Urban Computing

Dr. Mitra Baratchi

Leiden Institute of Advanced Computer Science - Leiden University

30 March 2020

Sixth Session: Urban Computing - Machine learning 2

Agenda for this session

- Part 1: Intro
  - Fundamentals of deep learning
- Part 2: Capturing spatial patterns (Convolutional neural networks)
  - Example: Crowd flow modeling using CNN
- Part 3: Capturing temporal patterns (Recurrent neural networks)
  - RNN and LSTM
  - Example: Trajectory modeling using LSTM
- Part 4: Representation learning
  - Embeddings
  - LINE embedding
  - Example: Spatio-temporal region embeddings
- Part 5: Transfer learning
  - Example: Cross-city transfer learning

Part 1: Intro

What is going on in Urban Computing research?

How is the Urban Computing research evolving?

What is going on in Urban Computing research?

How is the Urban Computing research evolving?

- Spatial, time-series, spatio-temporal statistics (auto-correlation function dates back to 1920s)

What is going on in Urban Computing research?

How is the Urban Computing research evolving?

- Spatial, time-series, spatio-temporal statistics (auto-correlation function dates back to 1920s)
- Pattern mining and machine learning algorithms (2007-2017) (Mobile phones, GPS sensors)

What is going on in Urban Computing research?

How is the Urban Computing research evolving?

- Spatial, time-series, spatio-temporal statistics (auto-correlation function dates back to 1920s)
- Pattern mining and machine learning algorithms (2007-2017) (Mobile phones, GPS sensors)
- Deep learning algorithms (2017-?)

Why is there an interest to use it for spatio-temporal data

- Performance in various data analysis tasks for unstructured data (image, sequential, graph)
  - **Spatio-temporal data is unstructured**
- Feature extraction from raw data instead of hand-crafted feature engineering
  - **Spatio-temporal data is high-dimensional and featureless**
- New solutions for handing unlabeled data
  - **Spatio-temporal is difficult to label**
- Learning features over data from multiple modalities
  - **Data collected from heterogeneous sensors and data sources**

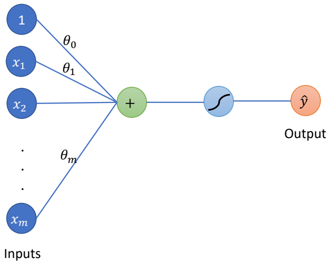
Why is there an interest to use it for spatio-temporal data

- Performance in various data analysis tasks for unstructured data (image, sequential, graph)
  - **Spatio-temporal data is unstructured**
- Feature extraction from raw data instead of hand-crafted feature engineering
  - **Spatio-temporal data is high-dimensional and featureless**
- New solutions for handing unlabeled data
  - **Spatio-temporal is difficult to label**
- Learning features over data from multiple modalities
  - **Data collected from heterogeneous sensors and data sources**

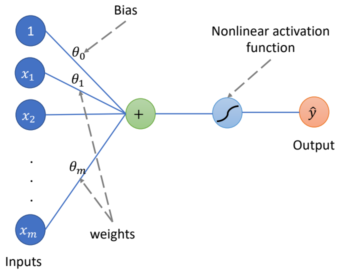
At the same time they are black box algorithms (Big limitation)

A perceptron (neuron)

The building block of neural networks



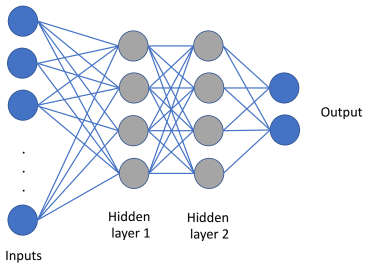
A perceptron (neuron)



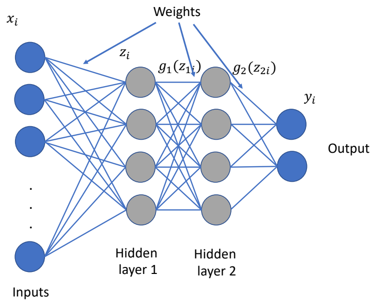
$$\hat{y} = g(\theta_0 + \sum_{i=1}^{m} \theta_i x_i)$$

A neural network is created by repeating this simple pattern

Neural networks with multiple hidden layers

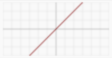
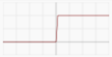
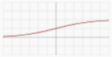
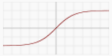
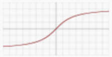
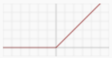
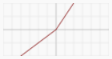
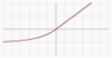
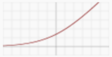


Neural networks with multiple hidden layers



Where is the power coming from?

