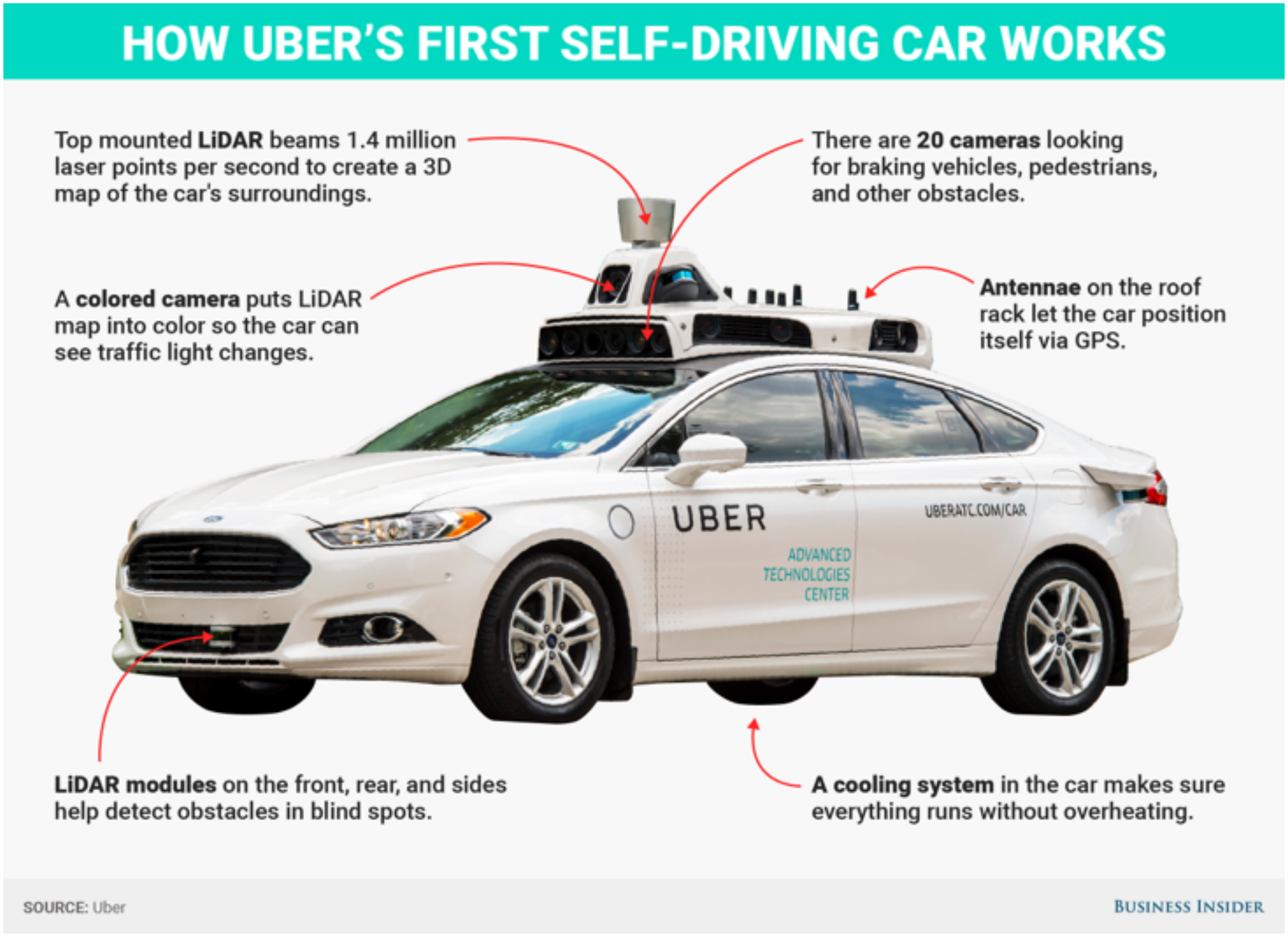
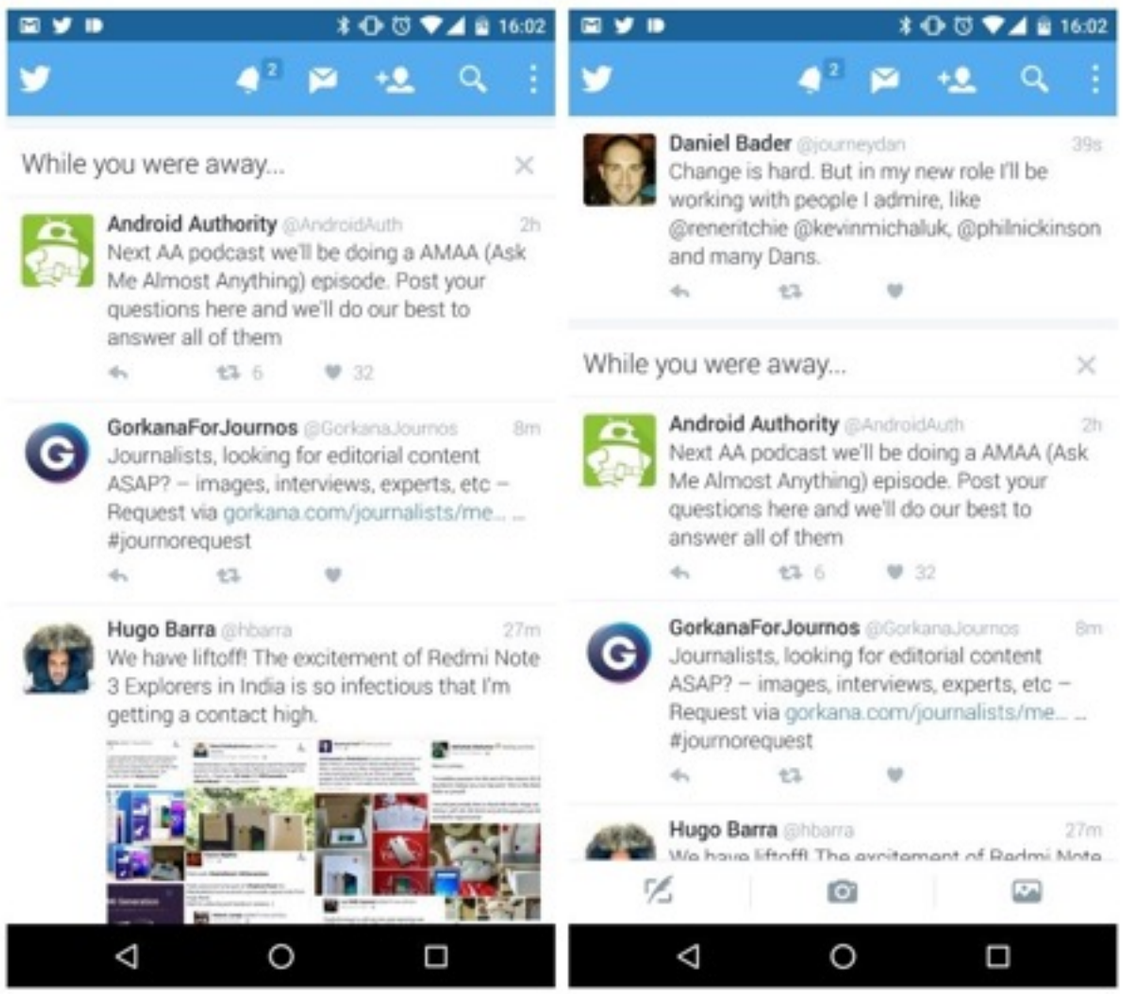
Application of RNNs to Multiple Asynchronous Event-Driven Data Streams

