted to look up from an instance of time series data. Then the user used the mouse to sketch this pattern on the sketch panel (200×200 pixels, located in the middle of the canvas). The reason for choosing a mouse as the input device was that we wanted as general as possible results and pen-based input is not generally available to many users.

2. Based on the user's sketch, we computed similarities by using a simple ED matching rule and a sliding window with step length of 20 pixels, over all the 50 variants of the prepared time series data, yielding a ranked list of matches. We then chose 20 candidates evenly from the matching results based on their similarity rank. The 20 candidates were distributed evenly around 0%, 25%, 50%, 75%, and 100%, where 0% means the most dissimilar and 100% amounting to the best match. The purpose of this procedure was to achieve training data within a healthy range of similarities so that the network could learn the similarity function properly.

3. In the next step, the user was asked to rate the similarities between their sketch and the 20 candidates we offered, one by one. The similarity rating was acquired on the Likert scale [46], which is commonly used to collect respondents' attitudes and opinions. We asked the users to choose their rating from five options: no match ($s = 0$), bad match ($s = 0.25$), half good / half bad match ($s = 0.5$), good match ($s = 0.75$), and excellent match ($s = 1$). All users were clearly informed about the correspondence between the similarities and the five rating options in the tutorial section of the user study. For the whole study, each user had to do five sketches and for each sketch he/she rated the similarity against 20 candidates based on their visual perception. In total, 100 pairs of time series data and sketch with rated similarities were recorded per user.

4. In the second step, we used simple ED to do an initial selection of candidates for rating, aiming at training data with balanced similarities. The actual situation, however, turned out to be so that only for very few candidates our users were satisfied enough to rate them as good match or even as excellent match. Therefore, in the fourth stage of this study, instead of asking the users to sketch a pattern in their mind, we showed them 10 additional candidates in sequence, asking them to sketch while aiming at the shown curve. Then, we asked them to rate how satisfied they were with their own drawing, again using the Likert scale. This way, we got some additional pairs of data with highly rated similarity, making sure that our training data covers a health range in terms of similarity.

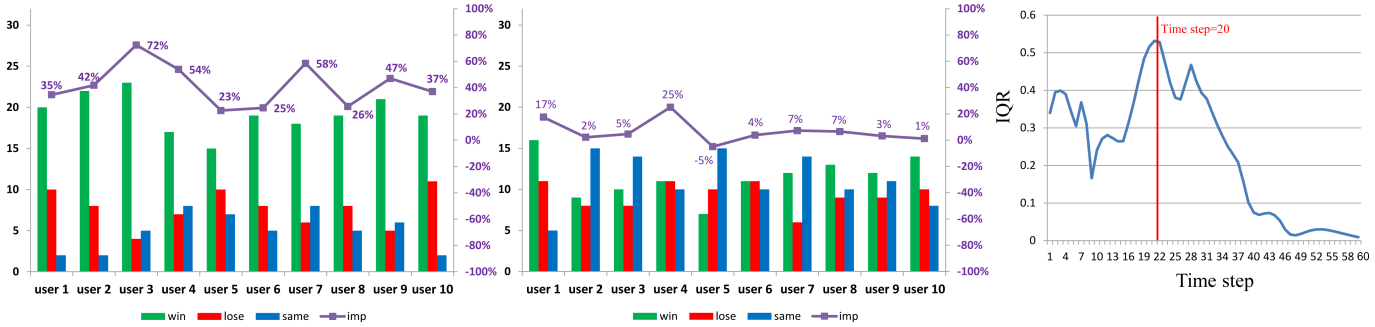

Fig. 8. Left: comparison between our method and the Qetch algorithm. Middle: comparison between our method and DTW. Green bars show how often our technique was preferred (red: how often Qetch/DTW was preferred; blue: tied) and the purple line indicates per user the improvement as achieved by our method (positive: average improvement due to our method). Right: IQR values showing output variation between different time series from the 1st time step to the 60th, indicating the change of variation over time.