- **Embedding non-linearity:** Through introducing nonlinearity we are able to find any form of real-world nonlinear pattern
- The activation function allows embedding non-linearity
- Examples
  - Sigmoid $g(z) = \sigma(z) = \frac{1}{1+e^{(-z)}}$
  - Relu
  - Hyperbolic tangent
  - Sigmoid function

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU)[2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU)[3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$

[1]

[1] Image source: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

Objective function

The goal is finding a network that minimizes loss on an objective function

- Find a set of parameters that help us minimize the loss
  - $\theta^* = argmin_\theta \frac{1}{n} \sum_{i=1}^{n} L(f(x^i)|\theta), y^i)$

Loss optimization

- **Gradient descent:**
  - Considers how the loss is changing with respect to each weight $\rightarrow$ gradient
- **Back-propagation:**
  - Calculates a gradient that is needed in the calculation of the weights to be used in the network
- **Batch gradient descent:**
  - Gradient descent in mini-batches
  - Allows parallelizing the work

Different types of neural networks

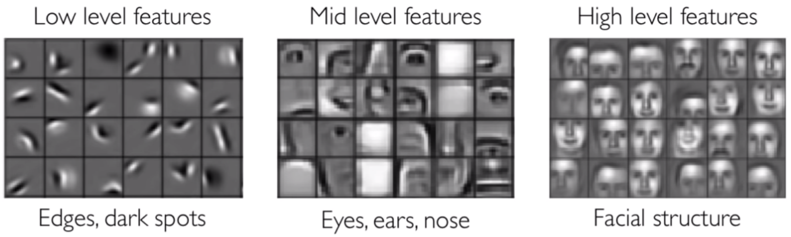
- Multilayer perceptron
- Convolutional neural networks
- Recurrent neural networks
- Auto-encoders
- Generative adversarial networks

Part 2: Capturing spatial patterns (Convolutional neural networks)

Convolutional neural networks

- Originally made for image data represented in 3D matrices
- Manual feature extraction used previously in image classification considers:
  - Manually designing features to detect edges, shapes, textures, etc.
  - Dealing with problems such as (lighting, rotation, etc)
- Convolutional neural networks allow extraction of these features hierarchically

Hierarchical feature extraction with convolutional neural networks

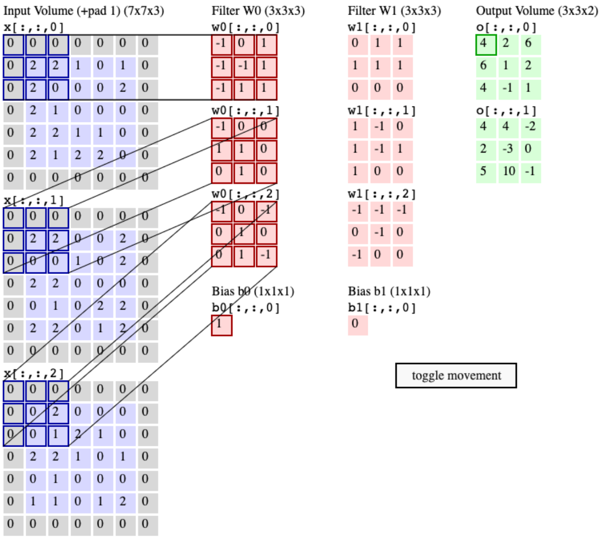


[2]

[2] Image source: [LGRN11]

Convolution

- Convolution layer is the main building block of a convolutional neural network
- The convolution layer is composed of independent filters that are convolved with data



Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	2	2	1	0	1	0
0	2	0	0	0	2	0
0	2	1	0	0	0	0
0	2	2	1	1	0	0
0	2	1	2	2	0	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	2	2	0	0	2	0
0	0	0	1	0	2	0
0	2	2	0	0	0	0
0	0	1	0	2	2	0
0	2	2	0	1	2	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	0	2	0	0	0	0
0	0	1	2	1	0	0
0	2	2	1	0	1	0
0	0	1	0	0	0	0
0	1	1	0	1	2	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

-1	0	1
-1	-1	1
-1	1	1

w0[:,:,1]

-1	0	0
1	1	0
0	1	0

w0[:,:,2]

-1	0	-1
0	1	0
0	1	-1

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

0	1	1
1	1	1
0	0	0

w1[:,:,1]

1	-1	0
1	-1	1
1	0	0

w1[:,:,2]

-1	-1	-1
0	-1	0
-1	0	0

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

o[:,:,0]