…and Deep Learning with Dynamic Computational Graphs

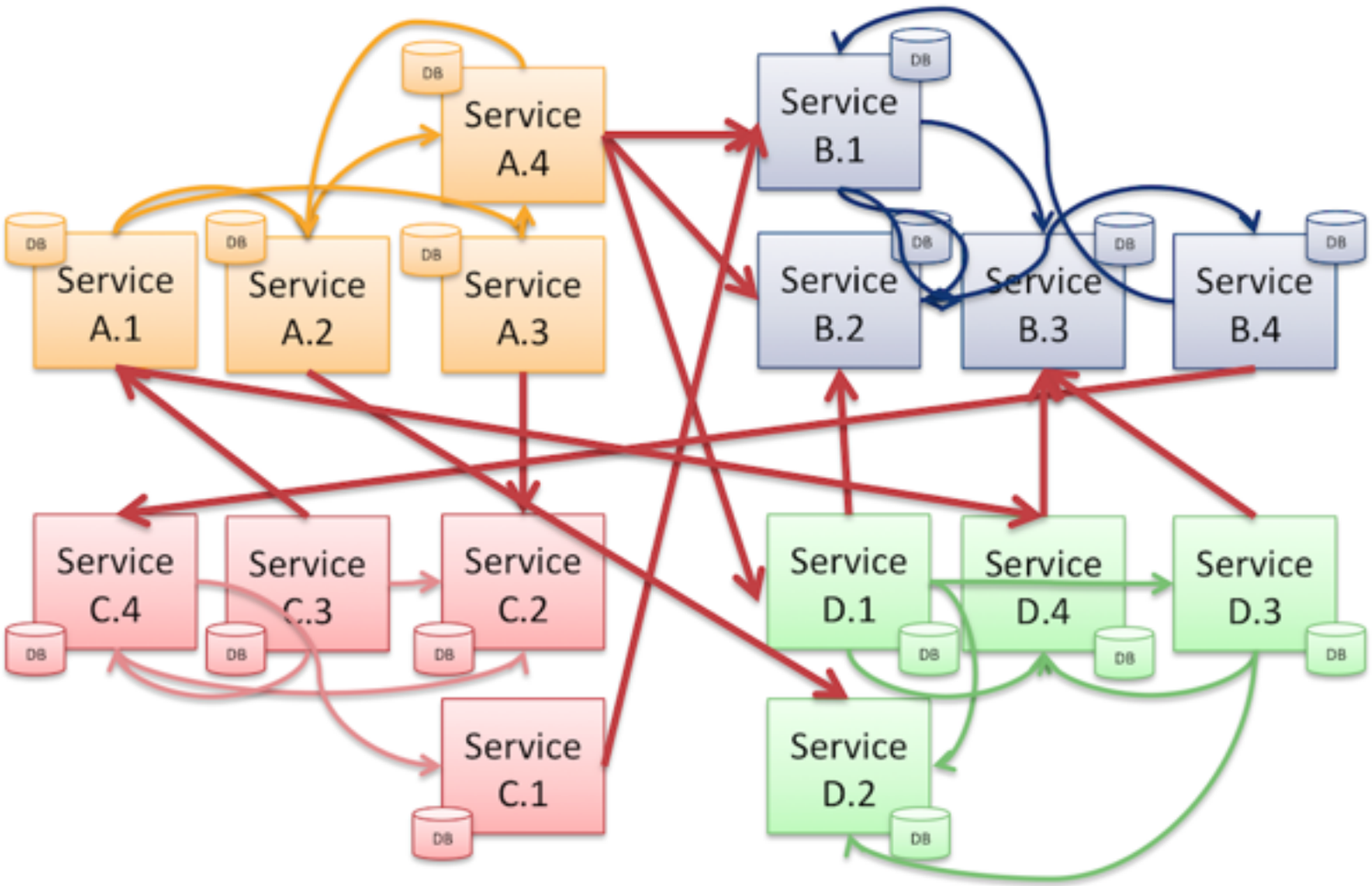
Data Streams



Data Streams



Data Streams



Data Streams

postnord

May 29, 2017

Thesis work in Machine Learning 2017/18

Do you want to be part of a revolution in logistics? Companies like Amazon and Alibaba has shown that logistics and e-commerce is a data-challenge. As the largest player in the Nordics with approx 1 million shipments/day we're in a great position to start applying machine-learning throughout our production-pipeline. We're currently looking for ambitious thesis workers to our headquarters in Stockholm that wants to help us get started.

Task

Build & deploy a predictive model that can work in a high-throughput production environment. GO!

Context

If you've ordered something online you've seen new events roll in as your package is moved about. Using anonymized data about individual shipments we want to approach this from a machine-learning point of view and use these sequences to extract knowledge and make predictions.

In each step of a chain of events you want to use what you know about the shipment and the surrounding system to predict a multitude of things:

1. Time to arrival/final event (estimate ETA)
2. Time To next Event (TTE)
3. Probability of delay (binary sequence classification)
4. Predict next event (think char-RNN)

We want to query the models in realtime but evaluate them only as new events roll in. It's your job to figure out the solution to the problem. One idea for this type of non-event prediction could be to apply Deep Learning techniques like WTTE-RNN[1].

We're looking for masters students heavily focused on Machine Learning. We assume that you're a quite confident programmer (knows git and R or Python and experience with Java/Scala is a big plus).

We offer well defined and generalizable ML problems, huge & great data in a quick-paced environment with a really fun stack. Reach out to Egil Martinsson [2] (Chalmers alumn) to talk more!

[1] github, blog, thesis

[2] egilm[at]gmail.com 0737416185

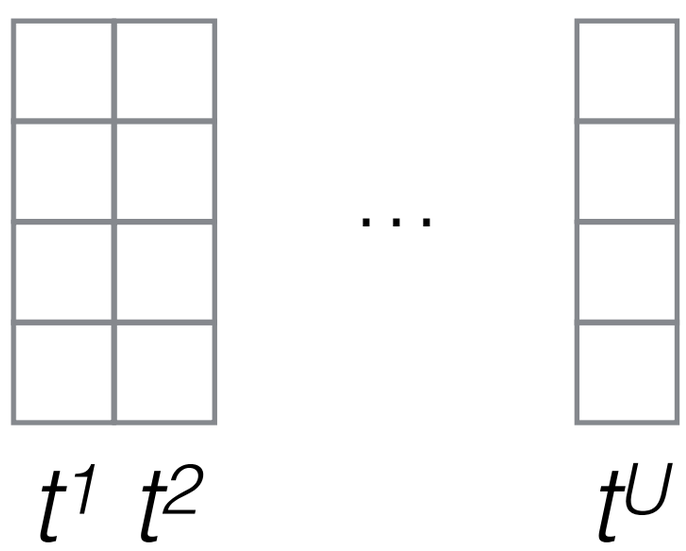
Challenges

- Different data types (data-dependent feature extraction)
    - Most popular dl libraries are based on *static computational graphs*
- Varying resolution/timescale
    - Achieve long enough effective memory
    - bursts of high-frequent events vs rare events

