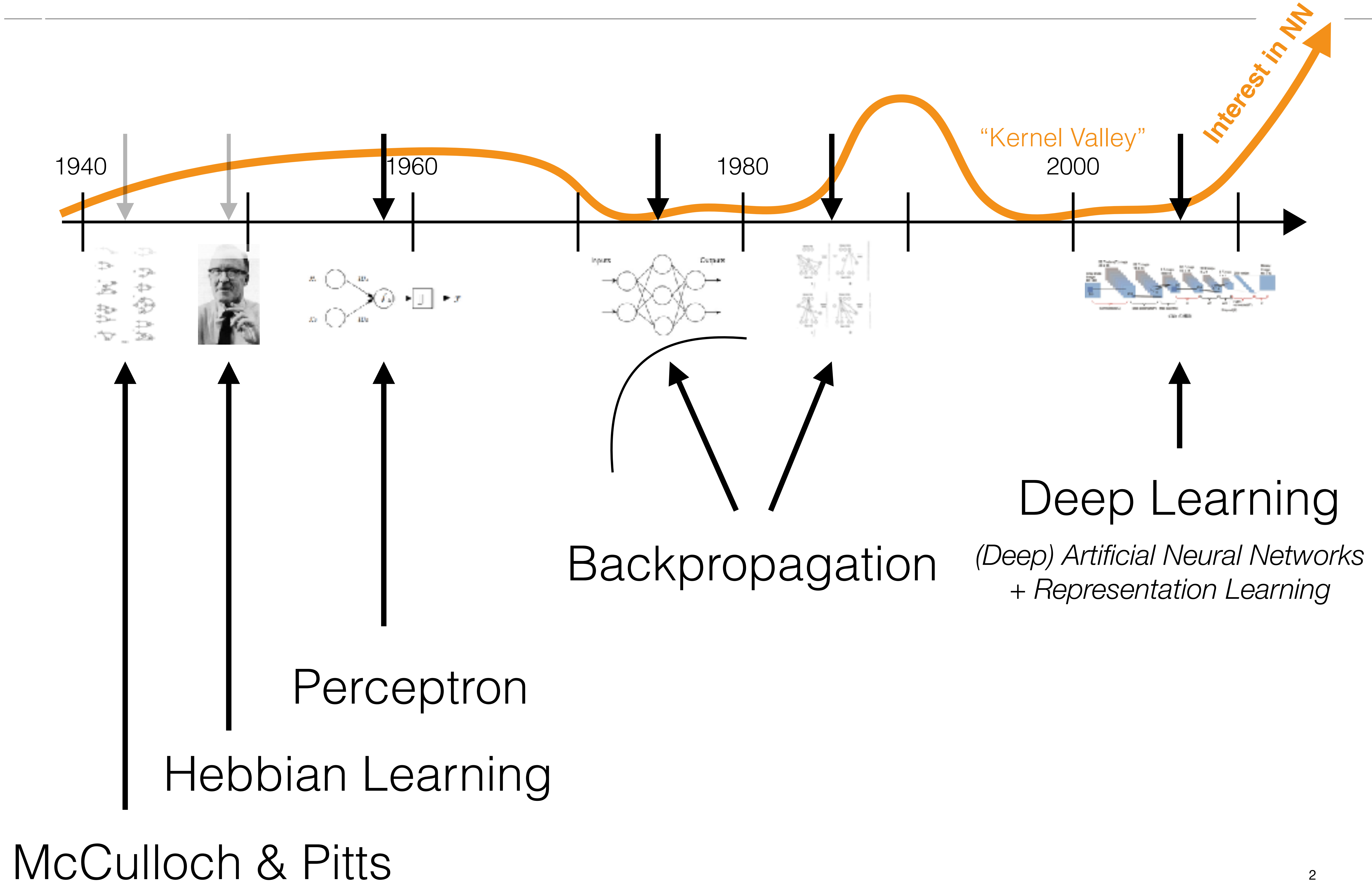
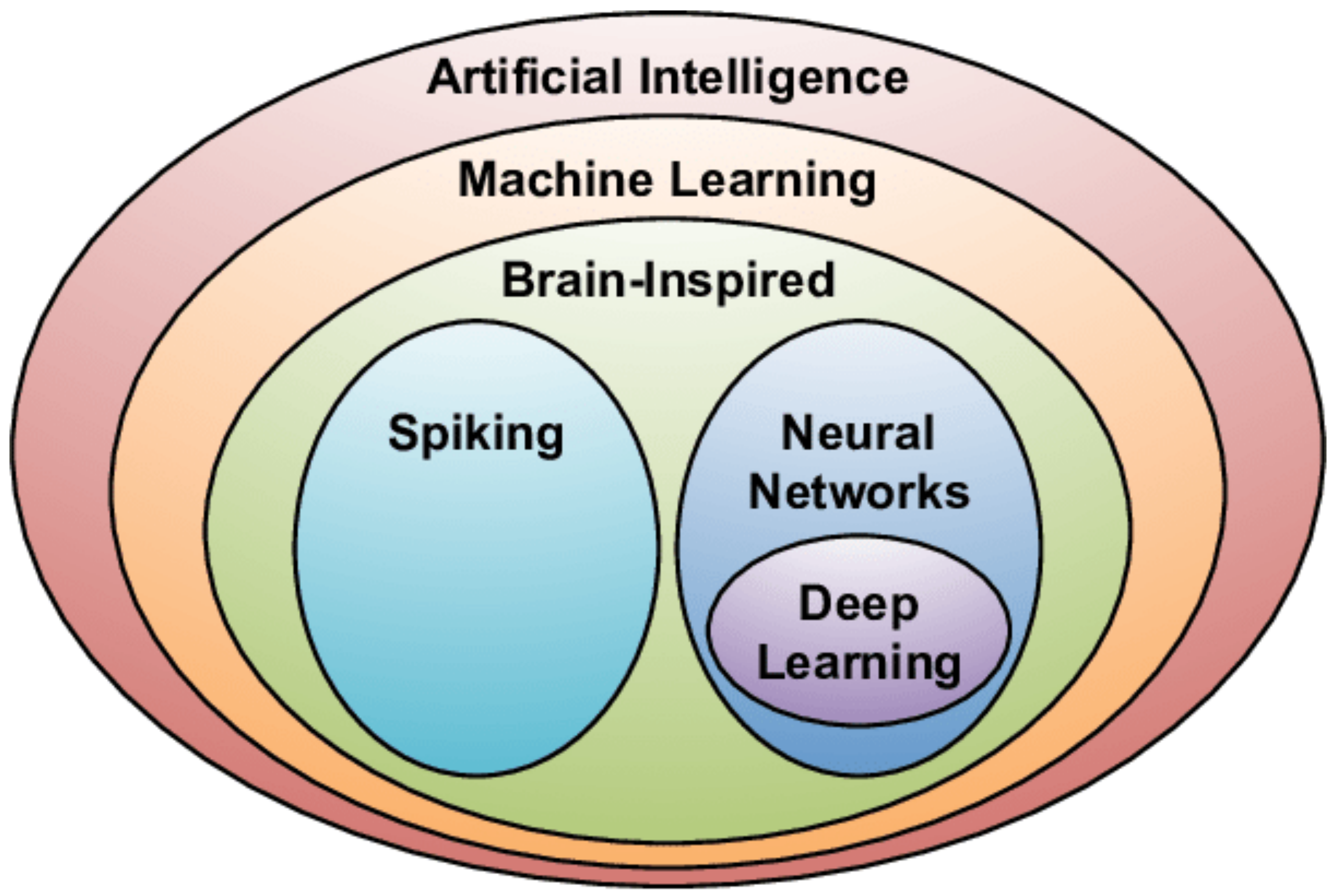
ECE209AS (Winter 2021)

Lecture 3: Deep Learning for Sensor Information Processing

Mani Srivastava
mbs@ucla.edu
Networked & Embedded Systems Lab
ECE & CS Departments
UCLA

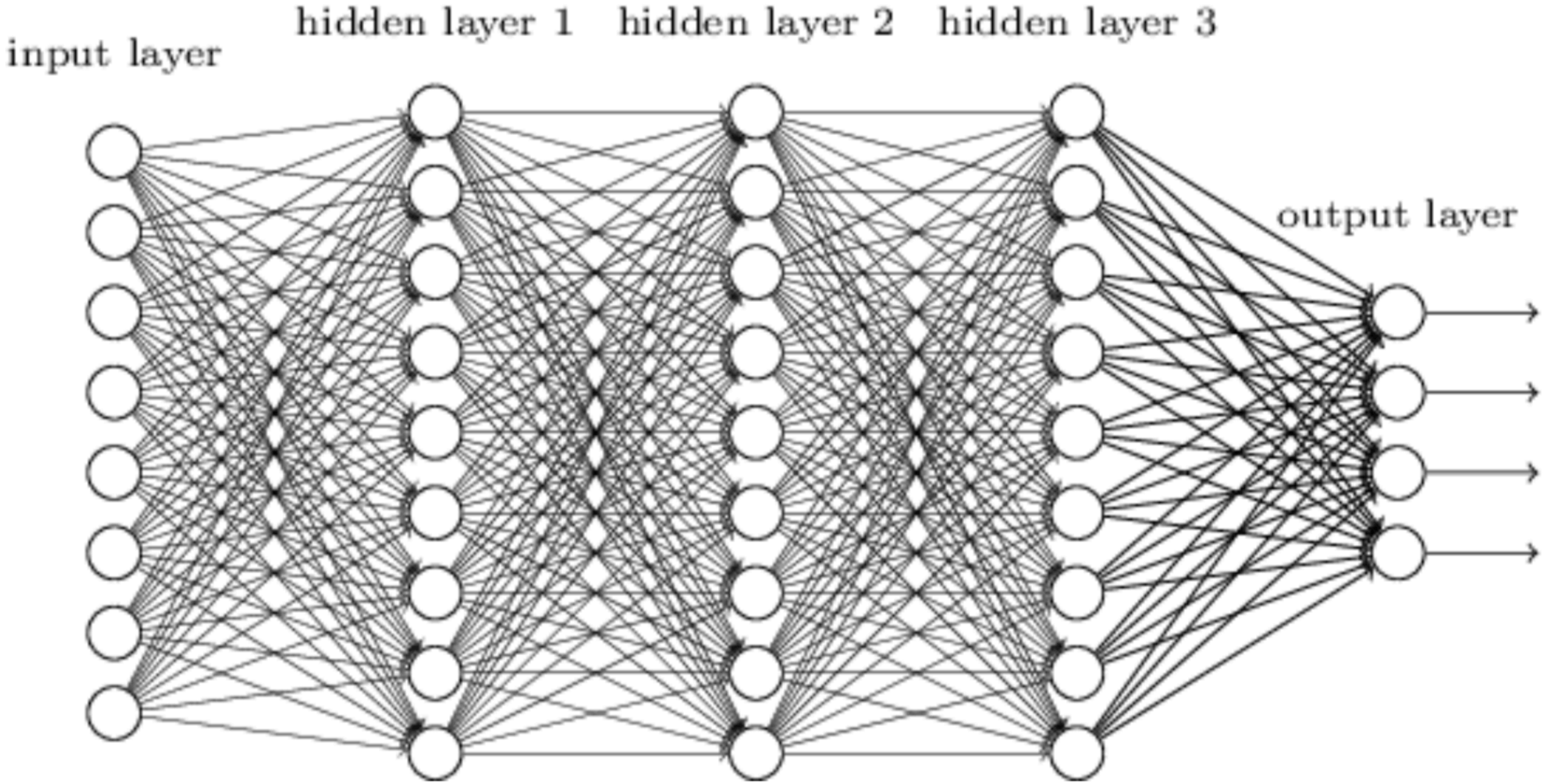
Machine Learning Revolutionized by Artificial Neural Networks



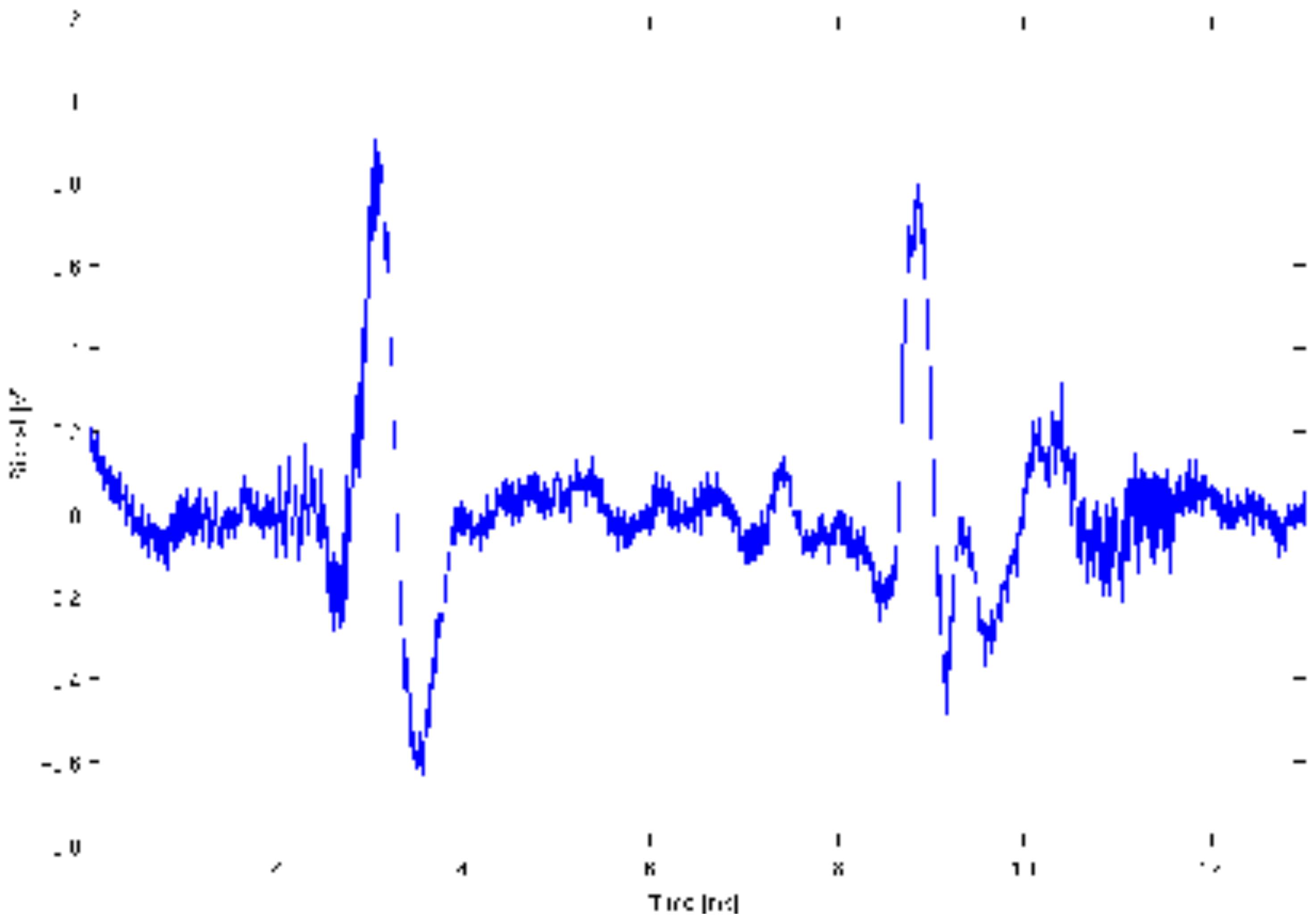


A Quick Backgrounder…

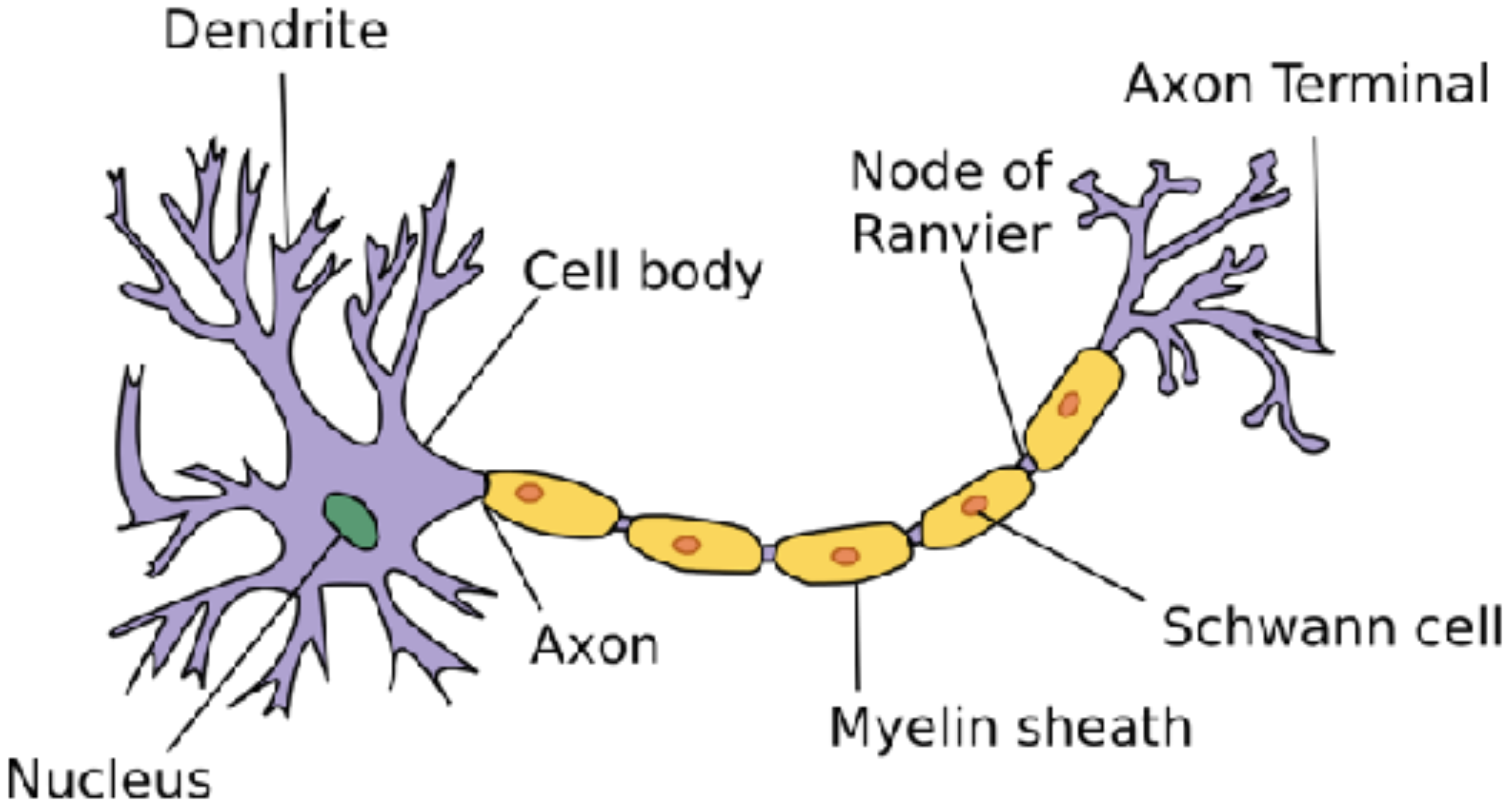
A Deep Feedforward Artificial Neural Network



Simple Model of a Biological Neuron

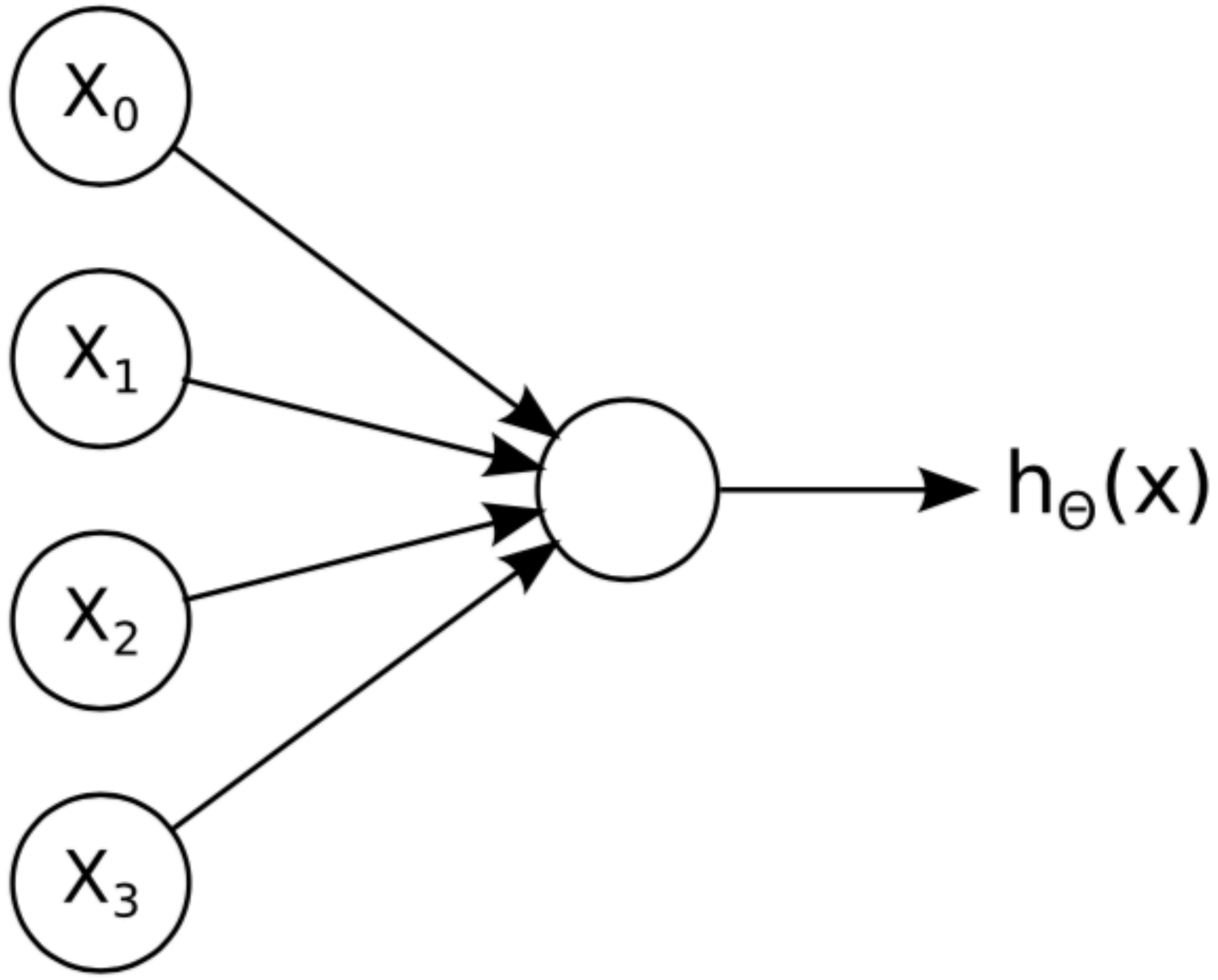


[Reichardt et al. 2009]



- Human brain contains approx. 10^{10} neurons, with ~ 10^{14} connections!
- Each neuron is connected to thousands of others and sends signals based on it's input
- With changing input, the output potential on the axon produces a "spike"
  - In easy terms this corresponds to a binary output (spike / no spike)
  - The perceptron is a very simple model for this behaviour of a neuron

