Time Series and Sequence Learning

Lecture 10 – Recurrent Neural Networks

Johan Alenlöv, Linköping University

2021-10-05

Summary of Lecture 9

Summary of Lecture 9: State transformations

Consider the LGSS model

$$\begin{aligned} \alpha_t &= T\alpha_{t-1} + R\eta_t, & \eta_t &\sim \mathcal{N}(0, Q), \\ y_t &= Z\alpha_t + \varepsilon_t, & \varepsilon_t &\sim \mathcal{N}(0, \sigma_\epsilon^2). \end{aligned}$$

We can obtain an **equivalent model** by a change of variables

$$\widetilde{\alpha}_t = \Gamma\alpha_t \iff \alpha_t = \Gamma^{-1}\widetilde{\alpha}_t,$$

resulting in

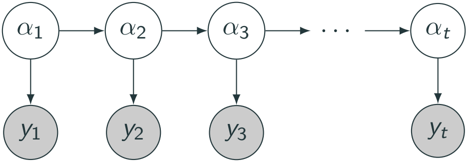
$$\begin{aligned} \widetilde{\alpha}_t &= \Gamma T \Gamma^{-1}\widetilde{\alpha}_{t-1} + \Gamma R\eta_t, & \eta_t &\sim \mathcal{N}(0, Q), \\ y_t &= Z\Gamma^{-1}\widetilde{\alpha}_t + \varepsilon_t, & \varepsilon_t &\sim \mathcal{N}(0, \sigma_\epsilon^2). \end{aligned}$$

The state representation is not unique!

Summary of Lecture 9: Innovation form

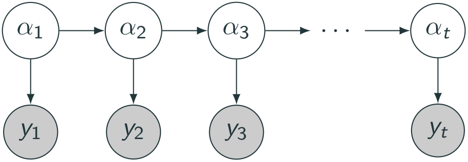
Original form:

$$\alpha_t = T\alpha_{t-1} + R\eta_t,$$
$$y_t = Z\alpha_t + \varepsilon_t.$$



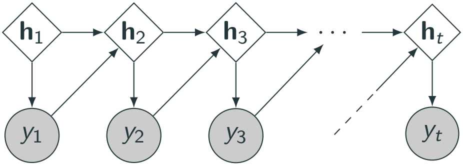
Original form:

$$\alpha_t = T\alpha_{t-1} + R\eta_t,$$
$$y_t = Z\alpha_t + \varepsilon_t.$$



Innovation form:

$$\mathbf{h}_t = W\mathbf{h}_{t-1} + Uy_{t-1},$$
$$y_t = C\mathbf{h}_t + \nu_t.$$



The hidden state variable $\mathbf{h}_t$ can be **deterministically and recursively computed** from the data.

Summary of Lecture 9: Going nonlinear

Innovation form of an LGSS model:

$$
\begin{aligned}
\mathbf{h}_t &= W\mathbf{h}_{t-1} + Uy_{t-1},\\
y_t &= C\mathbf{h}_t + \nu_t,
\end{aligned}
$$

By introducing a nonlinear activation function in the state update we obtain a simple RNN,

$$
\begin{aligned}
\mathbf{h}_t &= \sigma(W\mathbf{h}_{t-1} + Uy_{t-1} + b),\\
y_t &= C\mathbf{h}_t + c + \nu_t,
\end{aligned}
$$

with **learnable** parameters $\theta = \{W, U, b, C, c\}$.

The parameters are the same for all time steps ("weight sharing").

Summary of Lecture 9: Learning the parameters

We train the model by minimizing the negative log-likelihood,

$$L(\theta) = -\sum_{t=1}^{n} \log p_{\theta}(y_t \mid y_{1:t-1}) = \sum_{t=1}^{n} \left\{ y_t - \hat{y}_{t|t-1}(\theta) \right\}^2$$

using gradient-based numerical optimization.

The fact that there is no state noise means that we can compute $\hat{y}_{t|t-1}(\theta)$, $t = 1, \ldots, n$ by a forward pass through the network.

The gradient is computed by back-propagation on the "unrolled" network,

$\Longrightarrow$ **Back-propagation through time.**

Summary of Lecture 9: A (more) general RNN model

RNNs are not restricted to the simple networks discussed above.

A generalization of the Jordan-Elman network is,

$$\mathbf{h}_t = H_\theta(\mathbf{h}_{t-1}, y_{t-1}),$$

$$y_t = O_\theta(\mathbf{h}_t, y_{t-1}) + \nu_t, \qquad \nu_t \overset{\text{iid}}{\sim} \mathcal{N}(0, \sigma_\nu^2).$$

for arbitrary (parameterized) nonlinear functions H_θ and O_θ.