In the first user study, 20 users were invited to participate, all students or employees from the University of Bergen, and in total 2200 pairs of data with perceived similarities were collected for training (the details of the collected pairs are shown in the supplementary material). Before the user study, every user was given a training session to get familiar with the interface and the mouse operations for sketching. In addition, we showed them 10 typical patterns in time series data such as rounded-bottom, head-and-shoulders, sharp rise and down, falling peaks, and so on. This procedure is to help those users, who were unfamiliar with time series data, to do the sketching meaningfully. During the training session, we answered any questions they had about the tool until they were ready for the study.

5.2 Studying the variation of sketches

Naturally, for the same matching goal, the sketch done by the user will be a bit different every time. In order to understand the natural variation of the user's sketches, when having the same matching goal and sketch operation in mind, and to model this variation for synthesizing additional data for the training, we did this follow-up user study based on the first user study. In the second study, 10 individuals, all students or employees from the University of Bergen, participated. We chose 100 representative clips of the time series data from the first user study. These 100 clips have an even distribution in terms of shapes and frequency, which forms a healthy base to investigate the variation of human sketches. For each user, 10 different time series clips were displayed as the matching goal for sketching. The second user study then consisted of two parts:

In the first part, the users were asked to look at a clip of the time series data that we provided. Then, the users were required to draw a sketch with the goal to match the shown clip. The users were asked to repeat this interaction 12 times in each case. The traces of all sketches were recorded during the study. Altogether, 1200 sketches were collected and used for modeling the variation among sketches.

5.3 Evaluation user study

In order to evaluate our new model, learned by the double network, we conducted a third user study, asking users to tell whether the computed matching results were considered good, or not. As a baseline for comparison, we chose two state-of-the-art techniques, i.e., DTW and the recently published Qetch algorithm [1]. The well-established DTW metric was chosen as one of the best options for distance measures in time series data [18], while the authors of Qetch claimed its strength over DTW for freehand sketch and matching for high-level task (in terms of time spent). In this user study, the users were asked to rate the matching results computed by Qetch, DTW, and our new method, leading to a quantitative, comparative evaluation reflecting the user's perspective.

For this evaluation study, 10 users were invited. For each user, 8 new time series data (with lengths ranging from 40 to 1440, none of them used for training before) were provided in sequence to test the generality of the proposed model. The procedure of this user study consisted of two steps:

1. To start, a dataset with a useful default smoothing level (see above) was presented to the user. For each dataset, the user could interact with the interface and specify the targeted scale of the visualization by zooming with the mouse. In addition, users could also adjust the smoothing level by using a slider.

2. Then, the user was asked to sketch a pattern he/she wanted to look up from the time series data on the sketching panel. Based on the user' sketch, we computed the three best matching results from our new model, the Qetch algorithm, and using DTW, respectively. Then, we showed these three results (in random order and without telling which is which) to the user and asked them to rate the similarity between their sketch and these three results separately. As Qetch and DTW are highly competitive matching algorithms, capturing the difference between good and very good results requires more detail, so we offered a once refined nine-points range (also from 0 to 1, $s = 0, 0.125, 0.25, 0.375...1.0$) for rating instead of the courser five-points Likert scale, delivering the required details for a proper comparison.

During this study, we recorded the similarity rating from each user for a subsequent quantitative analysis (see below). For each data, the user sketched four times and did four sets (our method, Qetch, and DTW) rating. Accordingly, we collected altogether 320 sets of rating results from 10 users. In addition, the time cost of computation were recorded for comparing efficiency, also. Before the user study started,

every users was offered a training session to get familiar with the interface and the interaction of the user study.

6 Evaluation results

For evaluating our matching model, we did a quantitative comparison with Qetch and DTW – shown on the left and in the middle of Figure 8. The results (32 pairs from each user in the third study) are shown as bars (how often one technique was preferred) and as a line graph (average improvement). Green bars show (per user) the number of times that our method was rated with a higher similarity than Qetch (left) or DTW (middle), while red bars represents the contrary (blue bars a tied rating). Further, we also compute the average improvement (denoted as imp) of our method as compared with Qetch/DTW in terms of the similarity value: $imp = \frac{s_{\text{our}} - s_{\text{qetch}}}{s_{\text{qetch}}}$ or $\frac{s_{\text{our}} - s_{\text{dtw}}}{s_{\text{dtw}}}$ (s_{our}, s_{qetch}, and s_{dtw} denote the average similarity rating of our method, the Qetch algorithm, and DTW, respectively) and show it (also per user) as line graph in purple.