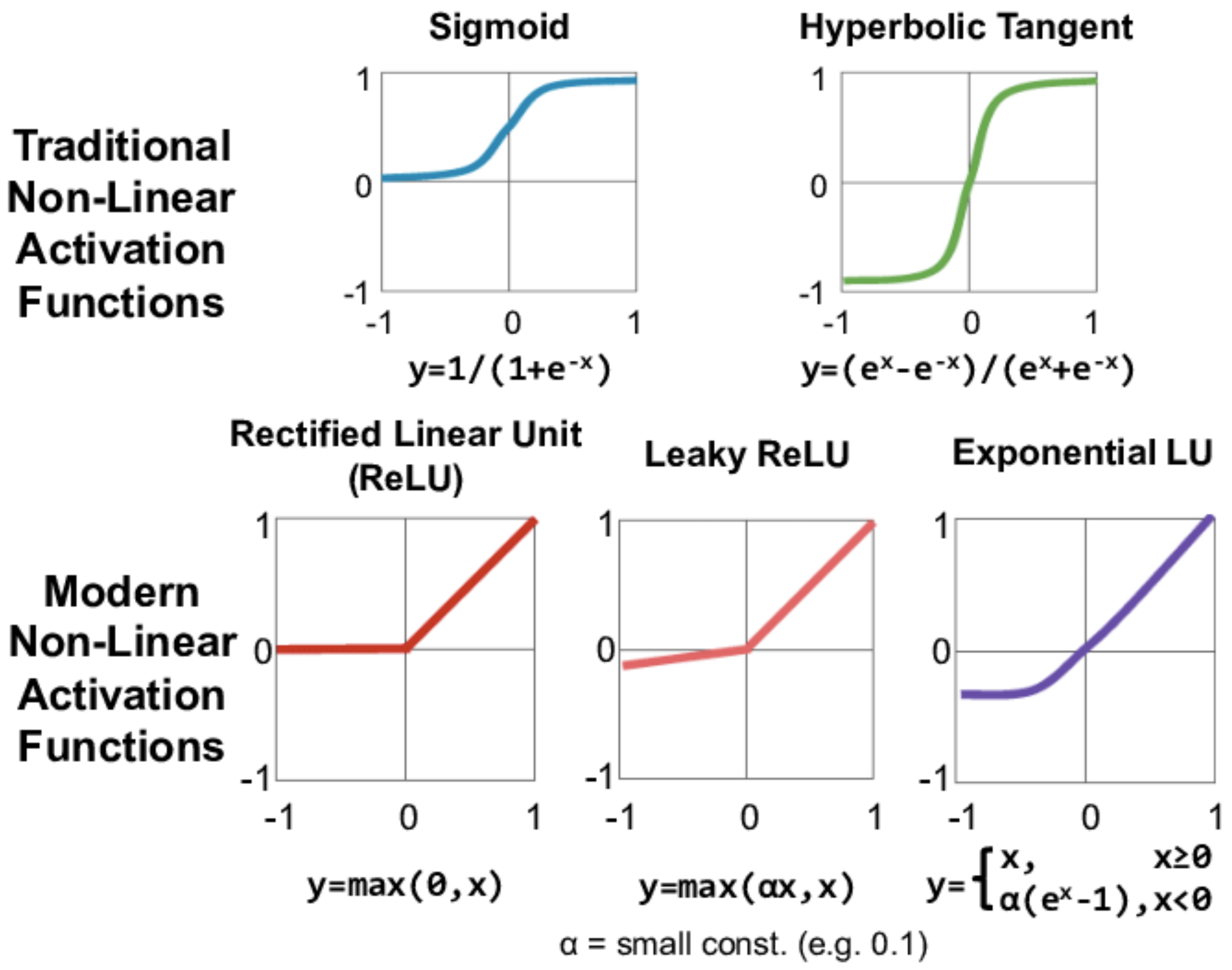
Simple Model of an Artificial Neuron



$$\mathbf{x} = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \boldsymbol{\Theta} = \begin{bmatrix} \Theta_0 \\ \Theta_1 \\ \Theta_2 \\ \Theta_3 \end{bmatrix}$$

- Has a number of “weighted" inputs $\mathbf{x}$
  - represented as a multidimensional array or “tensor” (different from tensors in math/physics)
- Calculates an activation $\mathrm{h}(\mathbf{x}) = \mathrm{g}(\boldsymbol{\Theta}^T \mathbf{x})$
- $\mathrm{g}()$ has to be differentiable

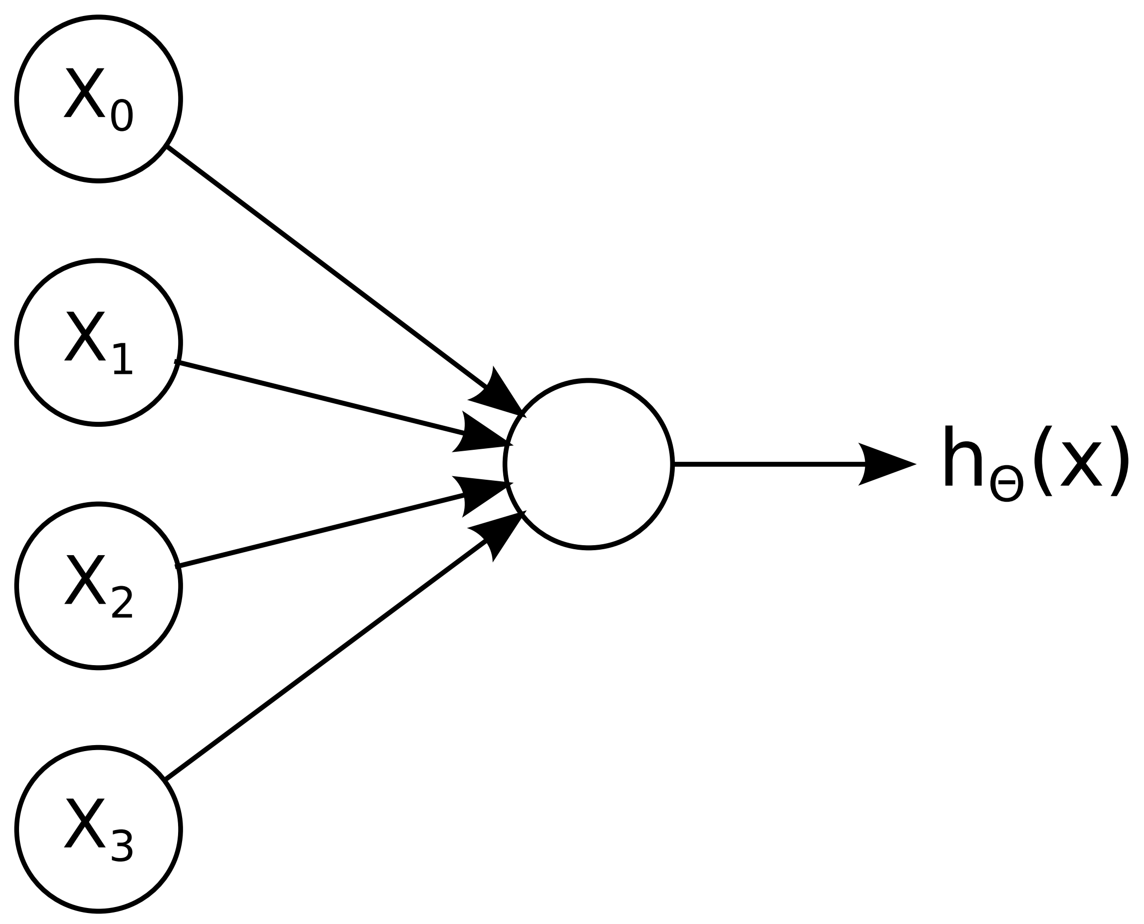
Infinitely Many Different Activation Functions Possible



- Different applications require different activation functions
- Linear or Gaussian useful for regression
- Sigmoid (logistics), ReLU etc. useful for classification
  - ReLU popular due to efficiency

Efficient Processing of Deep Neural Networks: A Tutorial and Survey

Finding Best Parameters

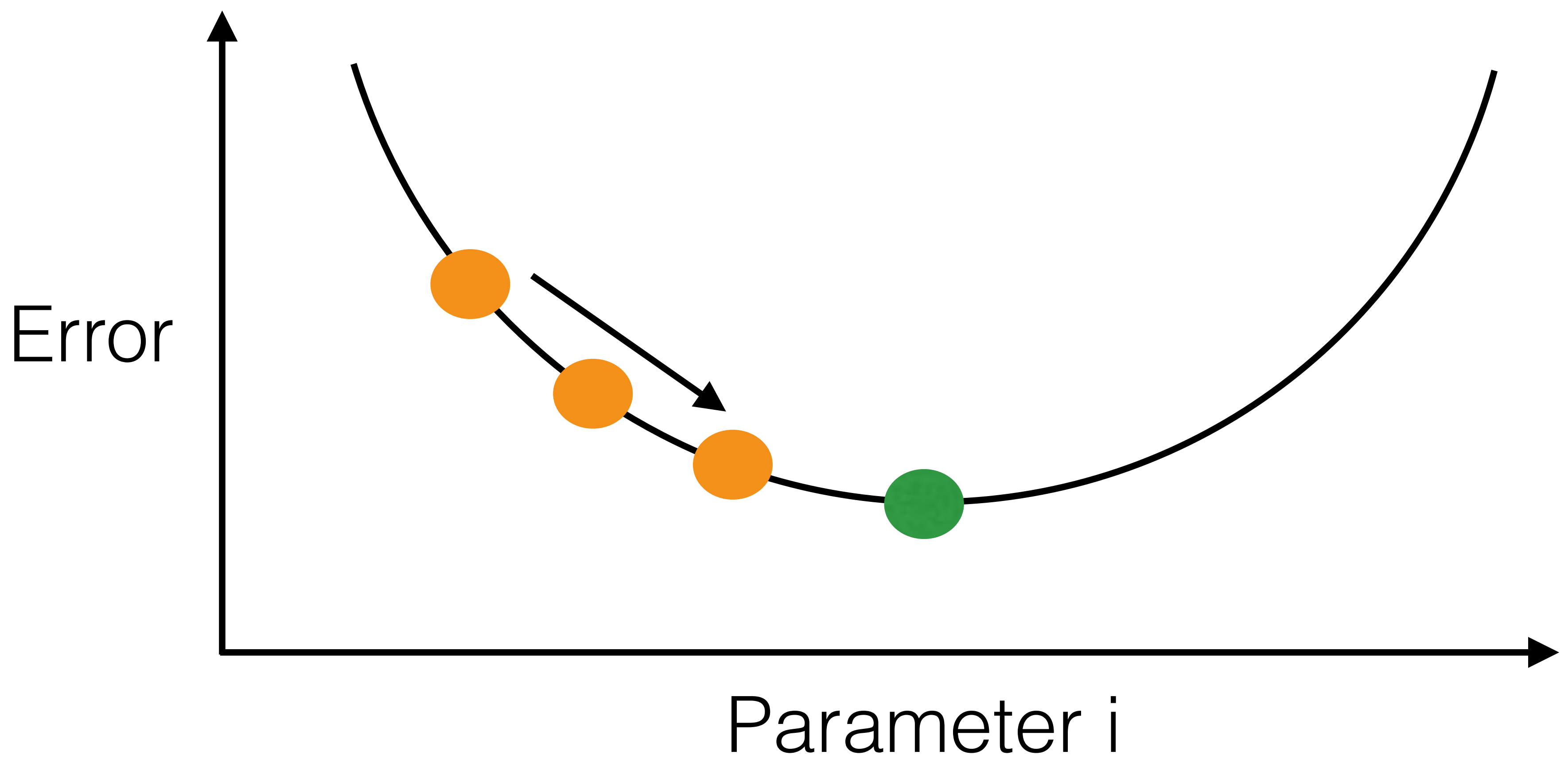


$$\mathbf{x} = \begin{bmatrix} \mathbf{1} \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \mathbf{\Theta} = \begin{bmatrix} \Theta_0 \\ \Theta_1 \\ \Theta_2 \\ \Theta_3 \end{bmatrix}$$

$$h_{\Theta}(\mathbf{x}) = g(\mathbf{\Theta}^T \mathbf{x})$$

Procedure:

1. Guess initial parameters
2. Calculate error
3. Calculate derivative of error
4. Change the parameters

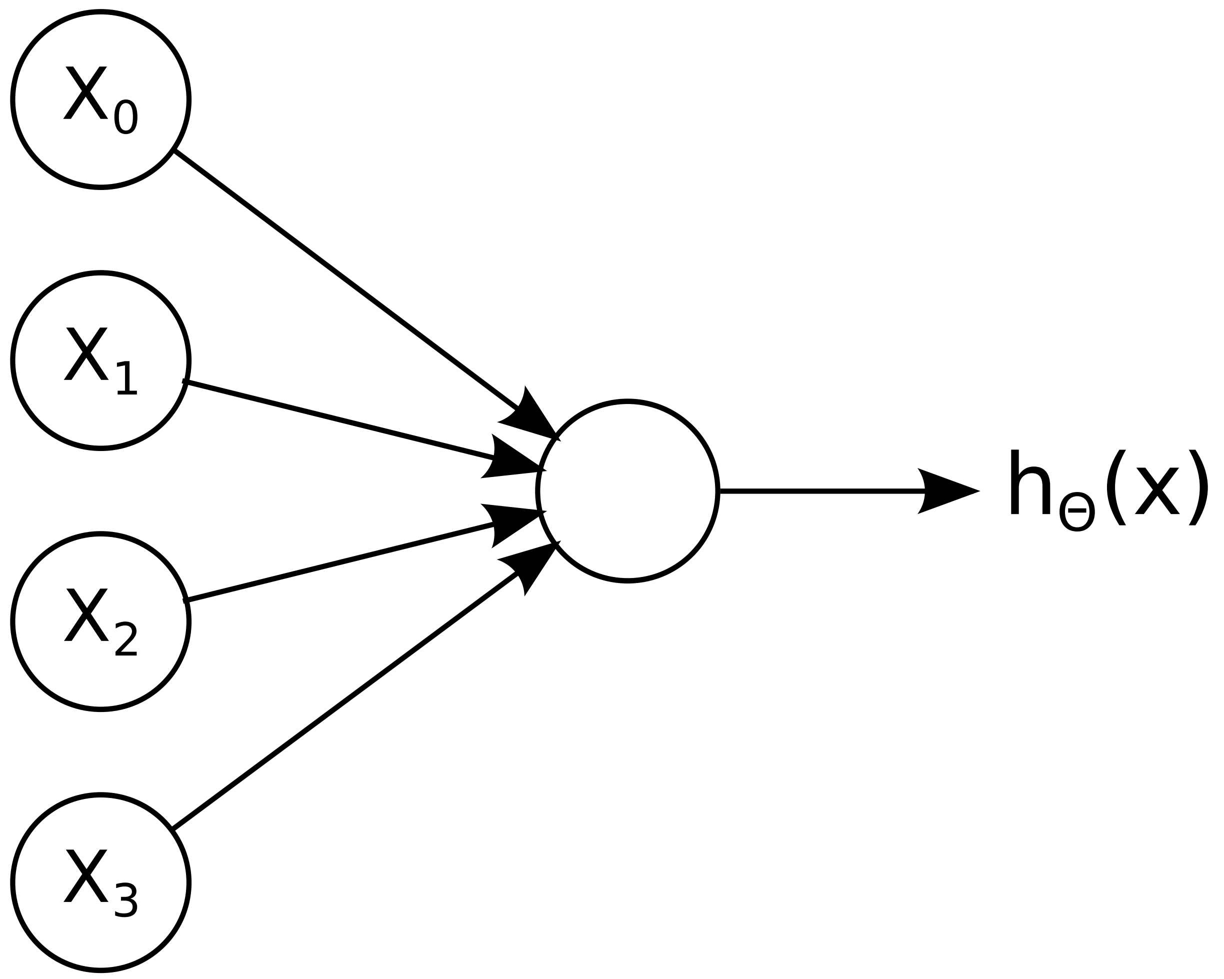


Finding Best Parameters

Using the chain rule for partial derivative we obtain:

$$E = \frac{1}{2}\sum_n (t^n - h(\mathbf{x}^n))^2$$

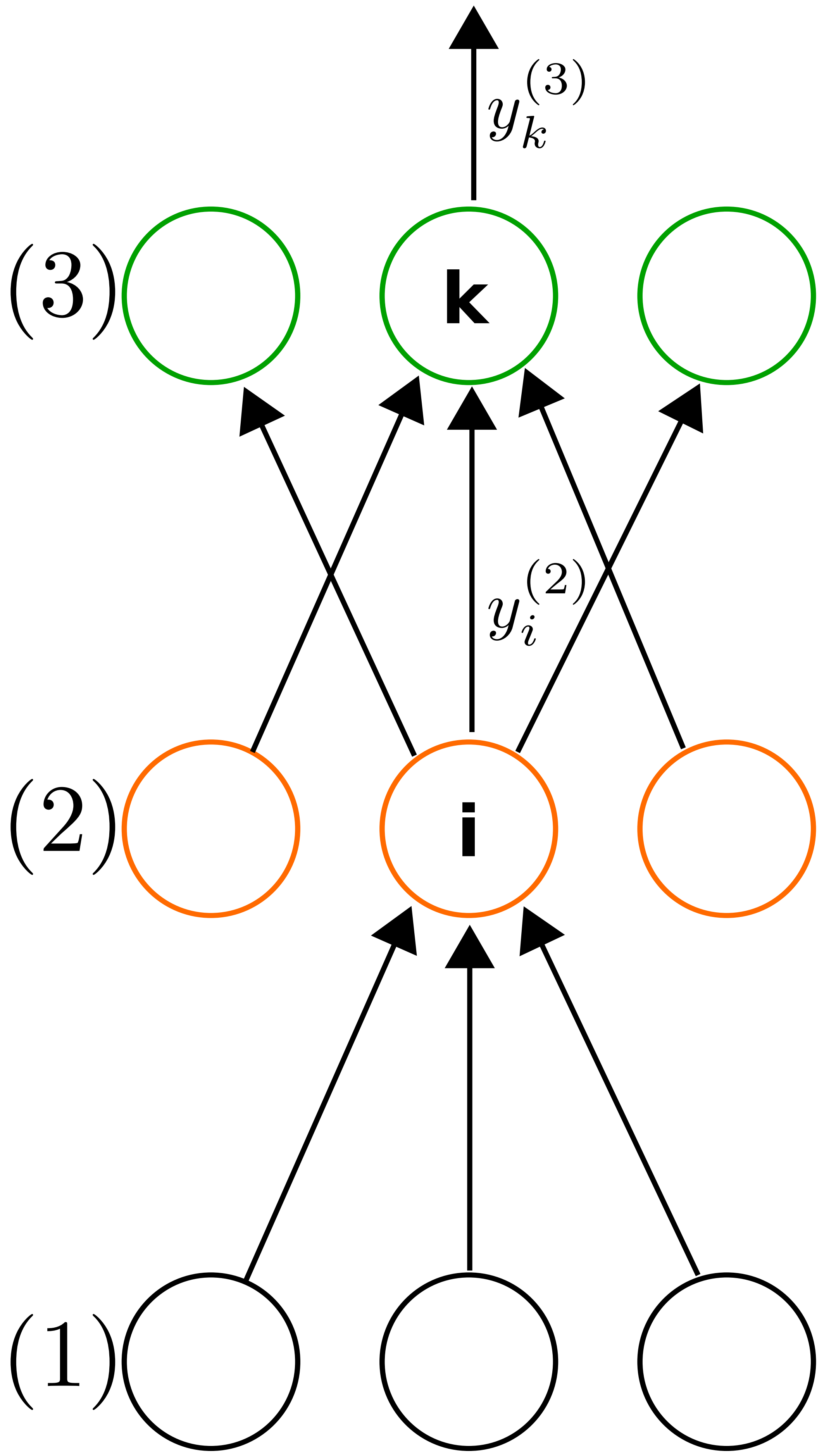
$$\frac{\partial E}{\partial \Theta_i} = \frac{1}{2}\sum_n \frac{\partial h(\mathbf{x}^n)}{\partial \Theta_i}\frac{dE}{dh(\mathbf{x}^n)}$$



How does the output change if the weight changes?

How does the error change if the output changes?

Finding Parameters of Non-output Layers via Backpropagation



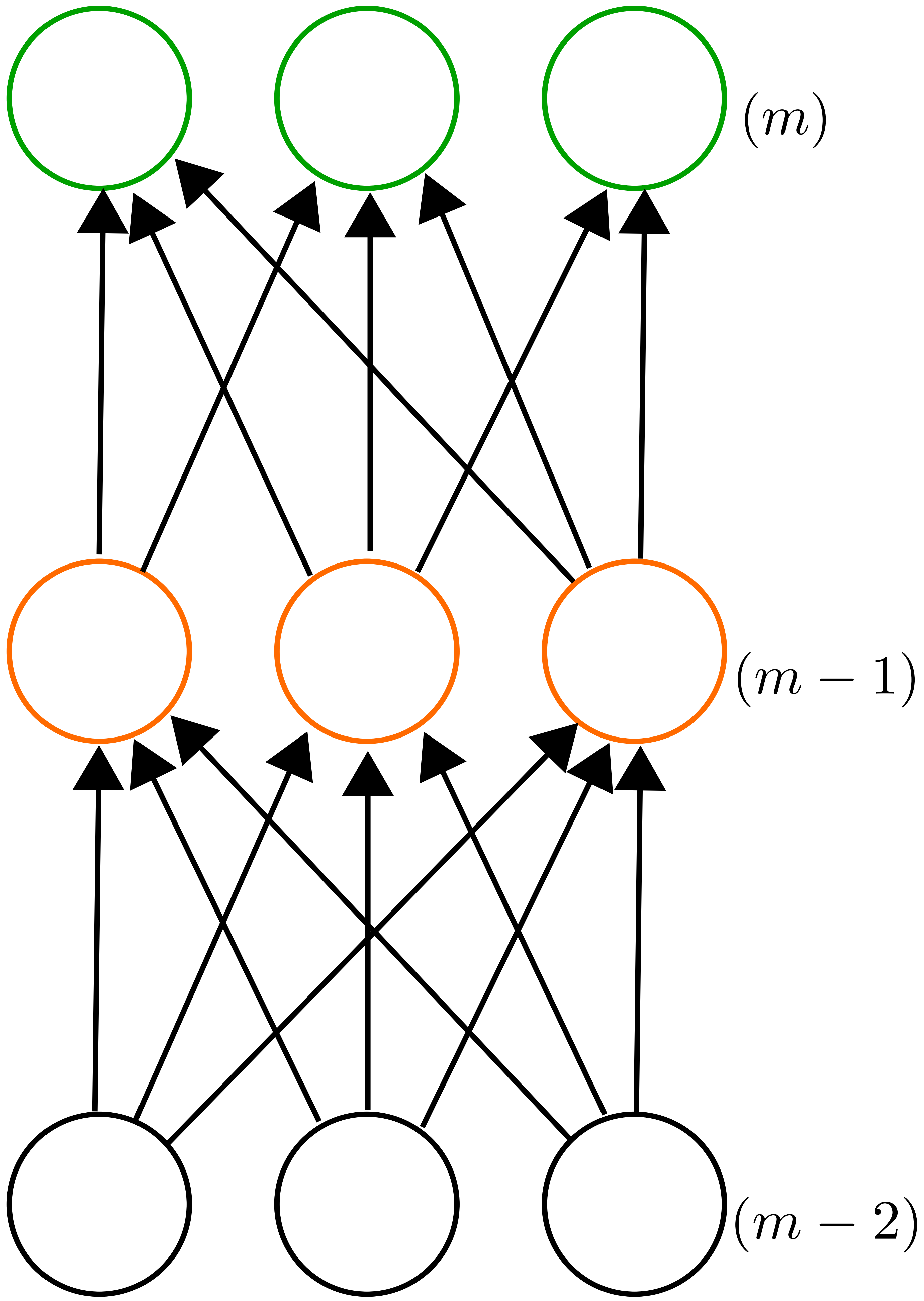
$$y_k^{(3)} = g(z_k^{(3)})$$
$$z_k^{(3)} = (\Theta_k^{(2)})^T \mathbf{y}^{(2)} = \sum_{n \in (2)} \Theta_{kn}^{(2)} y_n^{(2)}$$

$$\frac{\partial E}{\partial z_k^{(3)}} =$$ How much does the error change if I change the input to unit **k**?

$$\boldsymbol{\delta_2} = \frac{\partial E}{\partial y_i^{(2)}} =$$ How much does the error change w.r.t. the activation of unit **i**?

$$\frac{\partial E}{\partial \Theta_{ki}^{(2)}} =$$ How much does the error change w.r.t. the weight between unit **i** and **k**?