Regular Time Series

input data: $f^{1:U} = [f^1, \ldots, f^U]$

input index: $t^{1:U} = [t^1, \ldots, t^U]$

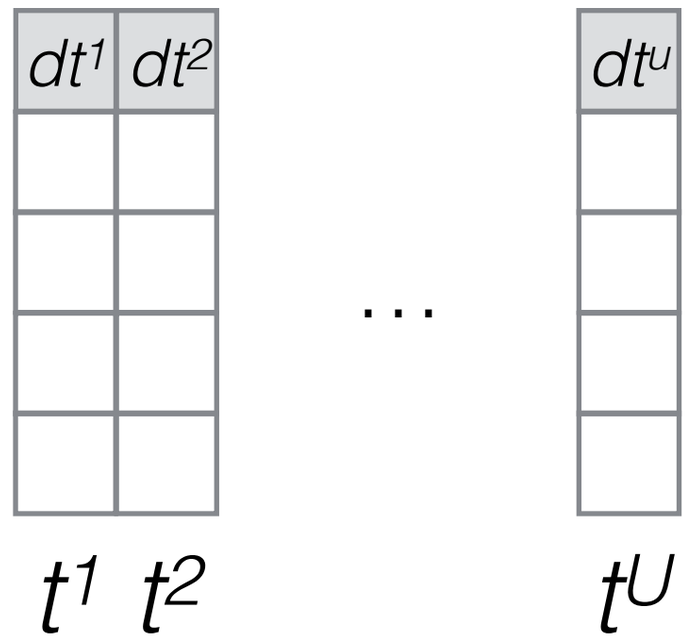


$$t^u - t^{u-1} = dt^u = k$$

Irregular Time Series

input data: $f^{1:U} = [f^1, \ldots, f^U]$

input index: $t^{1:U} = [t^1, \ldots, t^U]$

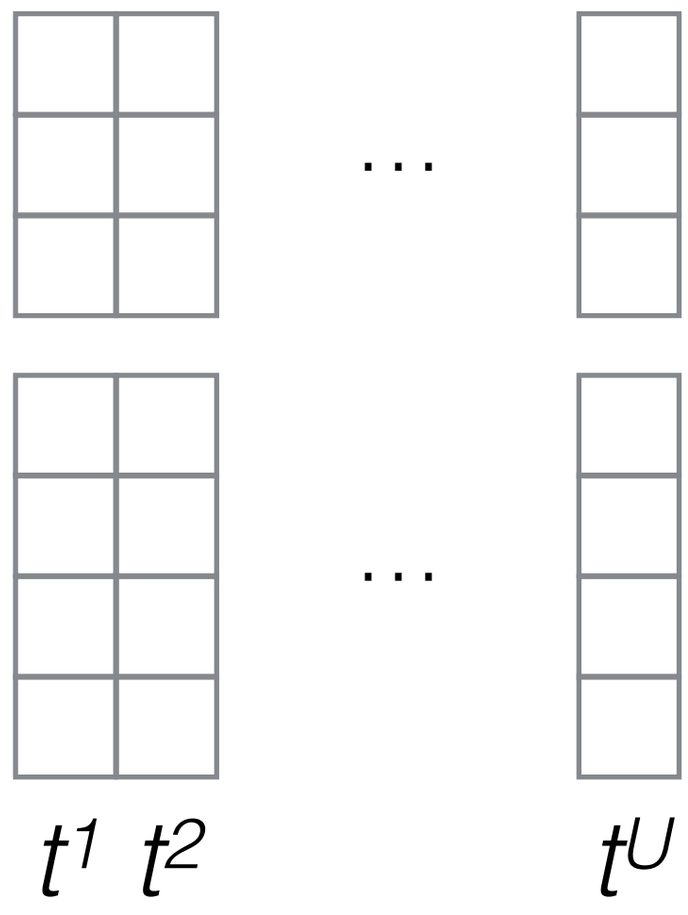


$$t^u - t^{u-1} = dt^u \neq k$$