By looking at the bar graph in the left of Figure 8, we clearly see that all users preferred our matching over the Qetch algorithm and that our method was preferred about 2.5 so often as the other way around (193:77). The line graph shows that all average improvements are positive, providing a clear evidence that our method is closer to the user's perception in terms of similarity. More specifically, the average improvement is ≈42% as $\bar{s}_{\text{our}}$=0.64 and $\bar{s}_{\text{qetch}} = 0.45$, where $\bar{s}_{\text{our}}$ and $\bar{s}_{\text{qetch}}$ are the average similarity values of our method as compared to the Qetch algorithm. Based on this evaluation, we are confident to conclude that in general our method performs significantly better than the Qetch algorithm, when the matching results are directly judged by the users.

By looking at the chart in the middle of Figure 8, 7 out of 10 users preferred our technique over DTW according to the bar graph, while the average improvements are positive (in our favor) for 9 out of 10 users, when looking at the line graph. The overall ratio of users preferring our method over DTM is around 1.2 (115:93) and the average improvement for each user is ≈7% with $\bar{s}_{\text{dtw}} = 0.6$ ($\bar{s}_{\text{dtw}}$ is the average similarity of DTW rated by all the users). Overall, and even though the improvement numbers are clearly smaller than in the comparison with Qetch, this suggests that our method is slightly better than DTW in terms of accuracy (at least not worse) when evaluated directly from the user's perception. The details of all user ratings are in the supplementary material.

Furthermore, and since a swift user–computer dialogue in visual query systems is highly dependent on the efficiency of the involved interactions, we also compared the computation cost of the three methods. According to the user study, the average computation costs of our method, Qetch, and DTW, are 33ms, 132ms and 3.6s, respectively. Based on this data, we see clearly that our method is the most efficient one due to its linear complexity, while DTW is the slowest and unable to achieve a real-time interaction.

Besides the quantitative evaluation in terms of accuracy and efficiency, we also examined the output of each step of the LSTM network to check whether the network learns meaningful information. The reason for doing this was that the network was implicitly taught to only remember the information of a specific length of previous time steps (we set this to 40 in our experiment) with a goal to mimic a sliding window for matching. To investigate whether the network has successfully achieved this important feature, we collect the output (namely the hidden state $\mathbf{h}$) of each time step of several time series data with different left lead data and compute the output variations over the time steps to analyze whether the network is able to discard long term dependencies. The details are illustrated below.

In our training, the time series data clips were sampled at 40 points with synthesized lead data "on the left" at 20 points. For looking into the information as learned by the network, we did an experiment that generated 20 time series snippets with length 60 (denoted as G_i, $i \in \{1, \ldots, 20\}$), where the trailing 40 entries for each time series were the same, but the first 20 varied according to our synthesis procedure. As a reference, we have another time series (denoted as ref) that has the same last 40 entries but with a real lead data that is different from all synthesized lead data G_i. We then iterated the time series and the reference time series over the 60 time steps by using the already trained model and obtained a set of outputs with format 20×1×60×20 and 1×60×20. We then computed the cosine similarity between all outputs of G_i and ref per time step, leading to a set of 20 cosine similarities. We compute the inter-quartile range (IQR), i.e., the difference between the 75%- and the 25%-percentile, as a robust indicator of the variation over the time steps.

After iterating over all the time steps from 1 to 60, we have 60 IQR values which represent the output variations over the 60 time steps and this information is shown in the right of Figure 8. As we mentioned, from the 1st time step to 20th time step, the network output corresponds to lead data – since all of the lead data was randomly synthesized the variation during this period is fluctuating at a relatively high level. After time step 20, however, we see a decline of the IQR values, correlated with the networking reading actual time series data. Towards the 60th time step, the IQR values approach 0, which meets our expectation that the output at the last time step almost only represents the information of the previous 40 time steps. In summary, this statistics gives us a strong indication that the LSTM network has been successfully trained to understand that only the information from a certain length of the previous time steps should be taken into account in each time step.