Training a Deep Neural Network



1) Randomly initialise all weights
2) For each randomly chosen sample **x**
3) Forward pass, for each layer m

$$y_i^{(m)} = g(z_i^{(m)})$$

$$z_i^{(m)} = \left(\Theta_k^{(m-1)}\right)^T \mathbf{y}^{(m-1)}$$

4) Calculate derivative of error at output layer

$$\nabla E = \left(\frac{\partial E}{\partial y_1^{(m)}}, \frac{\partial E}{\partial y_2^{(m)}}, \frac{\partial E}{\partial y_3^{(m)}}\right)$$

5) Backpropagate the error to the next layers

$$\frac{\partial E}{\partial y_i^{(m-1)}} = \sum_{n \in (m)} \Theta_{ni}^{(m-1)} y_n^{(m)} (1 - y_n^{(m)}) \frac{\partial E}{\partial y_n^{(m)}}$$

6) Adjust the weights using derivative

$$\Theta_{ji}^{(m)} \leftarrow \Theta_{ji}^{(m)} - \alpha \left(y_i^{(m)} \frac{\partial E}{\partial z_j^{(m+1)}} \right)$$

7) GOTO 2 if not converged

Problem: Too many parameters

- In general, every connection from output of one neuron in a layer to input of another in the text layer can have different weights
  - Called Multi Layer Perceptron

- Exploit two properties to reduce # of parameters
  - **Structural Locality**
    - an output neuron depends on input neurons in a window (space, time)
  - **Translational Equivariance**
    - shift in input causes similar shift in output

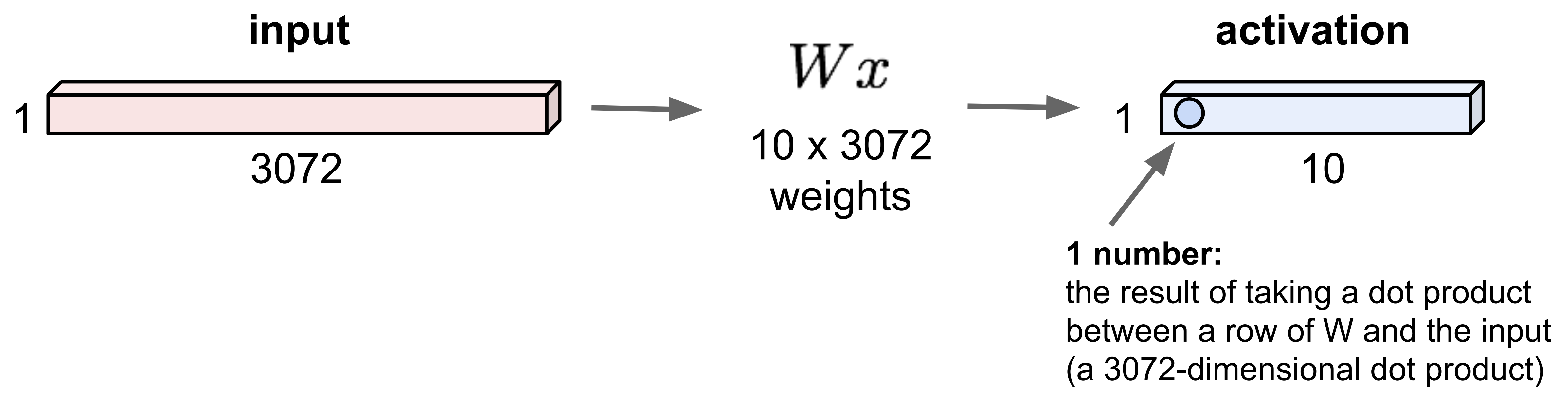
- **Convolutional Layers** (instead of fully connected layers)

Problem: Too many parameters

- Exploit two properties to reduce # of parameters
  - **Structural Locality**
    - an output neuron depends on input neurons in a window (space, time)
  - **Translational Equivariance**
    - shift in input causes similar shift in output

- **Convolutional Layers** (instead of fully connected layers)

32x32x3 image -> stretch to 3072 x 1

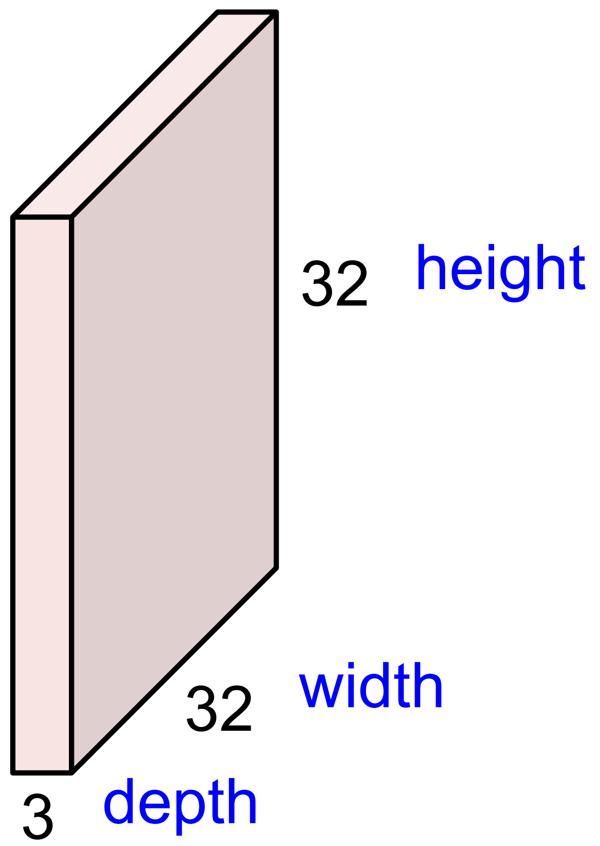


Problem: Too many parameters

- Exploit two properties to reduce # of parameters
  - **Structural Locality**
    - an output neuron depends on input neurons in a window (space, time)
  - **Translational Equivariance**
    - shift in input causes similar shift in output

- **Convolutional Layers** (instead of fully connected layers)

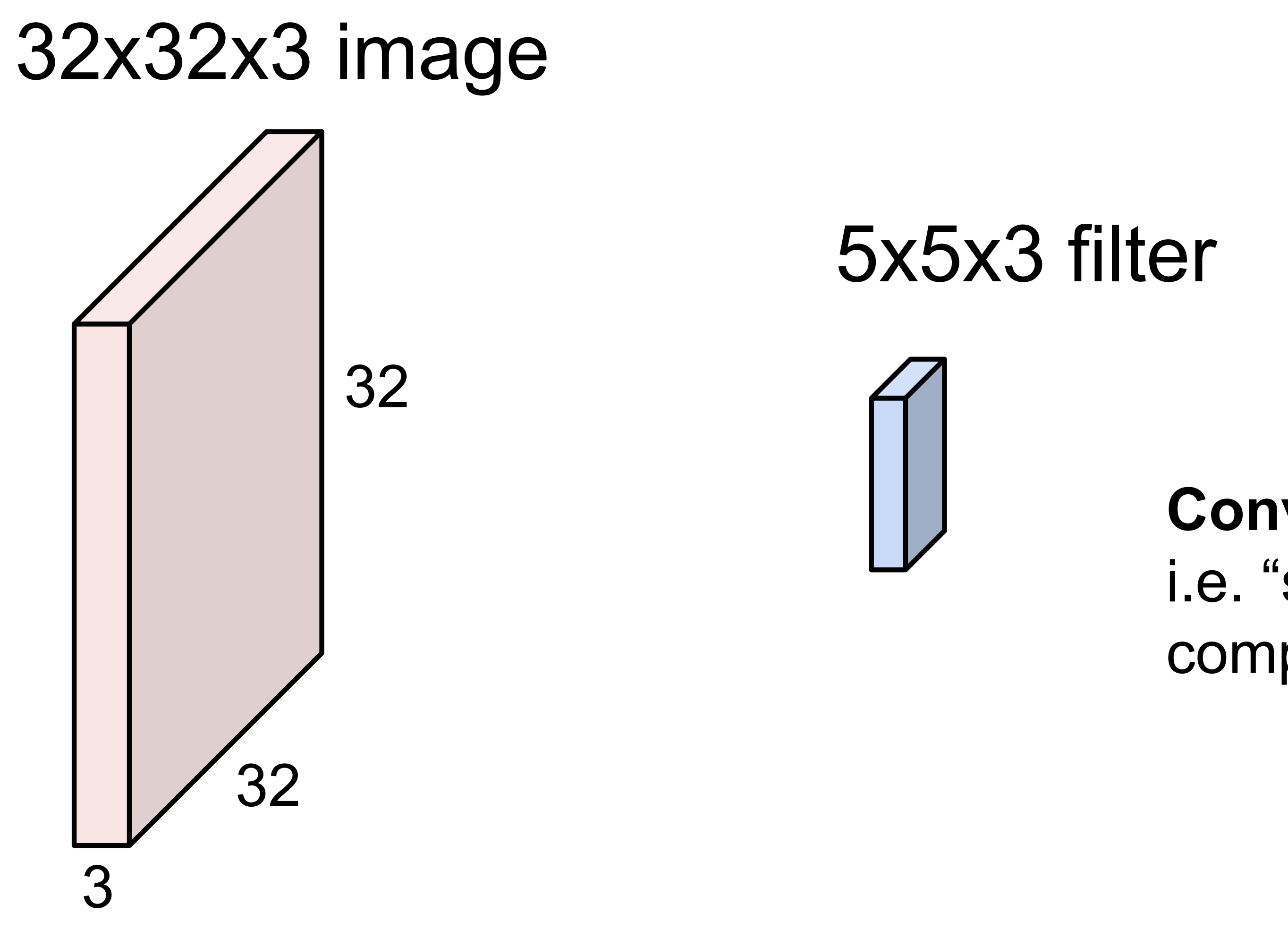
32x32x3 image -> preserve spatial structure



Problem: Too many parameters

- Exploit two properties to reduce # of parameters
  - **Structural Locality**
    - an output neuron depends on input neurons in a window (space, time)
  - **Translational Equivariance**
    - shift in input causes similar shift in output

- **Convolutional Layers** (instead of fully connected layers)

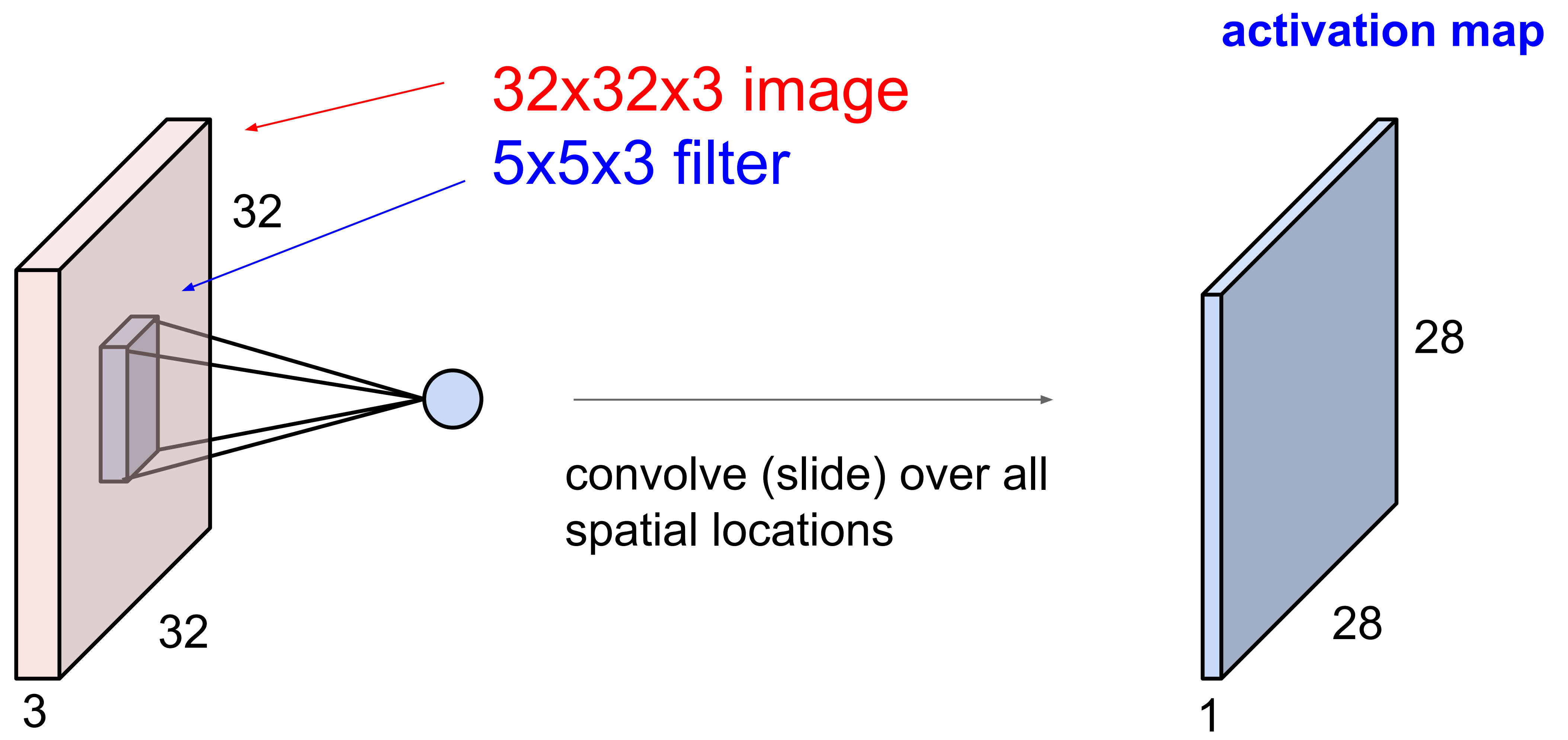


Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

Problem: Too many parameters

- Exploit two properties to reduce # of parameters
  - **Structural Locality**
    - an output neuron depends on input neurons in a window (space, time)
  - **Translational Equivariance**
    - shift in input causes similar shift in output

- **Convolutional Layers** (instead of fully connected layers)

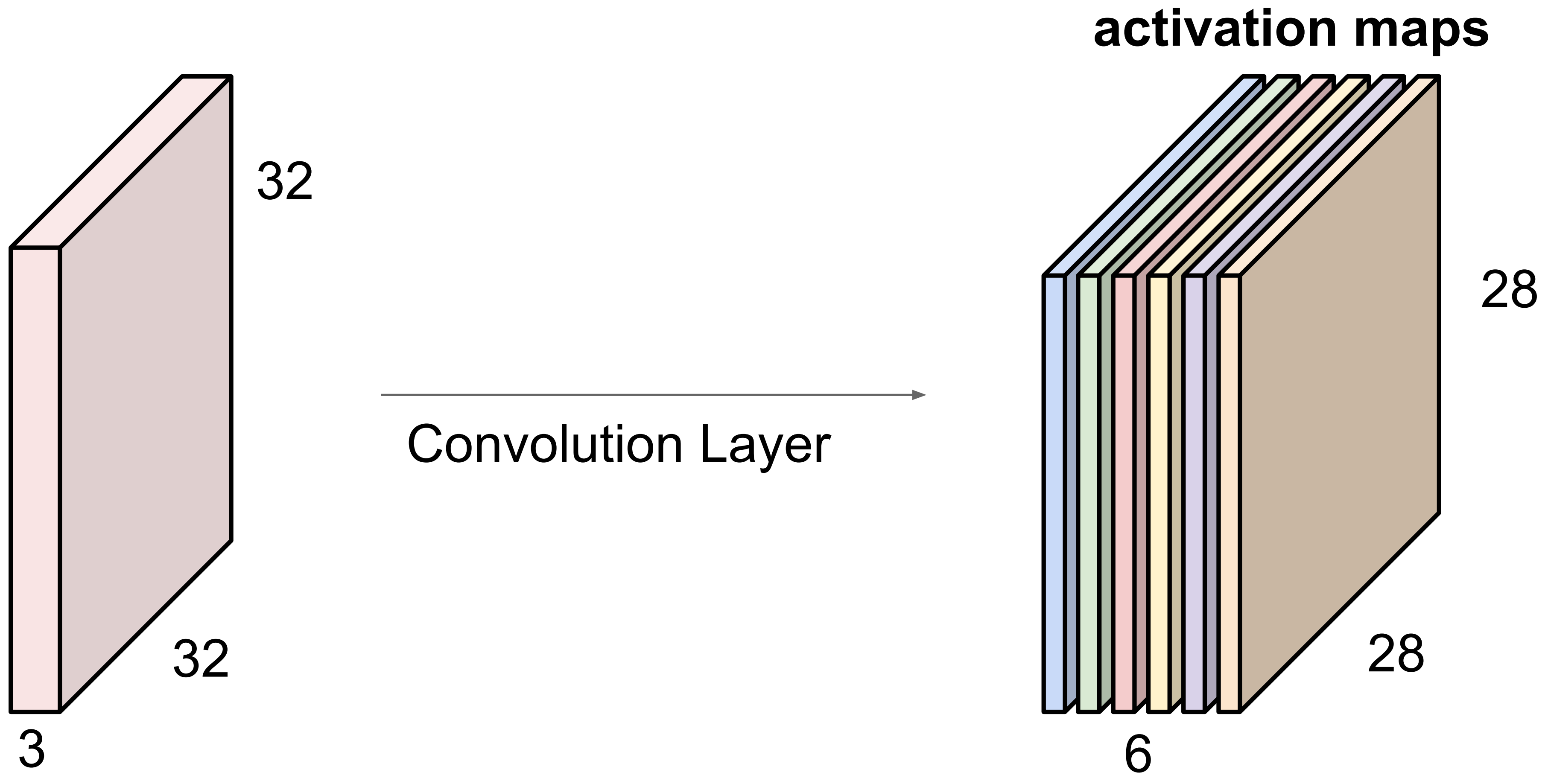


Problem: Too many parameters

- Exploit two properties to reduce # of parameters
  - **Structural Locality**
    - an output neuron depends on input neurons in a window (space, time)
  - **Translational Equivariance**
    - shift in input causes similar shift in output

- **Convolutional Layers** (instead of fully connected layers)

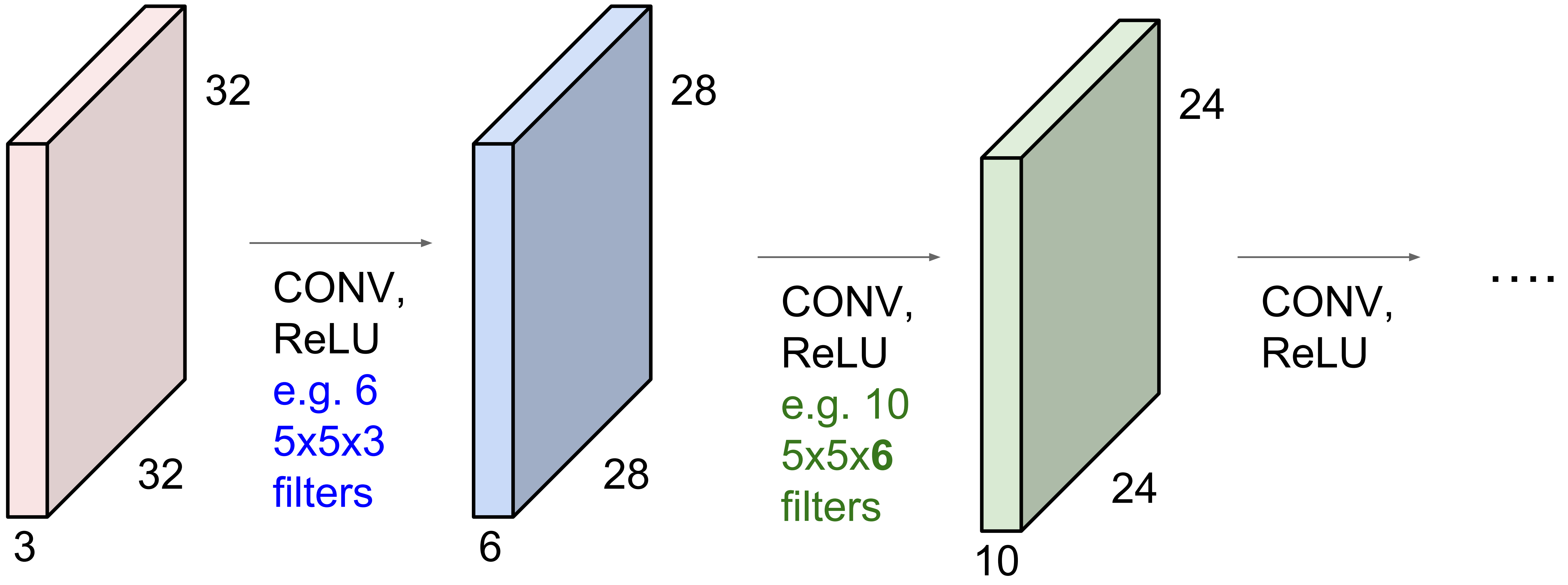
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



We stack these up to get a "new image" of size 28x28x6!