Summary of Lecture 9: A (more) general RNN model

RNNs are not restricted to the simple networks discussed above.

A generalization of the Jordan-Elman network is,

$$
\begin{aligned}
\mathbf{h}_t &= H_\theta(\mathbf{h}_{t-1}, y_{t-1}), \\
y_t &= O_\theta(\mathbf{h}_t, y_{t-1}) + \nu_t, \qquad \qquad \nu_t \overset{\text{iid}}{\sim} \mathcal{N}(0, \sigma_\nu^2).
\end{aligned}
$$

for arbitrary (parameterized) nonlinear functions H_θ and O_θ.

- This is a nonlinear state-space model with output feedback and **without state noise.**
- As before, the one-step prediction can be computed by a forward propagation

$$
\underline{p_\theta(y_t \mid y_{1:t-1})} = \mathcal{N}(y_t \mid \underline{O_\theta(\mathbf{h}_t, y_{t-1})}, \sigma_\nu^2).
$$

Aim and outline

Aim:

- Discuss different approaches to training RNNs in a time series context.
- Discuss different RNN architectures and application of these models in time series analysis.

Aim and outline

Aim:

- Discuss different approaches to training RNNs in a time series context.
- Discuss different RNN architectures and application of these models in time series analysis.

Outline:

1. Training RNNs: Different approaches to mini-batching
2. Long-range dependencies and specialized RNN architectures
3. Extensions and alternative use-cases

Training RNNs

Learning from multiple time series

In the RNN literature it is common that the training data consists of **multiple short sequences**, $\{y^j_{1:n}\}_{j=1}^S$

seq	data		
1	y^1_1	$\cdots$	y^1_n
		$\vdots$	
		$\vdots$	
		$\vdots$	
S	y^S_1	$\cdots$	y^S_n

With **loss-function**

$$L(\theta) = \sum_{j=1}^{S} \left\{ \sum_{t=1}^{n} (y^j_t - \hat{y}^j_{t|t-1}(\theta)) \right\}$$

Learning from multiple time series

In the RNN literature it is common that the training data consists of **multiple short sequences**, $\{y^j_{1:n}\}_{j=1}^S$

seq	data		
1	y^1_1	$\cdots$	y^1_n
		$\vdots$	
		$\vdots$	
		$\vdots$	
S	y^S_1	$\cdots$	y^S_n

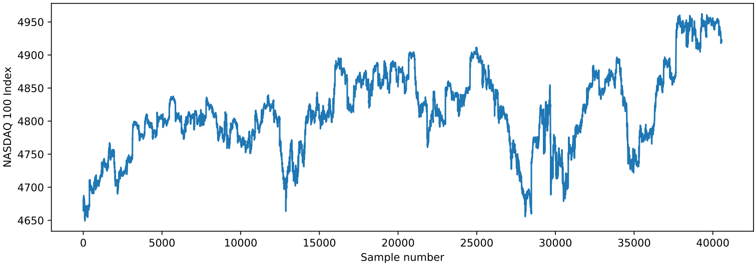
With **loss-function**

$$L(\theta) = \sum_{j=1}^{S} \left\{ \sum_{t=1}^{n} (y^j_t - \hat{y}^j_{t|t-1}(\theta)) \right\}$$

Typically, use mini batching by choosing a small batch of the sequences.

Learning from a single long time series

What if we instead have a **single, long time series?**



Possible approaches:

1. Do nothing
2. Split the data into shorter sequences that are assumed to be independent
3. Split the data with “statefulness” between sequences

Option 1. Do nothing

Optimize the loss function

$$L(\theta) = \sum_{t=1}^{n} \left\{ y_t - \hat{y}_{t|t-1}(\theta) \right\}^2$$

by gradient descent **without using mini-batching**.

Option 1. Do nothing

Optimize the loss function

$$L(\theta) = \sum_{t=1}^{n} \left\{ y_t - \hat{y}_{t|t-1}(\theta) \right\}^2$$

by gradient descent **without using mini-batching**.