7 Limitations and future work

Although our new model demonstrated its accuracy and efficiency over two state-of-the-art methods, there are still three limitations: 1. It cannot be excluded that scenarios exist that are not covered by the model due to limited training data. This is in general a known problem of deep learning approaches – high prediction accuracy requires tremendous amount of training data which is often not easy to obtain. 2. Deep learning models lack sufficient interpretability due to their black-box nature, especially when compared with the carefully crafted empirical models. 3. Using a fixed query size is not as flexible as the Qetch approach, which matches based on the salient parts in the time series data.

In the future, we see several opportunities to further extend our work, including:

- Further improving the synthesis of additional training data as this plays a crucial role for the learning process. One possible way may be to use a generative adversarial network (GAN) – similar to other successful applications in image generation and synthesis. We hypothesize that a GAN could synthesize more realistic variations of the user's sketch.
- The design of a matching algorithm, which is tailored for a particular user, using an appropriate method to learn this user's particular sketching behavior over time, would be interesting, as well.

8 Conclusion

In this paper, we have demonstrated how deep learning can be used to further improve a visual query system for exploring time series data in visual analytics. By learning the relation between the time series clip to be selected and a free-hand user sketch, we achieve a solution, which is both fast and accurate. To the best of our knowledge, this is the first study to report the successful application of a matching model, realized by a pair of parameter-sharing LSTM networks, to improve an important human interaction scenario in visual analytics. We demonstrate, quantitatively, and in comparison with the recently published Qetch algorithm as well as the classical distance measure DTW, that our LSTM-based solution leads to an improvement in terms of the overall similarity (≈42% to Qetch and ≈7% to DTW), rated by users in a user study, while enabling a fast interaction (≈4 times faster than Qetch and ≈100 times faster than DTW). The code of our prototype is available via github.com/reddestrabbit/LSTM-based-visual-query-system.git.

Acknowledgments

We thank all participants of the three user studies for taking their valuable time to support this research. We also thank the reviewers of an earlier version of this paper for their valuable input. Parts of this work were done in the context of CEDAS, UiB's Center for Data Science. VRVis is funded by BMK, BMDW, Styria, SFG and Vienna Business Agency in the scope of COMET - Competence Centers for Excellent Technologies (854174) which is managed by FFG.

References

[1] M. Mannino and A. Abouzied, "Expressive time series querying with hand-drawn scale-free sketches," in *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. ACM, 2018, p. 388.

[2] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, 2015, pp. 91–99.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[4] Y. Shen, X. He, J. Gao, L. Deng, and G. Mesnil, "Learning semantic representations using convolutional neural networks for web search," in *Proceedings of the 23rd International Conference on World Wide Web*. ACM, 2014, pp. 373–374.

[5] J. Mueller and A. Thyagarajan, "Siamese recurrent architectures for learning sentence similarity," in *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[6] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994.

[7] H. Sakoe and S. Chiba, "Dynamic programming algorithm optimization for spoken word recognition," *IEEE transactions on acoustics, speech, and signal processing*, vol. 26, no. 1, pp. 43–49, 1978.

[8] M. Novotnỳ and H. Hauser, "Similarity brushing for exploring multidimensional relations," in *Journal of WSCG*, vol. 14. Václav Skala-UNION Agency, 2006, pp. 105–112.

[9] C. Fan and H. Hauser, "User-study based optimization of fast and accurate mahalanobis brushing in scatterplots," in *Proceedings of the conference on Vision, Modeling and Visualization*. Eurographics Association, 2017, pp. 77–84.

[10] ——, "Fast and accurate cnn-based brushing in scatterplots," in *Computer Graphics Forum*, vol. 37, no. 3. Wiley Online Library, 2018, pp. 111–120.

