PURDUE UNIVERSITY Discovery Park

Machine Learning for Chemical Sensing

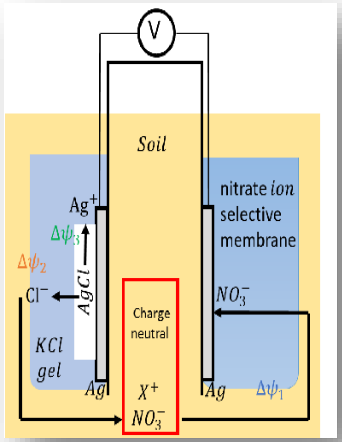
Chandra S. Mouli
Prof. Bruno Ribeiro
Department of Computer Science
Purdue University

15 May 2019

SMART
scalable manufacturing of aware & responsive thin films

Goal: Nitrate Sensing

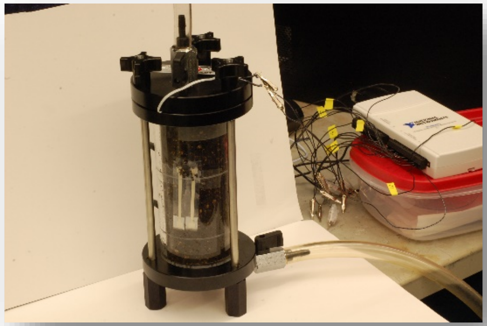
Nitrate Sensor



Prof. **Muhammad Alam (ECE)**
Xin Jin (ECE)

+

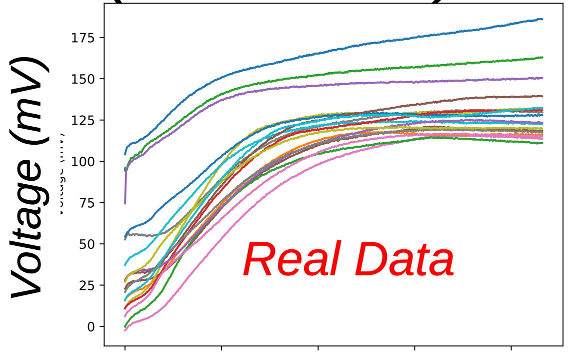
Nitrate in soil



Prof. Babak Ziaie (ECE)
Prof. Dimitri Peroulis (ECE)
Prof. Ali Shakouri (ECE)
Hongjie Jiang (ECE)
Rahim Rahimi (ECE)

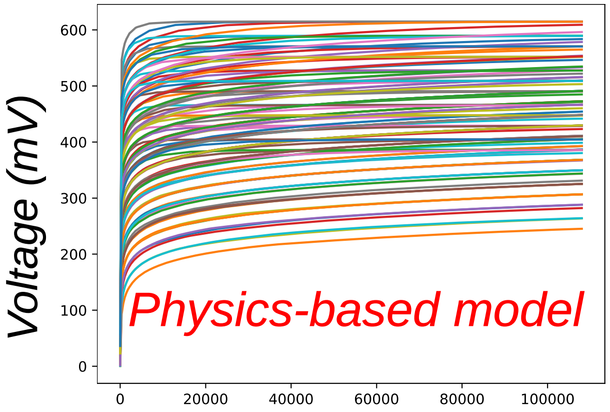
=

Sensor voltages (time series)



Time (seconds)

Not same transient



Time (seconds)

Challenges:

- Battery-powered (sensor must sleep often)
- Low-powered sensor (noise)
- Physics-based model ≠ real-world data
- Wireless communication (data transmission drains battery)

SMART scalable manufacturing of aware & responsive thin films

Advantages of Fast Chemical Sensing

Voltage potential takes hours to stabilize

- *Measuring data drains battery*
- *Communication of data drains battery*
- *Physics-model ≠ real-world data (currently)*

Fast chemical sensing:

- *Faster answers* ✓
- *Saves battery* ✓
  - *Short measurement time*
  - *Reduces wireless communication*