Multiple Sources - Sync

input data: $(f_1, f_2)^{1:U} = [(f_1, f_2)^1, \ldots, (f_1, f_2)^U]$

input index: $t^{1:U} = [t^1, \ldots, t^U]$

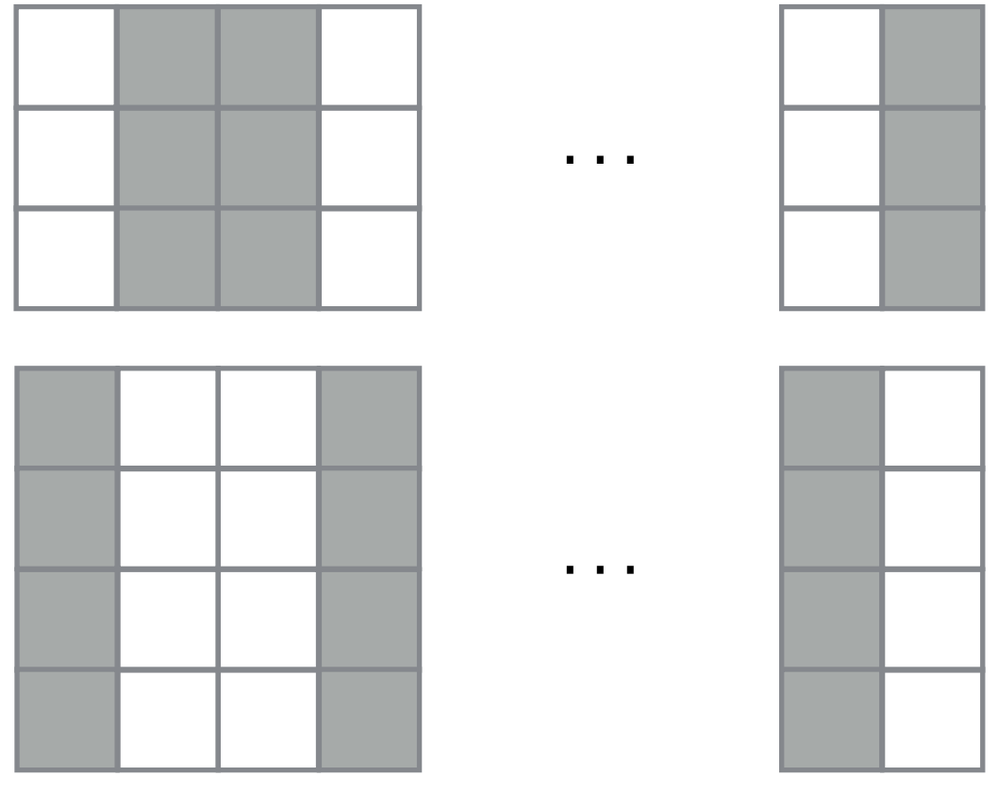


Multiple Sources - Async

input data: $(f_1 \text{ or } f_2)^{1:U} = [(f_i)^1,(f_i)^2,\ldots,(f_i)^U]$

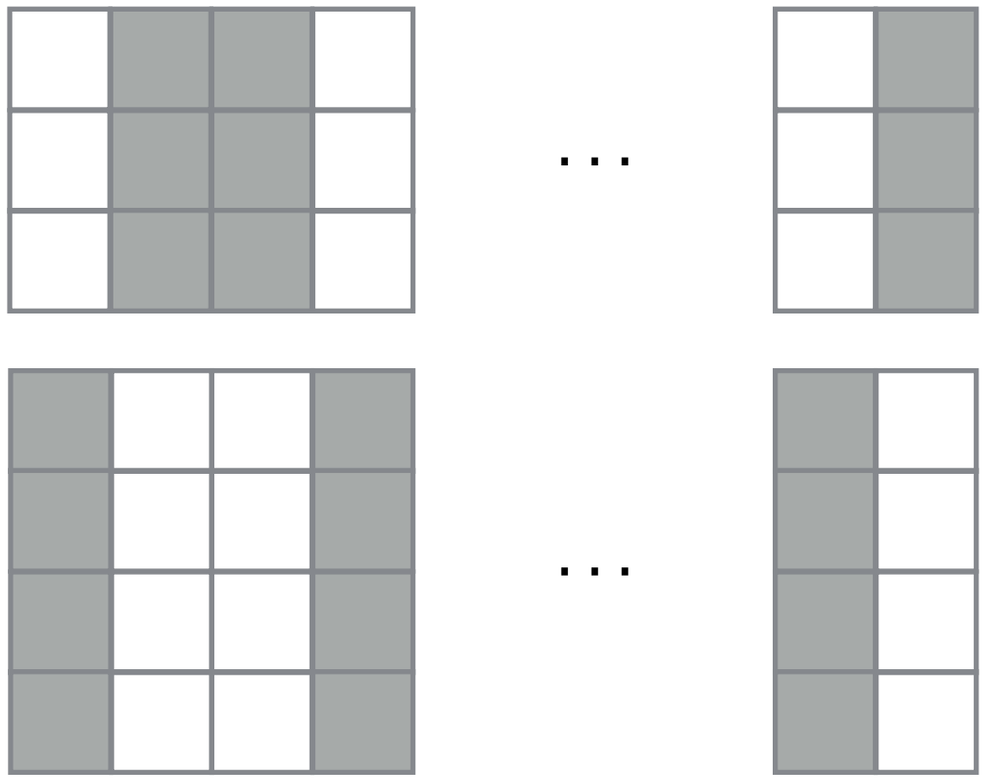
input index: $t^{1:U} = [t^1,\ldots,t^U]$