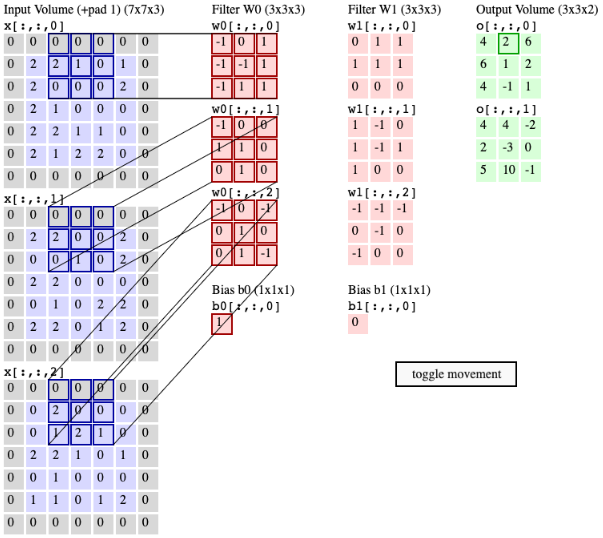
4	2	6
6	1	2
4	-1	1

o[:,:,1]

4	4	-2
2	-3	0
5	10	-1

3

[3] source: https://cs231n.github.io/convolutional-networks/



Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	2	2	1	0	1	0
0	2	0	0	0	2	0
0	2	1	0	0	0	0
0	2	2	1	1	0	0
0	2	1	2	2	0	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	2	2	0	0	2	0
0	0	0	1	0	2	0
0	2	2	0	0	0	0
0	0	1	0	2	2	0
0	2	2	0	1	2	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	0	2	0	0	0	0
0	0	1	2	1	0	0
0	2	2	1	0	1	0
0	0	1	0	0	0	0
0	1	1	0	1	2	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

-1	0	1
-1	-1	1
-1	1	1

w0[:,:,1]

-1	0	0
1	1	0
0	1	0

w0[:,:,2]

-1	0	-1
0	1	0
0	1	-1

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

0	1	1
1	1	1
0	0	0

w1[:,:,1]

1	-1	0
1	-1	1
1	0	0

w1[:,:,2]

-1	-1	-1
0	-1	0
-1	0	0

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

o[:,:,0]

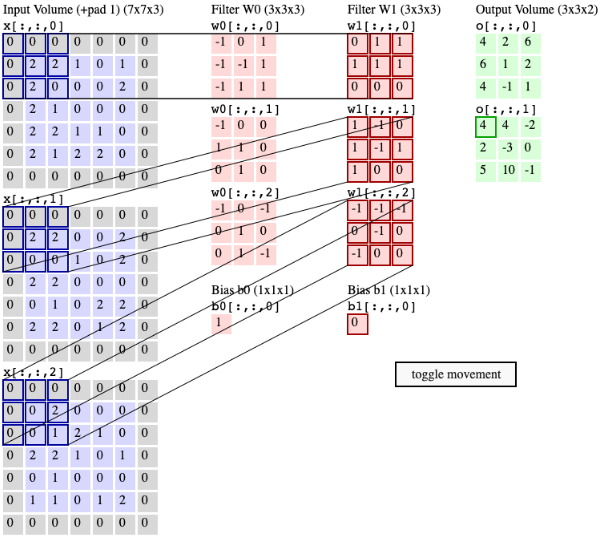
4	2	6
6	1	2
4	-1	1

o[:,:,1]

4	4	-2
2	-3	0
5	10	-1

[4]

[4] source: https://cs231n.github.io/convolutional-networks/


Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	2	2	1	0	1	0
0	2	0	0	0	2	0
0	2	1	0	0	0	0
0	2	2	1	1	0	0
0	2	1	2	2	0	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	2	2	0	0	2	0
0	0	0	1	0	2	0
0	2	2	0	0	0	0
0	0	1	0	2	2	0
0	2	2	0	1	2	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	0	2	0	0	0	0
0	0	1	2	1	0	0
0	2	2	1	0	1	0
0	0	1	0	0	0	0
0	1	1	0	1	2	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

-1	0	1
-1	-1	1
-1	1	1

w0[:,:,1]

-1	0	0
1	1	0
0	1	0

w0[:,:,2]

-1	0	-1
0	1	0
0	1	-1

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

0	1	1
1	1	1
0	0	0

w1[:,:,1]

1	-1	0
1	-1	1
1	0	0

w1[:,:,2]

-1	-1	-1
0	-1	0
-1	0	0

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

o[:,:,0]

4	2	6
6	1	2
4	-1	1

o[:,:,1]

4	4	-2
2	-3	0
5	10	-1

5

[5] source: https://cs231n.github.io/convolutional-networks/

Convolution

Convolution operation allows learning features in small pixel regions

- Filters are defined based on weights to detect local patterns
- Many filters are used to extract different patterns