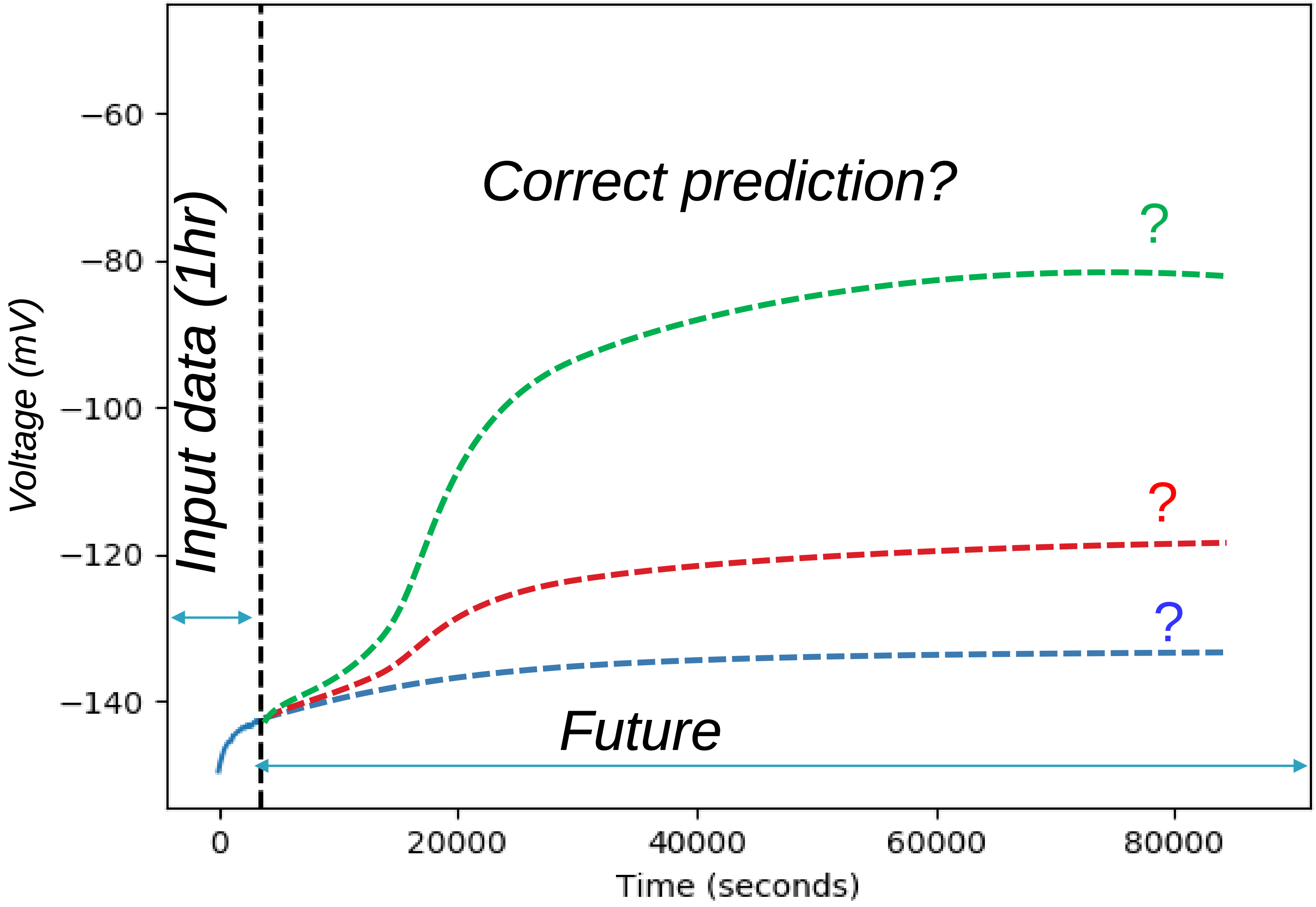
Q: *Can we measure Nitrate concentrations in minutes rather than hours?*

- *Fast Chemical Sensing,*
  Discovery Park Integrative Data Science Initiative, June 2018-June 2020