[11] ——, "Personalized sketch-based brushing in scatterplots," *IEEE computer graphics and applications*, vol. 39, no. 4, pp. 28–39, 2019.

[12] ——, "On kde-based brushing in scatterplots and how it compares to cnn-based brushing," in *Machine Learning Methods in Visualisation for Big Data*. The Eurographics Association, 2019.

[13] M. Wattenberg, "Sketching a graph to query a time-series database," in *CHI'01 Extended Abstracts on Human factors in Computing Systems*. ACM, 2001, pp. 381–382.

[14] C. Holz and S. Feiner, "Relaxed selection techniques for querying time-series graphs," in *Proceedings of the 22nd annual ACM symposium on User interface software and technology*. ACM, 2009, pp. 213–222.

[15] M. Correll and M. Gleicher, "The semantics of sketch: Flexibility in visual query systems for time series data," in *2016 IEEE Conference on Visual Analytics Science and Technology (VAST)*. IEEE, 2016, pp. 131–140.

[16] P. K. Muthumanickam, K. Vrotsou, M. Cooper, and J. Johansson, "Shape grammar extraction for efficient query-by-sketch pattern matching in long time series," in *2016 IEEE Conference on Visual Analytics Science and Technology (VAST)*. IEEE, 2016, pp. 121–130.

[17] D. Q. Goldin and P. C. Kanellakis, "On similarity queries for time-series data: constraint specification and implementation," in *International Conference on Principles and Practice of Constraint Programming*. Springer, 1995, pp. 137–153.

[18] H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, and E. Keogh, "Querying and mining of time series data: experimental comparison of representations and distance measures," *Proceedings of the VLDB Endowment*, vol. 1, no. 2, pp. 1542–1552, 2008.

[19] H. Hochheiser and B. Shneiderman, "Visual queries for finding patterns in time series data," *University of Maryland, Computer Science Dept. Tech Report, CS-TR*, vol. 4365, 2002.

[20] E. Keogh, H. Hochheiser, and B. Shneiderman, "An augmented visual query mechanism for finding patterns in time series data," in *International Conference on Flexible Query Answering Systems*. Springer, 2002, pp. 240–250.

[21] H. Hochheiser and B. Shneiderman, "Dynamic query tools for time series data sets: timebox widgets for interactive exploration," *Information Visualization*, vol. 3, no. 1, pp. 1–18, 2004.

[22] P. Buono, A. Aris, C. Plaisant, A. Khella, and B. Shneiderman, "Interactive pattern search in time series," in *Visualization and Data Analysis 2005*, vol. 5669. International Society for Optics and Photonics, 2005, pp. 175–187.

[23] K. Ryall, N. Lesh, T. Lanning, D. Leigh, H. Miyashita, and S. Makino, "Querylines: approximate query for visual browsing," in *CHI'05 Extended Abstracts on Human Factors in Computing Systems*. ACM, 2005, pp. 1765–1768.

[24] M. Gregory and B. Shneiderman, "Shape identification in temporal data sets," in *Expanding the Frontiers of Visual Analytics and Visualization*. Springer, 2012, pp. 305–321.

[25] P. Eichmann and E. Zgraggen, "Evaluating subjective accuracy in time series pattern-matching using human-annotated rankings," in *Proceedings of the 20th International Conference on Intelligent User Interfaces*. ACM, 2015, pp. 28–37.

[26] D. D. Hoffman and M. Singh, "Salience of visual parts," *Cognition*, vol. 63, no. 1, pp. 29–78, 1997.

[27] E. J. Keogh and P. Smyth, "A probabilistic approach to fast pattern matching in time series databases." in *Kdd*, vol. 1997, 1997, pp. 24–30.