- Treated as a "single sample"
- Batch size $= 1$, one gradient step/epoch.
- **Each gradient computation** using BPTT requires a full forward–backward pass through the data.

$$\implies O(n) \text{ computation per gradient step.}$$

Option 2. Splitting into sub-sequences

Original data

y_1	y_2	$\cdots$	$\cdots$	y_{n-1}	y_n

Split into S sequences of length m

seq	data		
1	y_1	$\cdots$	y_m
2	y_{m+1}	$\cdots$	y_{2m}
		$\vdots$	
		$\vdots$	
		$\vdots$	
S	y_{n-m+1}	$\cdots$	y_n

Option 2b. ...with random starting points

Original data

y_1	y_2	$\cdots$	$\cdots$	y_{n-1}	y_n

Choose a random starting point and take a window of length m.

seq	inputs			targets		
1	y_{s_1}	$\cdots$	y_{s_1+m}	y_{s_1+1}	$\cdots$	y_{s_1+m+1}
		$\vdots$			$\vdots$	
		$\vdots$			$\vdots$	
B	y_{s_B}	$\cdots$	y_{s_B+m}	$y_{s_B} + 1$	$\cdots$	y_{s_B+m+1}

Option 2c. ...and warm-up

Neglecting temporal dependencies between consecutive sequences can give rise to unwanted boundary effects.

Mitigated by allowing the hidden state to ”warm up” for a few time steps.

Basic windowing Loss is computed by summing prediction errors over the whole window.

With warmup Skip the initial r values in the loss computation.

Option 3. Stateful training

Stateful means that we keep the hidden state from the previous sub-sequence, when processing the next one.

Stateful training:

- Split the data into sub-sequences
- Process the sub-sequences in order
- When computing a gradient for sequence j, initialize the hidden state using the final state from sequence $j-1$

Teacher forcing

Maximum likelihood $\iff$ minimizing **one-step prediction errors**.

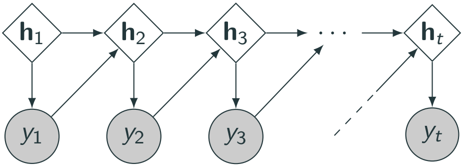
$$L(\theta) = \sum_{t=1}^{n} \left\{ y_t - \hat{y}_{t|t-1}(\theta) \right\}^2$$

Teacher forcing

Maximum likelihood $\iff$ minimizing **one-step prediction errors**.

$$L(\theta) = \sum_{t=1}^{n} \left\{ y_t - \hat{y}_{t|t-1}(\theta) \right\}^2$$

Note that we use the actual observations as **inputs** to the model!



In the neural network literature this is sometimes referred to as **teacher forcing.**

Training for multi-step prediction

With **teacher forcing**,

- **Training:** Observed data is used as input at each time step to compute one-step predictions.
- **Test:** Previous predictions are used as input to compute *k*-step predictions.

Training for multi-step prediction

With **teacher forcing**,

- **Training:** Observed data is used as input at each time step to compute one-step predictions.
- **Test:** Previous predictions are used as input to compute k-step predictions.

Alternative approach: If we are primarily interested in k-step predictions, a better approach might be to directly optimize

$$L_{k\text{-step}}(\theta; y_{1:n}) = \sum_{t=k}^{n} \left\{ y_t - \hat{y}_{t|t-k}(\theta) \right\}^2$$

Long-range dependencies

Challenges with long-range dependencies

Capturing long-range dependencies through a recurrence relation is challenging!

Challenges with long-range dependencies

Capturing long-range dependencies through a recurrence relation is challenging!

ex) A linear state space model is a simple special case of an RNN,

$$\mathbf{h}_t = W\mathbf{h}_{t-1}.$$

We know from before that the **eigenvalues of** W control the dynamic behavior:

- All eigenvalues within the unit circle $\Rightarrow$ $\mathbf{h}_t$ converges exponentially to zero.
- Some eigenvalue outside the unit circle $\Rightarrow$ norm of $\mathbf{h}_t$ explodes.

Challenges with long-range dependencies

Similarly, when training a **nonlinear RNN** we might experience:

- Vanishing gradients (operating in a "stable regime")
- Exploding gradients (operating in an "unstable regime")

Challenges with long-range dependencies

Similarly, when training a **nonlinear RNN** we might experience:

- Vanishing gradients (operating in a “stable regime”)
- Exploding gradients (operating in an “unstable regime”)

Various solutions:

- Scaling and clipping gradients
- Adding skip connections for easier information flow
- Specialized RNN architectures

Challenges with long-range dependencies

Similarly, when training a **nonlinear RNN** we might experience:

- Vanishing gradients (operating in a “stable regime”)
- Exploding gradients (operating in an “unstable regime”)