SMART scalable manufacturing of aware & responsive thin films

Challenges of Fast Chemical Sensing

PURDUE UNIVERSITY Discovery Park

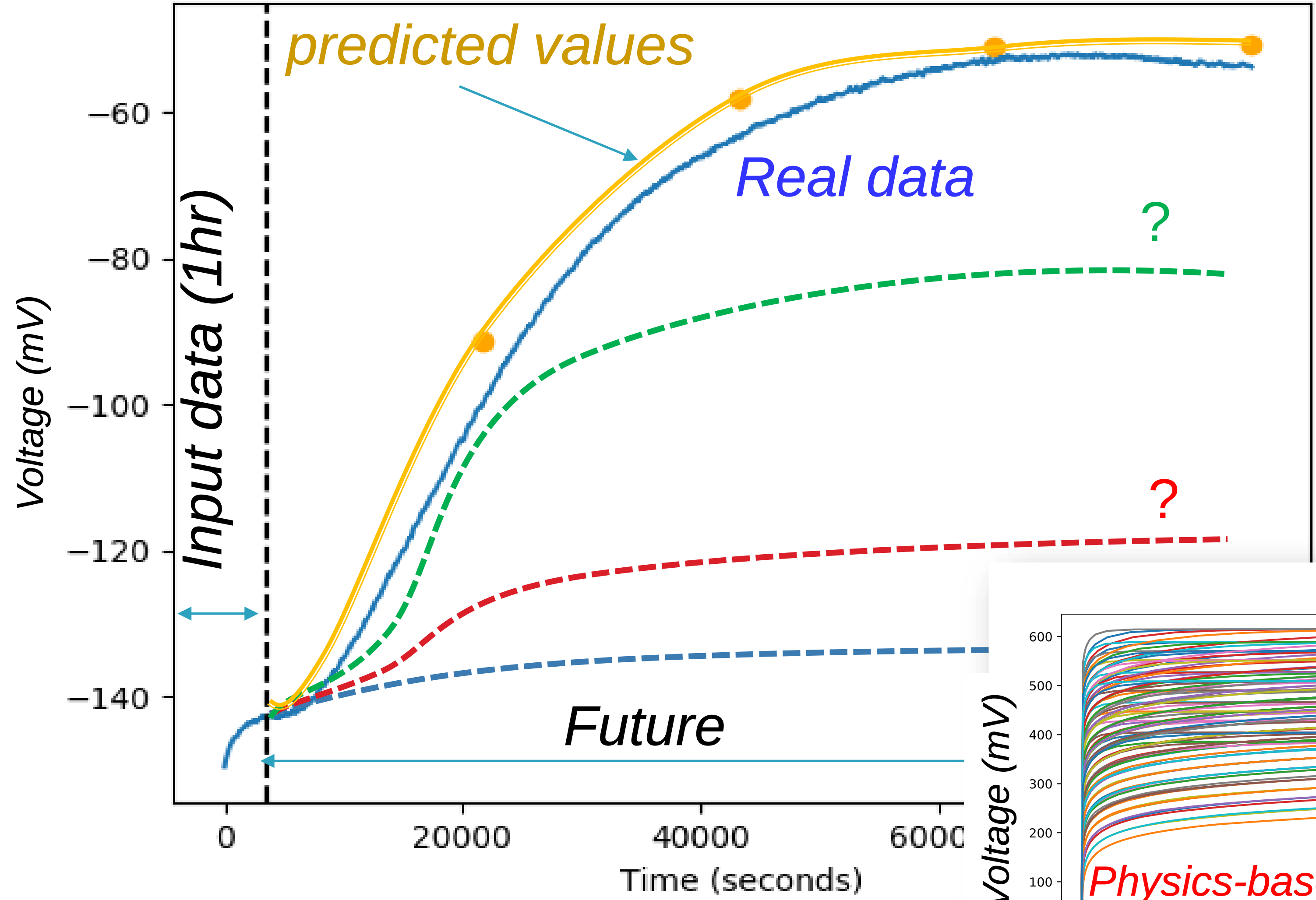


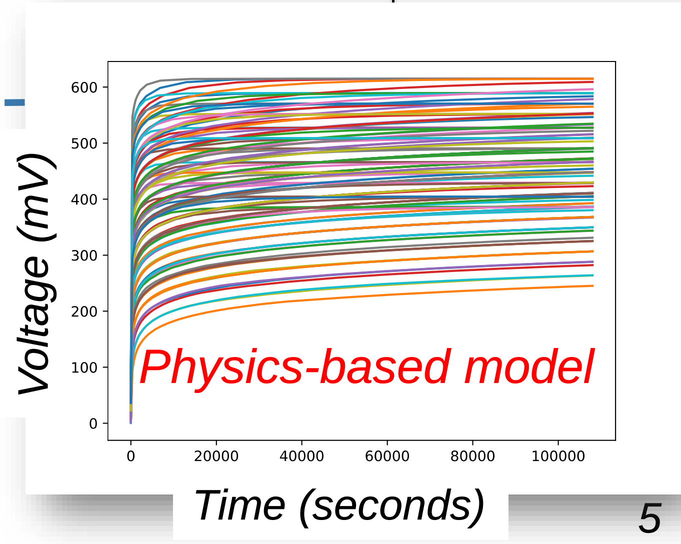
SMART
scalable manufacturing of aware & responsive thin films

Fast Chemical Sensing: Initial Results

PURDUE UNIVERSITY Discovery Park

Nitrate sensing under highly controlled lab conditions





SMART scalable manufacturing of aware & responsive thin films

Machine Learning Challenges

- ***Key Challenge:***
  - *Accurate chemical sensors readings depend on **slow chemical processes***

- ***Impact:***
  - *Chemical sensor models not yet capable of **fast estimation** of target chemical concentrations*

- ***Pure Data-driven Solutions:***
  - *Not enough training data; not enough environmental conditions*

SMART
scalable manufacturing of
aware & responsive thin films

Some Basic Concepts

Definition: Joint Probability

- Consider a **sequence** of random variables:

$X_1, \ldots, X_n$ *with* $X_i \in \Omega$ $\text{e.g. } \Omega = \{$ $\}$

with joint probability distribution

$$P(X_1, \ldots, X_n)$$

- The joint probability is a function

$$P : \Omega^n \rightarrow [0,1]$$ *(w/ normalization)*

  - *P* takes an ordered sequence and outputs a value between zero and one (w/ normalization)

SMART scalable manufacturing of aware & responsive thin films

Conditional Probability

- Consider a **sequence** of random variables:

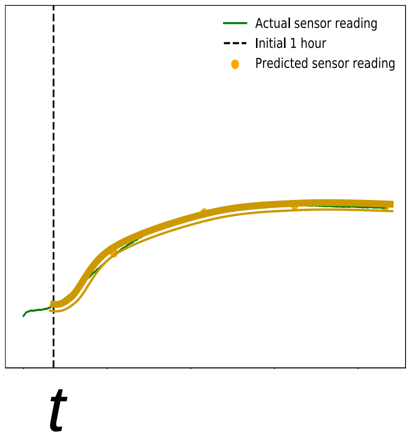
$$X_1, \dots, X_n \qquad \text{with } X_i \in \Omega \quad \text{e.g. } \Omega = \{\ \ \}$$

with joint probability distribution

$$P(X_1, \dots, X_n)$$

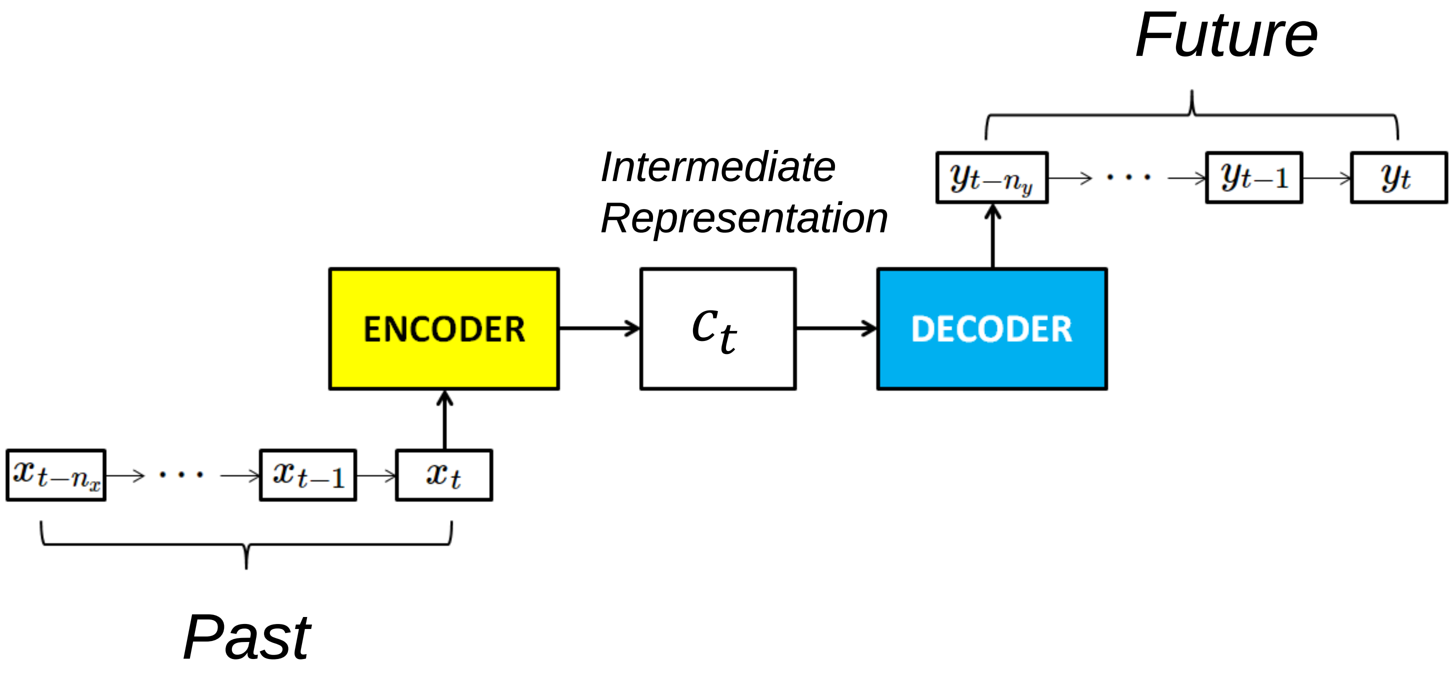
From the joint probability, we can compute the conditional probability:

$$P(X_n, \dots, X_t | X_t, \dots, X_1)$$



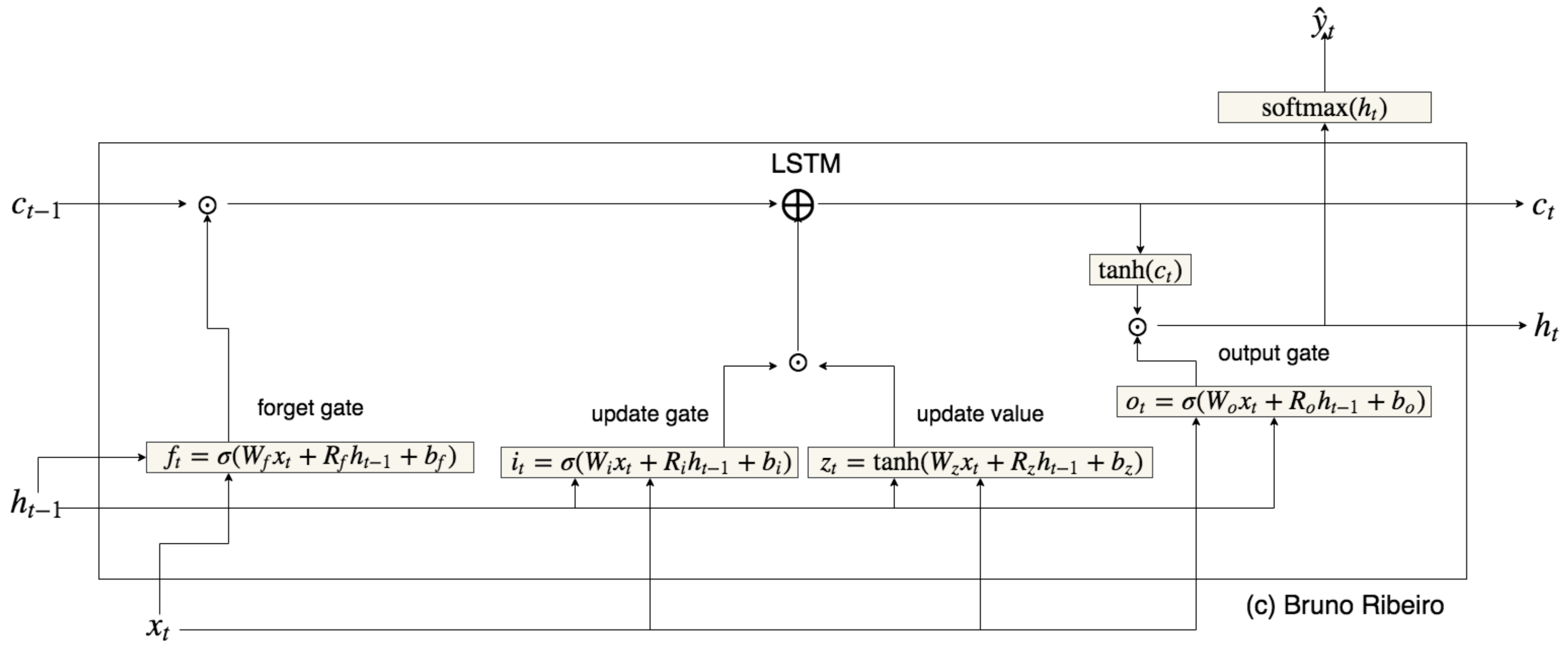
A Model for the Conditional Probability

- 2 models, one encoder, another decoder



Encoding/Decoding Sequences

- Example of encoder/decoder: Long Short Term Memory (LSTM)



$\hat{y}_t$
$\text{softmax}(h_t)$
LSTM
c_{t-1}
c_t
$\tanh(c_t)$
h_t
output gate
$o_t = \sigma(W_o x_t + R_o h_{t-1} + b_o)$
forget gate
$f_t = \sigma(W_f x_t + R_f h_{t-1} + b_f)$
update gate
$i_t = \sigma(W_i x_t + R_i h_{t-1} + b_i)$
update value
$z_t = \tanh(W_z x_t + R_z h_{t-1} + b_z)$
h_{t-1}
x_t
(c) Bruno Ribeiro

Details: Learning Sequence Models

Backpropagation-Through-Time in a Long Short Term Memory (LSTM) Neural Network

In theory, the structure of the RNN should allow it to make long-term predictions due to the variable collision. In practice, this is often not possible due to vanish gradients. That is, in standard RNNs the signal at H_t from its effect at the output of much later Hs is lost in time due to the multiplication of small gradients.