General architecture

- The goal is learning the weights on the filters from data
  - Convolution: Applying filters
  - Nonlinearity: Activation function
  - Pooling: Reduce the size of the feature map
  - Fully connected layer: in classification settings it allows to calculate the class scores

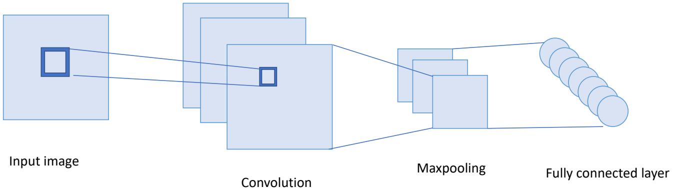


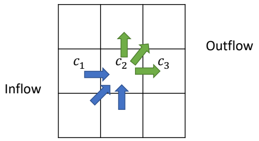
Figure: Feature learning and classification pipeline

Example: using CNNs for modeling spatial dependencies

Problem

Forecasting the crowd flows using mobility trajectories

- Inflow
- Outflow



- Given a tensor $\{X_i | t \in [1, n-1]\}$, $X \in \mathbb{R}^{2 \times I \times J}$ showing the inflow and outflow to cells of a grid of size $I \times J$
- We are interested in Forecasting the flow of crowds in X_n

Things that we need to model

Things that we need to model

- **Spatial dependencies:** The inflow of a region is affected by outflows of nearby regions as well as distant regions.

Things that we need to model

- **Spatial dependencies:** The inflow of a region is affected by outflows of nearby regions as well as distant regions.
- **Temporal dependencies:** (near and far)
  - **Near past:** A traffic congestion occurring at 8am will affect that of 9am.
  - **Periodicity:** Traffic conditions during morning rush hours may be similar on consecutive workdays, repeating every 24 hours
  - **Trend:** Morning rush hours may gradually happen later as winter comes. When the temperature gradually drops and the sun rises later in the day, people get up later and later.

Things that we need to model

- **Spatial dependencies:** The inflow of a region is affected by outflows of nearby regions as well as distant regions.
- **Temporal dependencies:** (near and far)
  - **Near past:** A traffic congestion occurring at 8am will affect that of 9am.
  - **Periodicity:** Traffic conditions during morning rush hours may be similar on consecutive workdays, repeating every 24 hours
  - **Trend:** Morning rush hours may gradually happen later as winter comes. When the temperature gradually drops and the sun rises later in the day, people get up later and later.
- **External influence.** e.g. Weather conditions, events

Things that we need to model

- **Spatial dependencies:** The inflow of a region is affected by outflows of nearby regions as well as distant regions.
- **Temporal dependencies:** (near and far)
  - **Near past:** A traffic congestion occurring at 8am will affect that of 9am.
  - **Periodicity:** Traffic conditions during morning rush hours may be similar on consecutive workdays, repeating every 24 hours
  - **Trend:** Morning rush hours may gradually happen later as winter comes. When the temperature gradually drops and the sun rises later in the day, people get up later and later.
- **External influence.** e.g. Weather conditions, events

What solutions did we learn before so far to address these?

Things that we need to model

- **Spatial dependencies:** The inflow of a region is affected by outflows of nearby regions as well as distant regions.
- **Temporal dependencies:** (near and far)
  - **Near past:** A traffic congestion occurring at 8am will affect that of 9am.
  - **Periodicity:** Traffic conditions during morning rush hours may be similar on consecutive workdays, repeating every 24 hours
  - **Trend:** Morning rush hours may gradually happen later as winter comes. When the temperature gradually drops and the sun rises later in the day, people get up later and later.
- **External influence.** e.g. Weather conditions, events

What solutions did we learn before so far to address these? (Spatial weight matrices, ARIMA, SARIMA, Autoregressive models....)

ST-ResNet uses residual networks to model these properties [ZZQ17]

How convolution can help?

- A city usually has many regions with different distances
- **Spatial correlation in nearby regions:** The flow of crowds in nearby regions may affect each other, which can be effectively handled by the convolutional neural network
- **Spatial correlation in distant regions:** subway systems and highways connect two locations with a far distance, leading correlation over distance.
- A CNN with many layers can capture the spatial dependency of any region

Capturing temporal dependence

How to capture temporal dependence?

ST-ResNet

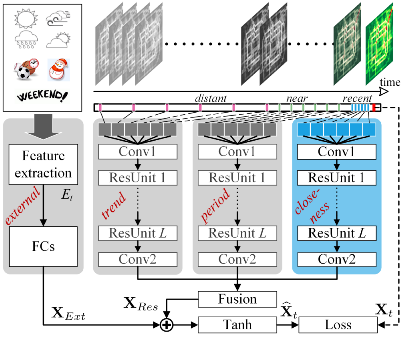


Figure 3: ST-ResNet architecture. Conv: Convolution; ResUnit: Residual Unit; FC: Fully-connected. [6]

[6] Image source: [ZZQ17]

ST-ResNet

Residual learning is technique for having numerous convolutional layers.

