

APPLICATION OF REPRESENTATION LEARNING TO DISCRETE EVENT SEQUENCES ACROSS DISCRETE DOMAINS

CROSS-REFERENCE TO RELATED APPLICATION

5 [0001] This application claims priority to, and the benefit of, United States Provisional Patent Application No. 63/621,032 filed on January 15, 2024, the teachings of which are hereby incorporated by reference.

TECHNICAL FIELD

[0002] The present disclosure relates to representation learning, and more particularly to the application of representation learning to time series data.

10 BACKGROUND

[0003] Representation learning is a type of machine learning in which unlabeled data is processed to extract patterns, which are used to generate representations that can then be further processed, for example to make predictions.

15 [0004] Representation learning can be applied to time series data, but there remains a challenge in integrating representations from disparate types of time series data.

SUMMARY

[0005] In one aspect, a method for constructing an entity representation engine is provided. The method receives time series training data representing a plurality of discrete training event sequences. Each training event sequence is associated with a respective one of a
20 plurality of discrete separate data domains and the training event sequences are time-ordered sequences of sample interactions within the respective discrete separate data domains. The method applies representation learning to the time series training data to generate a trained representation engine. The trained representation engine is adapted to ingest entity time series data representing a plurality of discrete entity event sequences for a particular entity. Each
25 entity event sequence is associated with a respective one of the discrete separate data domains and the entity event sequences are time-ordered sequences of entity interactions for that entity within the respective discrete separate data domains. The trained representation engine is

further adapted to, from the entity time series data, generate a corresponding single unified entity representation characterizing a distribution of the entity interactions within the discrete separate data domains.

5 [0006] The method may further comprise using labeled unified entity representation instances to train a downstream prediction model to predict an entity characteristic from an unlabeled unified entity representation instance.

10 [0007] In some embodiments, there are a plurality of entity characteristics and at least some of the labeled unified entity representation instances are only partially labeled for only a subset of the entity characteristics. Using the labeled unified entity representation instances to train the downstream prediction model to predict the plurality of entity characteristics from the unlabeled unified entity representation instance may comprise providing different ones of the labeled unified entity representation instances that are only partially labelled, wherein the different ones of the labeled unified entity representation instances have respectively different subsets of the entity characteristics that are labelled. The method may further comprise
15 providing other different ones of the labeled unified entity representation instances that are fully labelled.

[0008] The entity characteristic may be a characteristic of the entity in interacting with one of the distinct separate data domains. The entity characteristic may comprise an aggregate statistic of the entity interactions.

20 [0009] In some embodiments, the trained representation engine may be adapted to generate the single unified entity representation by combining a plurality of domain-specific entity representations into the single unified entity representation. In some such embodiments, the trained representation engine may be adapted to combine the plurality of domain-specific entity representations into the single unified entity representation by concatenation. In other
25 such embodiments, the trained representation engine may be adapted to combine the plurality of domain-specific entity representations into the single unified entity representation by aggregation. In particular embodiments, the aggregation may comprise summing or averaging.

[0010] In preferred embodiments, the time series data is asynchronous time series data.

[0011] According to some embodiments, the discrete separate data domains may be financial data domains, the training event sequences may be time-ordered sequences of sample financial transactions within the respective financial domains, the entity event sequences may be time-ordered sequences of financial transactions for that entity within the respective financial data domains, and the at least one entity characteristic may comprise credit capacity. According to some embodiments, the financial data domains may include two or more selected from the following list: credit cards, bank accounts, mortgages, and bill payments. In some embodiments, a computer system may automatically determine whether to grant or deny credit based on a prediction of the credit capacity.

[0012] In some embodiments, applying representation learning to the time series training data comprises applying self-supervised learning. The self-supervised learning may utilize a recurrent neural network (RNN). The self-supervised learning may utilize a temporal point process (TPP) model. In particular embodiments, the TPP model is a Meta TPP model. The self-supervised learning may utilize one or more of a conditional diffusion model, bidirectional encoder representations from transformers (BERTs), contrastive learning, alignment prediction, masked auto-encoding, aggregated prediction within different time windows, tenure, 1-dimensional convolutional architecture and graph convolutional architecture.

[0013] In some embodiments, applying representation learning to the time series training data comprises applying supervised learning. The supervised learning may be fully supervised learning.

[0014] In other aspects, the present disclosure is directed to a data processing system comprising at least one processor and memory coupled to the at least one processor wherein the memory contains instructions which, when executed by the at least one processor, cause the at least one processor to carry out the method as described above.

[0015] In still further aspects, the present disclosure is directed to at least one tangible, non-transitory computer-readable medium embodying instructions which, when executed by the at

least one processor, cause the at least one processor to carry out the method as described above.

BRIEF DESCRIPTION OF THE DRAWINGS

5 [0016] These and other features will become more apparent from the following description in which reference is made to the appended drawings wherein:

FIGURE 1 is a flow chart showing an illustrative method for constructing an entity representation engine according to an aspect of the present disclosure;

FIGURE 1A shows initial training of an MTL model, according to an aspect of the present disclosure;

10 FIGURE 1B shows inference of the MTL model of Figure 1A, according to an aspect of the present disclosure;

FIGURE 1C shows re-training of the MTL model of Figure 1A, according to an aspect of the present disclosure;

15 FIGURE 2 shows RNN-based self-supervised learning according to an aspect of the present disclosure;

FIGURE 3 shows conditional diffusion self-supervised learning according to an aspect of the present disclosure;

FIGURE 3A shows use of aggregation in conditional diffusion self-supervised learning according to an aspect of the present disclosure;

20 FIGURE 3B shows use of concatenation in conditional diffusion self-supervised learning according to an aspect of the present disclosure;

FIGURE 4 shows contrastive self-supervised learning according to an aspect of the present disclosure;

FIGURE 5 shows alignment prediction self-supervised learning according to an aspect of the present disclosure;

FIGURE 5A shows alignment prediction self-supervised learning using BERT according to an aspect of the present disclosure;

5 FIGURE 6 shows masked auto-encoding self-supervised learning according to an aspect of the present disclosure;

FIGURE 6A shows masked auto-encoding self-supervised learning using BERT according to an aspect of the present disclosure;

10 FIGURE 7 shows fully-supervised learning using multi-layer perceptrons according to an aspect of the present disclosure;

FIGURE 8 shows an illustrative computer system in association with which aspects of the present disclosure may be implemented;

FIGURE 9 shows an illustrative implementation of a transformer with causal attention according to an aspect of the present disclosure;

15 FIGURES 10A and 10B show aggregated statistics prediction across multiple time windows according to an aspect of the present disclosure;

FIGURE 11A shows an illustrative tenure network during training, according to an aspect of the present disclosure;

20 FIGURE 11B shows the tenure network of Figure 11A during inference, according to an aspect of the present disclosure;

FIGURE 11C shows use of an illustrative 1D CNN as an encoder network, according to an aspect of the present disclosure; and

FIGURE 11D shows use of an illustrative graph convolutional network architecture as an encoder network, according to an aspect of the present disclosure.

DETAILED DESCRIPTION

[0017] The present disclosure may provide an illustrative foundation model for representing an entity in terms of its interactions across multiple discrete separate data domains. Various implementations can be built upon the model, some of which are described below.

5 [0018] The term “entity” is used herein in its broadest sense. An entity may be animate or inanimate, and may be a machine, or a location, or a system that operates as a combination of one or more of the foregoing. Certain non-limiting illustrative examples of types of “entity” follow. An animal may have interactions with other animals or with its environment, and these interactions may be monitored. A machine, such as a washer or dryer in a laundromat,
10 or a vending machine, may have interactions with users. An autonomous system, like a self-driving vehicle, may have interactions with its environment or with other vehicles, or a robot dog may have interactions with humans and with biological dogs. A location may be a portion of land, or water, which may be arbitrarily defined in its extent and may be monitored, for example, for ingress and egress of animals. A hospital is a system that is a combination
15 of a location, machines and personnel. A machine combination may be, for example, a website or electronic mail system, or a streaming service, or an artificial intelligence agent (with or without physical effectors). A veterinary clinic is a system that combines a location with machines, personnel, and merchandise (e.g. dispensed medication and special medical foods). A toy store is a system that is a combination of a location, machines and personnel, as
20 well as merchandise and in some contexts a website and shipping backend. Another example of an entity is a taxicab or ride share vehicle, which may be operated by a human, or may be autonomous. Again, the foregoing are merely non-limiting illustrative examples.

[0019] The term “interaction” refers to something that happens to an entity, or is done by an entity, and which has a time of occurrence (e.g. a timestamp) and at least one attribute. An
25 attribute may be an implied attribute, or an explicit attribute. For example, where an entity is a hospital, an interaction may be an arrival of a person at that hospital, which will have a time of arrival (occurrence), one or more attributes, including the inherent attribute of hospital arrival, and possibly additional attributes such as age, gender, nature of medical complaint, etc. Where an entity is an individual, an interaction may be a bank account transaction, which

will have a time of occurrence (this may be a date only, or may be more granular), the inherent attribute of being a bank account transaction, as well as an amount (which may be positive or negative), and potentially other information about the nature of the transaction. Where an entity is a monitored wildlife refuge, an interaction may be an avian entering the
5 airspace of the refuge, which will have a time of occurrence, its inherent attribute (arrival of an avian) and possibly additional attributes, such as altitude, or type of avian. For a taxicab or ride share vehicle, an interaction may be an arrival at a specific position, or a trip (which may have the trip commencement as the time of occurrence).

[0020] A “data domain” is a grouping of one or more types of interaction having a defined
10 common circumstance. One data domain is discrete and separate from another data domain (“discrete separate data domains”) if the two domains have a different common circumstance. A common circumstance, and a difference in common circumstance, may be an inherent real difference which is recognized in defining the circumstance or difference, or may be an arbitrarily defined difference chosen by an implementer of the methods described herein
15 according to the context and purpose. Certain illustrative non-limiting examples follow. For example, where an entity is a hospital there may be a single data domain for “all arrivals”, or there may be discrete separate data domains for each of ambulance arrivals, walk-in arrivals and helicopter arrivals. Where an entity is an individual, there may be one discrete separate data domain for bank account transactions, and another discrete separate data domain for
20 credit card transactions, or there might be discrete separate data domains for each of chequing accounts and savings account, or for each credit card of a particular brand even if offered by the same bank. Where an entity is a monitored wildlife refuge, there may be one domain for all avians, or there may be discrete separate data domains for different species of avians. Where an entity is a ride share vehicle within a service that provides both transportation (e.g.
25 of humans) and delivery services (e.g. food delivery), there may be a single data domain for all trips, or there may be discrete separate data domains for each of transportation trips and delivery trips. Preferably, the discrete separate data domains are mutually exclusive, but in some contexts, there may be overlap. For example, a hospital may have discrete separate data domains for each of ambulance arrivals, walk-in arrivals and helicopter arrivals and may have
30 a further discrete separate data domain for heart attack arrivals (which overlaps the other

hospital data domains) but each is still a discrete separate data domain. Or, where an entity is an individual, there might be discrete separate data domains for each of credit card, mortgage, and line of credit, and then another discrete separate data domain for credit products generally.

5 [0021] Importantly, because each interaction relates to an entity, has a time of occurrence (e.g. a timestamp), and can be grouped into a data domain, an entity's interactions within a particular data domain can be represented as a time series, that is, a time-ordered event sequence where each event in the sequence has at least one attribute (which, as noted above, may be a single inherent attribute). Moreover, the time series data is asynchronous time series
10 data, that is, the interactions are not temporally evenly spaced apart.