Long Short Term Memory (LSTM) is a type of RNN that can learn long-term dependencies. The trick is to have an extra hidden state C_t with gates that determine how the information must be propagated through time. These gates allow the network to remember or to forget information. The graphical model of an LSTM is essentially the same as the graphical model of the RNN with an extra memory variable C_t.

Let $x_t \in \mathbb{R}^M$ be the the input at time t. Let N be the number of neurons in the hidden states of the LSTM. Then we get:

- Input Weights: $W_z, W_i, W_f, W_o \in R^{N\times M}$
- Recurrent Weights: $R_z, R_i, R_f, R_o \in R^{N\times N}$
- Bias Weights: $b_z, b_i, b_f, b_o \in R^N$

SMART scalable manufacturing of aware & responsive thin films

Details: Learning Sequence Models

The forward pass in the LSTM is given by:

$$\begin{aligned}
z_t &= \tanh(W_z x_t + R_z h_{t-1} + b_z) \\
i_t &= \sigma(W_i x_t + R_i h_{t-1} + b_i) \\
f_t &= \sigma(W_f x_t + R_f h_{t-1} + b_f) \\
c_t &= z_t \odot i_t + c_{t-1} \odot f_t \\
o_t &= \sigma(W_o x_t + R_o h_{t-1} + b_o) \\
h_t &= \tanh(c_t) \odot o_t,
\end{aligned}$$

where "$\odot$" is the element-wise multiplication and "σ" is the sigmoid function.

We will simplify our LSTM by directly mapping from the hidden variables h_t to the output probabilities through a softmax:

$$\hat{y}_t = \text{softmax}(h_t).$$

SMART scalable manufacturing of aware & responsive thin films

Details: Learning Sequence Models

Backpropagation-through-Time

The negative log-likelihood error for an input sequence $\mathbf{x} = (x_1, \ldots, x_T)$ of one-hot encoded values is:

$$L_{total}(\hat{\mathbf{y}}, \mathbf{x}) = \sum_{t=1}^{T} L(\hat{y}_t, x_t)$$

where

$$L(\hat{y}_t, x_t) = -\sum_{i=1}^{N} x_t[i] \cdot \log(\hat{y}_t[i])$$

and is a one-hot encoded vector $x_t = (0, \ldots, 1, \ldots, 0)$ of dimension N.

Define $L(\hat{y}_t, x_t)$ as L_t to simplify the notation. To backpropagate, we need to compute the gradient of the loss with respect to each predicted output $\hat{y}_t$, for all $t = 1, \ldots, T$ for each of the variables. We will start with the hidden variables.

Derivatives with respect to h_t

$$\frac{\partial L_{total}}{\partial h_t} = \frac{\partial L_1}{\partial h_t} + \frac{\partial L_2}{\partial h_t} + \ldots + \frac{\partial L_t}{\partial h_t} + \frac{\partial L_{t+1}}{\partial h_t} + \ldots + \frac{\partial L_T}{\partial h_t}, \quad t = 1, \ldots, T.$$

Since we assume that the future hidden states are not affecting the past observations, $\frac{\partial L_j}{\partial h_t} = 0$ for all $t > j$, and we can limit the influence of h_t to $j \geq t$. The derivative of the hidden state h_t w.r.t. L_j is

$$\frac{\partial L_t}{\partial h_t} = (x_t - \hat{y}_t),$$

since x_t is a one-hot vector and $\hat{y}_t[i]$ is the probability $x_t[i] = 1, i = 1, \ldots, N$.

This gives:

$$\frac{\partial L_{total}}{\partial h_t} = \sum_{j=t}^{T} \frac{\partial L_j}{\partial h_t}.$$

We will also note that for $j = t + m, m \geq 1$,

$$\frac{\partial L_{t+m}}{\partial h_t} = \frac{\partial L_{t+m}}{\partial h_{t+m}} \cdot \frac{\partial h_{t+m}}{\partial h_{t+m-1}} \cdots \frac{\partial h_{t+1}}{\partial h_t},$$

where "·" is the matrix multiplication operator.

The rest of the derivation is mostly bookeeping. The derivatives of h_{t-1} w.r.t. h_t are:

$$\frac{\partial h_t}{\partial h_{t-1}} = \tanh(c_t) \odot \frac{\partial o_t}{\partial h_{t-1}} + o_t \odot \frac{\partial \tanh(c_t)}{\partial h_{t-1}}$$

Noting that

$$\frac{\partial \tanh(c_t)}{\partial h_{t-1}} = (1 - \tanh^2(c_t)) \odot \frac{\partial c_t}{\partial h_{t-1}}$$

and

$$\frac{\partial c_t}{\partial h_{t-1}} = \frac{\partial z_t}{\partial h_{t-1}} \odot i_t + \frac{\partial i_t}{\partial h_{t-1}} \odot z_t + \frac{\partial c_{t-1}}{\partial h_{t-1}} \odot f_t + \frac{\partial f_t}{\partial h_{t-1}} \odot c_{t-1}$$

Since $\frac{\partial c_{t-1}}{\partial h_{t-1}} = 0$ as c_{t-1} only depends on h_{t-2} and the future does not affect the past, then

$$\frac{\partial h_t}{\partial h_{t-1}} = R_o \frac{\partial h_t}{\partial o_t} + R_z \frac{\partial h_t}{\partial z_t} + R_i \frac{\partial h_t}{\partial i_t} + R_f \frac{\partial h_t}{\partial f_t}$$

Details: Learning Sequence Models

Derivatives with respect to c_t

Another key variable in the backpropagation algorithm is c_t. Similar to h_t,

$$\frac{\partial L_{total}}{\partial c_t} = \frac{\partial L_1}{\partial c_t} + \frac{\partial L_2}{\partial c_t} + \ldots + \frac{\partial L_t}{\partial c_t} + \frac{\partial L_{t+1}}{\partial c_t} + \ldots + \frac{\partial L_T}{\partial c_t}, \quad t = 1, \ldots, T.$$