missing data approach:

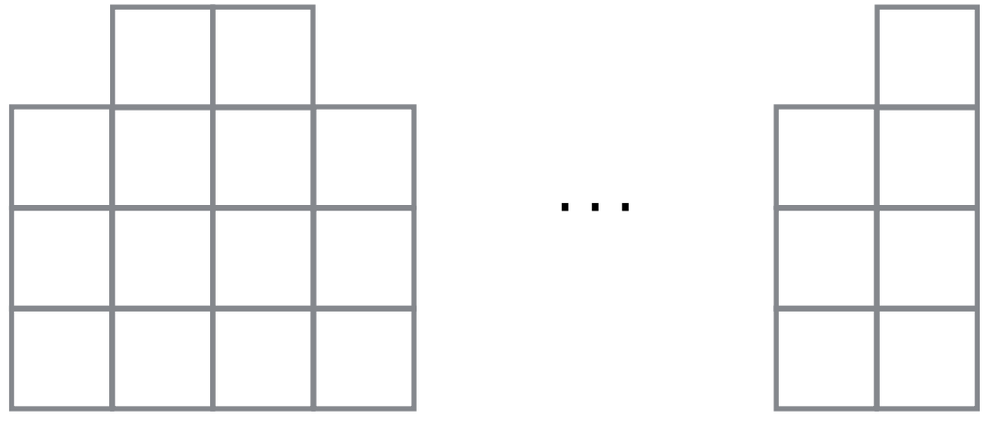


Multiple Sources - Async

missing data approach:



what we really want to do:

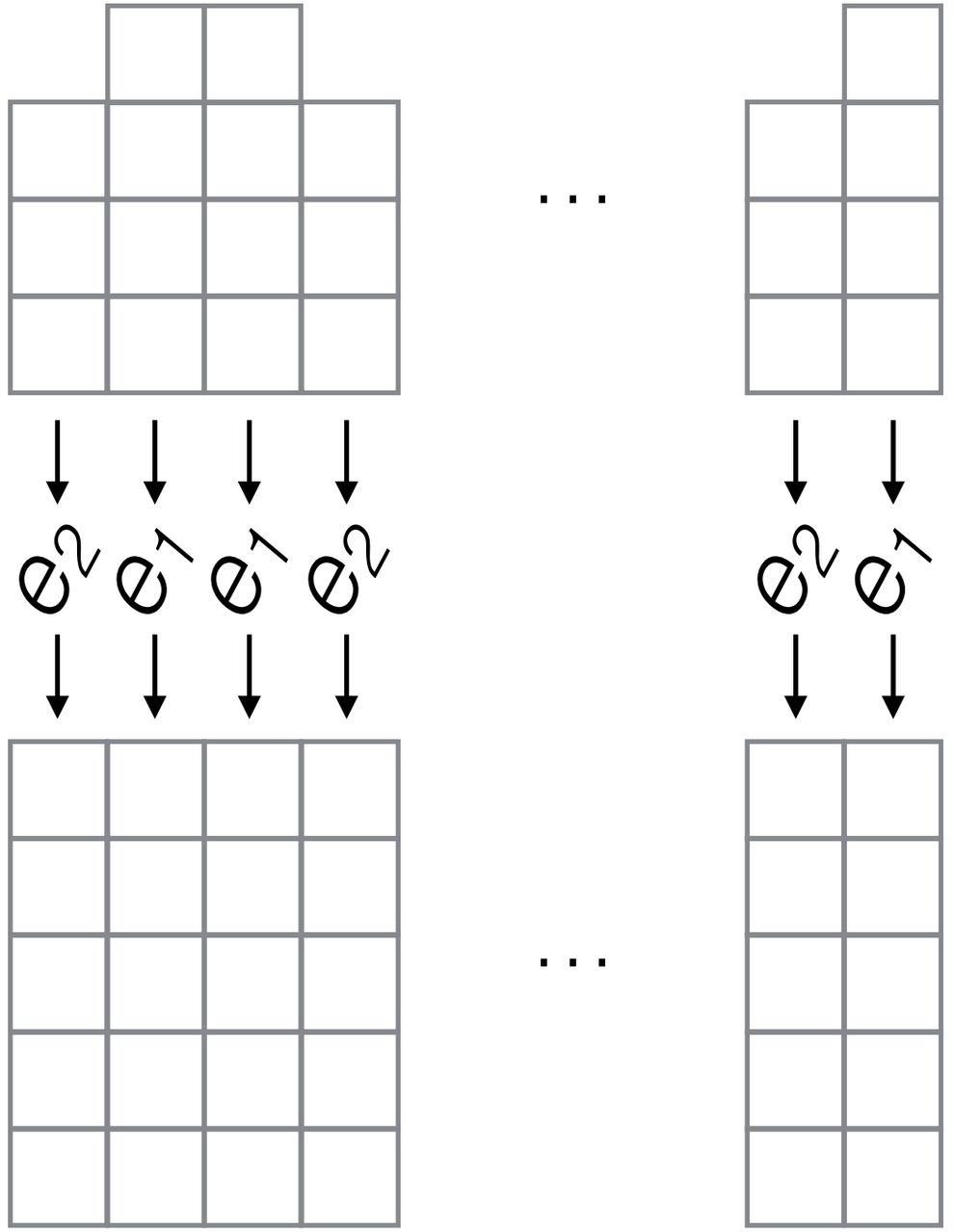


Multiple Sources - Async

Input data/features

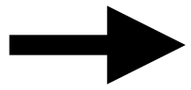
data-type-specific feature extraction

conformed/general features for RNN



Multiple Sources - Async

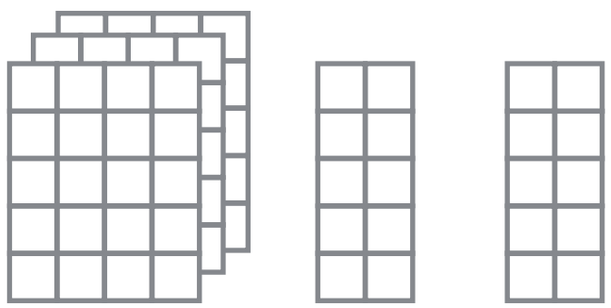
data-type-specific
feature extraction



data-dependent
dynamic graph

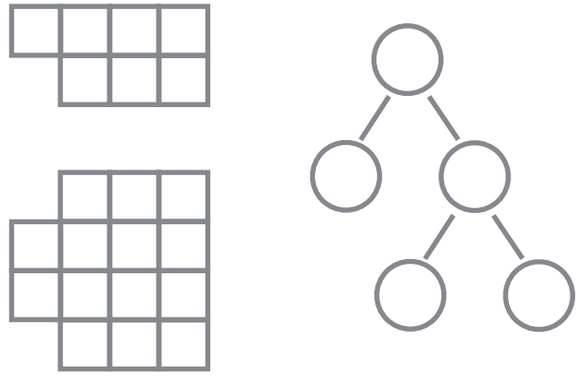
DL implementation regimes

Static Graphs



Tensorflow
Theano

Dynamic Graphs



TensorFlow-Fold
DyNet
PyTorch
Chainer

TensorFlow-Fold

Any transformation is referred to as a *Block*. We combine blocks by *function composition:*

F: x -> f(x)
G: x -> g(x)

H = F >> G
H: x -> h(x), h = (g o f)(x) = g(f(x))

Conform RNN features from different input types

```
A_SIZE = 3
B_SIZE = 5

input_sequence = [
    {'type': 'A', 'data': [1, 2, 3]},
    {'type': 'B', 'data': [5, 4, 3, 2, 1]},
    {'type': 'A', 'data': [3, 2, 1]},
]

RNN_FEATURE_SIZE = 8
```

Conform RNN features from different input types

```
feature = td.OneOf(
    key_fn=lambda x: x['type'],
    case_blocks={
        'A': td.GetItem('data') >> td.Vector(A_SIZE) >> td.FC(RNN_FEATURE_SIZE),
        'B': td.GetItem('data') >> td.Vector(B_SIZE) >> td.FC(RNN_FEATURE_SIZE)
    }
)
feature_sequence = td.Map(feature)
```

Supports variable length input sequences!