[0022] Considering a toy store as an entity, examples of interactions may include in-person sales and online sales, and in the case of an online sale, the action of shipping of the item(s). In-person sales and online sales may be treated as respective discrete separate data domains, or as a single data domain of "sales" which may have an attribute of "in-person" or "online".
15 Other attributes of a sale interaction may include the dollar value and the item(s) purchased, which may be as granular as the stock keeping unit (SKU) or universal product code (UPC) number(s) for the item(s), or less granular, such as a brand or a category, such as "doll" or "action figure" or "board game", or "boys toy" or "girls toy 8-10". These are merely non-limiting illustrative examples.

20 [0023] In the context of a veterinary clinic, an interaction may be an arrival at the clinic, which may have the same attributes as a hospital arrival, as well as additional attributes such as the species and breed of the patient (e.g. a cockapoo dog), and attributes relating to the accompanying human guardian (owner) of the patient (e.g. dog, cat, iguana, parrot, etc.). Scheduled arrivals based on appointments and unscheduled walk-in arrivals may be treated as
25 respective discrete separate data domains, or as a single data domain. The scheduling of an appointment may itself be an interaction within a data domain that is separate and discrete from the data domain for the corresponding arrival. Cancellation of an appointment may also be an interaction. Other examples of an interaction may include admission of a patient to the veterinary clinic (e.g. a veterinary clinic providing emergency care), and discharge of an

admitted patient. Still other examples of an interaction may include the prescription of medications, and the fulfillment of such prescription (where the clinic has an in-house pharmacy). A sale of ancillary item(s) such as food, animal toothpaste, animal vitamins, or animal toys would also be an interaction, whose attributes would relate to the items sold. A much sadder example of an interaction is euthanasia, which may include attributes such as transfer for necropsy, transfer for cremation, or release for burial.

[0024] An important objective of the technology described in the present disclosure is to be able to provide an overall holistic representation of an entity, in the form of a single unified entity representation characterizing a distribution of the entity interactions within the discrete separate data domains.

[0025] Reference is now made to Figure 1, which is a flow chart showing an illustrative method 100 for constructing an entity representation engine, that is, a trained machine learning engine, which can receive input data associated with an entity and then generate a representation of that entity.

[0026] At step 102, the method 100 receives time series training data representing a plurality of discrete training event sequences. Each training event sequence is associated with a respective one of a plurality of discrete separate data domains. The training event sequences are time-ordered sequences of sample interactions within the respective discrete separate data domains. Thus, there are multiple training event sequences, and each training event sequence is associated with one of the discrete separate data domains, so that multiple discrete separate data domains are represented in the training event sequences.

[0027] At step 104, the method 100 applies representation learning to the time series training data received at step 102 to generate a trained representation engine. Some non-limiting illustrative examples of application of representation learning to time series training data are described further below.

[0028] The trained representation engine generated at step 104 is adapted to ingest entity time series data and, from the entity time series data, generate a corresponding single unified entity representation characterizing a distribution of the entity interactions within the discrete

separate data domains. The entity time series data that the trained representation will ingest represents a plurality of discrete entity event sequences for a particular entity. Each entity event sequence is associated with a respective one of the discrete separate data domains, and the entity event sequences are time-ordered sequences of entity interactions for that entity within the respective discrete separate data domains. Thus, even though there are entity event sequences associated with a plurality of different discrete separate data domains, the output of the trained representation engine generated at step 104 is not multiple representations, but rather a single unified entity representation. The unified entity representation may be a reduced dimensionality characterization, for example, and without limitation, a feature vector, or weights of a neural network.

[0029] In some embodiments, the trained representation engine may be adapted to generate the single unified entity representation by combining a plurality of domain-specific entity representations into the single unified entity representation, for example by concatenation or aggregation, including summing and averaging.

[0030] At optional step 106, the method 100 uses labeled unified entity representation instances to train a downstream prediction model to predict an entity characteristic from an unlabeled unified entity representation instance. The labeled unified entity representation instances may be obtained by using the trained representation engine from step 104 to obtain unified entity representation instances, and then applying a target label for the entity characteristic to be predicted. The entity characteristic to be predicted may be a characteristic of the entity in interacting within one of the distinct separate data domains, or an aggregate statistic of the entity interactions, or another characteristic.

[0031] Some illustrative, non-limiting examples of entity characteristics are now provided. Where the entity is a hospital, the entity characteristic may be when the next patient is expected to arrive by ambulance, or the total number of patients expected in the next hour, or the gender distribution of the patients expected within the next six hours. As another example, where the entity is an individual and the discrete separate data domains relate to financial transactions, the entity characteristic may be the timing and/or amount of the next credit card payment, or a total amount of deposits expected within the individual's chequing

account within the next month, or the individual's credit capacity (the ability to take on debt). As another example, where the entity is a monitored wildlife refuge, the entity characteristic may be what type of avian is expected to arrive next, or the total number of avians expected to be present in the wildlife refuge at some future time. Where the entity is a taxicab or a ride
5 share vehicle, the prediction may be, for example, the time of the next trip, or the distance of the next trip, or the time of the next delivery trip. Where the entity is a toy store, the entity characteristic may be how many of a certain type of doll will be sold within the next week, or a ratio of boys toys to girls toys expected to be sold within the next month. Where the entity
10 is a veterinary clinic, the entity characteristic may be a number of cats expected to have rabies vaccination appointments during the next six months. Each of the foregoing is merely a non-limiting, illustrative example.

[0032] In one embodiment of the method 100, the discrete separate data domains are financial data domains, and the training event sequences are time-ordered sequences of sample financial transactions within the respective financial data domains, such as credit card charges
15 and payments, bank account deposits and withdrawals, mortgage payments, and bill payments, for example. Then, the entity event sequences are time-ordered sequences of financial transactions for that entity within the respective financial data domains. In this embodiment, an entity characteristic may be credit capacity, and the labeled unified entity representation instances may be used to train the downstream prediction model to predict the
20 credit capacity from the unlabeled unified entity representation instance. In some embodiments, a computer system may automatically determine whether to grant or deny credit based on a prediction of the credit capacity.

[0033] The downstream prediction model trained at step 106 may be trained using any suitable machine learning technology and, unlike step 104, the prediction model trained at
25 step 106 is not limited to representation learning. For example, in one embodiment, step 106 can be implemented using a multilayer perceptron (MLP), which consists of fully connected layers and non-linear layers and takes the unified entity representation as input, and directly outputs the predictions for the downstream tasks. In another embodiment, a linear

transformation may be directly applied to the unified entity representation to obtain the prediction.

[0034] In some embodiments of step 106, the method 100 uses the labeled unified entity representation instances to train a downstream prediction model to predict a plurality of entity characteristics from an unlabeled unified entity representation instance. In particular implementations of such embodiments, multi-task learning (MTL) may be used. In such embodiments, at least some of the labeled unified entity representation instances are only partially labeled for only a subset of the entity characteristics. Training of the downstream prediction model to predict the plurality of entity characteristics from the unlabeled unified entity representation instance comprises providing different ones of the labeled unified entity representation instances that are only partially labelled. The different ones of the labeled unified entity representation instances have respectively different subsets of the entity characteristics that are labelled. Training of the downstream prediction model may further comprise providing other different ones of the labeled unified entity representation instances that are fully labelled.

[0035] Reference is now made to Figures 1A to 1C. Let there be D datasets $\{D_1, D_2, \dots, D_D\}$, each consisting of input-output pairs, where x_i^j is an input sample, and y_i^j is the ground truth for one or more tasks. In Figures 1A to 1C, three datasets 120, 122, 124 are shown for purposes of illustration, there may be more or fewer datasets in practice. Let $T = \{t_1, t_2, \dots, t_t\}$ represent the set of tasks, where each task t_k has a corresponding prediction head f_{t_k} 130, 132, 134. Construct a batch β 126 of size N by selecting samples from each dataset D_i , such that in each batch β 126 there will be an equal number of samples per dataset 120, 122, 124. Each sample z in each batch β is passed through a shared feature extractor (e.g. shared layers) f_{shared} 128, resulting in a shared representation:

$$h_i = f_{shared}(x_i)$$

[0036] For each task $t_k \in T$, compute predictions $\hat{y}_i^{t_k}$ using the corresponding task-specific head f_{t_k} 130, 132, 134:

$$\hat{y}_i^{t_k} = f_{t_k}(h_i)$$

[0037] For each sample (x_i, y_i, t_k) in the batch β , compute the loss $\mathcal{L}_{i,k}$ as follows:

$$\mathcal{L}_{i,k} = \begin{cases} \ell(\hat{y}_i^{t_k}, y_i^{t_k}), & \text{if } y_i^{t_k} \text{ available} \\ 0, & \text{otherwise} \end{cases}$$

5 [0038] Then, aggregate the individual losses across the batch β to compute the total batch loss, where there are M tasks and N samples:

$$\mathcal{L}_\beta = \frac{1}{N} \frac{1}{M} \sum_{k=1}^M \sum_{i=1}^N \mathcal{L}_{i,k}$$

[0039] In Figures 1A to 1C, three datasets 120, 122, 124 are shown for purposes of illustration. Each dataset comprises a plurality of respective samples 120A, 122A, 124A.

10 Each dataset 120, 122, 124 can have a ground truth for one or many tasks. In the illustrated embodiment, the first dataset 120 has ground truth labels available for tasks X and Y, the second dataset 122 has ground truth labels available for task Z, and the third dataset 124 has ground truth labels available for task X and Z. After feeding each batch 126 to the feature extractor 128, output features are fed to the respective task-specific heads 130, 132, 134.

15 Each task has its own task-specific head. Thus, there is a task X head 130, a task Y head 132 and a task Z head 134.

[0040] During initial training of the MTL model, as shown in Figure 1A, loss is applied for the samples 120A, 122A, 124A that have ground truth labels for the task of the respective task-specific head 130, 132, 134 and loss for the samples 120A, 122A, 124A that do not have ground truth labels for that task will be masked (set equal to 0). Thus, as shown by the circled
20 “X” indicators in Figure 1A, each task-specific head 130, 132, 134 is used to generate pseudolabels for the samples 120A, 122A, 124A that did not have a ground truth label for the respective task. Accordingly, the task X head 130 generates pseudolabels for the samples 122A in the second dataset 122, the task Y head 132 generates pseudolabels for the samples

122A in second dataset 122 and for the samples 124A in the third dataset 124, and the task Z head 134 generates pseudolabels for the samples 120A in the first dataset 120.

[0041] Reference is now made to Figure 1B, which shows inference of the MTL model.

During inference, loss is not applied to the samples for which there is no ground truth label.

5 Thus, as denoted by the circled checkmark indicators, the task X head 130 does not apply loss to the samples 122A in the second dataset 122, the task Y head 132 does not apply loss to the samples 122A in the second dataset 122 or to the samples 124A in the third dataset 124, and the task Z head 134 does not apply loss to the samples 120A in the first dataset 120.

10 [0042] Reference is now made to Figure 1C, which shows retraining of the MTL model by combining the ground truth labels and the pseudolabels. Each sample 120A, 122A, 124A is labeled for all tasks. The label can be the ground truth label or the pseudolabel if the ground truth label is not available.

[0043] Some non-limiting, illustrative implementations of step 104 of the method 100 will now be described.

15 **Model-Specific Self-supervised Learning**

[0044] In some embodiments, model-specific self-supervised learning approaches are used to apply representation learning to the time series training data. Model-specific approaches are self-supervised learning approaches that rely on specific types of models.