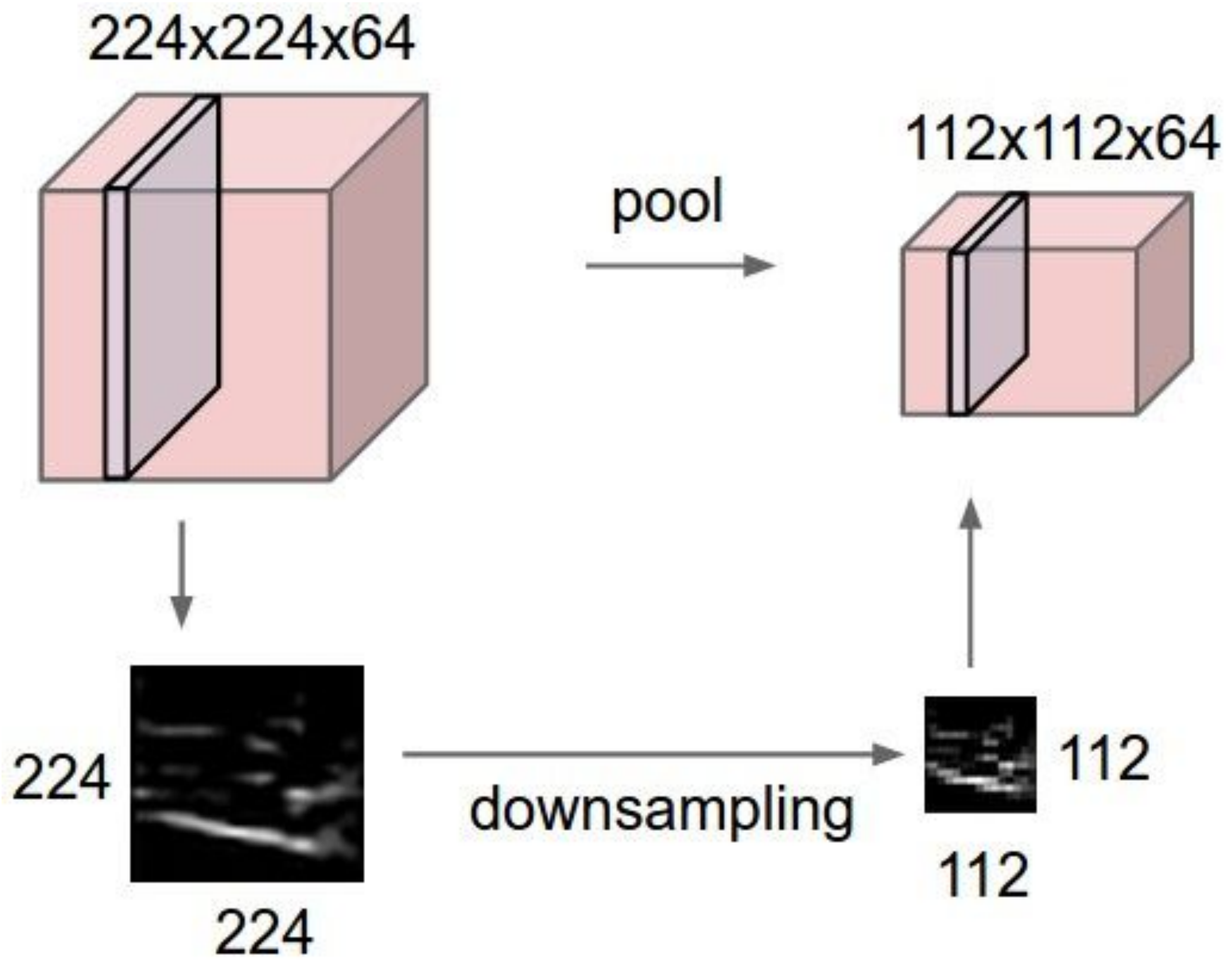
Problem: Too many parameters

- **Convolutional Neural Networks (CNNs)**
  - Sequence of Convolutional Layers

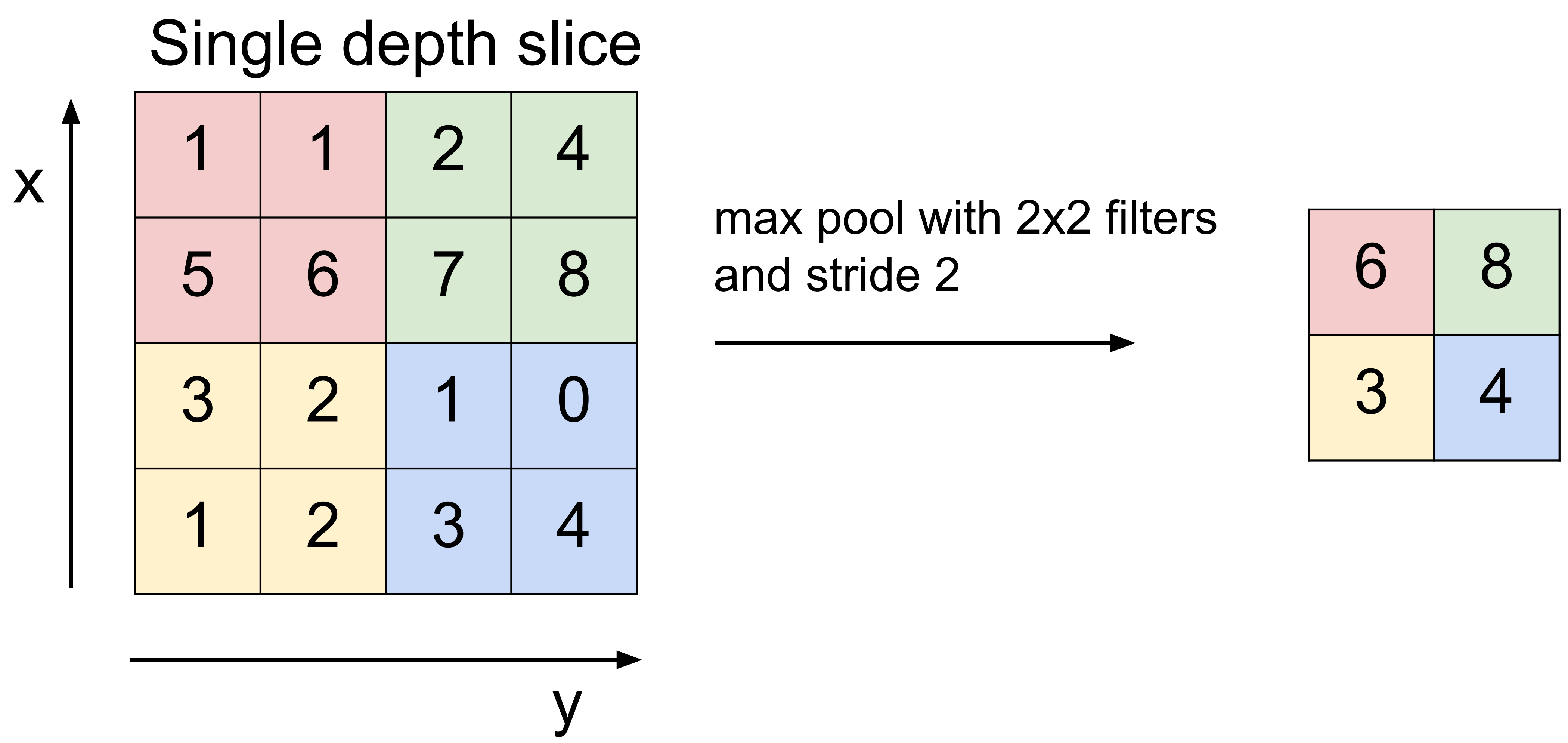


Problem: Too many parameters

- **Pooling**
  - Making representations more manageable
  - Getting some Shift Invariance

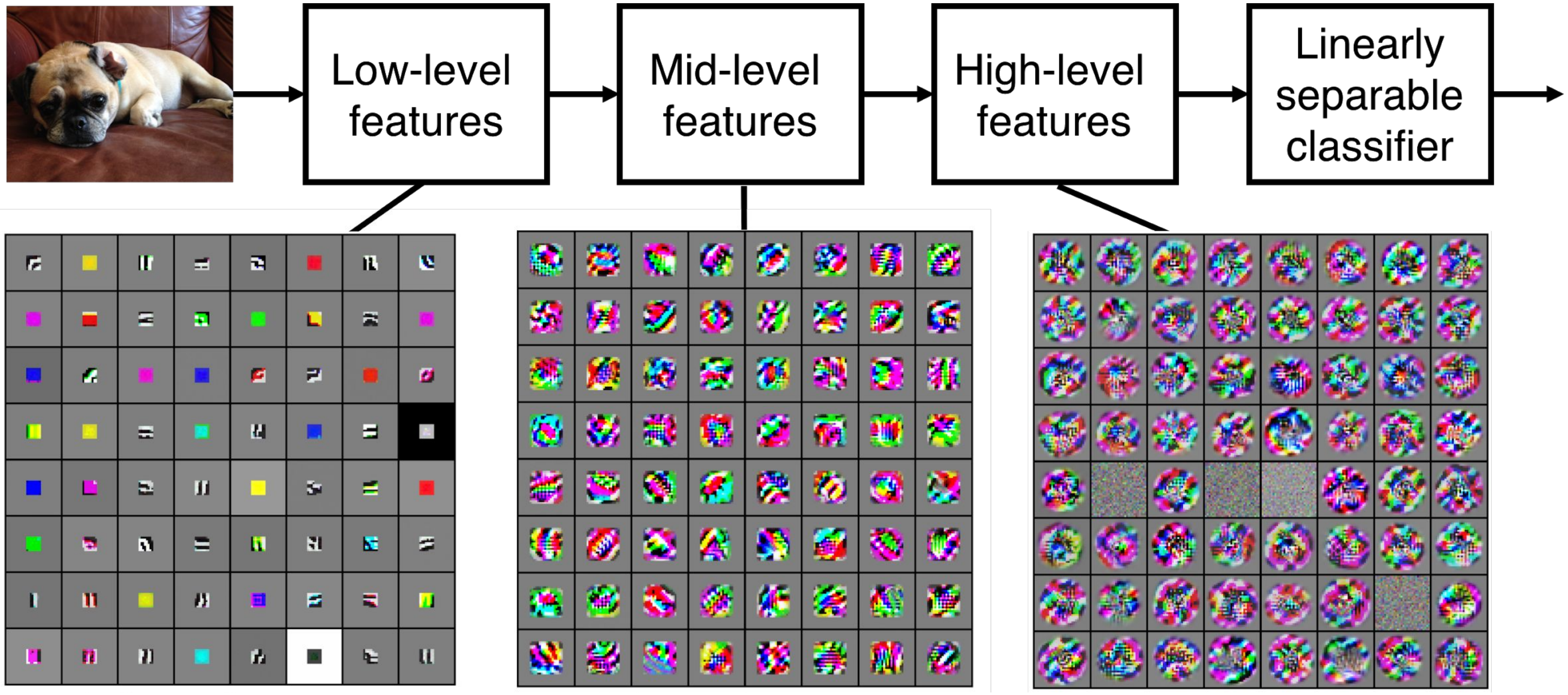


MAX POOLING



Problem: Too many parameters

- Putting it all together…



But other problems (that matter to us) remain

- How can we handle temporal sequences?
- How can we handle sensors that are distributed irregularly?
- How do we handle multiple sensors of different modalities?
- How do we handle missing data? bad data? misaligned data?

and more…

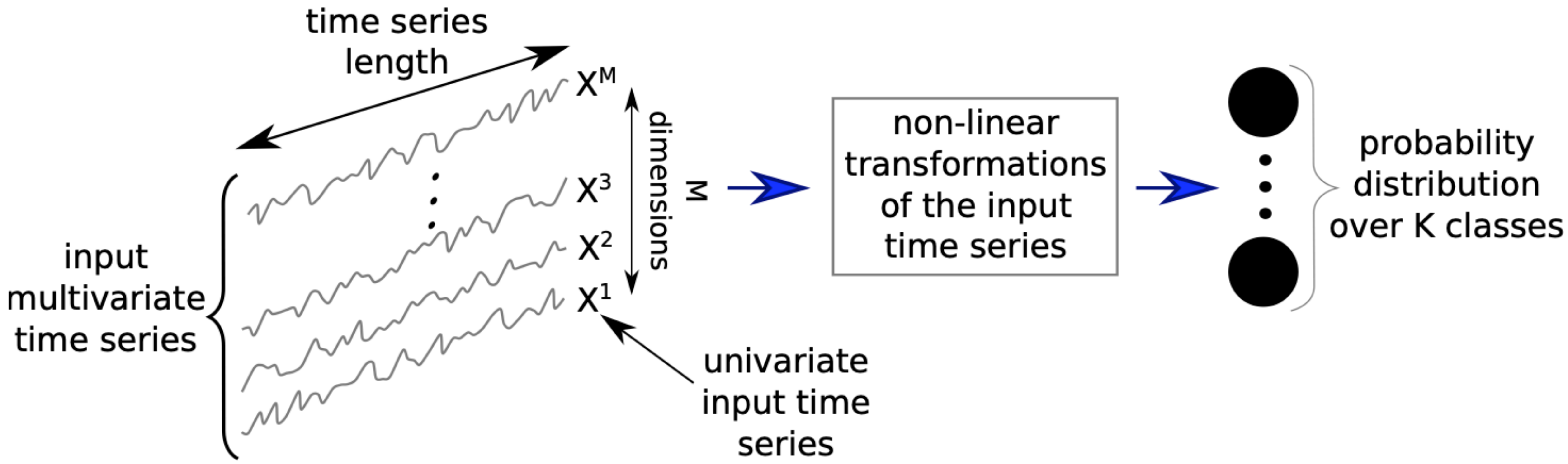
Deep Learning for Temporal Sequences

Time Series Classification (TSC) Problem

- A Univariate Time Series $X = [x_1, x_2, \cdots, x_T]$ is a sequence of real values
  - length of X is equal to the number of real values T
  - no explicit notion of time: either only order matters (e.g. text) or values are equally spaced in time, i.e. $x_{i+1} - x_i = x_{j+1} - x_j \; \forall \; i, j \in \{1,2,\cdots, T-1\}$
- An M-dimensional Multivariate Time Series $X = [X^1, X^2, \cdots, X^M]$ consists of M different univariate time series with $X^i \in \mathbb{R}^T$
- TSC task: Train a classifier on a dataset D in order to map from the space of possible inputs to a probability distribution over K class labels
  - $D = \{(X_1, Y_1), (X_2, Y_2), \cdots, (X_N, Y_N)\}$ where X_i could either be a univariate or multivariate time series with Y_i as its corresponding one-hot label vector of K-bits
  - $Y_i[j] = 1$ if the class of X_i is j, and 0 otherwise

Fawaz, Hassan Ismail, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. "Deep learning for time series classification: a review." Data Mining and Knowledge Discovery 33, no. 4 (2019): 917-963. https://arxiv.org/pdf/1809.04356.pdf

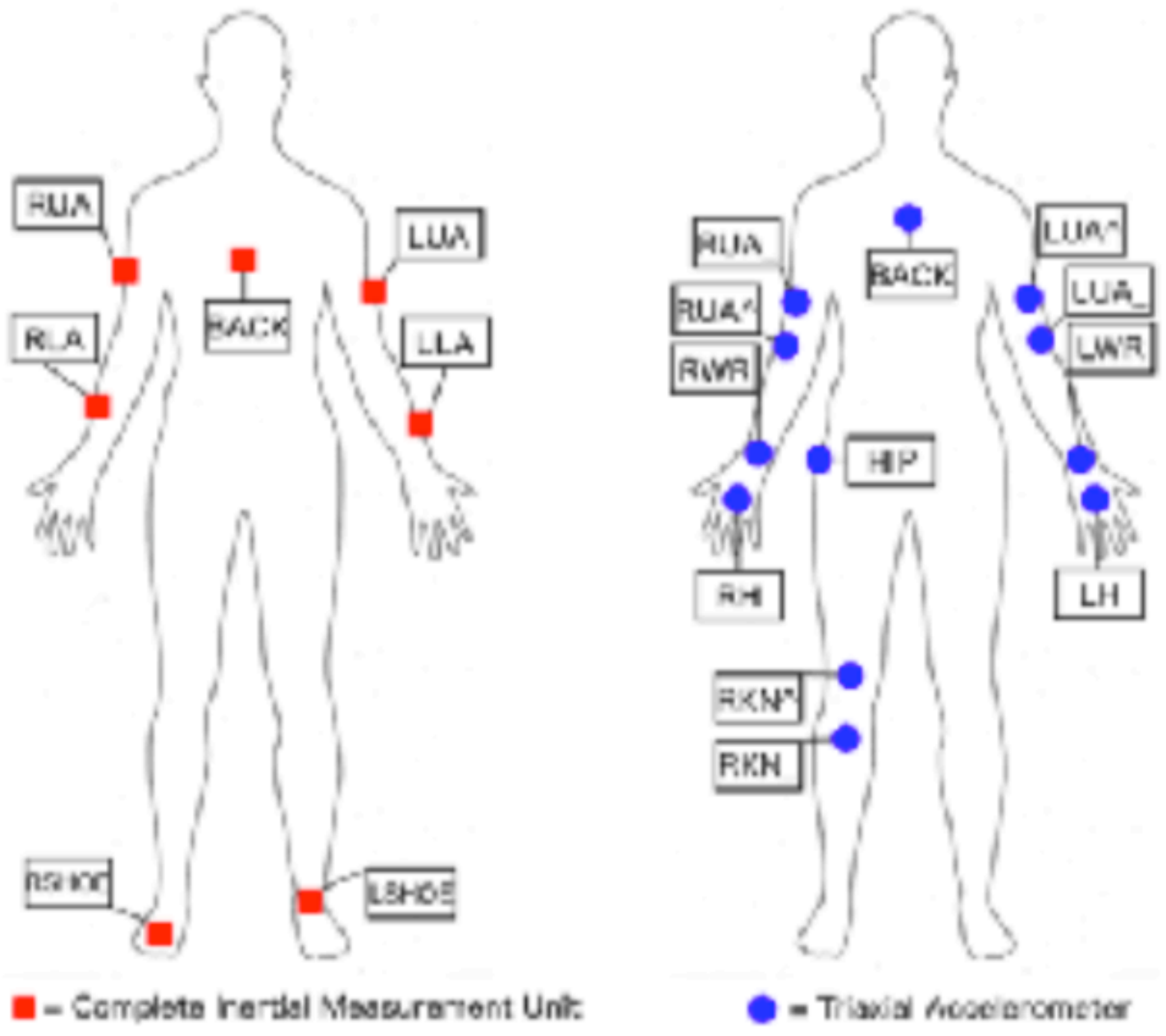
TSC Task



- What if time series is long and so labels change?
- E.g. Activity Recognition: we need a sequence of labels over a long time series
- Approach: TSC over windows with a higher level activity recognizer
    - window level output labels of TSC used as sequence of observations
        - essentially TSC is viewed as a noisy sensor of current state

Fawaz, Hassan Ismail, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller.
"Deep learning for time series classification: a review." Data Mining and Knowledge Discovery 33, no. 4 (2019): 917-963.
https://arxiv.org/pdf/1809.04356.pdf

Applications to Sensing Tasks



[Ordonez & Roggen, 2016]

Human Activity Recognition

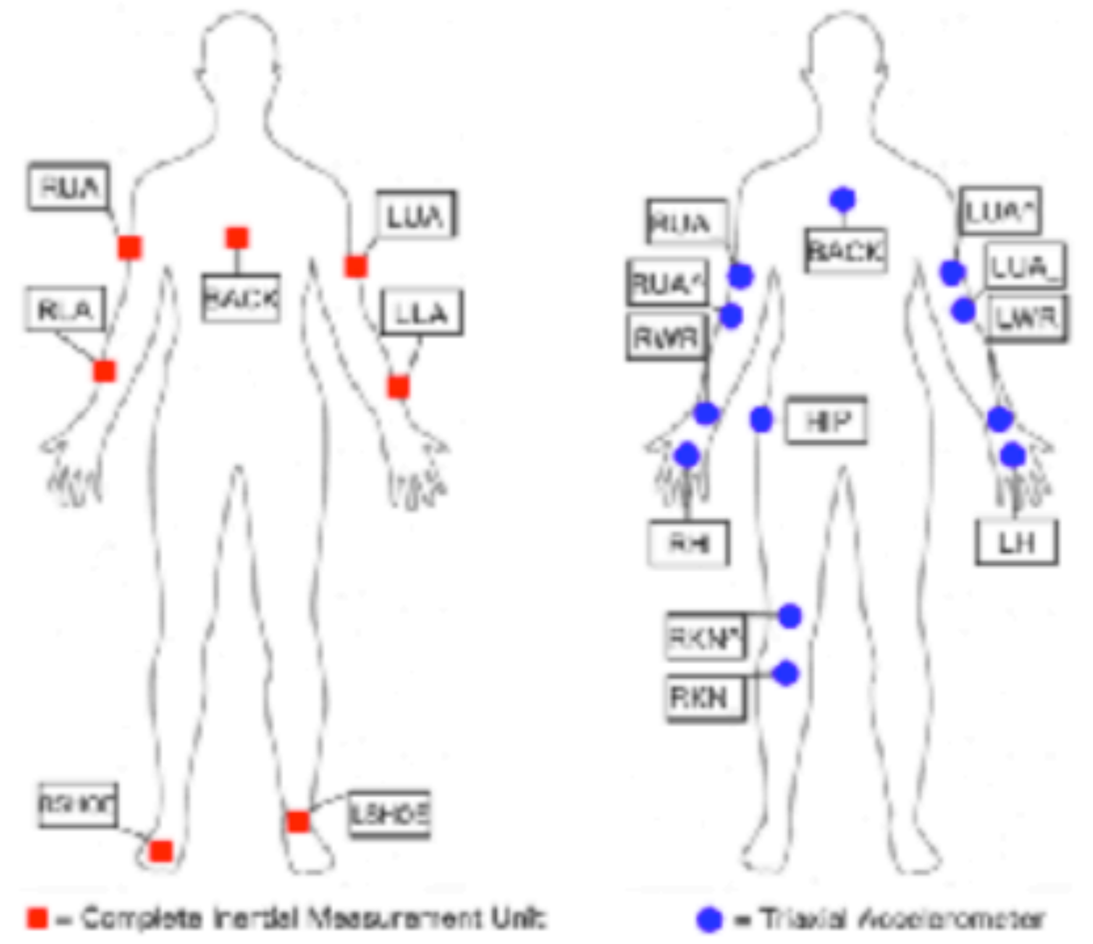
Audio Analysis

Human Activity Recognition

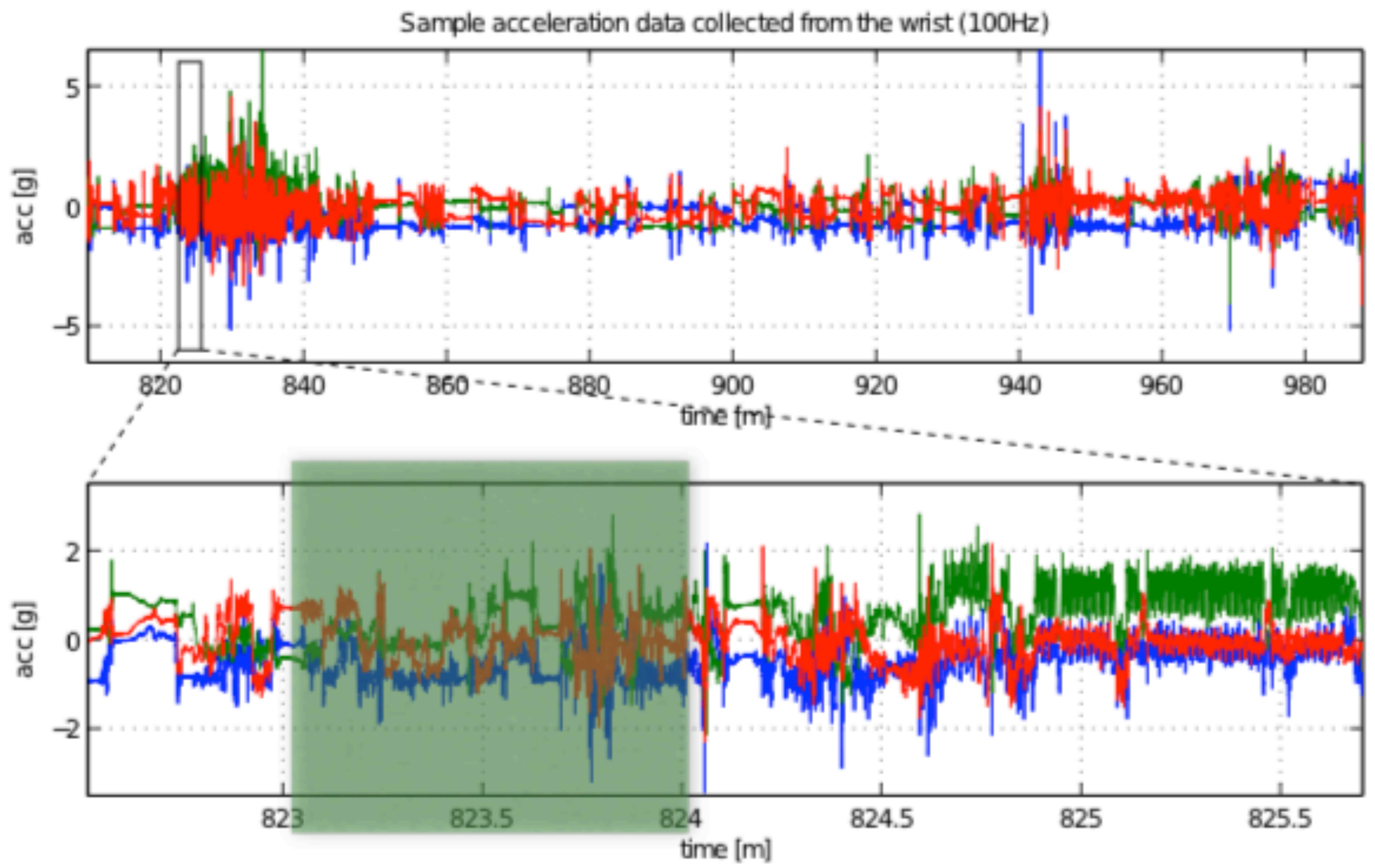
- Objective
  - Infer *when*? something of interest happened (*what*?), possibly *how*?
- Sensor data
  - (body worn) Inertial Measurement Units
- Standard approach
  Sliding window (frames)
  + *Feature engineering*
  + *Pattern classification*