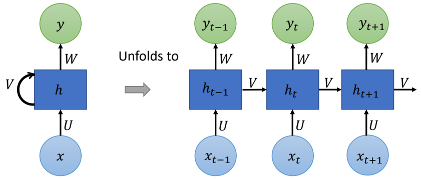
- Inflow and outflow is turned into a into a 2-channel matrix
- Time axis is turned into three fragments, denoting recent time, near history and distant history.
- The flow matrices in each time fragment are fed into the first three components separately to model the aforementioned three temporal properties: closeness, period and trend
- The first three components share the same network structure with a convolutional neural network followed by a Residual Unit sequence.
- In the external component some features from external datasets, such as weather conditions and events are fed into a two-layer fully-connected layer

Part 3: Capturing temporal patterns (Recurrent neural networks)

Recurrent neural networks (RNNs)

- A class of dynamic models (Like HMM, Dynamic Bayesian Networks)
- Connections between nodes form a directed graph along a temporal sequence
    - Allows capturing temporal dynamic behavior
    - RNNs can remember previous states to process sequences of inputs

RNNs



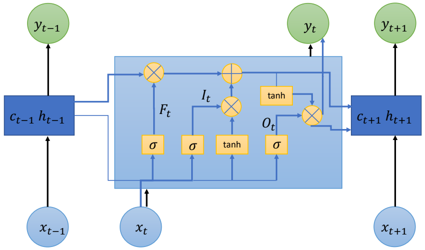
- h_t contains information from all previous past states
    - $h_t = f(h_{t-1}, x_t)$
- We learn the weights through back propagation
    - We have one loss at every timestamp

RNNs

- **Vanishing gradient problem:** weight receives an update proportional to the partial derivative of the error function with respect to the current weight in each iteration of training. The gradient will become very small, preventing the weight from changing its value.
- **Solution:** using more complex units (gated units, LSTMs)

LSTM

- Input, output, forget gates, cell state
- Forget irrelevant parts of previous state
- Selectively update cell state values
- Output certain parts of cell state

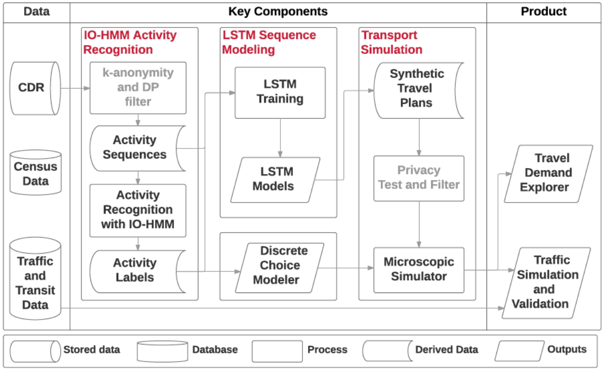


Example: Deep Generative Models of Urban Mobility [LYF$^+$17]

Problem

- **Given:** Call detail records
- **Goal:** Creating a traffic simulator
  - Synthetic daily travel itineraries
  - Traffic volumes that can be compared against real counts from highway sensors and transit agencies data
  - Estimating range of metrics for a given scenario including its environmental impact
  - Aggregated travel demand volumes to evaluate a specific policy

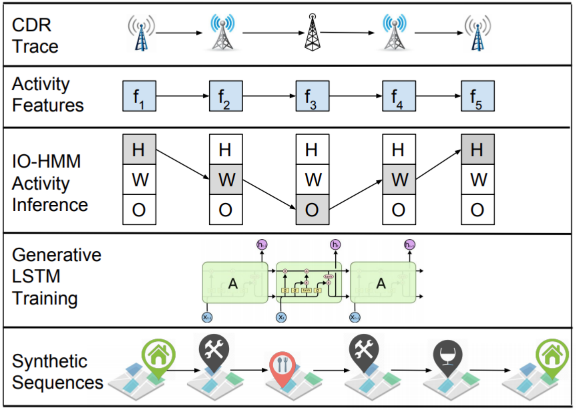
General simulation framework



7

[7] Image source: [LYF+17]