Recurrent Neural Network (RNN)-Based Self-Supervised Learning

20 [0045] In a preferred embodiment of RNN-based self-supervised learning, a Long Short Term Memory (LSTM) neural network is used as the backbone. The LSTM model is trained in a self-supervised way by framing a supervised learning task to predict only a subset of information, using the rest of the information. In this way, both inputs and labels are provided. For instance, given a sequence of interactions, a subset of the interactions is used as input and
25 the rest of the interactions are used as the target. The LSTM model can be trained to predict the next interaction in a sequence given previous interactions. For example, let

$\{x_1, x_2, \dots, x_n\}$ be a sequence of interactions. Feeding the interactions sequentially into an LSTM can be represented as $h_i = LSTM(x_i, h_{i-1})$ where h_i represents the internal states of the LSTM at step i , including both the hidden state and the cell state. The values of the initial internal state h_0 can be initialized to zero. A prediction of the next interaction x_{i+1} , $\widetilde{x}_{i+1}$, is made based on the internal state h_i by feeding it as input to a prediction network i.e., $\widetilde{x}_{i+1} = pred(h_i)$. The prediction network can be a multi-layer perceptron (MLP), a linear layer or any other network suitable for the prediction. The LSTM and the prediction network can be trained together by minimizing a loss between $\widetilde{x}_{i+1}$ and x_{i+1} . The loss could comprise L2 loss if the value of x_i is continuous or cross entropy if the value of x_i is categorical, or any combination of different losses that fit the data type of x_i . Selection of suitable loss(es) is within the capability of one of ordinary skill in the art, now informed by the present disclosure.

[0046] Thus, as shown in Figure 2, a sequence 202 of interactions 204 is fed to an LSTM model 206, which generates a prediction 208. For the LSTM model 206 to take the sequence 202 of interactions 204 as input, the interactions 204 must be given a fixed order. While only two interactions 204 are shown for simplicity of illustration, there may in practice be hundreds, thousands or even millions of interactions. A key challenge in determining the fixed order is that some interactions 204 can have the same time stamp if they happen in the same time period, depending on the granularity of time recording. This challenge may be addressed by sorting the sequence 202 of interactions 204 of an entity in a particular data domain by (1) the time stamp of the interaction 204 and (2) an additional variable other than the time stamp, e.g., interaction codes or interaction amounts, with the additional variable used as a tiebreaker if two interactions 204 have the same time stamp. To encode interaction amount values, high-frequency encoding may be used to transform the value of the interaction amount into a vector using a series of sine and cosine functions with different frequencies. In addition, sine and cosine transformations may be used to encode information in the time stamps for the interactions to capture cyclic patterns in the interaction sequences. Positional encoding may be used to encode the time stamps of the interactions 204. The learning task is to generate a prediction 208 for the next interaction in the sequence 202. To generate a representation of the entities for a particular data domain, the trained LSTM model 206 is fed

the entire sequence 202 of interactions 204. The last hidden state of the LSTM model is taken as the entity representation. To extend the approach to multiple data domains, different LSTM models 206 can be trained for each of the discrete separate data domains, and the entity representations for each of the discrete separate data domains can be combined using
5 aggregation approaches like (for example) concatenation, summation and mean pooling to obtain the single unified entity representation.

Meta-TPP-Based Self-Supervised Learning.

[0047] A temporal point process (TPP) is a stochastic process where its realization is a sequence of discrete events in time. Because interaction sequences are asynchronous i.e., they
10 are not evenly spaced apart, there may be a signal that can be extracted from how often the interactions occur, and how far apart they are. A time series (sequence of interactions) can be modeled as a temporal point process, to model the underlying process by learning to predict the distribution of the interaction time. This approach models the time difference between events, and predicts not only the interactions but also when the interactions occur.

[0048] An intermediate output of a model trained to predict the distribution of the interaction time can be used to obtain domain-specific entity representations that can be combined to obtain the single unified entity representation, or the model may be configured to output the single unified entity representation. In the former case, a separate TPP model can be trained for each of the discrete separate data domains to obtain the domain-specific entity
15 representations, which can then be fused together to obtain the final embedding as the single unified entity representation. In the latter case, time series (sequences of interactions) from the discrete separate data domains are fused together into a single combined time series and sorted by their time stamps. Interactions from different ones of the discrete separate data domains are treated as discrete categories within the single combined time series, and a TPP
20 model can then be trained on the single combined time series.

[0049] In some embodiments, the TPP model may be a Meta TPP model. Implementation of the Meta TPP approach is described in *Meta Temporal Point Processes* by Wonho Bae, Mohamed Osama Ahmed, Frederick Tung and Gabriel L. Oliveira, published in a conference

paper at ICLR 2023 and available at <https://arxiv.org/pdf/2301.12023.pdf> and also described in United States Patent Application No. 18/460,478 filed September 1, 2023 and Canadian Patent Application No. 3,211,408 filed September 7, 2023, each of which is hereby incorporated by reference.

5 Conditional Diffusion Model (CDM) for Representation Learning

[0050] Diffusion models are generative models that transform noise to data by iteratively denoising the input. During the training of diffusion models, Gaussian noise of variable scales is applied to corrupt the input data so that when the noise scale is large, inputs will be indistinguishable from pure noise. A denoising model is trained to predict the clean version of input data from the noisy version with knowledge of the noise scale. During the training of a conditional diffusion model, only a randomly selected portion of the sequence of interactions will be corrupted by noise and the remaining part of the sequence of interactions will be kept clean. Inputs to the denoising model will include both the noisy and clean portions of the sequence of interactions as well as the noise scale. The denoising model will predict the original values of the noisy portion of the sequence of interactions, conditioned on its clean portion.

[0051] Thus, as shown in Figure 3, within a sequence 302 of interactions 304A, 304B, some of the interactions 304A are clean (not corrupted by noise) and other interactions 304B are corrupted by Gaussian noise. Although only three interactions 304A, 304B are shown for simplicity of illustration, there may in practice be hundreds, thousands or even millions of interactions. Let $x = \{x_1, x_2, \dots, x_n\}$ be a sequence of interactions represented as continuous values. If the original values of each x_i are discrete, continuous values can be obtained by mapping the values of each x_i to their continuous embeddings. Let COR be a subset of $\{1, 2, \dots, n\}$ indicating the set of indices of events to be corrupted. Corruption can be applied to x_i for $i \in COR$ by directly adding noise to x_i , i.e. $\bar{x}_i = x_i + \alpha(s)\epsilon_i$ or performing some interpolation between $\bar{x}_i = (1 - \alpha(s))x_i + \alpha(s)\epsilon_i$ where ϵ_i represents independent standard Gaussian noise and $\alpha(s)$ is a value controlling the scale of corruption by noise. This results in the sequence 302 of interactions 304A, 304B including clean interactions 304A and

corrupted interactions 304B. The sequence 302 of interactions 304A, 304B is fed to a transformer encoder model 306, together with the information indicating which transactions 304B are noisy and the variance (scale) 308 of the noise. The transformer encoder model 306 will denoise the noisy transaction 304B and generate a prediction 310 of the clean value. Let

5 $y = \{y_1, y_2, \dots, y_n\}$ be a sequence such that $y_i = \text{concat}(\bar{x}_i, 1, \alpha(s))$ for $i \in COR$ and $y_i = \text{concat}(x_i, 0, 0)$ otherwise. Then, y is fed to the transformer encoder model 306, denoted mathematically as ENC and let $h = \{h_1, h_2, \dots, h_n\}$ be the output of the transformer encoder model 306, i.e. $h = ENC(y)$. Another prediction head is used to map each h_i for $i \in COR$ to a predicted clean version of $\bar{x}_i$, $\tilde{x}_i$, and minimize the distance between $\tilde{x}_i$ and x_i . The

10 prediction head could be an MLP or a linear layer or any other network architecture that fits the type of data. Selection of suitable architecture is within the capability of one of ordinary skill in the art, now informed by the present disclosure.

[0052] To obtain the domain-specific entity representation from the sequence of interactions for one of the discrete separate data domains, the denoising model (e.g. transformer encoder)

15 is fed the entire clean interaction sequence without adding any noise, and a suitable aggregation method, for example (and without limitation) mean-pooling, is used to aggregate some intermediate hidden states of the denoising model, the aggregated vector is taken as the domain-specific entity representation. The input sequence y to the transformer encoder model (denoted as ENC) is obtained by setting COR to an empty set, which means $y_i = \text{concat}(x_i, 0, 0)$ for each i . After obtaining h by feeding y to the transformer encoder model

20 ENC , the final domain-specific entity representation of the interactions is obtained by applying aggregation (e.g. mean pooling) over the sequence of h_i values, i.e. $REP = AGGREGATE(\{h_1, h_2, \dots, h_n\})$.

[0053] A key challenge of the conditional diffusion model approach is that diffusion models

25 typically work best with continuous data and there are discrete categorical data types, for example an interaction category code, in the sequences of interaction. An “interaction category code” describes an attribute of the interaction, within a data domain. To adapt the conditional diffusion model for discrete categorical data types, the categorical values may be converted into a set of embeddings, which are vectors of continuous values. Gaussian noise

can then be injected to these continuous embeddings consistent with the conditional diffusion model. To train the model for denoising the embeddings corrupted with Gaussian noise, the model is trained to produce a categorical distribution that assigns a set of weights to different interaction category codes and their embeddings. The weighted average of the category embeddings is used as the prediction of the original clean embedding vector, i.e. the domain-specific entity representation. Thus, denoising noisy category embeddings is achieved by predicting weights of the categories and taking the weighted sum of the categories' embeddings as the predicted clean embedding value. More concretely, for an interaction of discrete categorical data types with K categories, let $emb_1, emb_2, \dots, emb_K$ be the embeddings of the K categorical values. Assume x_i to take one of the embeddings. When predicting the clean version of the corrupted $\bar{x}_i$, first predict K weights over the K categories denoted as $w_i = \{w_i^{(1)}, w_i^{(2)}, \dots, w_i^{(K)}\}$ such that $\sum_{k=1}^K w_i^{(k)} = 1$ by applying a classification network to the output of the transformer encoder model at position i , i.e. $w_i = CLASS(h_i)$. Compute the predicted clean version $\tilde{x}_i$ as $\tilde{x}_i = \sum_{k=1}^K w_i^{(k)} emb_k$. The distance between $\tilde{x}_i$ and x_i can be minimized by training the classification model with a classification loss and maximizing the weight of the category of x_i . Some post-processing steps like clipping the prediction to the closest one in the set of category embeddings can also be taken.

[0054] A number of approaches can be used to generate the single unified entity representation.

[0055] One approach is to separately train different denoising models (e.g. transformer encoder model 306 in Figure 3) for each of the discrete separate data domains. Given sequences of interactions from multiple discrete separate data domains, a representation vector (i.e. a domain-specific entity representation) can be obtained for each individual data domain by using the different denoising models. The individual representation vectors from different data domains can be fused together using methods like concatenation, sum, or average to obtain the final representation vector of the entity, that is, the single unified entity representation. More concretely, let $REP^{(1)}, REP^{(1)}, \dots, REP^{(D)}$ be representations of interactions from D different data domains obtained using approaches described above in respect of Figure 3 using aggregation. The final representation of interactions REP (the

unified entity representation) may be obtained by fusing the D domain-specific entity representations. If concatenation is used for fusing, $REP = concatenate(\{REP^{(1)}, REP^{(1)}, \dots, REP^{(D)}\})$. If summation is used for fusing, $REP = \sum_{d=1}^D REP^{(d)}$. If averaging is used for fusing, $REP = \frac{1}{D} \sum_{d=1}^D REP^{(d)}$. The foregoing assumes

5 the representation vectors of all domains have the same dimension when summation or averaging is used for fusing. This approach of fusing individual domain-specific entity representations to obtain the unified entity representation is shown in Figure 3A. In Figure 3A, a respective sequence 302A of the clean (not corrupted by noise) interactions 304A for each discrete separate data domain (Domain 1, Domain 2 and Domain 3 in the simplified

10 representation shown in Figure 3A) is fed to a respective CDM encoder 306A for that domain. The CDM encoders 306A for each domain generate respective domain-specific entity representations 312, to which aggregation 314 is applied to obtain the unified entity representation 316A.