Various solutions:

- Scaling and clipping gradients
- Adding skip connections for easier information flow
- **Specialized RNN architectures**

Echo State Networks

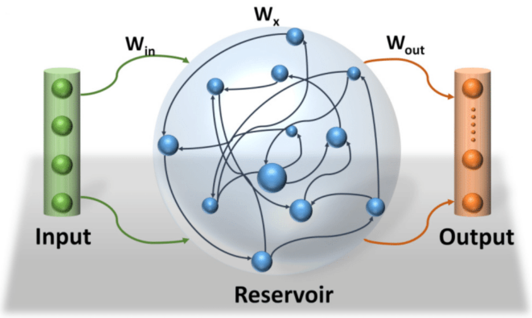
An **Echo State Network (ESN)** is a simple RNN model where the state transition matrices are **non-trainable!**

Def. Echo State Network:

$$\mathbf{h}_t = \sigma(W\mathbf{h}_{t-1} + Uy_{t-1} + b),$$
$$y_t = C\mathbf{h}_t + c + \nu_t,$$

with $\theta = \{C, c\}$.

Echo State Networks



Adapted from DOI: 10.3389/fnins.2015.00502 under license CC4.0.

Idea 1: The state vector $\mathbf{h}_t$ is though of as a "reservoir" of dynamical states that may (or may not) be useful for predicting y_t.

Echo State Networks

Idea 2: Set W, U, b randomly but in a way which ensures that $\mathbf{h}_t$ **stores information** about past values $y_{1:t-1}$. Specifically,

$$\left| \underline{\mathrm{eig}} \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} \right| \approx 1.$$

Idea 2: Set W, U, b randomly but in a way which ensures that $\mathbf{h}_t$ **stores information** about past values $y_{1:t-1}$. Specifically,

$$\left| \text{eig} \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} \right| \approx 1.$$

Echo State Networks:

- ▲ No learnable parameters in the dynamic part of the model $\Rightarrow$ no vanishing/exploding gradients!
- ▲ Extremely simple and fast to train
- ▼ Requires a large reservoir (high-dimensional $\mathbf{h}_t$) to be efficient.
- ▲ Can be used to initialize fully trainable RNNs.

Gated RNNs

Gated recurrent neural networks, such as the **LSTM** and **GRU**, are the go-to methods for dealing with long-range dependencies.

Gated RNNs

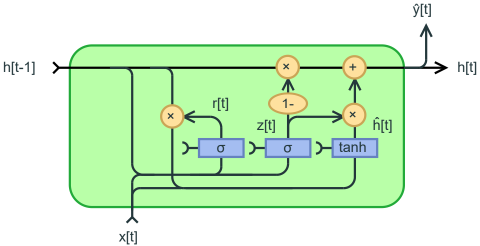
Gated recurrent neural networks, such as the **LSTM** and **GRU**, are the go-to methods for dealing with long-range dependencies.

Idea: Allow the dynamic mapping $\mathbf{h}_t = H_\theta(\mathbf{h}_{t-1}, y_{t-1})$ to be

1. **learnable**, but
2. **carefully designed** to enable gradients to propagate through time without vanishing or exploding.

This is based on **gating mechanisms** that allow the model to decide when to accumulate information and when to forget it.

ex) Gated Recurrent Unit



The GRU cell's hidden state transition $\mathbf{h}_t = H_\theta(\mathbf{h}_{t-1}, y_{t-1})$:

$$\mathbf{z}_t = \sigma(W_z\mathbf{h}_{t-1} + U_z y_{t-1} + b_z), \qquad \text{Update gate}$$

$$\mathbf{r}_t = \sigma(W_r\mathbf{h}_{t-1} + U_r y_{t-1} + b_r), \qquad \text{Reset gate}$$

$$\mathbf{c}_t = \tanh\left(W_c(\mathbf{r}_t \odot \mathbf{h}_{t-1}) + U_c y_{t-1} + b_c\right), \qquad \text{Candidate state}$$

$$\mathbf{h}_t = (1 - \mathbf{z}_t) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot \mathbf{c}_t. \qquad \text{State update}$$

Extensions

RNN extensions

We have discussed RNN models for **time series prediction**.

Model extensions and alternative use-cases:

- Stacked (deep) architectures
- Non-Gaussian likelihood (e.g., for discrete data)
- Conditioning on external inputs and context
- Time series classification
- Bidirectional connections
- Stochastic hidden layers
- ...