General simulation framework



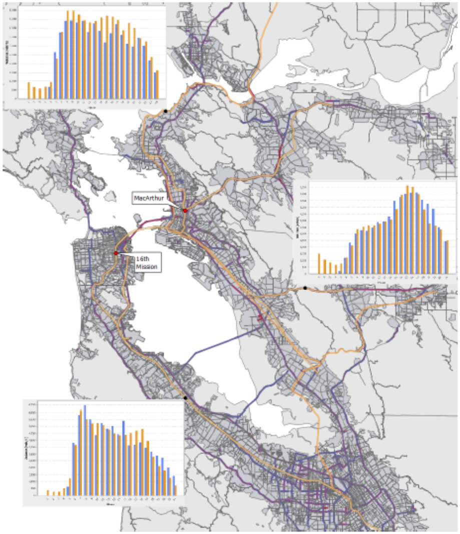
[8]

[8] Image source: [LYF+17]

Steps

- Anonymized CDR data is pre-processed to a sequence of stay location clusters corresponding to distinct unlabeled activities
- Features of activity, such as the start time, duration, location features, and the context of the activity (whether this activity happens during a home-based trip, work-based trip, or a commute trip) are extracted
- IO-HMMs are used to label each activity and uncover the activity patterns
- Labeled activities sequences are sent to a generative recurrent neural network with LSTM cells for training
- The trained model is able to learn explicit location choice with mixture density outputs for each type of activity, and thus capable of generating realistic activity chains

Evaluation



9

9 Image source: [LYF+17]

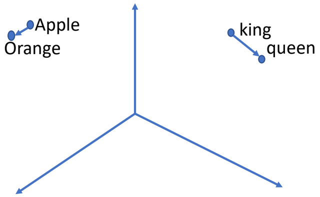
Part 4: Representation learning

Feature extraction

- Many type of data such as words of text, do not have a natural vector representation.
- Previously dealing with high-dimensional data using machine learning approaches relied on user-defined heuristics to extract features from data
  - Graph features (e.g., degree statistics or kernel functions)
  - Image features
  - Text features
- Deep learning provides potentials for automatic feature extraction
  - Automatically learn to encode high dimensional data (graph, text, images to low-dimensional embeddings)

What is an embedding?

- Given high dimensional data the goal is to encode data to low-dimensional vectors that summarize the important properties of data



What is an embedding?

- An embedding is a low-dimensional representation of high-dimensional vectors
- Individual dimensions of the new representation space do not have a meaning
- The patterns of locations and distances between vectors is the embedding space important
- Examples:
  - Embeddings for words: Word2Vec
  - Embeddings for graph: LINE

Modeling data in form of graphs

Graphs provide a flexible and general data structure for variety of applications using urban scale spatio-temporal data

- LBSN data
- Road network data

Let's see how we can learn embeddings for graphs

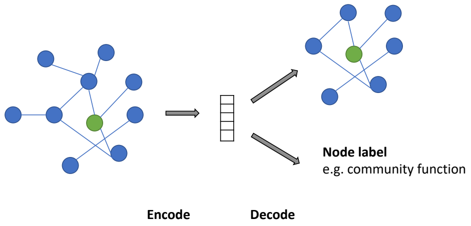
Factorization: Latent factor models

An example of how we did it before ...

- Assume that we can approximate the rating matrix R as a product of U and P^T

$$\underset{R}{\begin{array}{c|c|c|c|c|} & p_1 & p_2 & p_3 & p_4 \\ \hline u_1 & & 4.5 & 2 & \\ \hline u_2 & 4.0 & & 3.5 & \\ \hline u_3 & & 5.0 & & 2.0 \\ \hline u_4 & & 3.5 & 4.0 & 1.0 \\ \hline \end{array}} = \underset{U}{\begin{array}{c|c|c|} & \multicolumn{2}{c}{(k=2) \text{ factors}} \\ \hline u_1 & 1.2 & 0.8 \\ \hline u_2 & 1.4 & 0.9 \\ \hline u_3 & 1.5 & 1.0 \\ \hline u_4 & 1.2 & 0.8 \\ \hline \end{array}} \times \underset{P^T}{\begin{array}{|c|c|c|c|} p_1 & p_2 & p_3 & p_4 \\ \hline 1.5 & 1.2 & 1.0 & 0.8 \\ \hline 1.7 & 0.6 & 1.1 & 0.4 \\ \hline \end{array}}$$

The general Encoder-decoder approach



- **The encoder:** maps nodes of a graph to embeddings
- **The decoder:** maps the embeddings to structural information about the graph (neighborhood level information, or a community class label).[HYL17]

Steps in creating graph embeddings (graph embeddings)

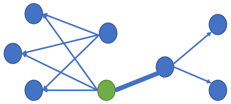
1. **Pairwise proximity function:** measures the connected-ness of nodes
2. **Encoder function:** generates node embeddings
3. **Decoder function:** reconstructs pairwise proximity values from the generated embeddings.
4. **Loss function**: measures the quality of the pairwise reconstructions [HYL17]