[0056] Another approach is to concatenate the sequences from all the discrete separate data

15 domains into a single long sequence and apply noise to a random portion of this concatenated sequence to train the denoising model (e.g. transformer encoder model 306 in Figure 3). Then, as shown in Figure 3B, a single concatenated long sequence 302B of the clean (not corrupted by noise) interactions 304A from the discrete separate data domains (Domain 1, Domain 2 and Domain 3 in the simplified representation shown in Figure 3B) is fed to a

20 single CDM encoder 306B and the single unified entity representation 316B can be obtained through the same aggregation approach described above to obtain the domain-specific entity representation. Let the output be $\{h_1^{(1)}, h_2^{(1)}, \dots, h_{n_1}^{(1)}, h_1^{(2)}, h_2^{(2)}, \dots, h_{n_2}^{(2)}, \dots, h_1^{(D)}, h_2^{(D)}, \dots, h_{n_D}^{(D)}\}$ and the domain-specific entity representation can be obtained by

AGGREGATE($\{h_1^{(1)}, h_2^{(1)}, \dots, h_{n_1}^{(1)}, h_1^{(2)}, h_2^{(2)}, \dots, h_{n_2}^{(2)}, \dots, h_1^{(D)}, h_2^{(D)}, \dots, h_{n_D}^{(D)}\}$), for example

25 concatenation, summation, average. More concretely, let $x^{(1)} = \{x_1^{(1)}, x_2^{(1)}, \dots, x_{n_1}^{(1)}\}$, $x^{(2)} = \{x_1^{(2)}, x_2^{(2)}, \dots, x_{n_2}^{(2)}\}$, ..., $x^{(D)} = \{x_1^{(D)}, x_2^{(D)}, \dots, x_{n_D}^{(D)}\}$ be sequences of interaction from D discrete separate data domains. A example of the long sequence 302B obtained by concatenating all of them could be $x =$

$\{x_1^{(1)}, x_2^{(1)}, \dots, x_{n_1}^{(1)}, x_1^{(2)}, x_2^{(2)}, \dots, x_{n_2}^{(2)}, \dots, x_1^{(D)}, x_2^{(D)}, \dots, x_{n_D}^{(D)}\}$. Optionally each $x_i^{(d)}$ can be augmented with a special token, $[TOKEN(d)]$, indicating the domain by concatenation along the data dimension of $x_i^{(d)}$. In other words, each $x_i^{(d)}$ is replaced by $append(x_i^{(d)}, [TOKEN(d)])$.

5 BERT-Style Generative Modelling.

[0057] Bidirectional encoder representations from transformers (BERTs) are a family of transformer-based language models pre-trained using self-supervised learning approaches on large-scale text corpora. The self-supervised learning approaches of BERT can be adapted for application to interaction sequence data. To apply BERT-styled self-supervised learning to sequences of interactions, the target data types, such as interaction amount values and interaction category codes, should be first represented as discrete values. For a data type that is continuous in nature, a binning approach can be used: a sequence of mutually exclusive intervals are created as bins for the continuous values, and each continuous value is encoded using the index of its interval (i.e. the bin into which it falls). A special integer value can be used to encode continuous values that do not fall into any of the bins. Since all of the relevant values for the sequence of interactions are reframed as discrete values, they can be transformed into tokens and therefore the self-supervised learning approaches of BERT can be applied. Positional encoding based on the position of each interaction in the respective sequence of interactions may be applied to the tokens.

20 [0058] The self-supervised learning tasks of BERT include (1) masked reconstruction and (2) alignment prediction. In the masked reconstruction task, a randomly selected subset of the discretized interaction values will be masked and replaced with a special mask value (e.g. "MASK") and the model is trained to predict the original discretized values from the contextual information provided by the other unmasked interaction values. A classification loss is used for this task as the data types of all the values to be predicted are discrete. In the alignment prediction task, the sequences of interactions for some entities will be altered by randomly replacing a portion of the sequence of interactions for that entity with a portion of the sequence of interactions for a different entity, while the sequences of interactions for some

entities will not be altered. The model is trained to predict whether a sequence of interactions consists entirely of interactions from the same entity, i.e. is aligned, or not. The masked reconstruction aspect of BERT is described below in the context of Figure 6A.

[0059] This approach will primarily rely on a transformer encoder model. The input to the transformer encoder is a sequence where the discretized values of different data types from individual ones of the discrete separate data domains will occupy different portions. To handle different data types (e.g., interaction amount and interaction category code) and the discrete separate data domains, a token indicating the data type and the domain would be appended to the token of the encoded value. Values of different data domains or modes can share the same embedding or have different embeddings when being transformed from discrete values into tokens. In mathematical notation, the operation above can be described as follows: Let $x_i^{(t,d)}$ be the i^{th} discretized interaction of type t in the interaction sequence of discrete separate data domain d and $emb_i^{(t,d)}$ of dimension dim_E be the embedding of $x_i^{(t,d)}$. Let $[TOKEN_T(t)]$ be the token of dimension dim_T indicating the type of data is t and $[TOKEN_D(d)]$ of dimension dim_D be the token indicating the discrete separate data domain of interaction is d . Appending the tokens to $emb_i^{(t,d)}$ would be $append(emb_i^{(t,d)}, [TOKEN_T(t)], [TOKEN_D(d)])$, a vector of dimension $dim_E + dim_T + dim_D$.

[0060] To obtain the single unified entity representation, a mean pooling over the outputs from the last layer of the transformer encoder can be taken as a vector to represent the entity, or a special class token can be appended to the input sequence and the vector from the transformer encoder's output at the position of the class token can be taken as the single unified entity representation. Let $x = \{x_1, x_2, \dots, x_n\}$ be the sequence of interactions. A special $[CLS]$ token can be appended to the beginning of x and the betokened x can be fed it to the transformer encoder. So, the input sequence is $\{[CLS], x_1, x_2, \dots, x_n\}$. Let $\{h_{CLS}, h_1, h_2, \dots, h_n\}$ be the sequence of outputs from the transformer encoders of the same length as x , where h_{CLS} is the output at the position corresponding to $[CLS]$ and h_i is the output at the position of x_i . Then h_{CLS} is the single unified entity representation of the entity.

[0061] Thus, this concatenates the data of different modes from discrete separate data domains into a long sequence and appends tokens to the data to indicate their data type and domain, and all of the data, whether originally discrete or originally continuous, are encoded as discrete values to be handled by a language model like BERT. The original BERT model is described in Devlin et al., *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* available at <https://arxiv.org/abs/1810.04805>, which is incorporated herein by reference.

Causal Attention for Transformers

[0062] Dense predictive coding may be used for self-supervised representation learning on sequential data (e.g. hospital arrivals, financial records). According to one aspect of the present disclosure, transformers may be used to produce dense future predictions. In one embodiment, the transformer executes the following steps: (1) mapping the sequences of interactions into sequences of embedding vectors; (2) concatenating the input embeddings with additional positional encoding vectors; (3) creating causal attention masks and (4) using a multi-head attention (MHA) operator with the causal masks to produce predictions for all timestamps in parallel. The causal attention mask is critical to ensure that the model can only attend to previous positions in the sequence of interactions, not future positions.

[0063] Embeddings from every timestamp may be aggregated by mean-pooling. The representation at the final time step or the mean-pooling aggregation of representations from all time steps could serve a domain-specific entity representation. To generate a corresponding single unified entity representation, different self-supervised transformers with causal attention can be trained for each of the discrete separate data domains, and their embeddings can be concatenated as the final representation vector. Thus, this approach leverages causal-attention transformers for self-supervised representation learning on event sequence data.

[0064] Let $x = \{x_1, x_2, \dots, x_n\}$ be a sequence of input to a transformer decoder after Step 2 and $\{h_1, h_2, \dots, h_n\}$ be the sequence of output. Since a transformer with causal attention is auto-regressive, one can have $h_i = \text{Last_state}(\text{TransformerDecoder}(\{x_1, x_2, \dots, x_i\}))$

where $Last_state()$ is the function taking the last state of a sequence of vectors and $\{h_1, h_2, \dots, h_i\} = TransformerDecoder(\{x_1, x_2, \dots, x_i\})$. As recurrent neural networks and transformer decoders are both auto-regressive in nature, the same training approach (next interaction prediction) can be used to obtain the domain-specific entity representation.

5 Alternatively, mean pooling can be applied over $\{h_1, h_2, \dots, h_n\}$ and obtain the domain-specific entity representation $h = \frac{1}{n} \sum_{i=1}^n h_i$.

[0065] A transformer with causal attention may be conceptualized as a transformer decoder. Figure 9 shows a sequence 902 of four interactions 904. Only four interactions 904 are shown for purposes of illustration; there may be hundreds, thousands or even millions of interactions. 10 The interactions 904 are fed to a transformer decoder layer 906, which feeds to an attention layer 916, with hidden states 920 for attending to the interactions 904. The first hidden state 920 attends to the first interaction 904. The second hidden state 920 attends to the first and second interactions 904. The third hidden state 920 attends to the first, second, and third interactions 904. The fourth hidden state attends to all of the interactions 904.

15 **Model-Agnostic Self-Supervised Learning**

[0066] In some embodiments, model-agnostic self-supervised learning approaches are used to apply representation learning to the time series training data. Model-agnostic approaches are self-supervised learning approaches that do not explicitly depend on specific types of models, and can be combined with the model-specific self-supervised learning approaches listed 20 above.

Prediction of Aggregated Interactions within a Time Period

[0067] Instead of predicting the next interaction in a sequence of interactions, it is also possible to predict aggregate statistics of interactions for the next time period (e.g. the next day). This is useful because there is no guarantee that interactions are ordered within a 25 particular time period. The model in this case would output a vector, with entries denoting the prediction of aggregated values for each interaction category code for the next time period. The predictions can be aggregated interaction amounts (how many total units are expended or

consumed for each interaction category code for that time period), interaction counts (how many interactions for each interaction category code for that day), as well as whether an interaction for that interaction category code is made at all for that time. This takes a higher-level view of the entity's interactions, without being limited to the fine-level detail of each interaction.

[0068] The losses to train a model for each kind of prediction are listed below. These are the simplest version of the losses, and there are many variations of the losses listed below that could be used to train the model. Suitable variations will be within the capability of one of ordinary skill in the art, now informed by the present disclosure.

[0069] A model can be trained to predict the aggregated interaction amounts with a mean squared error regression loss. As aggregated interaction amounts can vary from very small numbers to very large numbers, a symmetric logarithm function is also applied, which effectively applies the logarithm to numbers with large absolute values (both negative and positive).