ignores temporal aspects
treats windows in isolation

- Alternatives
  - sample-wise processing, memory

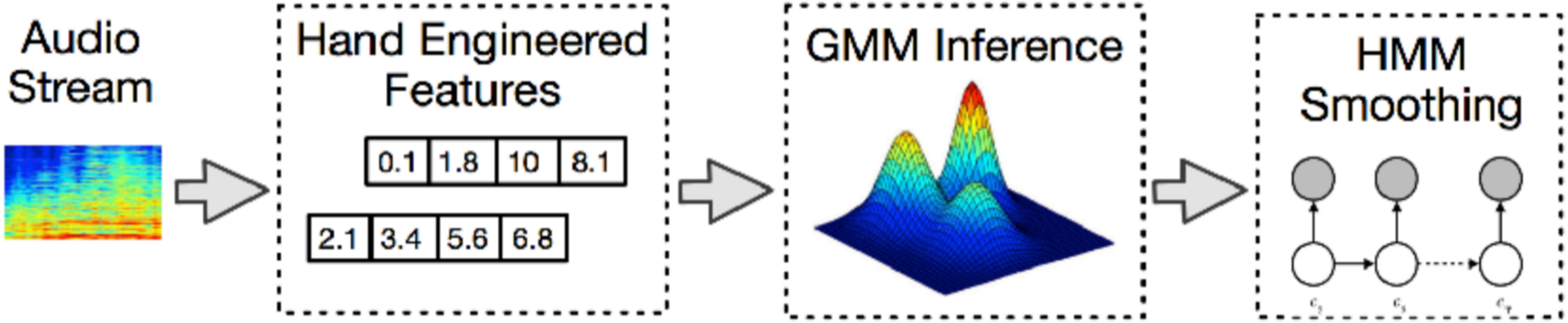


[Ordonez & Roggen, 2016]



Audio Analysis

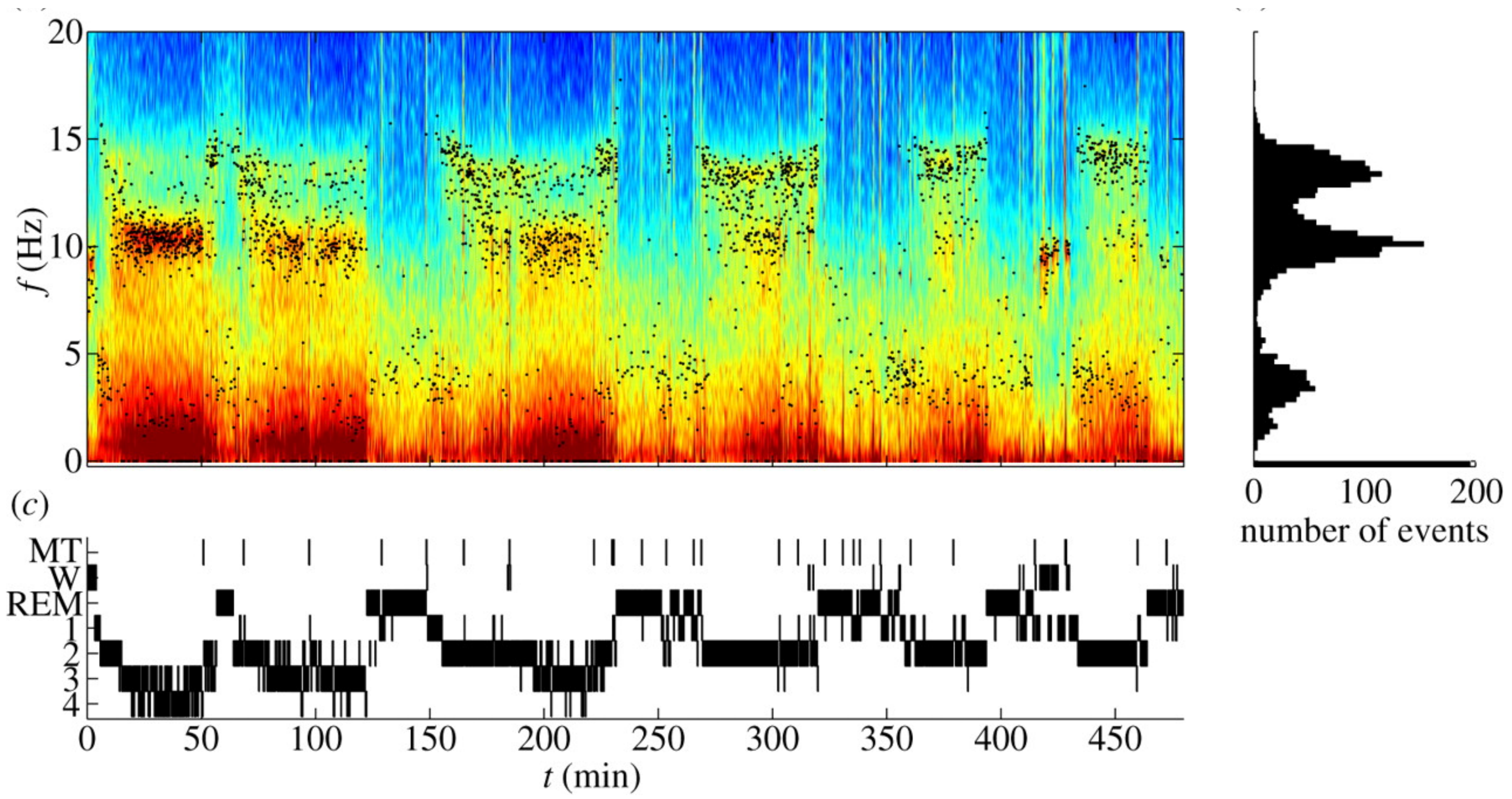
- Objective
  - Auditory scene analysis
  - Speaker recognition
  - Signal improvement
  - Speech recognition
- Sensor data
  - Microphone data
- Standard approach
  Sliding window
  \+ *Feature engineering*
  \+ *Pattern classification*
  \+ *Smoothing*



Lane, Nicholas D., Petko Georgiev, and Lorena Qendro. "Deepear: robust smartphone audio sensing in unconstrained acoustic environments using deep learning." In Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing, pp. 283-294. 2015.

Approach 1: Imaging Time Series

Main Idea: Turn the problem into a Vision Problem



Spectrogram

Movement Trajectory in 2D

Spectrograms and 2D Trajectories

- Spectrogram
  - An image with time and frequency as dimensions
  - In real-world image: nearby pixels normally belong to the same object: CNNs exploit this
  - Unlike pictures, spectrograms have non-local relationships, which complicates CNN
  - Limited to 1D values

- Trajectories in 2D
  - CNNs work with textures and not edges
  - Time dimension is lost
  - Doesn’t work beyond 2D

- Would like to have an image representation than can handle multidimensional values and retrieve information about any pair (x_i, x_j) given time series $(x_1, x_2, \cdots, x_n)$

Recurrence Plot

- Extracts *trajectories* from time series and computes the pairwise distances between these trajectories
  - capture recurrent behavior such as periodicities & irregular cyclicities

- The trajectories are defined as:

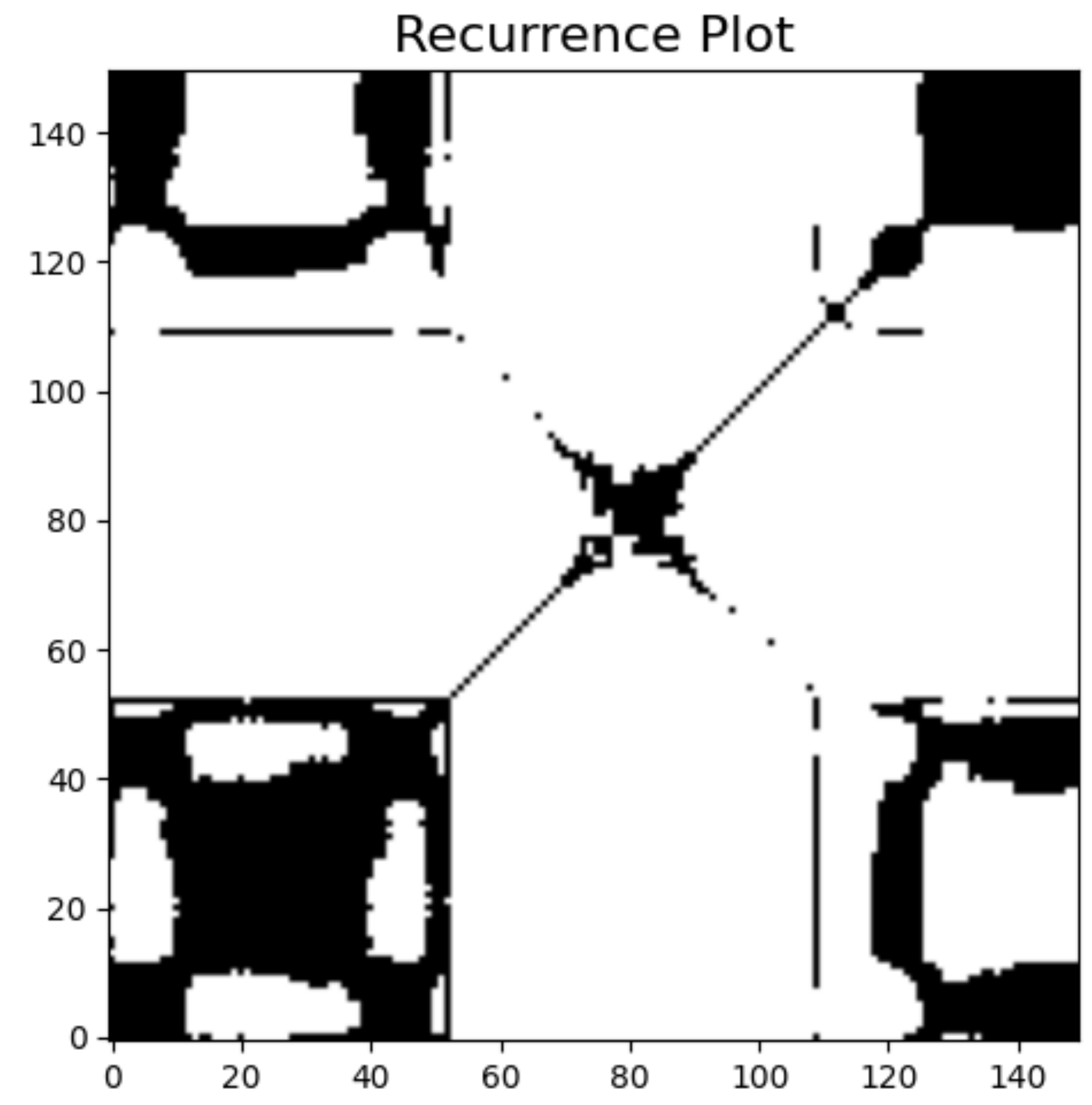
$$\vec{x}_i = (x_i, x_{i+\tau}, \dots, x_{i+(m-1)\tau}), \quad \forall i \in \{1, \dots, n-(m-1)\tau\}$$

  where m is the dimension of the trajectory, and τ is the time delay

- The recurrence plot, denoted R, is the binarized pairwise distance matrix between the trajectories

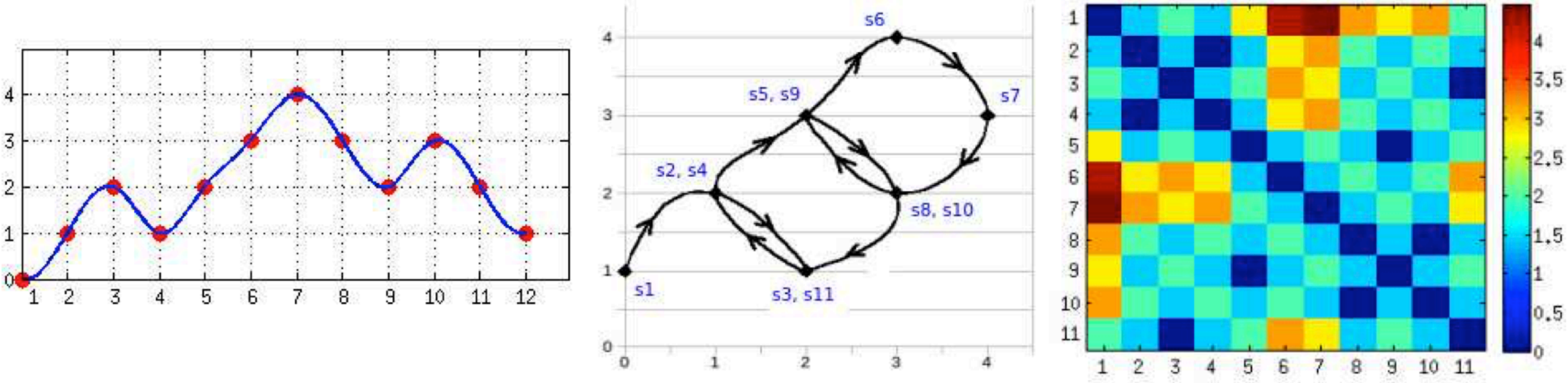
$$R_{i,j} = \Theta(\varepsilon - \|\vec{x}_i - \vec{x}_j\|), \quad \forall i, j \in \{1, \dots, n-(m-1)\tau\}$$

  where Θ is the Heaviside step function and ϵ is the threshold



https://pyts.readthedocs.io/en/stable/modules/image.html

From time-series signal to recurrence plot (binarization skipped)



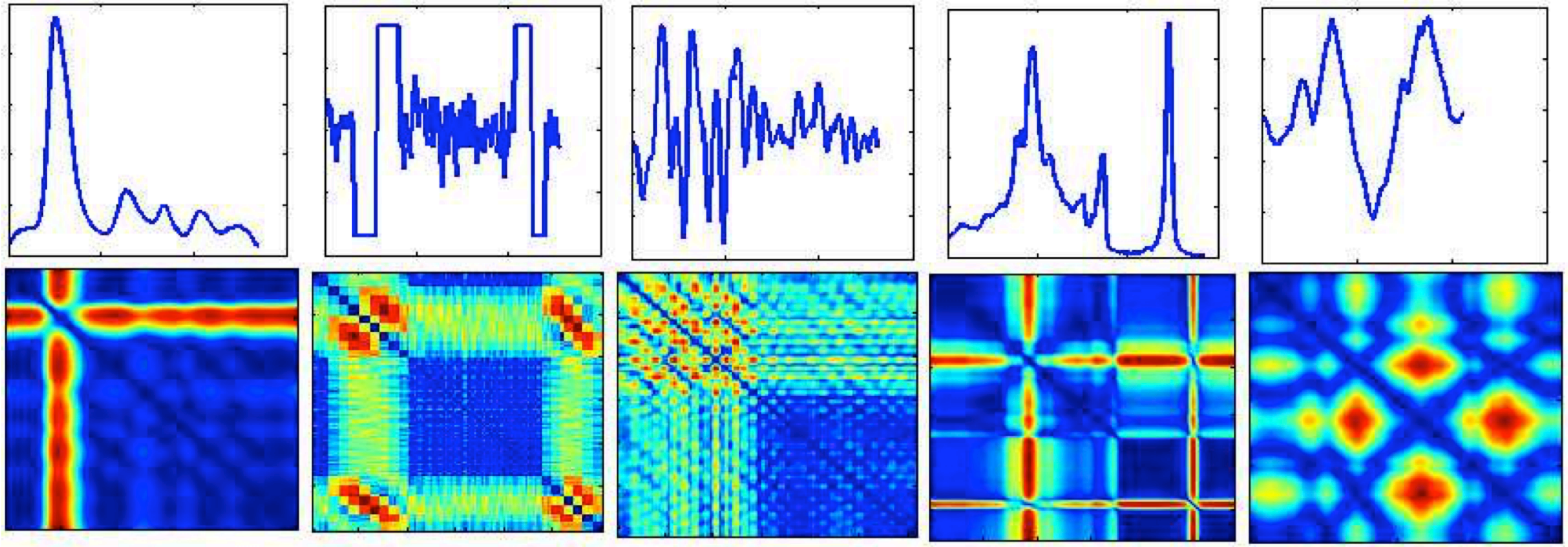
Left: A simple example of time-series signal (x) with 12 data points.

Middle: The 2D phase space trajectory is constructed from x by the time delay embedding ($\tau = 1$). States in the phase space are shown with bold dots: $s_1 : (x_1, x_2),\ s_2 : (x_2, x_3),\ \ldots,\ s_{11} : (x_{11}, x_{12})$.

Right: The recurrence plot R is a 11 × 11 square matrix with $R_{i,j} = \mathsf{dist}(s_i, s_j)$.

Hatami, Nima, Yann Gavet, and Johan Debayle. "Classification of time-series images using deep convolutional neural networks."
In Tenth international conference on machine vision (ICMV 2017), vol. 10696, p. 106960Y. International Society for Optics and Photonics, 2018.
https://arxiv.org/pdf/1710.00886.pdf

Recurrence Plot



Application of Recurrence Plot ($m = 3, \tau = 4$) time-series to image encoding on five different datasets from the UCR archive: 50words, TwoPatterns, FaceAll, OliveOil and Yoga data (from left to right, respectively)

Hatami, Nima, Yann Gavet, and Johan Debayle. "Classification of time-series images using deep convolutional neural networks." In Tenth international conference on machine vision (ICMV 2017), vol. 10696, p. 106960Y. International Society for Optics and Photonics, 2018. https://arxiv.org/pdf/1710.00886.pdf

Gramian Angular Fields (GAF)

- Creates a matrix of temporal correlations for each (x_i, x_j) via the following steps:
  1. Rescale the time series in a range $[a, b]$ where $-1 \leq a < b \leq 1$
  2. Compute the polar coordinates of the scaled time series by taking the arccos
     - novel way to understand time series: as time increases, corresponding values warp among different angular points on the spanning circles, like water rippling.
  3. Compute the cosine of the sum of the angles for the *Gramian Angular Summation Field* (GASF) or the sine of the difference of the angles for the *Gramian Angular Difference Field* (GADF).