Similarly, since we assume that the memory hidden states are not affecting the current or past observations, $\frac{\partial L_j}{\partial c_t} = 0$ for all $t \geq j$, and we can limit the influence of c_t to $j > t$. This gives

$$\frac{\partial L_{total}}{\partial c_{t-1}} = \sum_{j=t}^{T} \frac{\partial L_j}{\partial c_{t-1}}, \quad t = 2, \ldots, T.$$

Note that

$$\frac{\partial L_t}{\partial c_{t-1}} = \frac{\partial L_t}{\partial c_t}\frac{\partial c_t}{\partial c_{t-1}} = \frac{\partial L_t}{\partial c_t} \odot f_t,$$

where

This imples for $m \geq 1$,

$$\frac{\partial L_{t+m}}{\partial c_{t-1}} = \frac{\partial L_{t+m}}{\partial c_{t+m}}\frac{\partial c_{t+m}}{\partial c_{t+m-1}} \cdots \frac{\partial c_t}{\partial c_{t-1}} = \frac{\partial L_t}{\partial c_t} \odot f_{t+m} \odot \cdots \odot f_t,$$

that is, whether the memory hidden variable c_{t-1} influences the long-term future or not depends on the multiplications of the forget gate.

Looking at the foward equations, we can directly derive

$$\frac{\partial L_t}{\partial c_t} = \frac{\partial L_t}{\partial h_t}\frac{\partial h_t}{\partial c_t} = \frac{\partial L_t}{\partial h_t} \odot o_t \odot (1 - \tanh^2(c_t))$$

This makes the total gradient for c_t as

$$\frac{\partial L_{total}}{\partial c_t} = \frac{\partial L_t}{\partial h_t} \odot o_t \odot (1 - \tanh^2(c_t)) + \sum_{m=1}^{T-t} \frac{\partial L_{t+m}}{\partial c_{t+m}} \odot f_{t+m} \odot \cdots \odot f_{t+1}$$

The remaining derivatives are all local, do not propagate over time

For instance,

$$\frac{\partial L_{total}}{\partial W_z} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W_z} = \sum_{t=1}^{T}\sum_{m=0}^{T-t} \frac{\partial L_{t+m}}{\partial z_t}\frac{\partial z_t}{\partial W_z} = \sum_{t=1}^{T} \frac{\partial z_t}{\partial W_z}\left(\sum_{m=0}^{T-t} \frac{\partial L_{t+m}}{\partial z_t}\right),$$

noting that

$$\frac{\partial L_{t+m}}{\partial z_t} = \frac{\partial L_{t+m}}{\partial c_t}\frac{\partial c_t}{\partial z_t} = \frac{\partial L_{t+m}}{\partial c_t} \odot i_t$$

Note that during backpropagation, at time t we only need to know

$$\left(\sum_{m=0}^{T-t} \frac{\partial L_{t+m}}{\partial c_t}\right) \odot i_t$$

to perform the above calculation, which can be calculated cumulatively at each backward step.

In a similar fashion, we can calculate the derivatives of the other parameters. Some will be easier to compute with

$$\left(\sum_{m=0}^{T-t} \frac{\partial L_{t+m}}{\partial c_t}\right),$$

others will be easier to compute with

$$\left(\sum_{m=0}^{T-t} \frac{\partial L_{t+m}}{\partial h_t}\right).$$

Because of the simpler form of $\frac{\partial L_{t+m}}{\partial c_t}$, we will use c_t whenever we can.

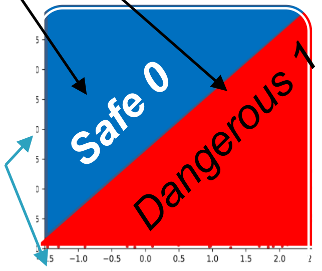
Deep Learning + Physics

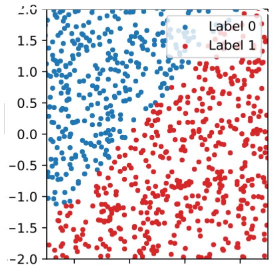
Illustration of approach:

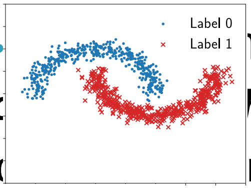
Chemical concentration

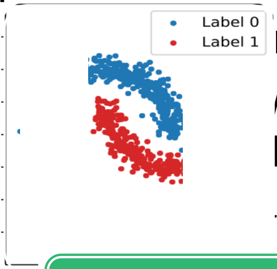
Electrical signal inputs

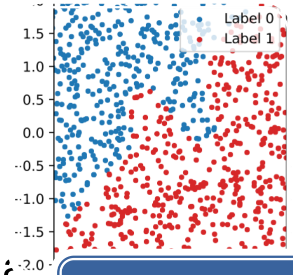
Real data under all soil / weather conditions:











n:

- **Deep Learning Mo** **hysics as prior**
  - Data-driven pred r soil / weather c enough data
  - g., Nortl rms)
  - P rediction veather where c le
  - g., Sout rms)