[0070] Denote the prediction and ground truth aggregated interaction amount for category c at time t as $x_{t,c}$ and as $y_{t,c}$ respectively. The total period of time is T and total number of categories is C . The loss is as follows:

$$\mathcal{L} = \frac{1}{T \cdot C} \sum_{t=1, c=1}^{t=T, c=C} (\text{symlog}(y_{t,c}) - x_{t,c})^2$$

where $\text{symlog}()$ is the symmetric logarithm function shown below, wherein $\text{sgn}()$ is the sign function that is 1 if a value is positive, -1 if the value is negative, and 0 otherwise:

$$\text{symlog}(y) = \text{sgn} \cdot \log(|y| + 1)$$

[0071] Similarly for predicting counts, denote the prediction and ground truths as $x_{t,c}$ and $s_{t,c}$ respectively, where $s_{t,c}$ is the total number of interactions of category c that occurred at time t . The loss is as follows:

$$\mathcal{L} = \frac{1}{T \cdot C} \sum_{t=1, c=1}^{t=T, c=C} (s_{t,c} - x_{t,c})^2$$

[0072] To predict whether an interaction for a given category happened at time t , denote the ground truth as $b_{t,c}$ which is equal to 1 if an interaction of category c happened at time t , and equal to 0 if it did not. For this task, the binary cross entropy loss is used as shown below:

$$\mathcal{L} = \frac{1}{T \cdot C} \sum_{t=1, c=1}^{t=T, c=C} b_{t,c} \cdot \log x_{t,c} + (1 - b_{t,c}) \cdot \log(1 - x_{t,c})$$

[0073] To fuse multiple discrete separate data domains together, an approach similar to the TPP model approach described above can be used, and embeddings (domain-specific entity representations) from the different discrete separate data domains can be fused together post-training. Another approach is to predict aggregate amounts for the various interaction category codes from all of the discrete separate data domains combined, and treat that prediction as the single unified entity representation.

[0074] More concretely, given a data domain D_1 that has a set of categories C_1 and a data domain D_2 that has a set of categories C_2 a model can be trained on the combined set of categories $C_1 \cup C_2$. For example, the loss for two categories to predict the aggregated amounts could be written as:

$$\mathcal{L} = \frac{1}{T \cdot |C_1|} \left[\sum_{t=1, c_1=1}^{t=T, c_1=|C_1|} (\text{symlog}(y_{t,c_1}) - x_{t,c_1})^2 \right] + \frac{1}{T \cdot |C_2|} \left[\sum_{t=1, c_2=1}^{t=T, c_2=|C_2|} (\text{symlog}(y_{t,c_2}) - x_{t,c_2})^2 \right]$$

where $|C|$ is the size of each set, and could be extended to any number of data domains. This approach is referred to herein as “aggregated combined loss”.

[0075] For this task, any suitable machine learning model that can take in sequential information up to time $t - 1$ and make a prediction for time t of the types mentioned above (an aggregated prediction for a given category) can be used. Examples of such models include, but are not limited to, transformer and LSTM models.

5 [0076] In one embodiment, an LSTM model that consumes aggregated interaction amounts from prior timesteps can be used to make a prediction for the aggregated interaction amount, count, or whether or not an interaction of this category occurs.

[0077] The parameters of the neural network, such as the number of LSTMs to use in the stack of LSTMs, the hidden size of the LSTM, and dropout rates for different layers of the
10 model, can be varied. Another important choice is K , the number of categories to predict. In a preferred embodiment, the K most common categories are predicted as some categories are very rare. Categories outside of the K most common categories (i.e. less common than the K most common categories) can be mapped to a pseudo-category value during model training.

[0078] A dataset of interactions can be collected and the interactions for each category can be
15 aggregated for a given time period (e.g. a day). Combining the dataset and the neural network via the loss, a model can be trained and learns a representation. If a model is trained on each domain separately, a domain-specific representation may be obtained. If a model is trained for multiple domains together using the aggregated combined loss approach as described above, a single unified entity representation may be obtained.

20 Contrastive Learning

[0079] Representations can also be learned from sequences of interactions using contrastive learning, which seeks to learn discriminative features between positive and negative sample pairs. More specifically, the contrastive loss would create a similarity matrix for each sample pair of data and the objective is to produce high similarity scores for data that are alike (i.e.,
25 positive pairs) and low similarity scores for data that are distant (i.e., negative pairs). In the contrastive learning context, the quality of positive and negative samples is critical to

successful learning. Data augmentation is the key in this approach - it creates many discrete views of input data that can serve as positive and negative samples for efficient training.

[0080] For contrastive learning, different views of one sequence of interactions are created, which contain similar semantic information. One example of creating a different view of a sequence of interactions is to randomly mask some interactions in a sequence of interactions by replacing the interaction with a MASK token. Let $x = \{x_1, x_2, \dots, x_{i-1}, x_i, x_{i+1}, \dots, x_n\}$ be a sequence of interactions belonging to an entity. A different view of the sequence could be $x^{(m)} = \{x_1, x_2, \dots, x_{i-1}, [MASK], x_{i+1}, \dots, x_n\}$, where $[MASK]$ is a dedicated token indicating the interaction is masked. Thus, referring to Figure 4, there is an original view 402A of a first sequence of transactions 404 and a modified view 402B of the first sequence of transactions 404; the modified view 402B of the first sequence of transactions 404 is identical to the original view 402A of the first sequence of transactions 404 except that one of the transactions the modified view 402B is masked 406. There is also a second sequence 408 of transactions 410; this is an original view, i.e. none of the transactions 410 in the second sequence 408 of transactions 410 is masked. Although only three interactions 404, 410 are shown for simplicity of illustration, there may in practice be hundreds, thousands or even millions of interactions. Similarly, only a single modified view 402B is shown for simplicity, in practice there would be many modified views. The original view 402A and the modified view(s) 402B of the first sequence of transactions 404 are fed to an encoder model 412 to obtain an embedding 414A, 414B for each view. The second sequence 408 of transactions 410 is also fed to the encoder model 412 to obtain an embedding 416. The learning task is to maximize the similarity between the embedding 414A of the original view 402A of the first sequence and the embedding 414B of the modified view(s) 402B of the first sequence, and minimize the similarity between the embeddings of different sequences (e.g. between the embedding 414A of the original view 402A of the first sequence of transactions 404 and the embedding 416 of the second sequence 408 of transactions 410 (which as noted above is an original view). Let $y = \{y_1, y_2, \dots, y_n\}$ be a sequence of interactions belonging to a different entity and $h^{(x)}, h^{(x^{(m)})}$, and $h^{(y)}$ be outputs of the encoder by feeding $x, x^{(m)}$, and y respectively. The training objective is to minimize $sim(h^{(x)}, h^{(y)})$ and maximize

$\text{sim}(h^{(x)}, h^{(x^{(m)})})$ where $\text{sim}(\cdot)$ is a similarity function, e.g. cosine similarity. A pairwise similarity algorithm is described in Chen et al., *A Simple Framework for Contrastive Learning of Visual Representations* available at <https://arxiv.org/pdf/2002.05709>, which is incorporated herein by reference.

5 [0081] The approach is not restricted to specific models for the encoder model 410, which may use, for example, a transformer model or an LSTM model. For a transformer model, the contrastive learning may be conducted with the following steps: (1) creating different views of input data by any one or more of (a) randomly masking transactions, (b) clipping the original sequences of transactions into subsequences, and (c) adding noise; (2) adding a
10 special token indicating the end of a sequence, e.g. “[EOS]”, to the time series (interaction) data; (3) using transformers to encode input sequences of interactions to embeddings and taking the embedding at the location of the [EOS] token as the final representation, and (4) computing the contrastive loss. Thus, sequence-level augmentations including random
15 learning on event sequences. Let $x = \{x_1, x_2, \dots, x_{i-1}, x_i, x_{i+1}, \dots, x_{k-1}, x_k, x_{k+1}, \dots, x_n\}$ be a sequence of interactions. Random masking $x^{(m)}$ may be applied as described above. A different view of x created by clipping into subsequence can be $x^{(c)} = \{x_i, x_{i+1}, \dots, x_{k-1}, x_k\}$ where $1 \leq i < k \leq n$. A different view of x created by adding noise can be $x^{(n)} = \{x_1 + \epsilon_1, x_2 + \epsilon_2, \dots, x_{i-1} + \epsilon_{i-1}, x_i + \epsilon_i, x_{i+1} + \epsilon_{i+1}, \dots, x_{k-1} + \epsilon_{k-1}, x_k + \epsilon_k, x_{k+1} + \epsilon_{k+1}, \dots, x_n + \epsilon_n\}$ where ϵ_i s are noise added to x_i s. After training, the output of the
20 transformer at the location of the [EOS] token is taken as the global representation of a sequence of interactions. Different contrastive learning transformers can be trained for each of the discrete separate data domains to obtain embeddings for domain-specific entity representations, and these embeddings can be concatenated into a final representation vector
25 to obtain the unified entity representation.

Alignment Prediction

[0082] Learning to align is another way to achieve self-supervised learning. The training interaction data has multiple feature dimensions across time (e.g., interaction category code, interaction amount, timestamp, etc.), and their alignment can be used as a supervisory signal.

5 [0083] For example, for a given sequence of interactions, a number of misaligned versions of that sequence of interactions can be created by any one or more of (1) randomly permuting the order of interactions within a sequence of interactions; (2) replacing a particular feature dimension (e.g., timestamps) of the original sequence of interactions with the same feature dimension from another sequence of interactions in the same training batch and (3) replacing

10 a certain percentage of interactions in the original sequence of interactions with interactions from another sequence of interactions in the same batch. Thus, in Figure 5, there is an original sequence 502 of interactions 504, a first misaligned sequence 506 which includes interactions 504 from the original sequence 502 as well as misfit transactions 508 from another sequence of interactions in the same training batch, and a second misaligned sequence

15 510 in which the order of the interactions 504 from the original sequence 502 have been randomly permuted. Although only three interactions 504, 508 are shown for simplicity of illustration, there may in practice be hundreds, thousands or even millions of interactions. The sequences 502, 506, 510 of interactions 504, 508 are fed to a predictor model 512, for example (and without limitation) a transformer encoder model or an LSTM model and a

20 binary classifier 514 is used to decide which of the sequences 502, 506, 510 is correctly aligned (all interactions belong to the same entity and are also in the right order, i.e. not permuted) and which are not aligned. The output or intermediate hidden states from the predictor model 512 will be the final representation, i.e. the domain-specific entity representation. A plurality of domain-specific entity representations can be combined into the

25 single unified entity representation by training a discrete predictor model 512 and alignment classifier 514 for every discrete separate data domain and concatenating their embeddings (domain-specific entity representations) as the final representation vector (unified entity representation). Let $x = \{x_1, x_2, \dots, x_{i_1}, \dots, x_{k_1}, \dots, x_n\}$ and $y = \{y_1, y_2, \dots, y_{i_2}, \dots, y_{k_2}, \dots, y_m\}$ be two interaction sequences. For simplicity, it is assumed that each interaction has two

feature dimensions t and c . Therefore each x_i can also be represented (x_i^t, x_i^c) . Randomly permuting sequence x would create $permute(x) = \{x_{p_1}, x_{p_2}, \dots, x_{p_n}\}$ where $p_1, p_2, \dots, p_n$ is a random permutation of $1, 2, \dots, n$. Replacing a certain percentage of interactions in x with interactions in y could create a misaligned sequence like $\{x_1, x_2, \dots, y_{i_2}, \dots, y_{k_2}, \dots, x_n\}$.

5 Replacing feature dimension c of x with the same feature from y would create a sequence where the i^{th} interaction is (x_i^t, y_i^c) . In the case where $n > m$, y can be padded with artificial interactions to be the same length as x or dimensions can be replaced in sequence x up to the m^{th} interaction. The training loss would be a binary classification loss where the ground truth label for an original sequence is “True” and the label for a misaligned sequence is “False”.