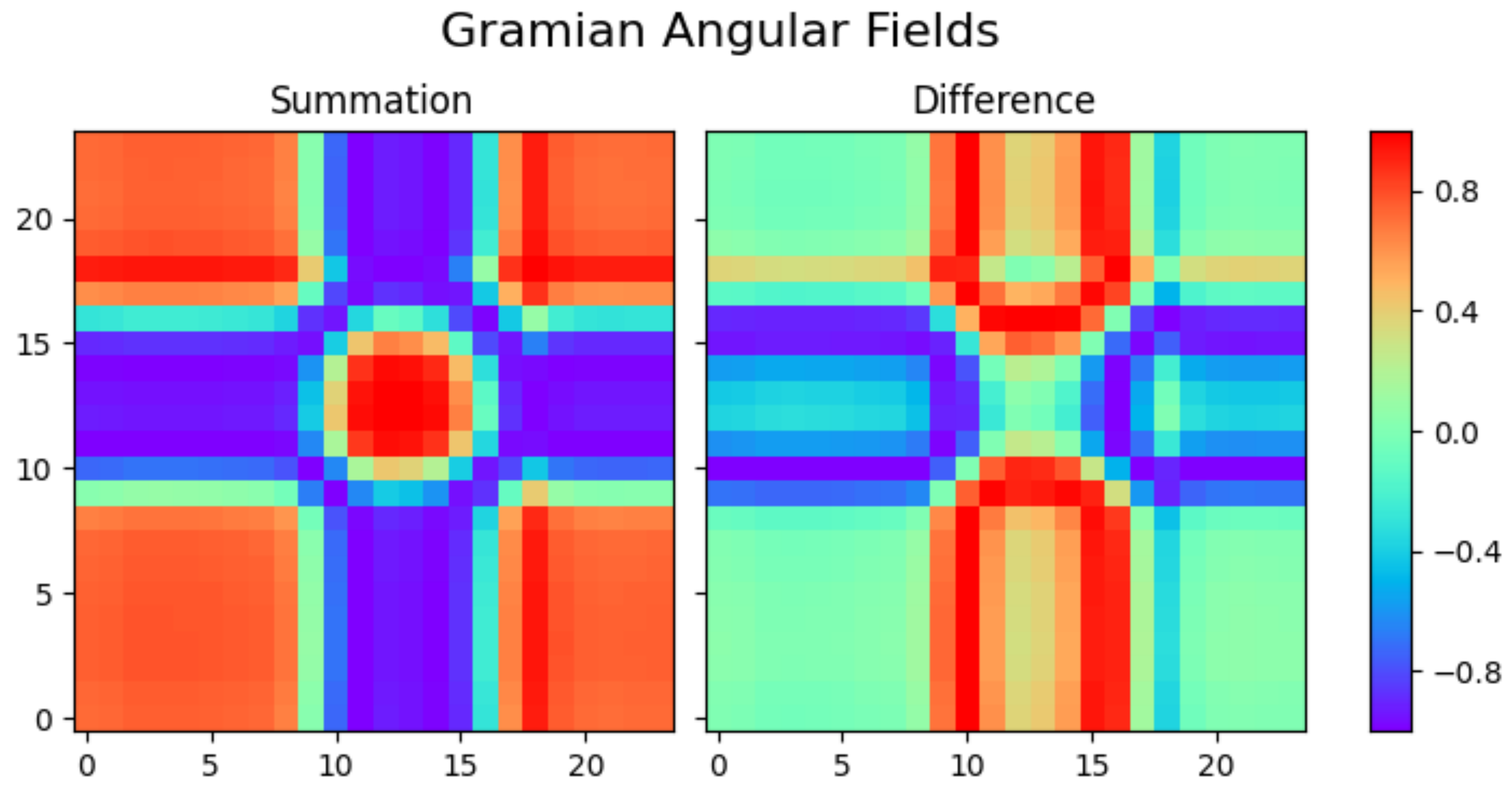
$$\tilde{x}_i = a + (b - a) \times \frac{x_i - \min(x)}{\max(x) - \min(x)}, \forall i \in \{1, 2, \cdots, n\}$$

$$\phi_i = \arccos(\tilde{x}_i), \forall i \in \{1, 2, \cdots, n\}$$

$$r_i = \frac{t_i}{n}, t_i \in \mathbb{N} \ \& \ \forall i \in \{1, 2, \cdots, n\}$$

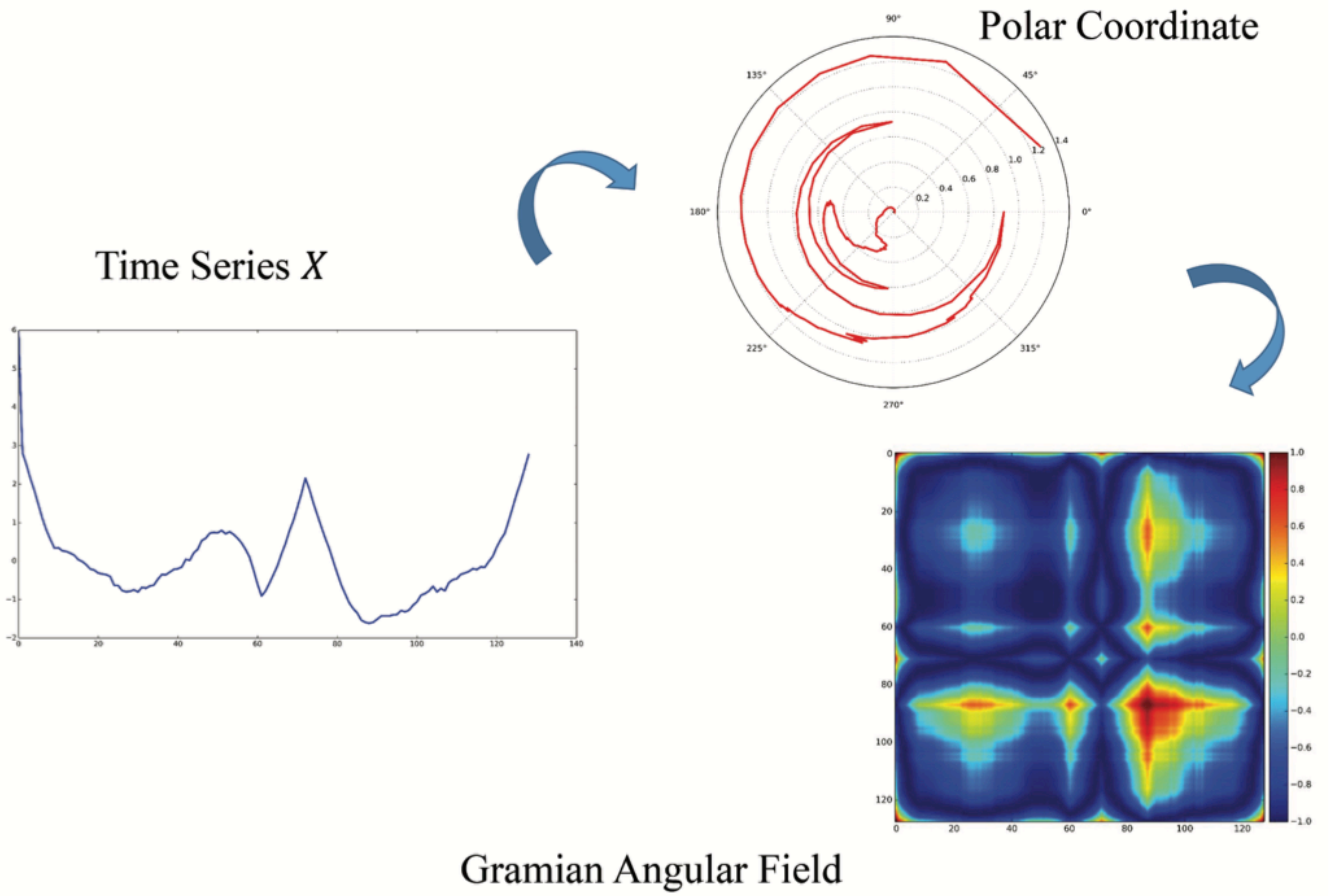
$$GASF_{ij} = \cos(\phi_i + \phi_j), \forall i, j \in \{1, 2, \cdots, n\}$$

$$GADF_{ij} = \sin(\phi_i - \phi_j), \forall i, j \in \{1, 2, \cdots, n\}$$



https://pyts.readthedocs.io/en/stable/modules/image.html

GAF Encoding Under the Hood

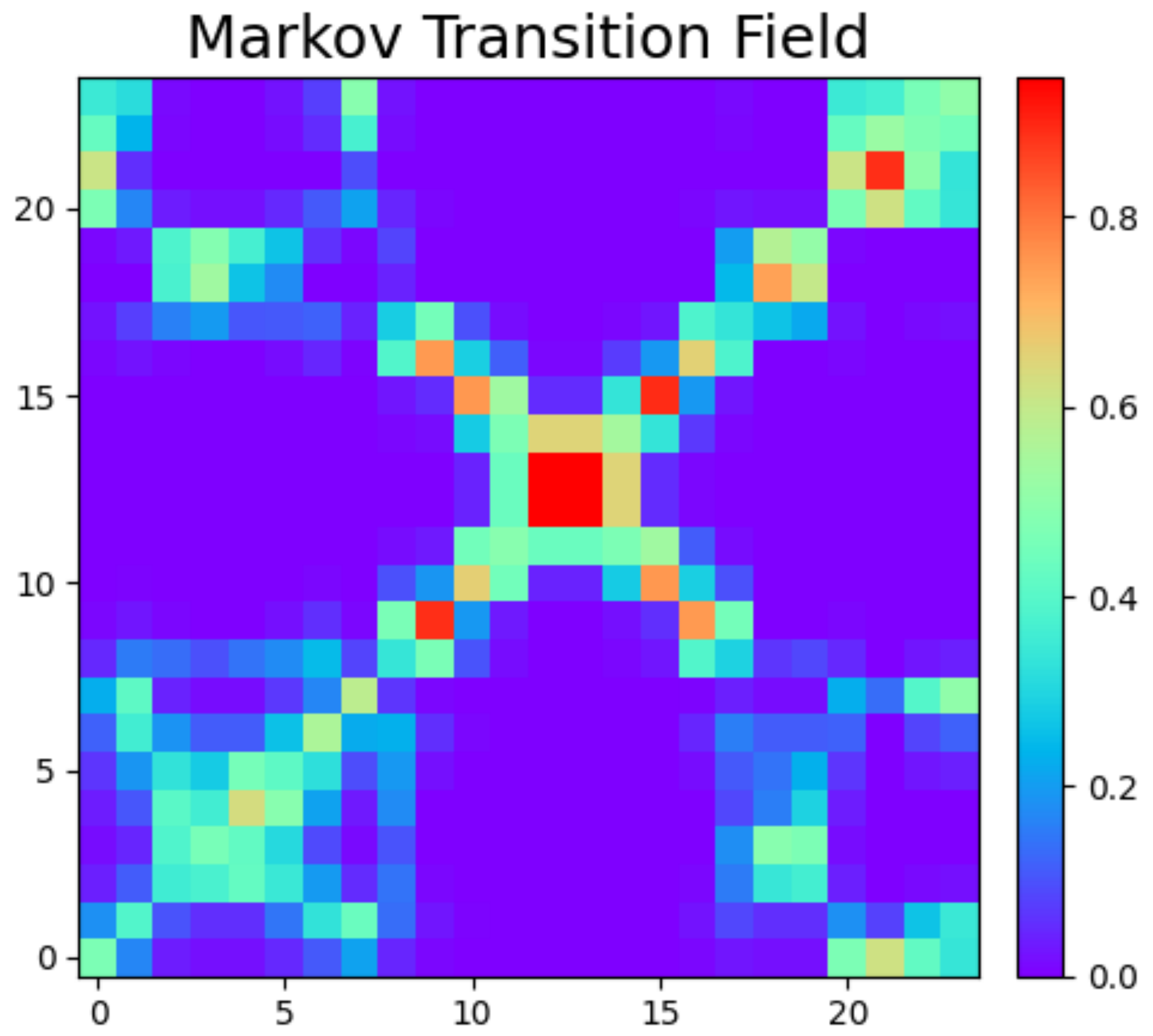


Wang, Zhiguang, and Tim Oates. "Encoding time series as images for visual inspection and classification using tiled convolutional neural networks." In Workshops at the twenty-ninth AAAI conference on artificial intelligence, vol. 1. 2015.

Markov Transition Fields (MTF)

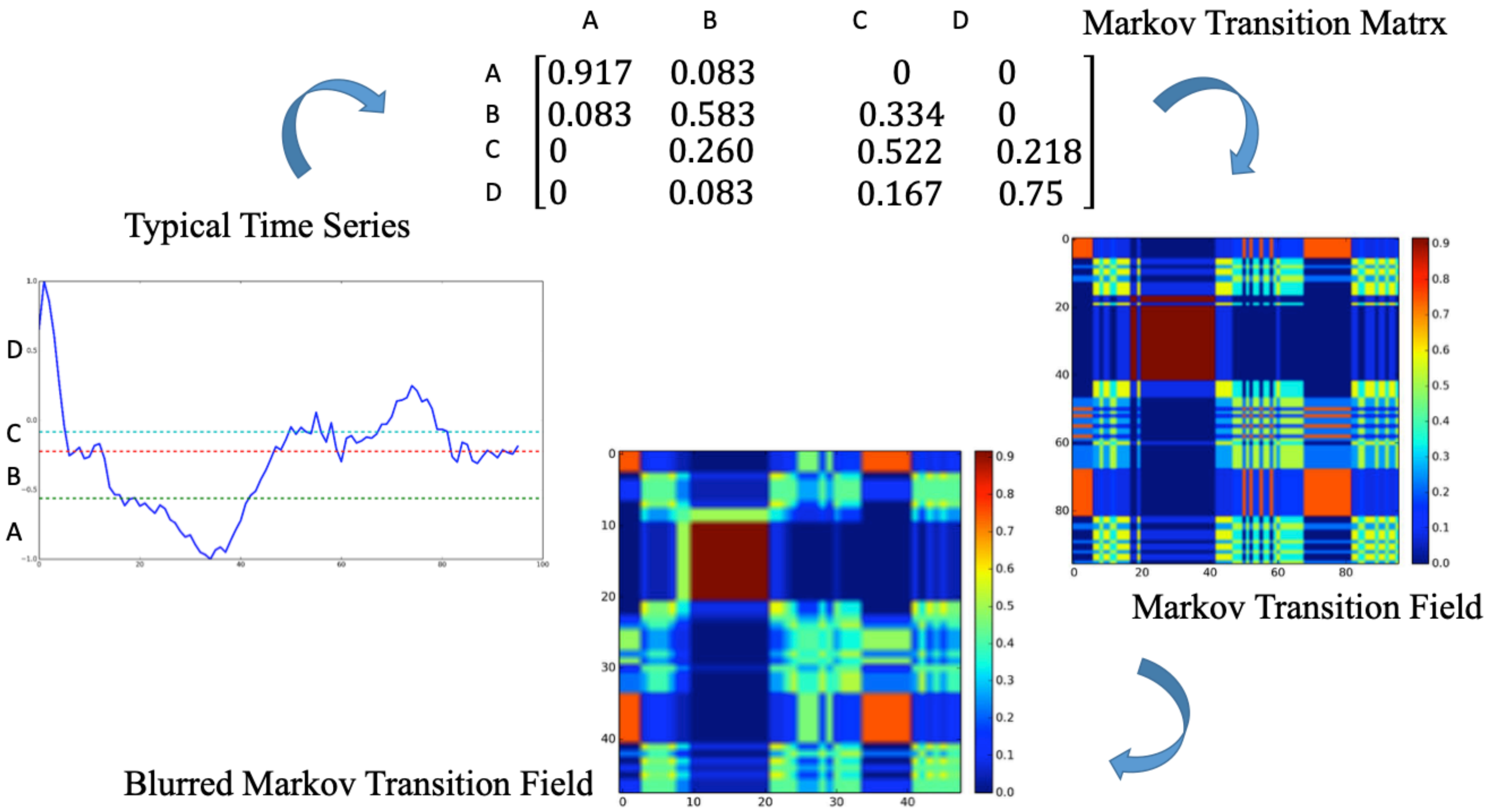
- Discretize a time series into Q quintile bins
- Construct a $Q \times Q$ weighted adjacency matrix W by counting transitions among quantile bins
  - as a first-order Markov chain along the time axis: $w_{i,j}$ = probability that a point in q_j is followed by a point in q_i
- Finally spread out the adjacency matrix to a field in order to reduce the loss of temporal information
  - Compute the Markov Transition Field matrix M of the discretized time series
  - $M_{i,j}$ = probability of transition $q_i \rightarrow q_j$ where q_i and q_j are quantile bins for data at timestamps i and j
  - M encodes the multi-span transition probabilities of the time series
  - $M_{i,j||i-j|=k}$ = transition probability between the points with time interval k
  - MTF size reduced by averaging pixels in non-overlapping $m \times m$ patches

$$M = \begin{bmatrix} w_{ij|x_1 \in q_i, x_1 \in q_j} & \cdots & w_{ij|x_1 \in q_i, x_n \in q_j} \\ w_{ij|x_2 \in q_i, x_1 \in q_j} & \cdots & w_{ij|x_2 \in q_i, x_n \in q_j} \\ \vdots & \ddots & \vdots \\ w_{ij|x_n \in q_i, x_1 \in q_j} & \cdots & w_{ij|x_n \in q_i, x_n \in q_j} \end{bmatrix}$$



https://pyts.readthedocs.io/en/stable/modules/image.html

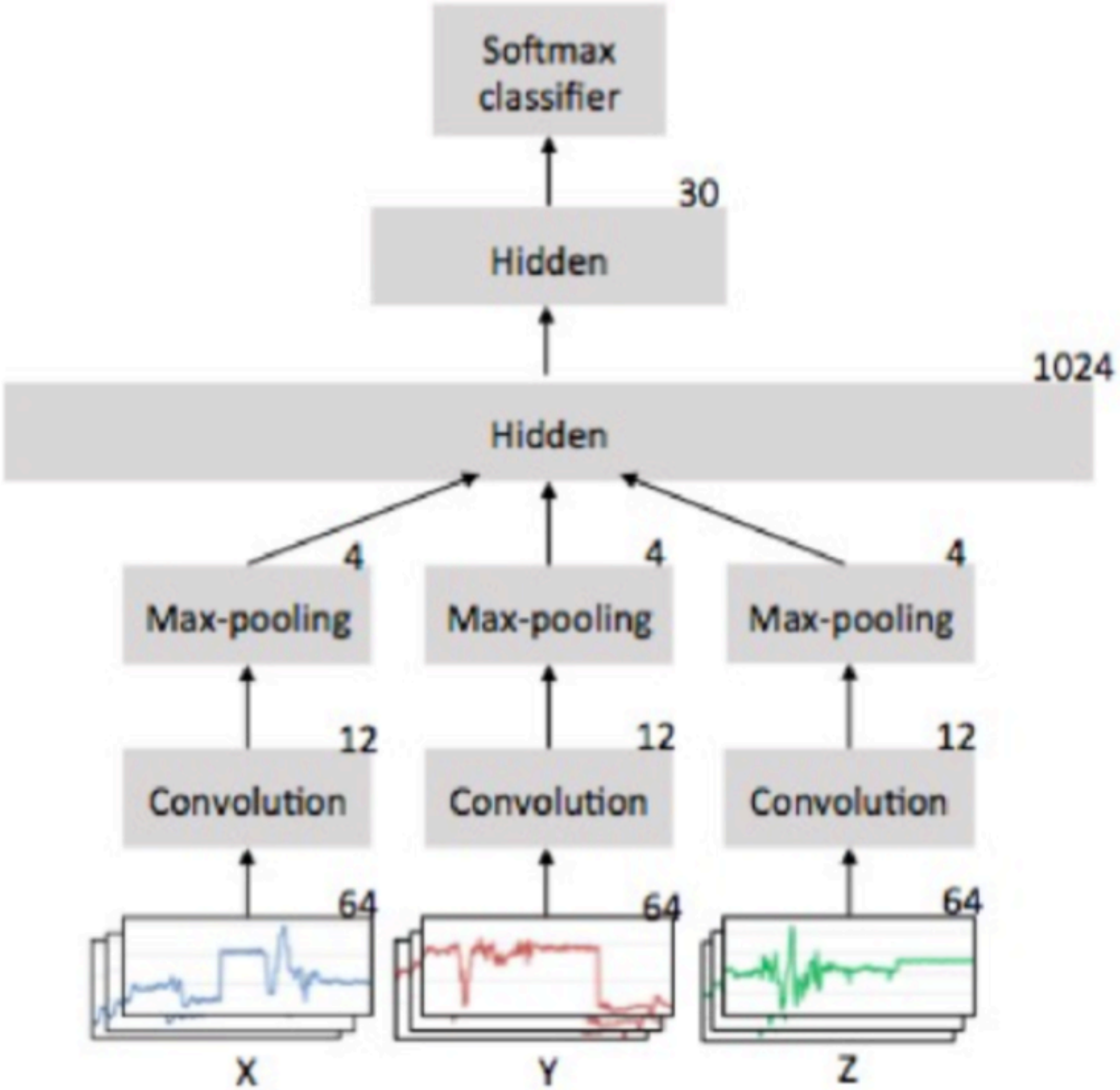
MTF Encoding Under the Hood



Wang, Zhiguang, and Tim Oates. "Encoding time series as images for visual inspection and classification using tiled convolutional neural networks." In Workshops at the twenty-ninth AAAI conference on artificial intelligence, vol. 1. 2015.

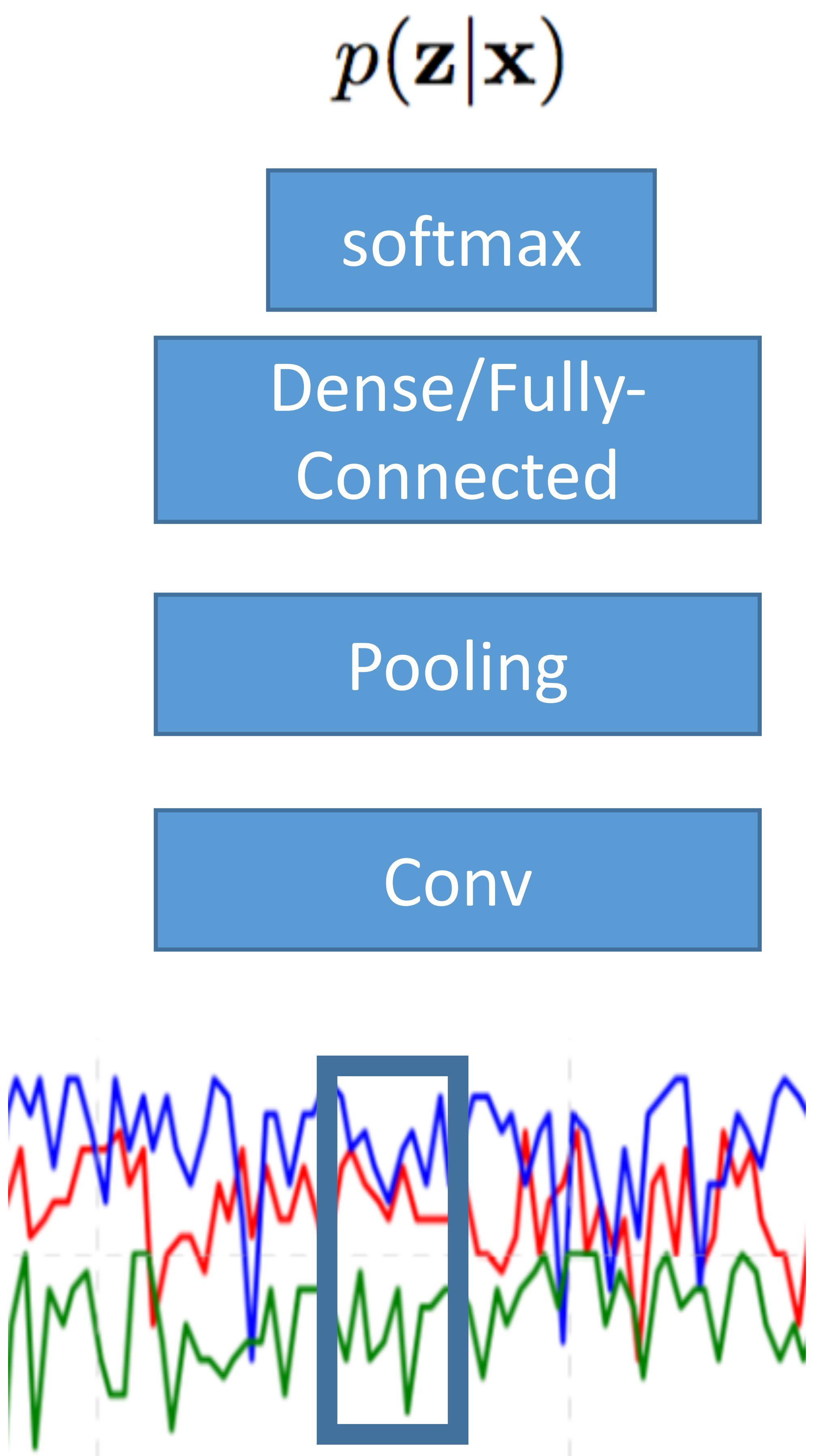
Approach 2: One-Dimensional CNN

TSC using a CNN



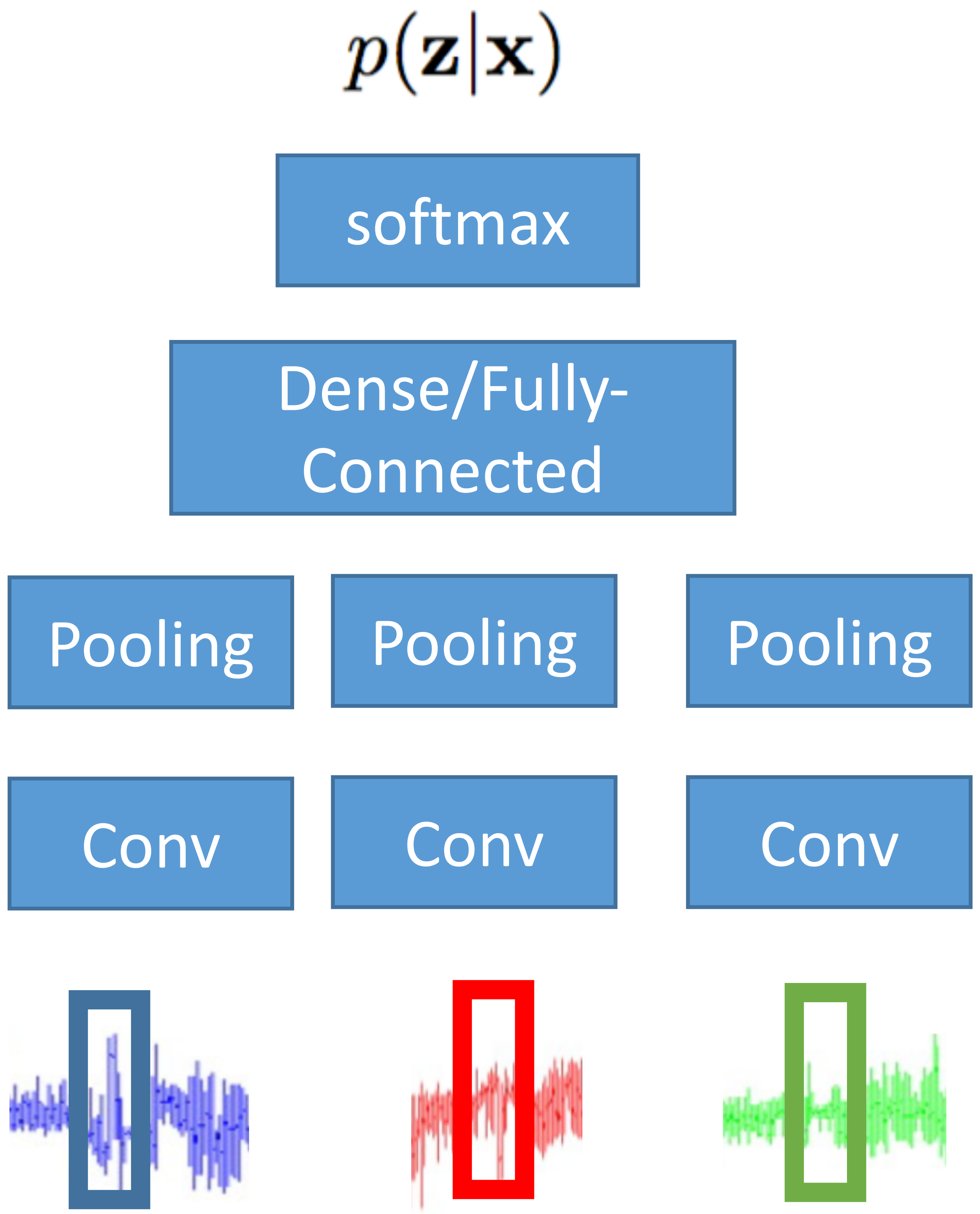
Zeng, Ming, Le T. Nguyen, Bo Yu, Ole J. Mengshoel, Jiang Zhu, Pang Wu, and Joy Zhang. "Convolutional neural networks for human activity recognition using mobile sensors." In 6th International Conference on Mobile Computing, Applications and Services, pp. 197-205. IEEE, 2014. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7026300