Autom from se hen da will be r ble

Final Predictions Combine Physics & Data

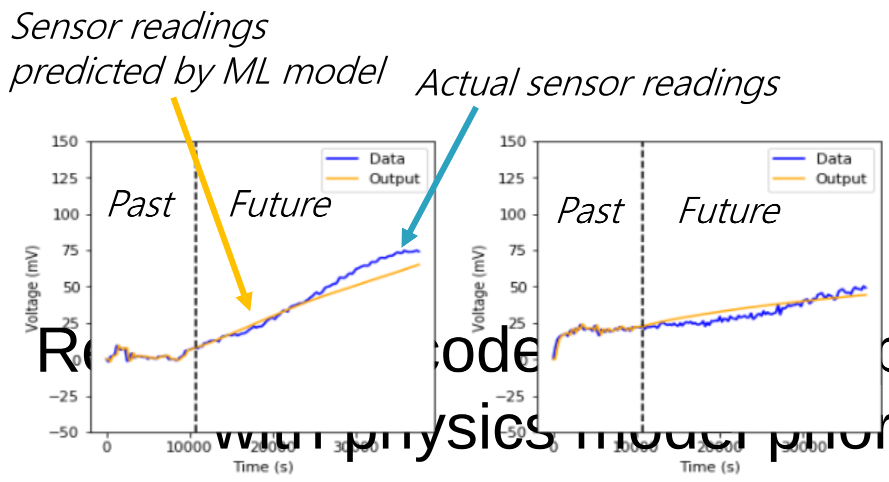
Physics-based Approximation

Neural net learns physics by example

Limited Real Sensor Data

Final Neural Network Predictions

Re code pproach with physics model priors



Learning over Sets

A sensor will have multiple detectors

How to combine a set of measurements?

How should we model sets?

Probabilities over Unordered Sequences (Sets)

- Consider a **set** of random variables:

$$X_2 \quad X_1 \quad X_4 \quad X_3 \quad X_5$$

with $X_i \in \Omega$

how should we define their joint probability distribution?

- Definition: The probability distribution P such that

$$P(X_1, \ldots, X_n) = P(X_{\pi(1)}, \ldots, X_{\pi(n)})$$

is true for any permutation π of (1,...,n)

Useful reference: (Murphy et al., ICLR 2019) Janossy Pooling: Learning Deep Permutation-Invariant Functions for Variable-Size Inputs

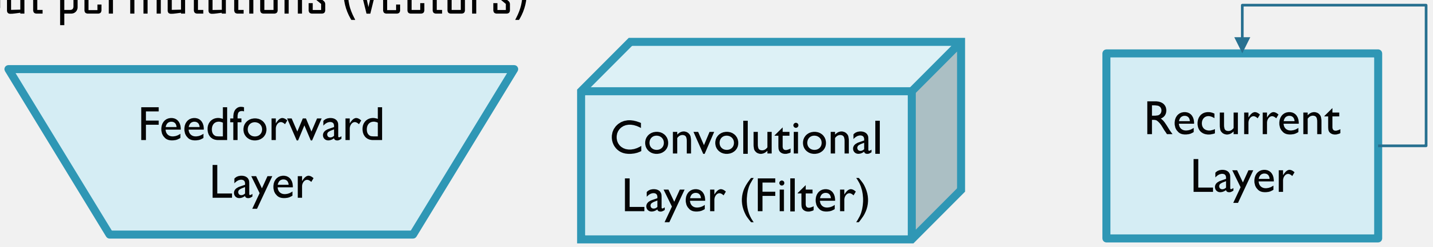
SMART scalable manufacturing of aware & responsive thin films

Pooling for Input Invariances

Deep learning has four basic building blocks:

Learnable

Sensitive to input permutations (vectors)



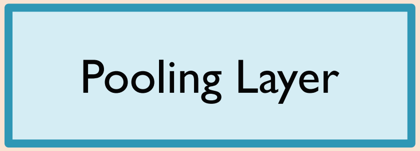
and

Deterministic

Insensitive to input permutations (set inputs)