LINE: Large Scale Information Networks Embedding [TQW^{+}15]

Node embedding

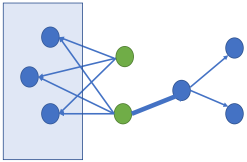
- Automatically creating features (embeddings) for different types of graphs
- Clear objective function
  - loss function is defined based on first and second order proximity

First-order proximity



Proximity between nodes based on the local pairwise proximity

Second-order proximity



- Proximity between neighbors of a node
- The general notion of the second-order proximity can be interpreted as nodes with shared neighbors being likely to be similar

Optimization

Goal: Embeddings should preserve both the first-order and second-order proximities

- Loss on the first order proximity
- Loss on the second order proximity

Two objective functions (O_1, O_2)

Loss on the first order proximity

- **Joint distribution** of first-order proximity
    - $p_1(v_i, v_j) = \frac{1}{1+exp(-u_i^T.u_j)}$ (u_i and u_j are low dimensional vector representation)
- **Empirical distribution** of first-order proximity (w_{ij} is the weight of edges between nodes)
    - $\hat{p}_1(v_i, v_j) = \frac{w_{ij}}{\sum_{i,j \in E} w_{ij}}$
- Optimize the loss based on the distance between two distributions (joint probability and empirical probability)
    - $O_1 = d(\hat{p}_1(.,.), p_1(.,.))$

Loss on the second order proximity

- **Joint distribution** of neighborhood structure (defined on the directed edge i $\rightarrow$ j)
  - $p_2(v_i|v_j) = \frac{exp(u_j^T / u_i)}{\sum_{k=1}^{|V|} exp(u_k^T . u_i)} w_{ik}$
- **Empirical distribution** of neighborhood structure defined on the directed edge i $\rightarrow$ j (d_i is the out-degree of node v_i)
  - $\hat{p}_2(v_i|v_j) = \frac{w_{ij}}{d_i}$
  - where N_i is the set of out neighbors of node i
- Optimize the loss based on the distance between two distributions (joint probability and empirical probability)
  - $O_2 = d(\hat{p}_2(.,.), p_2(.,.))$

Example: Using LINE for representing regions

Given a large set of spatio-temporal trajectories, how can you use graph embeddings?

Region representation learning via Mobility flow [WL17]

- **Goal:** is to learn vector representations for regions using mobility data (e.g. taxi trajectories) and later use the representations in different modeling application
- LINE-based proximities:
  - **First order proximity:** if there is a large volume of flow from region x to region y
  - **Second order proximity**: if there is a flow from x and y to similar regions

Generalized inference model

Using embedding in a general inference model

- Infer a regional property (i.e. crime rate, personal income, and real estate price) from observed auxiliary urban features.
- Learning region embedding from mobility flow data to enhance the following inference model
  $y_i = \alpha . X_i + \beta \sum_{i \in N_i} w(i,j) . y_j + \gamma$
  - y_i is the target value
  - α, β, γ are parameters of the regression model
  - w_{ij} are weights coming from embeddings
  - X_i auxiliary features

Graph embeddings for spatio-temporal data

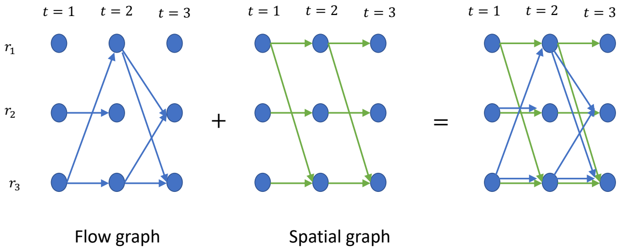
- Can be captured in a graph embedding:
  - First order proximity
  - Second order proximity
- Can't be capture in a graph embedding:
  - Spatial structures
  - Temporal structures

Region embedding method

Region embedding method:

- **Flow graph:** a layered graph with a set of time enhanced vertices. The edge weight are volumes of mobility between two vertices
- **Spatial graph:** With vertices exactly the same as that of flow graph. The edge set only contains edges connecting two vertices from consecutive layers. The edge weights represent the spatial similarity of two regions.

Region embedding



Validating region embeddings

Using the embedding in inference tasks

- Crime data
- House price data
- ...

Part 5: Transfer learning

Transfer learning

- Supervised learning models requires access to label
- When using neural networks for supervised learning we would need even more labels
- Transfer learning methods aim at transfering the knowledge gained while solving one problem and applying use this knowledge in a different solving a different problem

Transfer learning and deep learning

- Pre-training and fine-tuning
- Domain adaptation
- Domain confusion
- Multi-task learning
- One-shot learning
- Zero-shot learning

Transfer learning for Urban Computing

Example: Cross-city Transfer Learning for Deep Spatio-temporal Prediction [GLZ$^+$18]

Goal

- We are interested in prediction of air quality, traffic flows, and other urban parameters
- In some cities we do not have means to collect data that can be used for extracting a model
- How can we transfer the knowledge we can get from the data-rich cities to data-scarce cities?