Alternative CNNs with 1D or 2D Convolution Kernels



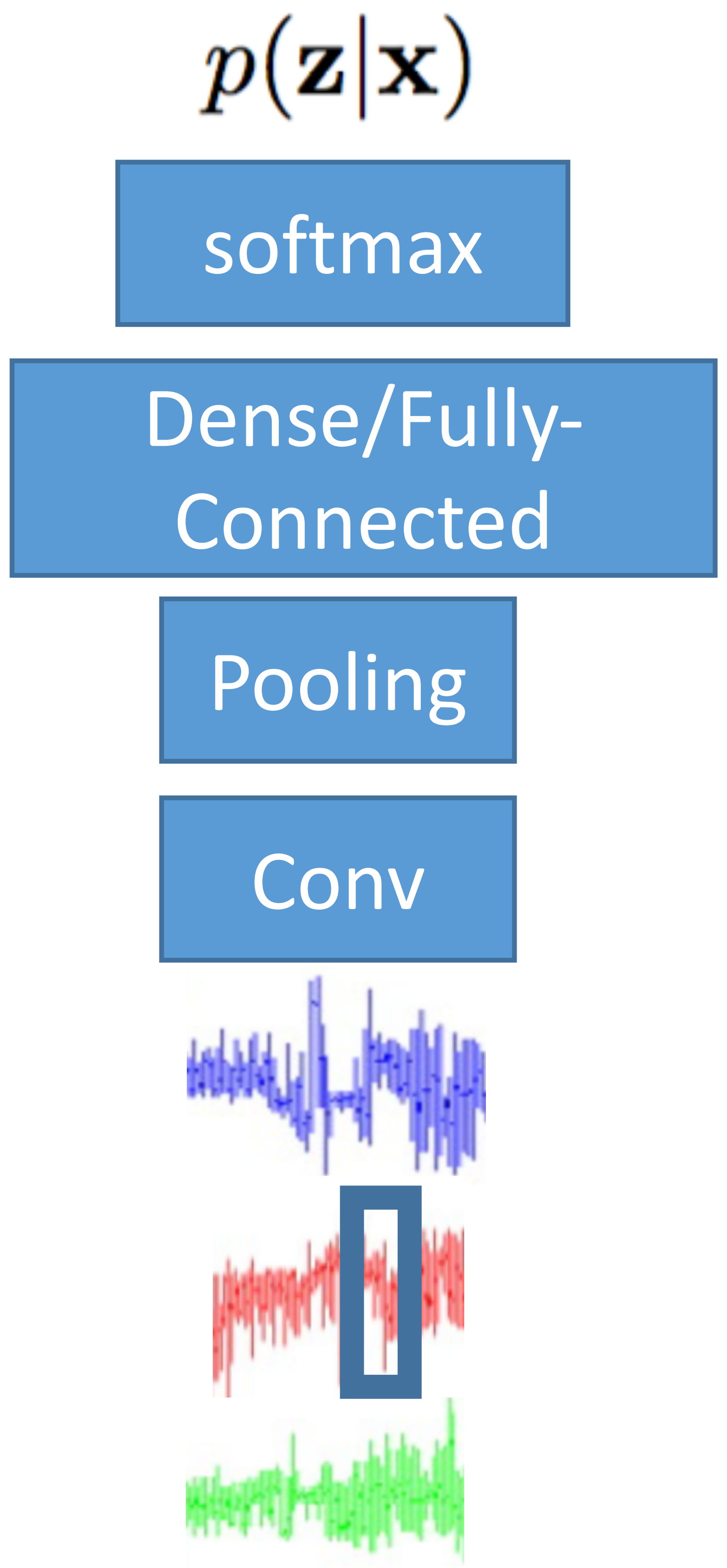
2d kernel

Neverova, et al. 2015



1d kernel

Zeng et al., 2014

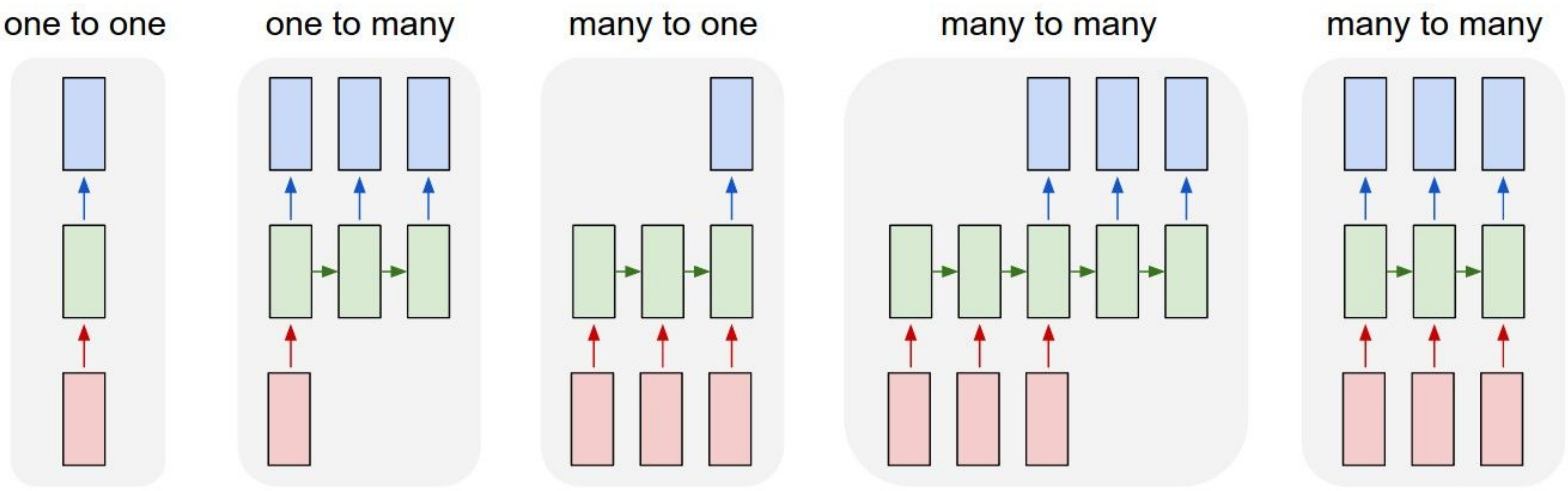


1d kernel

Yang et al. 2015

Approach 3: Recurrent Neural Networks

Sequence Processing with Recurrent Neural Networks (RNN)



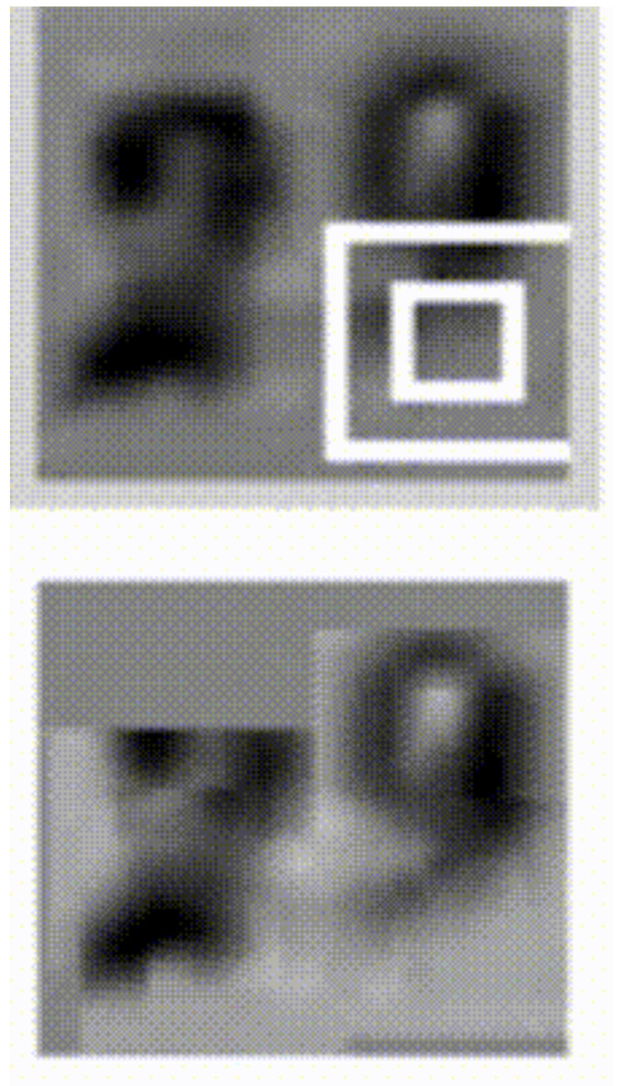
Label an Image | **Give a Caption for an Image** | **Give a Label for a VIdeo** | **Translate Sentence from English to French** | **Classify Frames of a Video**

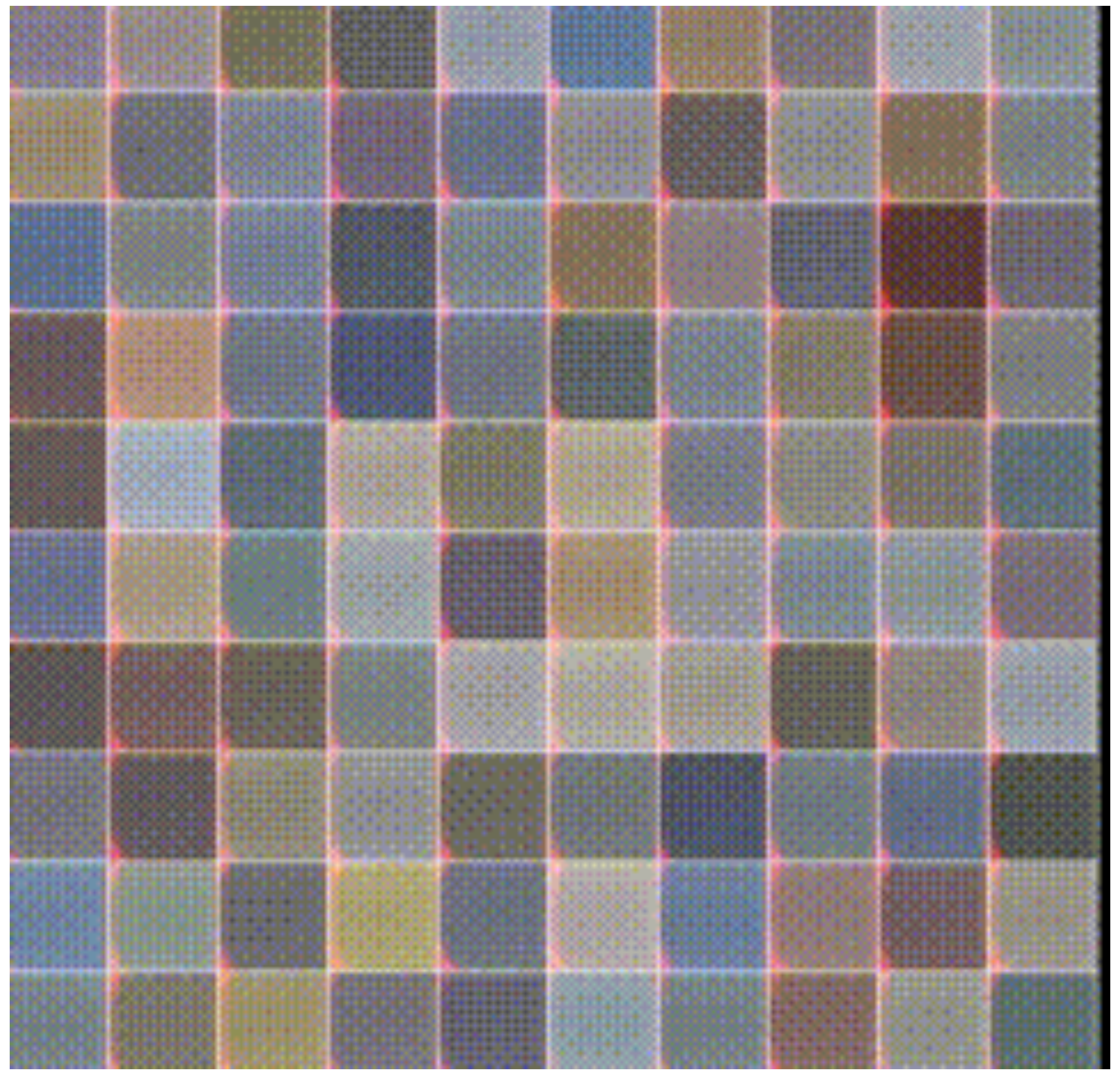
http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Sequence Processing

- RNNs combine the input vector with their state vector with a fixed (but learned) function to produce a new state vector
  - This can in programming terms be interpreted as running a fixed program with certain inputs and some internal variables
  - Viewed this way, RNNs essentially describe programs
  - While CNNs describe pure functions
- Sequential processing is useful in absence of sequences
  - even if your inputs/outputs are fixed vectors, it is still possible to use RNNs to process them sequentially

An algorithm learns a recurrent network policy that steers its attention around an image (specifically, it learns to read out house numbers from left to right)





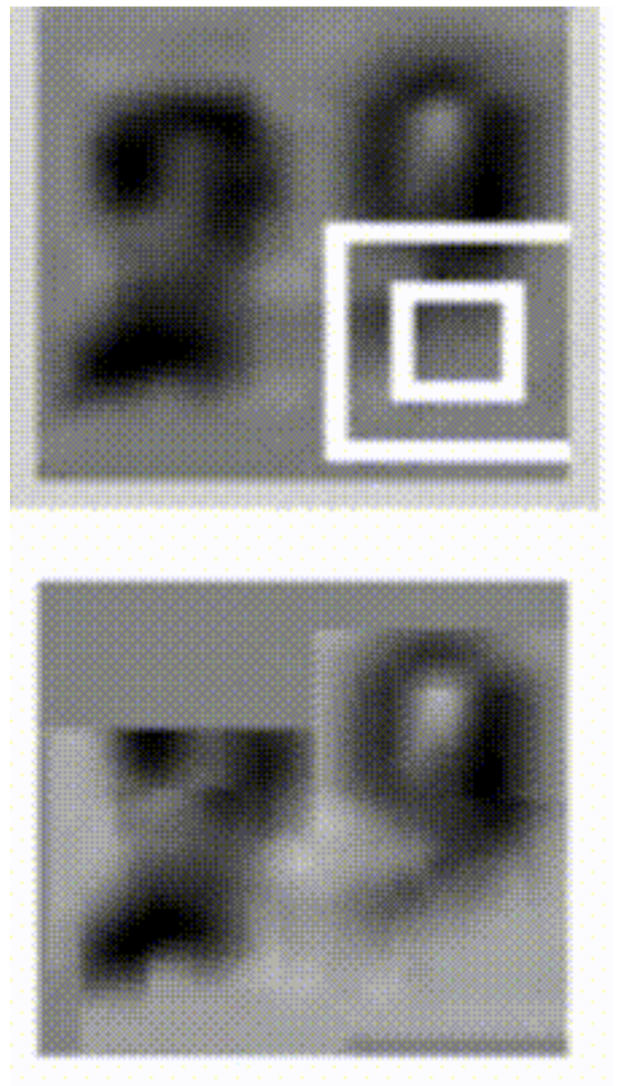
A recurrent network generates images of digits by learning to sequentially add color to a canvas

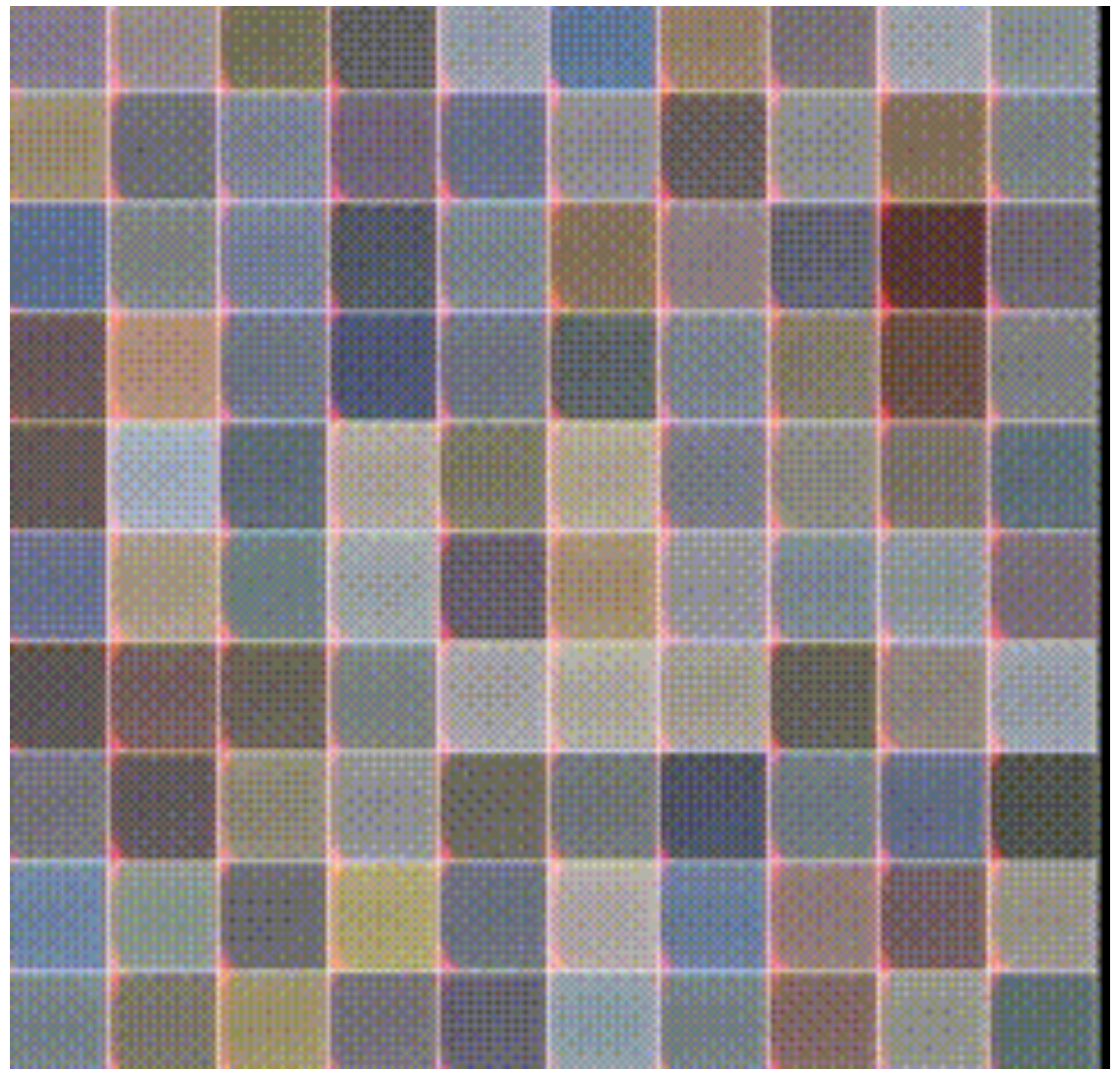
http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Sequence Processing

- RNNs combine the input vector with their state vector with a fixed (but learned) function to produce a new state vector
  - This can in programming terms be interpreted as running a fixed program with certain inputs and some internal variables
  - Viewed this way, RNNs essentially describe programs
  - While CNNs describe pure functions
- Sequential processing is useful in absence of sequences
  - even if your inputs/outputs are fixed vectors, it is still possible to use RNNs to process them sequentially

An algorithm learns a recurrent network policy that steers its attention around an image (specifically, it learns to read out house numbers from left to right)





A recurrent network generates images of digits by learning to sequentially add color to a canvas