RNN extensions

We have discussed RNN models for **time series prediction**.

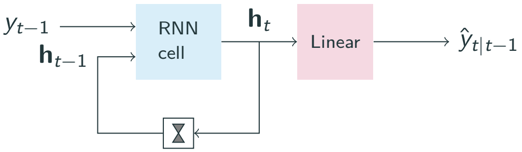
Model extensions and alternative use-cases:

→ Stacked (deep) architectures

- Non-Gaussian likelihood (e.g., for discrete data)
- Conditioning on external inputs and context
- Time series classification
- Bidirectional connections
- Stochastic hidden layers
- ...

Graphical illustration of the Jordan-Elman network

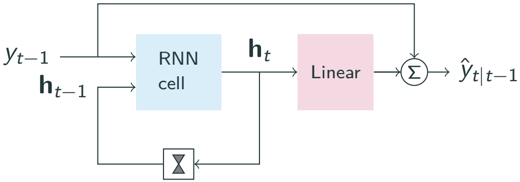
$$\mathbf{h}_t = \sigma(W\mathbf{h}_{t-1} + Uy_{t-1} + b),$$
$$\hat{y}_{t|t-1} = C\mathbf{h}_t + c,$$



Stacked RNNs

Graphical illustration of the Jordan-Elman network with residual connection

$$\mathbf{h}_t = \sigma(W\mathbf{h}_{t-1} + Uy_{t-1} + b),$$
$$\hat{y}_{t|t-1} = y_{t-1} + C\mathbf{h}_t + c,$$

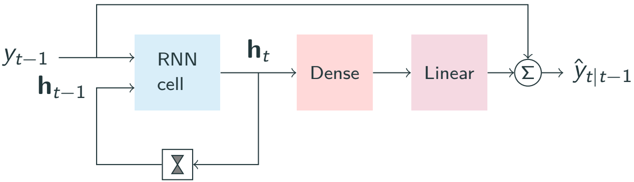


Stacked RNNs

We can build more complex (deep) models by stacking additional neural network blocks,

$$\mathbf{h}_t = H_{\theta}(\mathbf{h}_{t-1}, y_{t-1}),$$
$$\hat{y}_{t|t-1} = O_{\theta}(\mathbf{h}_t, y_{t-1}).$$

ex) Adding a densely connected layer for the output mapping

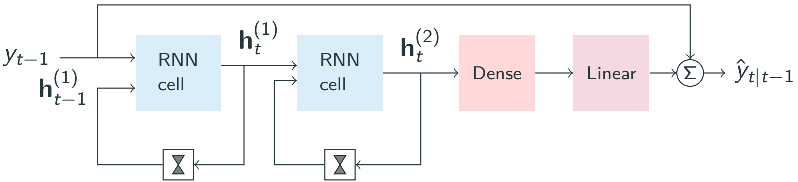


Stacked RNNs

We can build more complex (deep) models by stacking additional neural network blocks,

$$\mathbf{h}_t = H_{\theta}(\mathbf{h}_{t-1}, y_{t-1}),$$
$$\hat{y}_{t|t-1} = O_{\theta}(\mathbf{h}_t, y_{t-1}).$$

ex) Adding a second layer of RNN cells



RNN extensions

We have discussed RNN models for **time series prediction**.

Model extensions and alternative use-cases:

- Stacked (deep) architectures
- Non-Gaussian likelihood (e.g., for discrete data)
- Conditioning on external inputs and context

$\rightarrow$ Time series classification

- Bidirectional connections
- Stochastic hidden layers
- ...

Time series classification

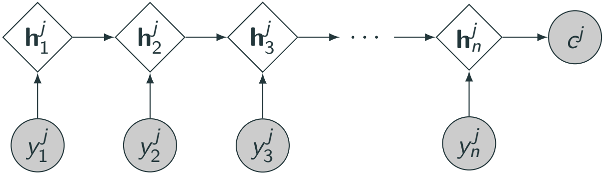
Time series prediction: Given $y_{1:n}$ build a **causal model** that can be used to **predict** y_{t+k} conditionally on $y_{1:t}$.

Time series classification

Time series prediction: Given $y_{1:n}$ build a **causal model** that can be used to **predict** y_{t+k} conditionally on $y_{1:t}$.

Alternative use case:

Time series classification: Given $\{y^j_{1:n}\}_{j=1}^S$ build a (non-causal) model that can be used to **classify** $y^\star_{1:n}$ as belonging to one of K possible classes.



ex) Learning to diagnose...