[28] N. Kong and M. Agrawala, "Perceptual interpretation of ink annotations on line charts," in *Proceedings of the 22nd annual ACM symposium on User interface software and technology*. ACM, 2009, pp. 233–236.
[29] Y. Luo, X. Qin, N. Tang, G. Li, and X. Wang, "Deepeye: Creating good data visualizations by keyword search," in *Proceedings of the 2018 International Conference on Management of Data*, 2018, pp. 1733–1736.
[30] B. Saket, D. Moritz, H. Lin, V. Dibia, C. Demiralp, and J. Heer, "Beyond heuristics: Learning visualization design," *arXiv preprint arXiv:1807.06641*, 2018.
[31] B. Saket, A. Endert, and Ç. Demiralp, "Task-based effectiveness of basic visualizations," *IEEE transactions on visualization and computer graphics*, vol. 25, no. 7, pp. 2505–2512, 2018.
[32] K. Hu, S. Gaikwad, M. Hulsebos, M. A. Bakker, E. Zgraggen, C. Hidalgo, T. Kraska, G. Li, A. Satyanarayan, and Ç. Demiralp, "Viznet: Towards a large-scale visualization learning and benchmarking repository," in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, 2019, pp. 1–12.
[33] X. Qin, Y. Luo, N. Tang, and G. Li, "Making data visualization more efficient and effective: a survey," *The VLDB Journal*, pp. 1–25, 2019.
[34] N. Tang, E. Wu, and G. Li, "Towards democratizing relational data visualization," in *Proceedings of the 2019 International Conference on Management of Data*, 2019, pp. 2025–2030.
[35] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European conference on computer vision*. Springer, 2014, pp. 818–833.
[36] S. Liu, X. Wang, M. Liu, and J. Zhu, "Towards better analysis of machine learning models: A visual analytics perspective," *Visual Informatics*, vol. 1, no. 1, pp. 48–56, 2017.
[37] F. M. Hohman, M. Kahng, R. Pienta, and D. H. Chau, "Visual analytics in deep learning: An interrogative survey for the next frontiers," *IEEE transactions on visualization and computer graphics*, 2018.
[38] J. Han, J. Tao, and C. Wang, "Flownet: A deep learning framework for clustering and selection of streamlines and stream surfaces," *IEEE transactions on visualization and computer graphics*, 2018.
[39] B. Kim and T. Günther, "Robust reference frame extraction from unsteady 2d vector fields with convolutional neural networks," *arXiv preprint arXiv:1903.10255*, 2019.
[40] K. Hu, M. A. Bakker, S. Li, T. Kraska, and C. Hidalgo, "Vizml: A machine learning approach to visualization recommendation," in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, 2019, pp. 1–12.
[41] V. Dibia and Ç. Demiralp, "Data2vis: Automatic generation of data visualizations using sequence-to-sequence recurrent neural networks," *IEEE computer graphics and applications*, vol. 39, no. 5, pp. 33–46, 2019.
[42] Z. Chen, W. Zeng, Z. Yang, L. Yu, C.-W. Fu, and H. Qu, "Lassonet: Deep lasso-selection of 3d point clouds," *IEEE transactions on visualization and computer graphics*, 2019.
[43] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
[44] S. Chopra, R. Hadsell, Y. LeCun *et al.*, "Learning a similarity metric discriminatively, with application to face verification," in *CVPR (1)*, 2005, pp. 539–546.
[45] K. Schittkowski, "Easy-fit: a software system for data fitting in dynamical systems," *Structural and Multidisciplinary Optimization*, vol. 23, no. 2, pp. 153–169, 2002.
[46] R. Likert, "A technique for the measurement of attitudes." *Archives of psychology*, 1932.

Chaoran Fan is currently working toward the Ph.D. degree at the Department of Informatics, University of Bergen, Bergen, Norway. His research interest focuses on information visualization, specifically for improving the user interaction in visual analytics by leveraging the machine learning knowledge. Contact him at Chaoran.Fan@uib.no.

Krešimir Matković is a senior researcher at VRVis Research Center and head of Interactive Visualization Group. He is interested in extending visual analysis technology to challenging heterogeneous data, in particular to a combination of multi-variate data and more complex data types. Contact him at matkovic@vrvis.at.

Helwig Hauser is a professor in visualization at the Department of Informatics at the University of Bergen, Norway, where he leads a research group on visualization as well as CEDAS, UiB's Center for Data Science. His research interests include visual data science, interactive visual data exploration and analysis, visualization in general, and the application of visualization to various domains.