10 [0084] Alternatively, cross-modal representation learning may be applied to train a single alignment model to align data from different ones of the discrete separate data domains but coming from the same entity. Let x^1, x^2 and x^3 be representations of three sequences of interactions from three discrete separate data domains for a first entity and y^1, y^2 and y^3 be representations of three sequences of interactions from three discrete separate data domains of
15 a second entity that is different from the first entity. The approach aims at maximizing the similarity score (e.g. cosine similarity) between x^i and y^j for $i = j$ and minimizing the similarity for $i \neq j$ by maximizing the cross entropy computed by $\sum_{i=1}^3 \frac{\exp(\text{sim}(x^i, y^i))}{\sum_{j=1}^3 \exp(\text{sim}(x^i, y^j))}$.

The network architecture for creating representations x^1, x^2 and x^3 could be transformer encoders, recurrent neural networks (including but not limited to LSTM) or any other suitable
20 networks appropriate for handling interaction sequence data.

[0085] BERT may also be used for alignment prediction. As shown in Figure 5A, there is an original sequence 502 of six interactions 504, and an artificial sequence 506 which includes three interactions 504 from the original sequence 502 as well as three misfit transactions 508 from another sequence of interactions in the same training batch. Although only six
25 interactions 504, 508 are shown for simplicity of illustration, there may in practice be hundreds, thousands or even millions of interactions. The sequences 502, 506 of interactions 504, 508 are fed to a BERT model 512A, which is trained to predict 516 the original sequence

502 as aligned 518 (all interactions belong to the same entity and are also in the right order, i.e. not permuted) and to predict 516 the artificial sequence 506 as not aligned 520.

[0086] Thus, one aspect of the disclosure provides for learning a representation by aligning for event sequence data.

5 Masked Auto-Encoding

[0087] Masked auto-encoding is another self-supervised learning approach that can be applied to learn an entity representation from interaction sequence data. In this approach, some interactions in a sequence, or some parts of some interactions in a sequence, are replaced with special values indicating that the original data has been masked (e.g. "MASK").

10 A model is trained to predict the original values of the masked portions in the sequence of interactions, conditioned on the remaining unmasked interaction data. As shown in Figure 6, within a sequence 602 of interactions 604A, 604B, some of the interactions 604A are unmasked and other interactions 604B are masked (in whole or in part). Although only three interactions 604A, 604B are shown for simplicity of illustration, there may in practice be

15 hundreds, thousands or even millions of interactions. The sequence 602 of interactions 604A, 604B is fed to a transformer encoder model 606, which generates a prediction 608 for the masked interaction 604B.

[0088] Specifically, let $x = \{x_1, x_2, \dots, x_{i-1}, x_i, x_{i+1}, \dots, x_n\}$ be a sequence of interactions belonging to an entity. A masked version of the sequence could be $x^{(m)} =$

20 $\{x_1, x_2, \dots, x_{i-1}, [MASK], x_{i+1}, \dots, x_n\}$, where x_i is replaced by $[MASK]$, a dedicated token indicating the transaction is masked. Figures 6 and 6A show a single interaction replaced by $[MASK]$ for purposes of illustration; in practice there may be multiple interactions masked. After pre-processing steps like positional encoding, the sequence is fed to a model producing a sequence of hidden states $\{h_1, h_2, \dots, h_n\}$. The hidden states are used to produce

25 predicted/reconstructed versions of x , $\tilde{x} = \{\tilde{x}_1, \tilde{x}_2, \dots, \tilde{x}_n\}$ and the models are trained with a reconstruction loss (e.g. L2 loss for continuous data or classification loss for categorical data) between $\tilde{x}_i$ and x_i for each x_i that is masked. The network architecture for producing $\{h_1, h_2, \dots, h_n\}$ from $x^{(m)}$ can be any model suitable for processing sequential data, including

without limitation recurrent neural networks and transformers. The model architecture for reconstructing $\tilde{x}_i$ can also be flexible, like an MLP.

[0089] An example of masked auto-encoding is the masked reconstruction task in the self-supervised learning of BERT-style generative modelling, that is, the transformer encoder 606 may be a BERT transformer encoder.

[0090] In Figure 6A, a sequence 602 includes unmasked interactions 604A and a masked interaction 604B. Only two unmasked interactions 604A and one masked interaction 604B is shown for simplicity of illustration; this is not limiting. The sequence 602 of interactions 604A, 604B is fed to a BERT transformer encoder model 606, which generates a prediction 608 for the masked interaction 604B; the prediction is a reconstruction of the original masked interaction 604B. Then, the reconstruction loss 610 between the prediction 608 and the original interaction 604A (which was masked to become the masked interaction 604B) is calculated. The entire BERT transformer encoder model 606 is trained using the reconstruction loss 610.

[0091] To obtain a domain-specific entity representation for an entity from a masked auto-encoding model, a sequence of interactions without any masking can be fed to the masked auto-encoding model, and aggregation functions, including but not limited to mean pooling, may be used to turn the hidden states of the masked auto-encoding model into a vector of fixed length. In mathematical notation, x is used directly instead of $x^{(m)}$ as input to produce hidden states $\{h_1, h_2, \dots, h_n\}$. In one embodiment, domain-specific entity representations can be combined into a single unified entity representation by training separate masked auto-encoding models for different ones of the discrete separate data domains and fusing the representation vectors (domain-specific entity representations) from the different domains into a single entity representation vector (unified entity representation). In another embodiment, the sequences of interactions from different ones of the discrete separate data domains may be concatenated into a single long sequence and a single masked auto-encoding model may be used for self-supervised learning to obtain a representation vector as the single unified entity representation.

[0092] Thus, masked auto-encoding can be used to learn an entity representation from asynchronous interaction sequence data, and this can be extended to asynchronous event sequence data from multiple discrete separate data domains.

Aggregated Statistics Prediction within Different Time Windows

5 [0093] In both supervised and unsupervised approaches, an entity's interactions over every M sequential time periods (e.g. days) can be aggregated across the observation window into a fixed-length vector for each interaction category code. These vectors can contain information such as the sum of positive and negative interactions, the maximum and minimum, the variance, and the number of interactions of a specific interaction category code across each
10 sequential set of time periods (e.g. days) in the interaction sequence.

[0094] As shown in Figure 10A, there are three sequential time periods 1002, each containing the interactions 1004 of a particular entity within the respective time period 1002. The first time window 1002 has three interactions 1004, the second time period 1002 has two
15 interactions 1004 and the third time period 1002 has four interactions 1004. This is merely illustrative, and in practice there may be hundreds, thousands or even millions of time periods, and each time period may contain hundreds, thousands or even millions of interactions. Three vectors 1006 are then produced, one for each time period 1002. The vectors 1006 contain statistics 1008 relating to the interactions 1004 during the respective time periods 1002. Thus,
20 the first vector 1006 contains statistics 1008 relating to the interactions 1004 during the first time period 1002, the second vector 1006 contains statistics 1008 relating to the interactions 1004 during the second time period 1002, and the third vector 1006 contains statistics 1008 relating to the interactions 1004 during the third time period 1002.

[0095] These vectors can then be concatenated into a fixed-length input to the model and can be interpreted using simple multi-layer perceptrons (MLPs), one-dimensional convolutional
25 neural networks (CNNs), or transformer architectures. This enables an understanding of an entity's interactions to be produced at once as opposed to iterating over interactions sequentially. Taking input from multiple discrete separate data domains can be achieved by increasing the number of input aggregation vectors passed to the network. This information

can be interpreted directly in the supervised setting for downstream predictions. In the self-supervised setting, aggregated statistics can be masked out across the vector and the model can be asked to recover the complete vector, similarly to leveraging masked auto-encoders for images (see Masked Autoencoders Are Scalable Vision Learners located at the website <https://arxiv.org/pdf/2111.06377.pdf> and which is hereby incorporated herein by reference) but modified for one dimension rather than two dimensions. Referring now to Figure 10B, during training, masked auto-encoding may be used as the self-supervised learning approach. A mask 1010 is applied to the second vector 1006, and the model 1012 is trained to generate a reconstruction (prediction) 1014 for the second vector 1006, based on the other unmasked vectors 1006 for which the statistics 1008 are available. Then, the reconstruction (prediction) 1014 is compared to the actual second vector 1006, and the reconstruction loss 1016 is used to train the model 1012.

[0096] An intermediate neural network layer output would be used as the entity's embedding. The method may be applied to each of the discrete separate data domains individually, and then the resulting embeddings may be concatenated or combined by (for example) averaging.

[0097] The above-described sequential interaction methods can also be adapted for aggregated statistical prediction. Instead of (for example) predicting the next interaction amount and interaction category code, the model would be asked to predict the aggregated statistics over the next set of M time periods (e.g. days) for each interaction category code. This would be a simpler target due to the significantly reduced variance in the target signal.

[0098] Thus, the present disclosure enables learning distributions over aggregated behaviours from multiple discrete separate data domains simultaneously, both learning *over* these distributions as input to the model and learning *from* these distributions as the target output of the model.

Supervised Learning

[0099] Supervised learning approaches can also be used to learn a unified entity representation from entity interactions with discrete separate data domains. Supervised

learning refers to the use of labeled data to train a model for specific tasks. It is possible to obtain meaningful entity representations from supervised learning models.

[00100] One such approach is fully supervised learning using a neural network. Referring now to Figure 7, in one embodiment of fully-supervised learning using multi-layer perceptrons (MLPs), each interaction 702, 704 in a sequence 706 is fed to an MLP 708 to obtain an intermediate output. Although two blocks are shown for the MLP 708 for purposes of illustration, the interactions 702, 704 are fed to the same MLP 708. The intermediate outputs for the interactions are aggregated 710, for example by mean pooling. The aggregated state 710 is used to make predictions 712 for a specific task and the training loss 714 can be computed based on the predictions.

[00101] In the fully supervised learning approach, a sequence of interactions is taken as an input and used to directly predict the label for a downstream task. An intermediate output of the neural network is then used as the entity's embedding. More concretely, let $y = f(x) = g(h(x))$ represent a network $f()$ that takes interaction sequence x as input and produces an output y for training where $h()$ represent the part of f up to the intermediate layer, the output of which is used to produce embeddings and g comprises the layers following $h()$. Thus, $h(x)$ is used to produce the entity's embeddings. If the output of $h(x)$ is a sequence, a suitable aggregation method, for example mean pooling or summing, can be used to produce a vector of fixed length. For example, a vector embedding from the penultimate layer of the network can then be used as the entity representation for the downstream task, for that particular data domain. Embeddings generated from the trained model can then be reused for the downstream task alongside the base features of the downstream task. The network architecture and training method will depend on the supervised learning task itself, and selection of a suitable architecture and training method is within the capability of one of ordinary skill in the art, now informed by the present disclosure.

[00102] To learn on multiple training event sequences for discrete separate data domains (interaction sequences for the same entity across discrete separate data domains), the supervised interaction model can be extended by using multiple "interaction blocks". An

“interaction block” may be any sub-network that takes in an interaction sequence, and outputs some vector. The vectors from multiple interaction blocks, each taking in a separate interaction sequence, can then be concatenated together to form the entity embedding, which serves as the unified entity representation. This can then be passed onto another prediction block that produces the final prediction output.

[00103] The choice of model for the interaction block is flexible. For example, but without limitation, any of a multi-layer perceptron, LSTM, or transformer encoder may be used. Once the unified entity representation is obtained from the interaction block, it can be passed through (e.g.) a multi-layer perceptron to get the final prediction for the downstream task.

[00104] Unlike traditional task-specific supervised learning which typically only uses a model’s final outputs, this approach takes the intermediate results from the model as representations. The supervised approach also makes representation learning simpler by directly using the downstream task to learn the representation based on an interaction sequence.