$f_{\text{sum-pooling}}(\mathbf{h}) = \sum_i h_i$

$f_{\text{max-pooling}}(\mathbf{h}) = \max(h_1, \ldots, h_{|\boldsymbol{h}|})$

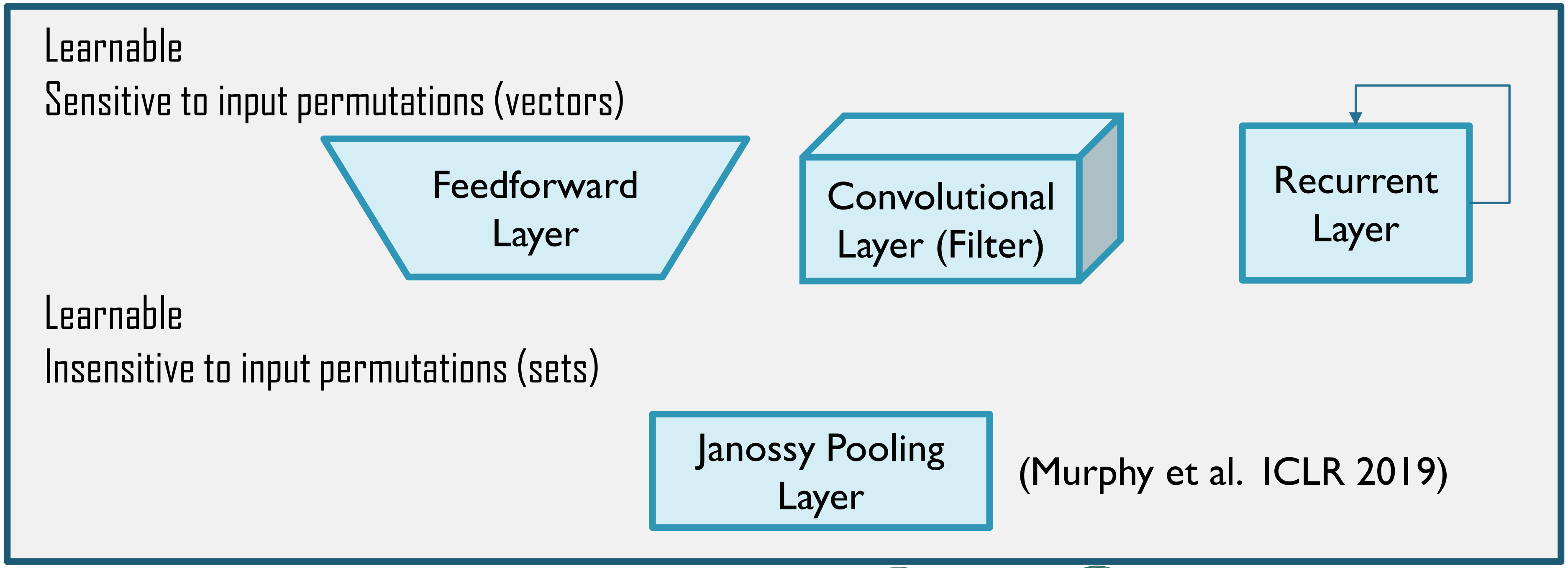


$f_{\text{mean-pooling}}(\mathbf{h}) = \frac{1}{|\boldsymbol{h}|} \sum_i h_i$

$f_{\text{min-pooling}}(\mathbf{h}) = \min(h_1, \ldots, h_{|\boldsymbol{h}|})$

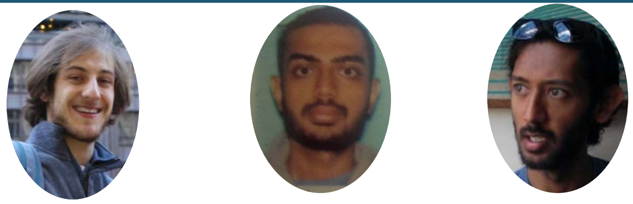
- Growing interest in developing learnable pooling layers (Zaheer et al., 2017), (Ravanbakhsh et al., 2017), (Vinyals et al., 2016) , (Rezatofighi et al., 2018), (Lee et al., 2017), (Lee et al., 2018)

A Seed Growing...


Learnable
Sensitive to input permutations (vectors)

Learnable
Insensitive to input permutations (sets)

(Murphy et al. ICLR 2019)



Joint work with R. Murphy, B. Srinivasan*, V. Rao*

(Murphy et al., ICLR 2019) R. Murphy, B. Srinivasan, V. Rao, and B. R.,
"Janossy Pooling: Learning Deep Permutation-invariant Functions for Variable-size Inputs", *ICLR* 2019.

Details

Janossy Pooling (Murphy et al. ICLR 2019)

- **Janossy pooling:**

$$\overline{\overline{f}}(\boldsymbol{h};\boldsymbol{\theta}) = \frac{1}{|\boldsymbol{h}|!}\sum_{\pi\in\Pi} \vec{f}(\boldsymbol{h}_\pi;\boldsymbol{\theta})$$

where $\vec{f}$ is a learnable permutation-sensitive function.

Many choices of $\vec{f}$:

- RNNs, LSTMs, GRUs (*for variable-size inputs*)
- Feedforward Networks
- Convolutional Neural Networks (CNNs)

Computational Tractability of Janossy Pooling

Janossy pooling work describes 3 approaches to tractably learn $\overline{\overline{f}}$

1. Tractability through canonical ordering

2. Tractability through k-ary dependencies

3. Tractability through stochastic optimization

Janossy pooling provides unified framework:

All literature falls into these 3 categories

1. Tractability through canonical ordering

Details

- Learning-to-sort input: (Vinyals et al., 2016) , (Rezatofighi et al., 2018), (Lee et al., 2017), (Lee et al., 2018)

- Find the best permutation
  - Discrete optimization problem
  - O(n!) complexity

- Special cases:
  - Max-pooling
  - Min-pooling

2. Tractability through k-ary dependencies

Details

- Assume k-ary dependencies over input
- Sums over $\binom{n}{k}$ combinations of elements (rather than n!)
- Special cases:
  - Deep Sets (Zaheer et al., 2017): k=1, n scaling
  - Mean-pooling: k=1, 1/n scaling

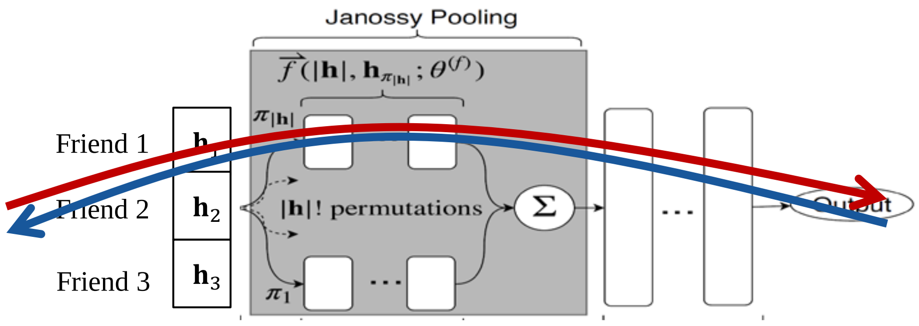
3. Tractability through Stochastic Optimization

Details

- SGD: **standard** Stochastic Gradient Descent
  1. sample a mini-batch
  2. backpropagate to compute gradients of batch
  3. update model with one gradient descent step
  4. GOTO 1:

- ***π*-SGD** (*as fast as SGD*)
  1. sample a mini-batch
  2. **sample one permutation $\pi^{(j)}$** for each example $\mathbf{x}^{(j)}$ in mini-batch
  3. perform forward pass with the sampled permutation
  4. backpropagate to compute gradients
  5. update model with one gradient descent step
  6. GOTO 1:

Backpropagation computes gradient over sampled permutation only



Forward pass chooses one permutation randomly

π-SGD

Details

- **Theorem 2.1** (Murphy et al. 2018): Learns proper permutation-invariant model
  - But model "changes"!

- Explains great results of LSTMs as pooling in graph models
  - *GraphSAGE (Hamilton, Ying & Leskovec, 2017)*
  - *Deep Collective Inference (Moore & Neville, 2017)*
  - more ways to improve model trained by π-SGD (see paper)

- inference at test time: average outputs over all permutations
  - avg. 5 sampled permutations enough for GraphSAGE tasks