LSTM trained to **diagnose using medical time series data** from pediatric ICU patients.

- Input $y^j_{1:n}$ for ICU j is a 13-dimensional time series with measurements, such as blood preasure, blood glucose, heart rate, etc.
- Output c^j is a classification into one of $K = 128$ possible diagnoses.
- $S = 10\,401$ ICU cases (with varying length observation sequences).

Published as a conference paper at ICLR 2016

LEARNING TO DIAGNOSE WITH LSTM RECURRENT NEURAL NETWORKS

Zachary C. Lipton *†
Department of Computer Science and Engineering
University of California, San Diego
La Jolla, CA 92093, USA
zlipton@cs.ucsd.edu

David C. Kale *‡
Department of Computer Science
University of Southern California
Los Angeles, CA 90089
dkale@usc.edu

Charles Elkan
Department of Computer Science and Engineering
University of California, San Diego
La Jolla, CA 92093, USA
elkan@cs.ucsd.edu

Randall Wetzel
Laura P. and Leland K. Whittier Virtual PICU
Children's Hospital Los Angeles
Los Angeles, CA 90027
rwetzel@chla.usc.edu

ABSTRACT

Clinical medical data, especially in the intensive care unit (ICU), consist of multivariate time series of observations. For each patient visit (or *episode*), sensor data and lab test results are recorded in the patient's Electronic Health Record (EHR). While potentially containing a wealth of insights, the data is difficult to mine effectively, owing to varying length, irregular sampling and missing data. Recurrent Neural Networks (RNNs), particularly those using Long Short-Term Memory (LSTM) hidden units, are powerful and increasingly popular models for learning from sequence data. They effectively model varying length sequences and capture long range dependencies. We present the first study to empirically evaluate the ability of LSTMs to recognize patterns in multivariate time series of clinical measurements. Specifically, we consider multilabel classification of diagnoses, training a model to classify 128 diagnoses given 13 frequently but irregularly sampled clinical measurements. First, we establish the effectiveness of a simple LSTM network for modeling clinical data. Then we demonstrate a straightforward and effective training strategy in which we replicate targets at each sequence step. Trained only on raw time series, our models outperform several strong baselines, including a multilayer perceptron trained on hand-engineered features.

1 INTRODUCTION

Time series data comprised of clinical measurements, as recorded by caregivers in the pediatric intensive care unit (PICU), constitute an abundant and largely untapped source of medical insights. Potential uses of such data include classifying diagnoses accurately, predicting length of stay, predicting future illness, and predicting mortality. However, besides the difficulty of acquiring data, several obstacles stymie machine learning research with clinical time series. Episodes vary in length, with stays ranging from just a few hours to multiple months. Observations, which include sensor data, vital signs, lab test results, and subjective assessments, are sampled irregularly and plagued by missing values (Marlin et al., 2012). Additionally, long-term time dependencies complicate learning with many algorithms. Lab results that, taken together, might imply a particular diagnosis may be separated by days or weeks. Long delays often separate onset of disease from the appearance of symptoms. For example, symptoms of acute respiratory distress syndrome may not appear until 24-48 hours after lung injury (Mason et al., 2010), while symptoms of an asthma attack may present shortly after admission but change or disappear following treatment.

*Equal contributions
†Author website: http://zacklipton.com
‡Author website: http://www-scf.usc.edu/~dkale/

Efficient way of mining information from electronic health records!

Learning to Diagnose with LSTM Recurrent Neural Networks. **Zachary C. Lipton, David C. Kale, Charles Elkan, Randall Wetzel.** *ICLR*, 2016.

RNN extensions

We have discussed RNN models for **time series prediction**.

Model extensions and alternative use-cases:

- ✓ Stacked (deep) architectures
- Non-Gaussian likelihood (e.g., for discrete data)
- Conditioning on external inputs and context
- ✓ Time series classification
- Bidirectional connections
- Stochastic hidden layers
- ...

Summary

Summary lecture 10:

- **Windowing:** Speeding up gradient computations in an RNN by only processing a window of observations at a time.
- **Teacher forcing:** Using the observed data (instead of the predictions made by the model) as inputs during training. Arises naturally from a maximum likelihood perspective, but can be suboptimal if we wish to train explicitly for k-step prediction.
- **Vanishing and exploding gradients:** (In-)stability of the gradients when propagated through time.
- **Echo State Network:** RNN where the parameters related to the state update are non-learnable.
- **GRU:** Specialized gated RNN for handling long-range dependencies.