Tenure

[00105] In some embodiments, a domain-specific entity representation can be learned by training the representation engine, R , to predict the amount of time the entity has been interacting with the specific data domain, i.e. the entity's tenure, based on recent interactions over a specific time frame. Here the representation engine R will have both an encoder, E , which produces the domain-specific entity representation, and a decoder, D , which predicts a tenure value from the domain-specific entity representation. In one embodiment, the decoder model is a simple multi-layer perceptron. To train the representation engine R , a time frame over which to observe interactions, W , is chosen, and random entities are sampled from the dataset. W is a hyperparameter. For each sampled entity, e_i , a random time period of length W is sampled, $[T_i, T_i + W]$, and the time between the beginning of the entity's interaction within the domain and the end of the sampled time period is recorded as the ground truth tenure, $Tenure_{GT} = T_i + W$. All interactions between the entity and the domain in this

observation window are then passed through the representation engine and a tenure prediction is made:

$$Tenure_{pt} = D(E(e_i)[T_i, T_i + W])$$

[00106] The distance metric, L2, is then applied to compute the difference between the predicted and ground truth value. The difference is used as a loss function to update the representation engine:

$$Loss = (Tenure_{pt} - Tenure_{GT})^2$$

[00107] Figure 11A shows an illustrative tenure network during training. An entity is randomly selected. The selected entity has a sequence 1102 of individual interactions 1104. A random subsequence 1106 is sampled from the sequence 1102. The subsequence 1106 has a start time T_0 and an end time T_1 . The subsequence 1106 is passed to an encoder 1108, which produces a representation 1110 of the selected entity at time T_1 . This representation 1110 is passed through a decoder 1112 to produce a prediction 1114 of the tenure of the selected entity at time T_1 (i.e. $T_1 - T_0$). Both the prediction 1114 and the true tenure 1116 are passed to a distance metric 1118, and the resultant loss is used to update the parameters of the representation engine R .

[00108] When predicting an entity's current representation, all interactions over the most recent time period K observed for this entity are passed through the encoder E to produce the domain-specific entity representation:

$$Representation = (e_c[T_n - W, T_n])$$

where T_n is the current time, and e_c is the current entity.

[00109] As shown in Figure 11B, at inference time a subsequence 1120 representing the most recent time period K observed for the entity is extracted from the sequence 1102 of interactions 1104. The subsequence 1120 is passed to the encoder 1108, which produces a representation 1122 of the selected entity at the current time, that is, a domain-specific entity

representation. This type of learning is an example of auxiliary task learning which is described in Liebel et al. *Auxiliary Tasks in Multi-task Learning* available at <https://arxiv.org/pdf/1805.06334> and incorporated herein by reference.

[00110] In some embodiments, a 1-dimensional convolutional architecture, e.g. a 1-dimensional convolutional neural network (1D CNN) may be used for the self-supervised learning. Thus, a 1D CNN may be used for the encoder network E of the representation engine R . The architecture for a 1D CNN is known to those of ordinary skill in the art, and is described in Kiranyaz et al., *1D Convolutional Neural Networks and Applications – A Survey* available at <https://arxiv.org/pdf/1905.03554> and incorporated herein by reference. Stated briefly, a 1D CNN operates by sequentially applying multiple 1-dimensional convolution filters with chosen kernel sizes, strides, channel dimensions and padding to the input observation sequence for a given entity. The main caveat to this approach, as applied to time series input sequences, is that the sequence length can change depending on the entity. However, the output channel will be fixed. The domain-specific entity representation is defined as the mean value across the sequence for each channel where, for each i in the number of output channels:

$$Representation[i] = mean(E(e)[: , i])$$

where e is the current entity being considered.

[00111] Figure 11C shows the use of an illustrative a 1D CNN as the encoder network E of the representation engine R . The selected entity has a sequence 1102 of individual interactions 1104. A random subsequence 1106 is extracted from the sequence 1102, with a start time T_0 and an end time T_1 . The subsequence 1106 is defined as a feature vector 1124 and passed through a series of 1D convolutional layers 1126 with chosen kernel sizes, strides, channel dimensions and padding. The kernel and channel sizes shown in Figure 11C are merely illustrative and not limiting. The result is then a series of C feature vectors 1128, where C is the number of output channels. Mean pooling 1130 is applied across the sequence, and the domain-specific entity representation 1132 is defined as the mean value across the sequence for each channel.

[00112] In other embodiments, the encoder network E may leverage a graph convolutional network architecture. This architecture for a graph convolutional network is known to those of ordinary skill in the art, and is described in Kipf et al., *Semi-Supervised Classification with Graph Convolutional Networks*, which is available at <https://arxiv.org/pdf/1609.02907> and is incorporated herein by reference. A graph convolutional network operates by sequentially applying graph convolutional layers with varying hidden features sizes at the nodes, and skip connections between layers. To convert the observation sequence into a graph, each observation is considered as a node and its features defined as the observed data. Edges between nodes are constructed based on temporal relationships between observations. This can be that a node shares an edge with its G_W closest observations in time, or with all observations which took place within a threshold T time of its own occurrence, where both T and G_W are hyperparameters. The output of the network will be a graph with the same number of nodes as at input, and a fixed size feature vector at each node. The representation is defined as the mean value at each index of the feature vectors, across all nodes, where for each index i in the nodes' feature vector:

$$Representation[i] = mean(E(e)[: , i])$$

where R is the representation, and e is the current entity considered, and $E(e)$ is the set of output feature vectors across all nodes.

[00113] Figure 11D shows the use of an illustrative graph convolutional network architecture as the encoder network E of the representation engine R . The selected entity has a sequence 1102 of individual interactions 1104. A random subsequence 1106 is extracted from the sequence 1102, with a start time T_0 and an end time T_1 . The subsequence 1106 is defined as a set of interaction nodes (graph nodes) 1134 with their features defined as the observed data. Temporal distance analysis 1136 is applied, and edges between the interaction nodes (graph nodes) 1134 are constructed based on temporal relationships between the observations to obtain an interaction graph 1138. The interaction graph 1138 is then passed through a graph convolutional network 1140 (a sequence layers with chosen hidden dimension size, and residual connections). The output of the graph convolutional network 1140 will be a graph with the same number of nodes as at input, and a fixed size feature

vector at each node. The domain-specific entity representation 1142 is defined as the mean value at each index of the feature vectors, across all nodes.

[00114] For the embodiments shown in Figures 11A to 11D, the domain-specific representation is the output of the encoder, i.e. the encoder 1108 for the tenure
5 implementation, the 1D CNN used as the encoder network E of the representation engine R , as shown in Figure 11C, and the graph convolutional network used as the encoder network E of the representation engine R , as shown in Figure 1D. Any combination of these domain-specific representations may then be (for example) concatenated together to obtain the single unified entity representation.

10 [00115] The above description and examples have been simplified for the purposes of illustration, and such simplification is not intended to be limiting. One of ordinary skill in the art will understand that the mathematical notation and variable names are limited to the specific mathematical technique in respect of which they are described, and may have other meanings in the context of other mathematical techniques described herein.

15 [00116] As can be seen from the above description, applying representation learning to time-ordered sequences of interactions within a plurality of discrete separate data domains as described herein represents significantly more than merely using categories to organize, store and transmit information and organizing information through mathematical correlations. The application of representation learning to time-ordered sequences of interactions within a
20 plurality of discrete separate data domains is in fact an improvement to the technology of machine learning, as it provides for the creation of a single unified entity representation characterizing a distribution of the entity's interactions within the discrete separate data domains. Moreover, although the technology can be applied across a wide range of contexts, the particular context in which the technology is applied does not change the fundamental
25 nature of the technology described and claimed herein, which is entirely confined to machine learning applications.

[00117] The present technology may be embodied within a system, a method, a computer program product or any combination thereof. The computer program product may

include a computer readable storage medium or media having computer readable program instructions thereon for causing a processor to carry out aspects of the present technology. The computer readable storage medium can be a tangible device that can retain and store instructions for use by an instruction execution device. The computer readable storage medium may be, for example, but is not limited to, an electronic storage device, a magnetic storage device, an optical storage device, an electromagnetic storage device, a semiconductor storage device, or any suitable combination of the foregoing.

[00118] A non-exhaustive list of more specific examples of the computer readable storage medium includes the following: a portable computer diskette, a hard disk, a random access memory (RAM), a read-only memory (ROM), an erasable programmable read-only memory (EPROM or Flash memory), a static random access memory (SRAM), a portable compact disc read-only memory (CD-ROM), a digital versatile disk (DVD), a memory stick, a floppy disk, a mechanically encoded device such as punch-cards or raised structures in a groove having instructions recorded thereon, and any suitable combination of the foregoing.

A computer readable storage medium, as used herein, is not to be construed as being transitory signals per se, such as radio waves or other freely propagating electromagnetic waves, electromagnetic waves propagating through a waveguide or other transmission media (e.g., light pulses passing through a fiber-optic cable), or electrical signals transmitted through a wire.

[00119] Computer readable program instructions described herein can be downloaded to respective computing/processing devices from a computer readable storage medium or to an external computer or external storage device via a network, for example, the Internet, a local area network, a wide area network and/or a wireless network. The network may comprise copper transmission cables, optical transmission fibers, wireless transmission, routers, firewalls, switches, gateway computers and/or edge servers. A network adapter card or network interface in each computing/processing device receives computer readable program instructions from the network and forwards the computer readable program instructions for storage in a computer readable storage medium within the respective computing/processing device.

[00120] Computer readable program instructions for carrying out operations of the present technology may be assembler instructions, instruction-set-architecture (ISA) instructions, machine instructions, machine dependent instructions, microcode, firmware instructions, state-setting data, or either source code or object code written in any combination of one or more programming languages, including an object oriented programming language or a conventional procedural programming language. The computer readable program instructions may execute entirely on the user's computer, partly on the user's computer, as a stand-alone software package, partly on the user's computer and partly on a remote computer or entirely on the remote computer or server. In the latter scenario, the remote computer may be connected to the user's computer through any type of network, including a local area network (LAN) or a wide area network (WAN), or the connection may be made to an external computer (for example, through the Internet using an Internet Service Provider). In some embodiments, electronic circuitry including, for example, programmable logic circuitry, field-programmable gate arrays (FPGA), or programmable logic arrays (PLA) may execute the computer readable program instructions by utilizing state information of the computer readable program instructions to personalize the electronic circuitry, in order to implement aspects of the present technology.

[00121] Aspects of the present technology have been described above with reference to flowchart illustrations and/or block diagrams of methods, apparatus (systems) and computer program products according to various embodiments. In this regard, the flowchart and block diagrams in the Figures illustrate the architecture, functionality, and operation of possible implementations of systems, methods and computer program products according to various embodiments of the present technology. For instance, each block in the flowchart or block diagrams may represent a module, segment, or portion of instructions, which comprises one or more executable instructions for implementing the specified logical function(s). It should also be noted that, in some alternative implementations, the functions noted in the block may occur out of the order noted in the Figures. For example, two blocks shown in succession may, in fact, be executed substantially concurrently, or the blocks may sometimes be executed in the reverse order, depending upon the functionality involved. Some specific examples of the foregoing may have been noted above but any such noted examples are not necessarily the

only such examples. It will also be noted that each block of the block diagrams and/or flowchart illustration, and combinations of blocks in the block diagrams and/or flowchart illustration, can be implemented by special purpose hardware-based systems that perform the specified functions or acts, or combinations of special purpose hardware and computer instructions.