Conform RNN features from different input types

```
inputs_A = tf.placeholder('float32', shape=(None, None, A_SIZE), name='inputs_A')
inputs_B = tf.placeholder('float32', shape=(None, None, B_SIZE), name='inputs_B')

feature_from_A = tf.layers.dense(inputs_A, RNN_FEATURE_SIZE, activation=tf.nn.relu)
feature_from_B = tf.layers.dense(inputs_B, RNN_FEATURE_SIZE, activation=tf.nn.relu)
```

```
mask_A = tf.placeholder('float32', shape=(None, None, 1), name='mask_A')
mask_B = tf.placeholder('float32', shape=(None, None, 1), name=‘mask_B')

feature_sequences = feature_from_A_masked + feature_from_B_masked

#...followed by data padding and formatting
```

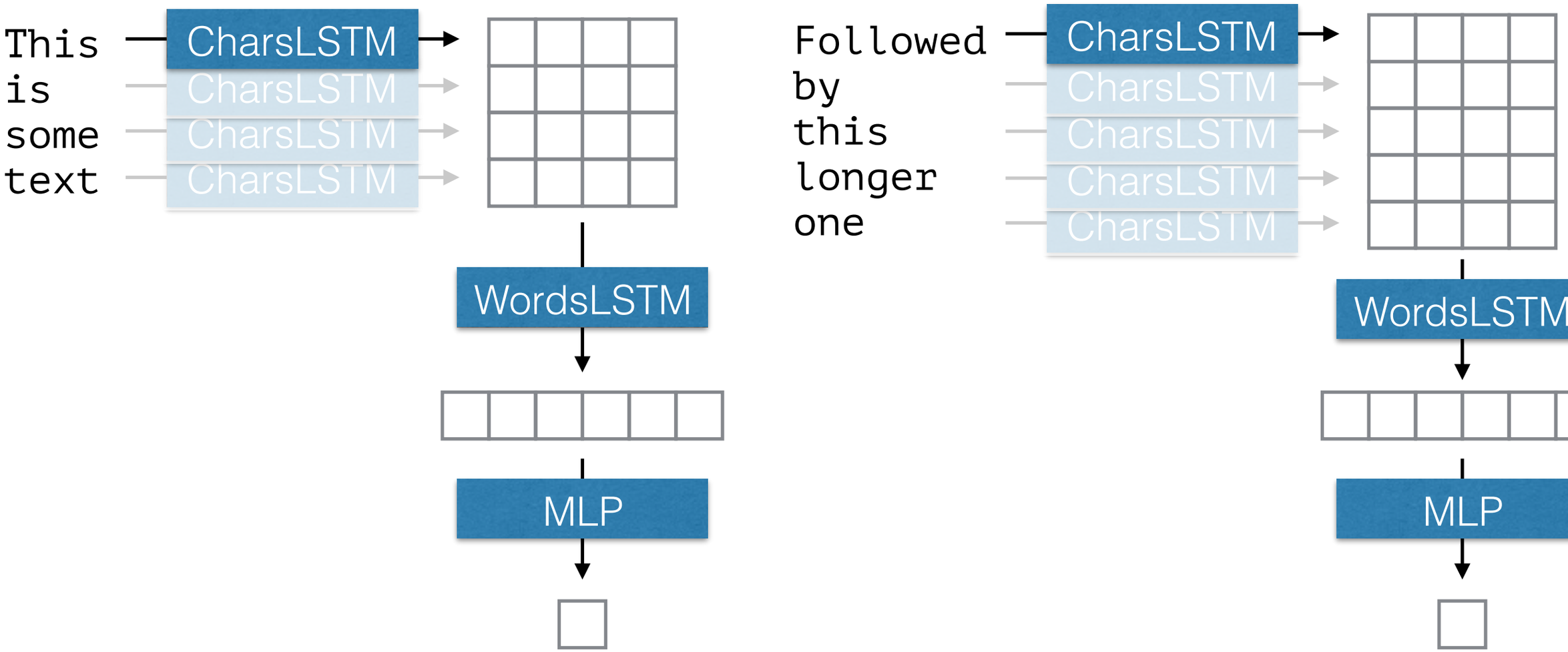
Requires fixed length of input sequences or implementation of additional padding

Conform RNN features from different input types

```
input_sequence = [‘words’, ’in’, ‘a’, ‘sentence’]
```

```
input_sequence = [
   [‘w’, ‘o’, ‘r’, ‘d’, ’s’],
   [‘i’, ’n’],
   [‘a’],
   [’s’,’e’,’n’,’t’,’e’,’n’,’c’,’e’]
]
```

Hierarchical LSTM (for sentiment analysis)



And what about the “varying resolution/timescale”

- Let the RNN know time difference between events - and/or absolute time
  - meaningful encoding of absolute time - is there e.g. expected to be a correlation with time during day?
- Allow only a subset of LSTM’s cell activations to be updated based on event/input type
  - this way we designate a subset of cell activations for a specific input type - especially useful for making use of rare event types
- General way of capturing underlying periodicity and extend effective memory horizon: *Phased LSTMs*

Links and References

- TODO