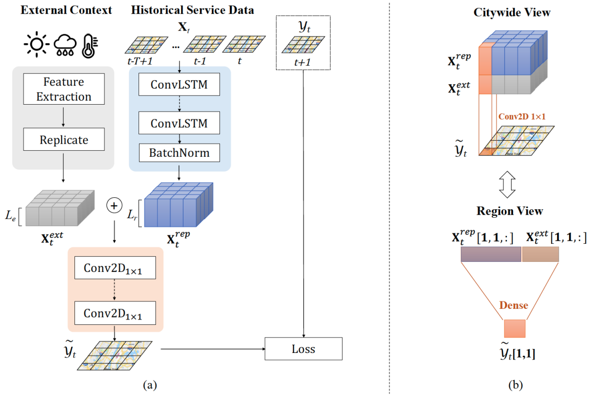
Problem

- **Given:**
  - **Urban image time-series:** $I_D = \{i_{r,t}| \in D\}$
    - where D is the grid of the city, r is a regions in city
    - weather condition, air quality, crowd flow,

Problem

- **Given:**
  - **Urban image time-series:** $I_D = \{i_{r,t} | \in D\}$
    - where D is the grid of the city, r is a regions in city
    - weather condition, air quality, crowd flow,
  - **Service spatio-temporal data:** $S_D = \{s_{r,t} | r \in D\}$
  - **Source city** D': Rich in terms of service
  - **Target city** D: With little service data in
  - Different temporal data durations in different cities
- **Goal:**
  - Learn a function model for predicting the service data in the target city data over time

Transferring the knowledge across cities



10

Figure: Pre-training a model in the source city

10 Image source: [GLZ+18]

Transferring the model to the target city

- Pre-trained model on the source city (we have the weights of the neural work)
- Refine the weights of the pertained model θ on the target city
  - **Objective 1:** Reducing the error on prediction of service data on the target city: $min_\theta = \sum ||\tilde{Y}_t - Y_t||^2$
  - **Objective 2:** Reducing the representation divergence between matched region in the target city $x_{r,t}$ and source city $x_{r^*,t}$ based on a correlation coefficient

Baselines

- ARIMA
- DeepST
- ST-RestNet

Lessons learned

- The strength of neural networks lies in **automatic feature extraction** and **encoding non-linearity**
- There are already neural network models for extracting spatial and temporal feature from data automatically
    - These models still need to be **adapted** to spatio-temporal data for urban applications
- Representations learning is a suitable technique that can create **generic (spatio-temporal) features** from data usable for different modeling tasks
    - We need to think about how to define the right objective function for creating representations
- Transfer learning that provide the possibility of **transferring the knowledge** from data-rich urban areas to data-scarce areas

References I

Bin Guo, Jing Li, Vincent W. Zheng, Zhu Wang, and Zhiwen Yu, *Citytransfer: Transferring inter- and intra-city knowledge for chain store site recommendation based on multi-source urban data*, Proc. ACM Interact. Mob. Wearable Ubiquitous Technol. **1** (2018), no. 4.

William L Hamilton, Rex Ying, and Jure Leskovec, *Representation learning on graphs: Methods and applications*, arXiv preprint arXiv:1709.05584 (2017).

Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y Ng, *Unsupervised learning of hierarchical representations with convolutional deep belief networks*, Communications of the ACM **54** (2011), no. 10, 95–103.

References II

Ziheng Lin, Mogeng Yin, Sidney Feygin, Madeleine Sheehan, Jean-Francois Paiement, and Alexei Pozdnoukhov, *Deep generative models of urban mobility*, IEEE Transactions on Intelligent Transportation Systems (2017).

Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei, *Line: Large-scale information network embedding*, Proceedings of the 24th international conference on world wide web, International World Wide Web Conferences Steering Committee, 2015, pp. 1067–1077.

Hongjian Wang and Zhenhui Li, *Region representation learning via mobility flow*, Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, ACM, 2017, pp. 237–246.

References III

Ying Wei, Yu Zheng, and Qiang Yang, *Transfer knowledge between cities*, Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016, pp. 1905–1914.

Junbo Zhang, Yu Zheng, and Dekang Qi, *Deep spatio-temporal residual networks for citywide crowd flows prediction*, Thirty-First AAAI Conference on Artificial Intelligence, 2017.