[00122] It also will be understood that each block of the flowchart illustrations and/or block diagrams, and combinations of blocks in the flowchart illustrations and/or block diagrams, can be implemented by computer program instructions. These computer readable program instructions may be provided to a processor of a general purpose computer, special purpose computer, or other programmable data processing apparatus to produce a machine, such that the instructions, which execute via the processor of the computer or other programmable data processing apparatus, create means for implementing the functions/acts specified in the flowchart and/or block diagram block or blocks.

[00123] These computer readable program instructions may also be stored in a computer readable storage medium that can direct a computer, other programmable data processing apparatus, or other devices to function in a particular manner, such that the instructions stored in the computer readable storage medium produce an article of manufacture including instructions which implement aspects of the functions/acts specified in the flowchart and/or block diagram block or blocks. The computer readable program instructions may also be loaded onto a computer, other programmable data processing apparatus, or other devices to cause a series of operational steps to be performed on the computer, other programmable apparatus or other devices to produce a computer implemented process such that the instructions which execute on the computer or other programmable apparatus provide processes for implementing the functions/acts specified in the flowchart and/or block diagram block or blocks.

[00124] An illustrative computer system in respect of which the technology herein described may be implemented is presented as a block diagram in Figure 8. The illustrative computer system is denoted generally by reference numeral 800 and includes a display 802, input devices in the form of keyboard 804A and pointing device 804B, computer 806 and

external devices 808. While pointing device 804B is depicted as a mouse, it will be appreciated that other types of pointing device, or a touch screen, may also be used.

[00125] The computer 806 may contain one or more processors or microprocessors, such as a central processing unit (CPU) 810. The CPU 810 performs arithmetic calculations and control functions to execute software stored in an internal memory 812, preferably random access memory (RAM) and/or read only memory (ROM), and possibly additional memory 814. The additional memory 814 may include, for example, mass memory storage, hard disk drives, optical disk drives (including CD and DVD drives), magnetic disk drives, magnetic tape drives (including LTO, DLT, DAT and DCC), flash drives, program cartridges and cartridge interfaces such as those found in video game devices, removable memory chips such as EPROM or PROM, emerging storage media, such as holographic storage, or similar storage media as known in the art. This additional memory 814 may be physically internal to the computer 806, or external as shown in Figure 8, or both.

[00126] The computer system 800 may also include other similar means for allowing computer programs or other instructions to be loaded. Such means can include, for example, a communications interface 816 which allows software and data to be transferred between the computer system 800 and external systems and networks. Examples of communications interface 816 can include a modem, a network interface such as an Ethernet card, a wireless communication interface, or a serial or parallel communications port. Software and data transferred via communications interface 816 are in the form of signals which can be electronic, acoustic, electromagnetic, optical or other signals capable of being received by communications interface 816. Multiple interfaces, of course, can be provided on a single computer system 800.

[00127] Input and output to and from the computer 806 is administered by the input/output (I/O) interface 818. This I/O interface 818 administers control of the display 802, keyboard 804A, external devices 808 and other such components of the computer system 800. The computer 806 also includes a graphical processing unit (GPU) 820. The latter may also be used for computational purposes as an adjunct to, or instead of, the (CPU) 810, for mathematical calculations.

[00128] The external devices 808 include a microphone 826, a speaker 828 and a camera 830. Although shown as external devices, they may alternatively be built in as part of the hardware of the computer system 800.

5 [00129] The various components of the computer system 800 are coupled to one another either directly or by coupling to suitable buses.

[00130] The term “computer system”, “data processing system” and related terms, as used herein, is not limited to any particular type of computer system and encompasses servers, desktop computers, laptop computers, networked mobile wireless telecommunication computing devices such as smartphones, tablet computers, as well as other types of computer systems.

10

[00131] Thus, computer readable program code for implementing aspects of the technology described herein may be contained or stored in the memory 812 of the computer 806, or on a computer usable or computer readable medium external to the computer 806, or on any combination thereof.

15 [00132] Finally, the terminology used herein is for the purpose of describing particular embodiments only and is not intended to be limiting. As used herein, the singular forms “a”, “an” and “the” are intended to include the plural forms as well, unless the context clearly indicates otherwise. It will be further understood that the terms “comprises” and/or “comprising,” when used in this specification, specify the presence of stated features, integers, steps, operations, elements, and/or components, but do not preclude the presence or addition of one or more other features, integers, steps, operations, elements, components, and/or groups thereof.

20

[00133] The corresponding structures, materials, acts, and equivalents of all means or step plus function elements in the claims below are intended to include any structure, material, or act for performing the function in combination with other claimed elements as specifically claimed. The description has been presented for purposes of illustration and description, but is not intended to be exhaustive or limited to the form disclosed. Many modifications and variations will be apparent to those of ordinary skill in the art without

25

departing from the scope of the claims. The embodiment was chosen and described in order to best explain the principles of the technology and the practical application, and to enable others of ordinary skill in the art to understand the technology for various embodiments with various modifications as are suited to the particular use contemplated.

5 [00134] One or more currently preferred embodiments have been described by way of example. It will be apparent to persons skilled in the art that a number of variations and modifications can be made without departing from the scope of the claims. In construing the claims, it is to be understood that the use of a computer to implement the embodiments described herein is essential.

10 [00135] The following list of references is provided for convenience only, and no limitation is implied. None of the references are admitted to be relevant to the invention as claimed and none of the references are admitted to be citable as prior art.

Bae et al., *Meta Temporal Point Processes* available at <https://arxiv.org/pdf/2301.12023.pdf>.

15 Chen et al., *A Simple Framework for Contrastive Learning of Visual Representations* available at <https://arxiv.org/pdf/2002.05709>.

Devlin et al., *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* available at <https://arxiv.org/abs/1810.04805>.

He et al., *Masked Autoencoders Are Scalable Vision Learners* located at the website <https://arxiv.org/pdf/2111.06377.pdf>,

20 Kiranyaz et al., *1D Convolutional Neural Networks and Applications – A Survey* available at <https://arxiv.org/pdf/1905.03554>.

Kipf et al., *Semi-Supervised Classification with Graph Convolutional Networks*, which is available at <https://arxiv.org/pdf/1609.02907>.

25 Liebel et al. *Auxiliary Tasks in Multi-task Learning* available at <https://arxiv.org/pdf/1805.06334>.

Radford et al., *Learning Transferable Visual Models from Natural Language Supervision* available at <https://arxiv.org/pdf/2103.00020v1>.

Wei et al., *Diffusion Models as Masked Autoencoders* available at <https://arxiv.org/abs/2304.03283>.

WHAT IS CLAIMED IS:

1. A computer-implemented method for constructing an entity representation engine, the method comprising:

receiving time series training data representing a plurality of discrete training event
5 sequences, wherein each training event sequence is associated with a respective one of a plurality of discrete separate data domains and the training event sequences are time-ordered sequences of sample interactions within the respective discrete separate data domains;

applying representation learning to the time series training data to generate a trained representation engine, wherein the trained representation engine is adapted to:

10 ingest entity time series data representing a plurality of discrete entity event sequences for a particular entity, wherein each entity event sequence is associated with a respective one of the discrete separate data domains and the entity event sequences are time-ordered sequences of entity interactions for that entity within the respective discrete separate data domains; and

15 from the entity time series data, generate a corresponding single unified entity representation characterizing a distribution of the entity interactions within the discrete separate data domains.

2. The method of claim 1, further comprising:

20 using labeled unified entity representation instances to train a downstream prediction model to predict at least one entity characteristic from an unlabeled unified entity representation instance.

3. The method of claim 2, wherein:

the at least one entity characteristic is a plurality of entity characteristics;

at least some of the labeled unified entity representation instances are only partially labeled for only a subset of the entity characteristics;

5 using the labeled unified entity representation instances to train the downstream prediction model to predict the plurality of entity characteristics from the unlabeled unified entity representation instance comprises:

providing different ones of the labeled unified entity representation instances that are only partially labelled, wherein the different ones of the labeled unified entity representation
10 instances have respectively different subsets of the entity characteristics that are labelled.

4. The method of claim 3, further comprising providing other different ones of the labeled unified entity representation instances that are fully labelled.

15 5. The method of claim 2, wherein the entity characteristic is a characteristic of the entity in interacting within one of the distinct separate data domains.

6. The method of claim 2, wherein the entity characteristic comprises an aggregate statistic of the entity interactions.

20

7. The method of claim 1, wherein the trained representation engine is adapted to generate the single unified entity representation by combining a plurality of domain-specific entity representations into the single unified entity representation.

8. The method of claim 7, wherein the trained representation engine is adapted to combine the plurality of domain-specific entity representations into the single unified entity representation by concatenation.
- 5 9. The method of claim 7, wherein the trained representation engine is adapted to combine the plurality of domain-specific entity representations into the single unified entity representation by aggregation.
- 10 10. The method of claim 9, wherein the aggregation comprises one of summing and averaging.
11. The method of claim 1, wherein the time series data is asynchronous time series data.
- 15 12. The method of claim 1, wherein applying representation learning to the time series training data comprises applying self-supervised learning.
13. The method of claim 12, wherein the self-supervised learning utilizes a recurrent neural network (RNN).
- 20 14. The method of claim 12, wherein the self-supervised learning utilizes a temporal point process (TPP) model.

15. The method of claim 14, wherein the TPP model is a Meta TPP model.
16. The method of claim 12, wherein the self-supervised learning utilizes at least one of a conditional diffusion model, bidirectional encoder representations from transformers (BERTs), contrastive learning, alignment prediction, masked auto-encoding, aggregated prediction within different time windows, tenure, 1-dimensional convolutional architecture, and graph convolutional architecture.
17. The method of claim 1, wherein applying representation learning to the time series training data comprises applying supervised learning.
18. The method of claim 17, wherein the supervised learning is fully supervised learning.
19. The method of claim 17, wherein the supervised learning utilizes at least one of 1-dimensional convolutional architecture and graph convolutional architecture.
20. A data processing system comprising at least one processor and memory coupled to the at least one processor wherein the memory contains instructions which, when executed by the at least one processor, cause the at least one processor to carry out a method comprising:
- receiving time series training data representing a plurality of discrete training event sequences, wherein each training event sequence is associated with a respective one of a plurality of discrete separate data domains and the training event sequences are time-ordered sequences of sample interactions within the respective discrete separate data domains;

applying representation learning to the time series training data to generate a trained representation engine, wherein the trained representation engine is adapted to:

5 ingest entity time series data representing a plurality of discrete entity event sequences for a particular entity, wherein each entity event sequence is associated with a respective one of the discrete separate data domains and the entity event sequences are time-ordered sequences of entity interactions for that entity within the respective discrete separate data domains; and

10 from the entity time series data, generate a corresponding single unified entity representation characterizing a distribution of the entity interactions within the discrete separate data domains.

21. At least one tangible, non-transitory computer-readable medium embodying instructions which, when executed by the at least one processor, cause the at least one processor to carry out a method comprising:

15 receiving time series training data representing a plurality of discrete training event sequences, wherein each training event sequence is associated with a respective one of a plurality of discrete separate data domains and the training event sequences are time-ordered sequences of sample interactions within the respective discrete separate data domains;

20 applying representation learning to the time series training data to generate a trained representation engine, wherein the trained representation engine is adapted to:

25 ingest entity time series data representing a plurality of discrete entity event sequences for a particular entity, wherein each entity event sequence is associated with a respective one of the discrete separate data domains and the entity event sequences are time-ordered sequences of entity interactions for that entity within the respective discrete separate data domains; and

from the entity time series data, generate a corresponding single unified entity representation characterizing a distribution of the entity interactions within the discrete separate data domains.