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

RNN = CNN + Memory

We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

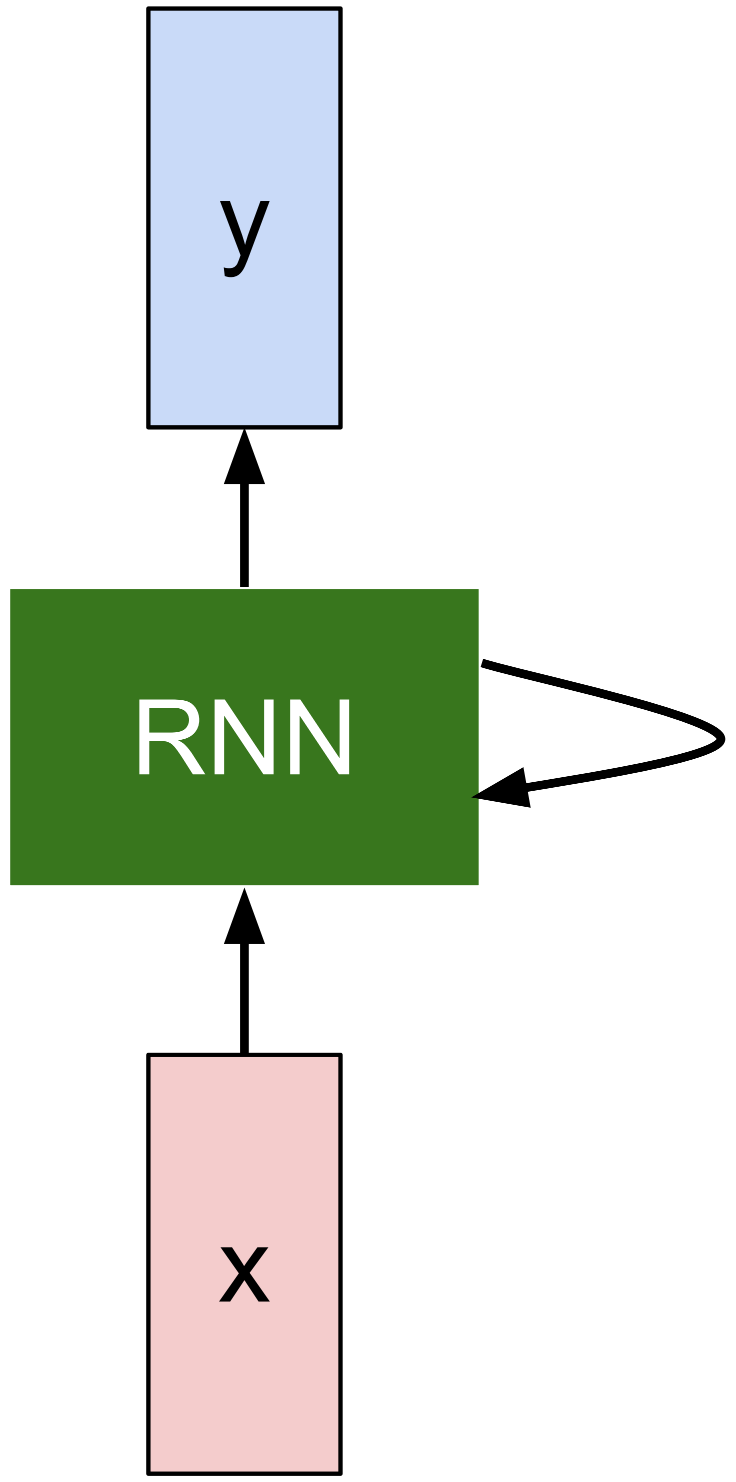
$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

new state

some function with parameters W

old state

input vector at some time step



RNN computation

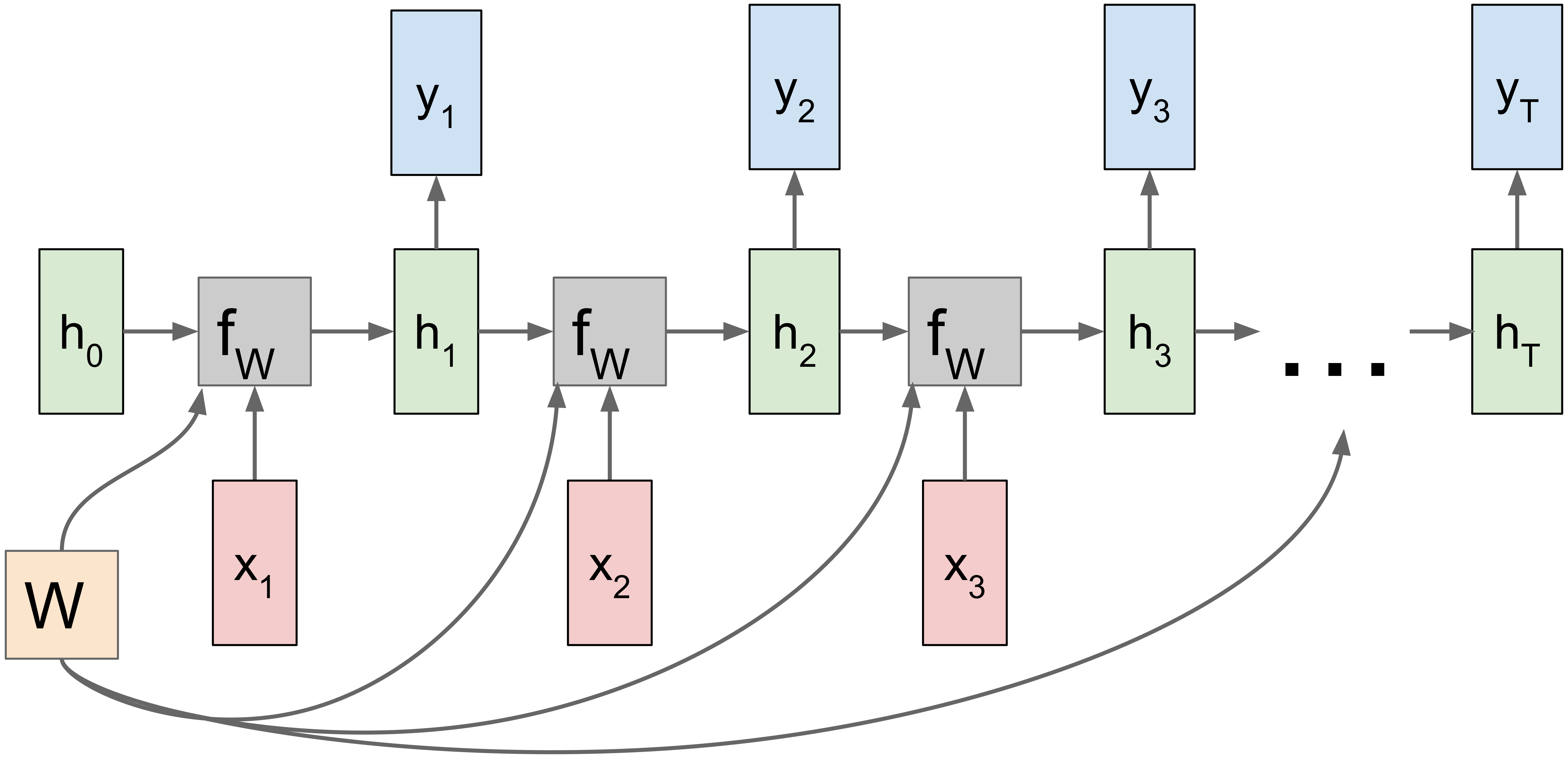
```
rnn = RNN()
y = rnn.step(x) # x is an input vector, y is the RNN's output vector
```

The RNN class has some internal state that it gets to update every time `step` is called. In the simplest case this state consists of a single *hidden* vector `h`. Here is an implementation of the step function in a Vanilla RNN:

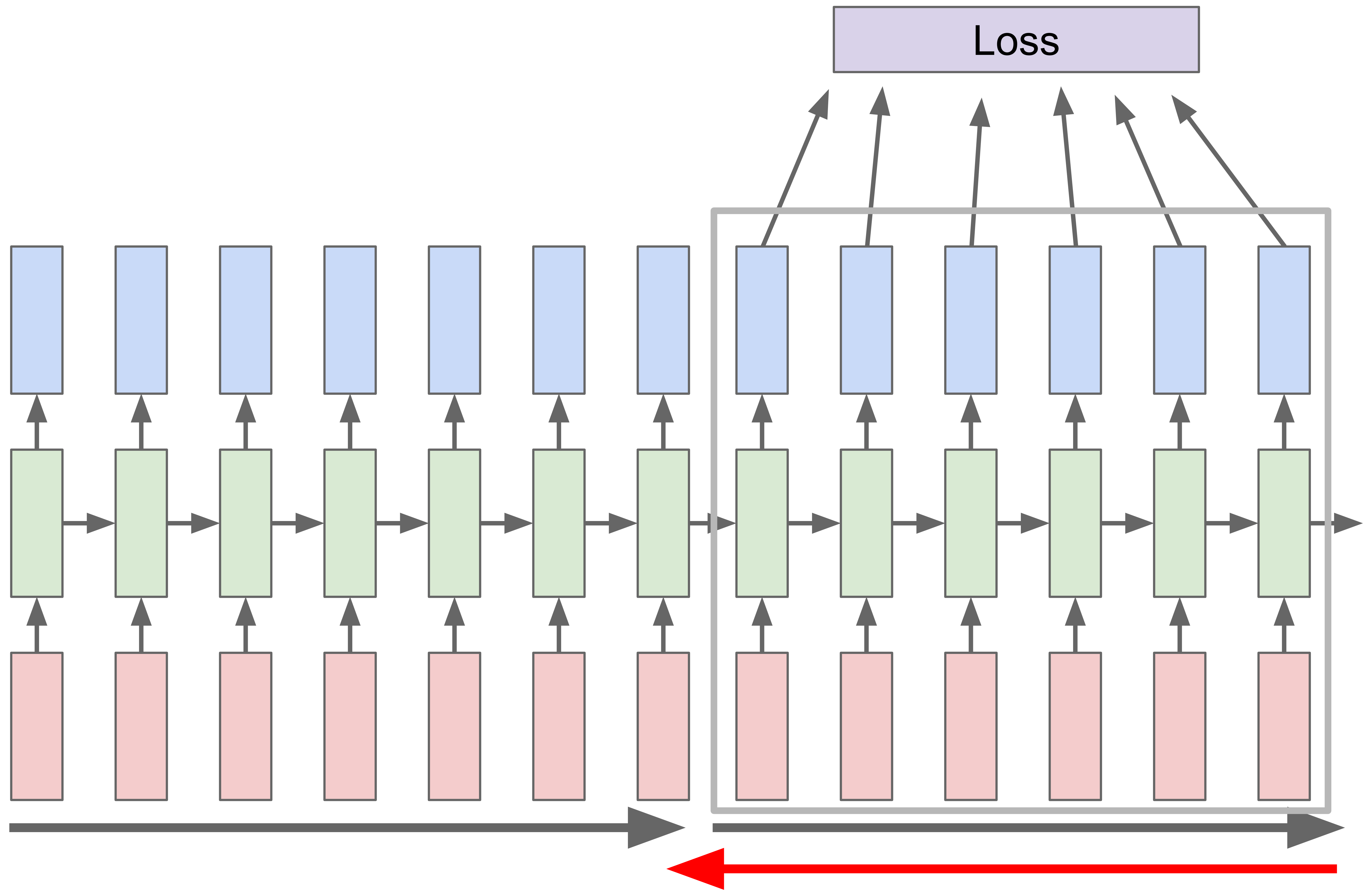
```
class RNN:
  # ...
  def step(self, x):
    # update the hidden state
    self.h = np.tanh(np.dot(self.W_hh, self.h) + np.dot(self.W_xh, x))
    # compute the output vector
    y = np.dot(self.W_hy, self.h)
    return y
```


http://karpathy.github.io/2015/05/21/rnn-effectiveness/

RNN Processing Unfolded in Time

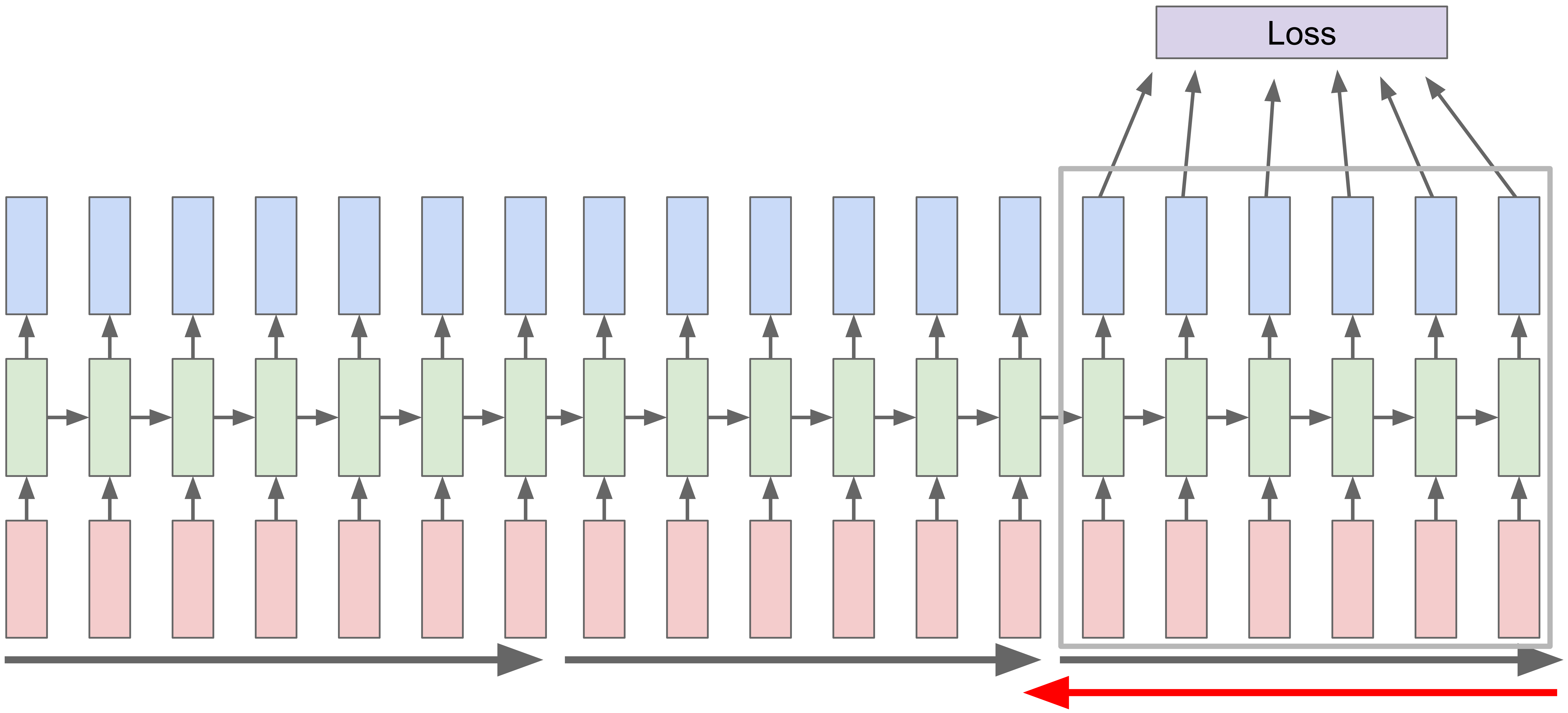


Backpropagation Over (Truncated) Time

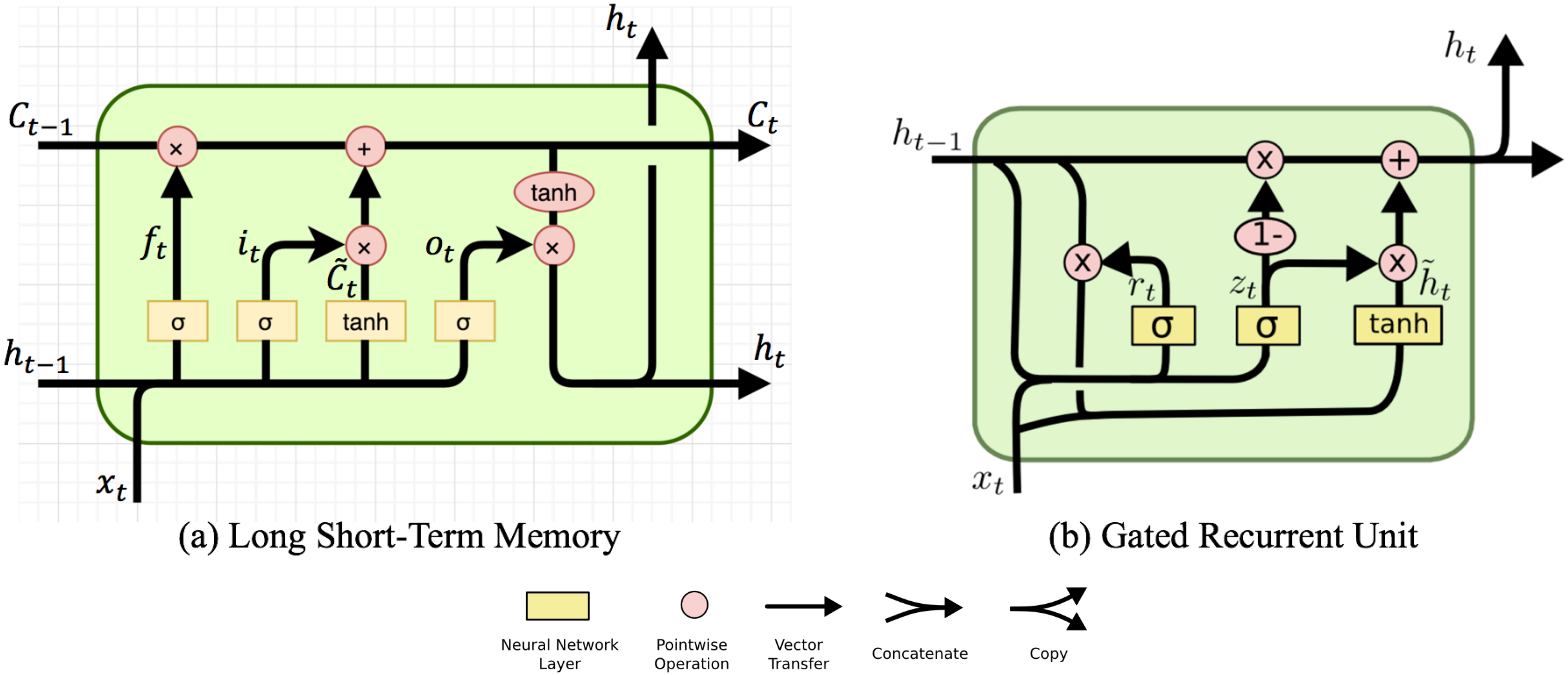


Carry hidden states forward in time forever, but only backpropagate for some smaller number of steps

Backpropagation Over (Truncated) Time



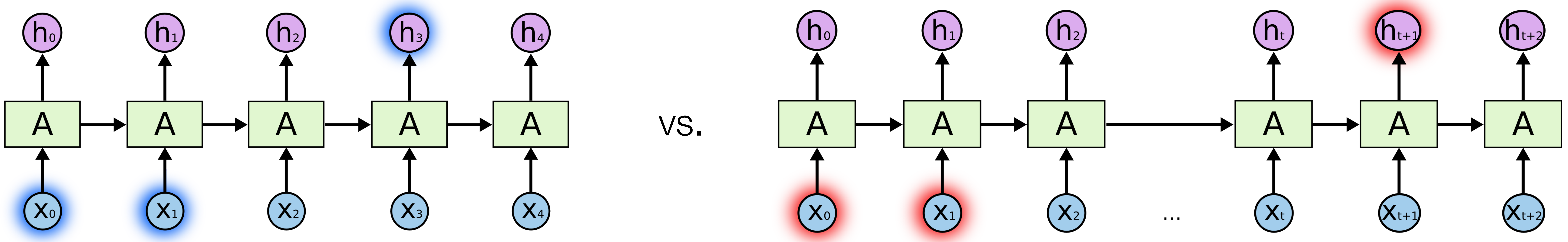
Inside the Green Box



http://colah.github.io/posts/2015-08-Understanding-LSTMs/

The Problem of Long-Term Dependencies

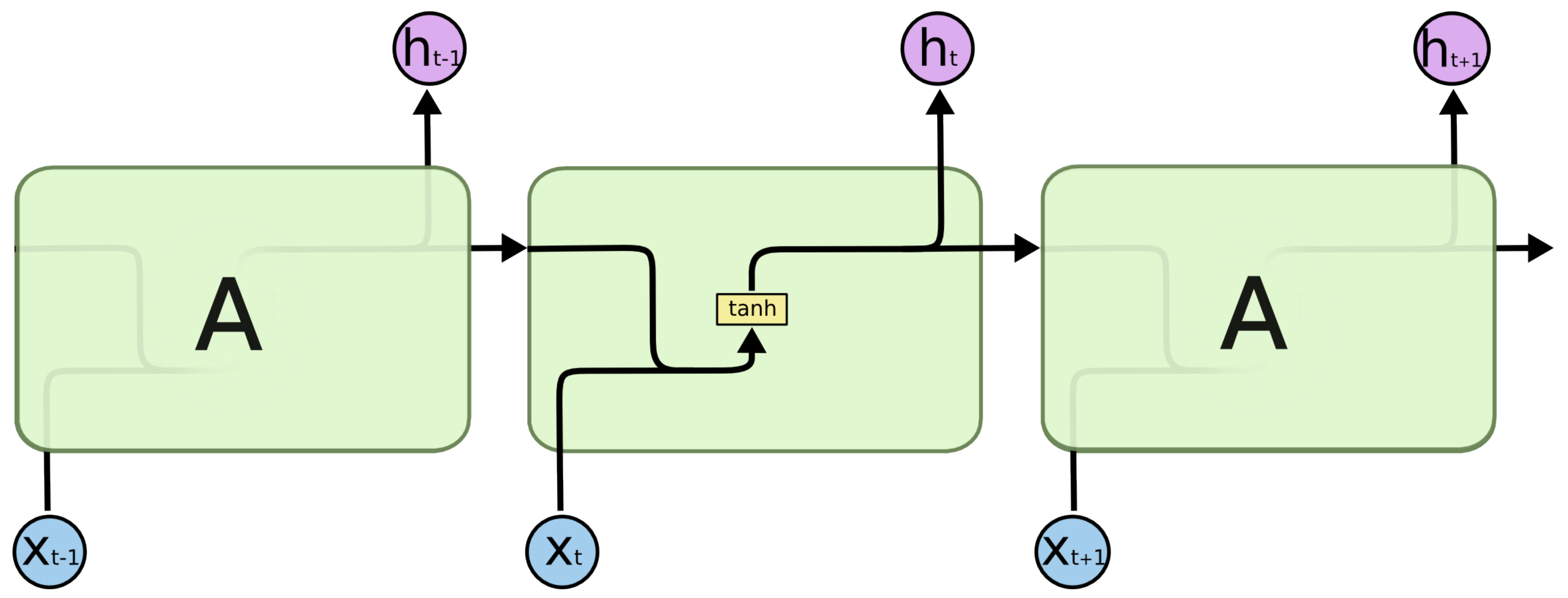
- One of the appeals of RNNs is that they might be able to connect previous information to the present task
  - e.g. using previous video frames might inform the understanding of the present frame.
- If RNNs could do this, they'd be extremely useful. But can they? Answer: It depends.
- Sometimes, we only need to look at recent information to perform the present task.
  - In such cases, where the gap between the relevant information and the place that it's needed is small, RNNs can learn to use the past information.
- But there are also cases where we need more context.
  - Unfortunately, as that gap grows, RNNs become unable to learn to connect the information.



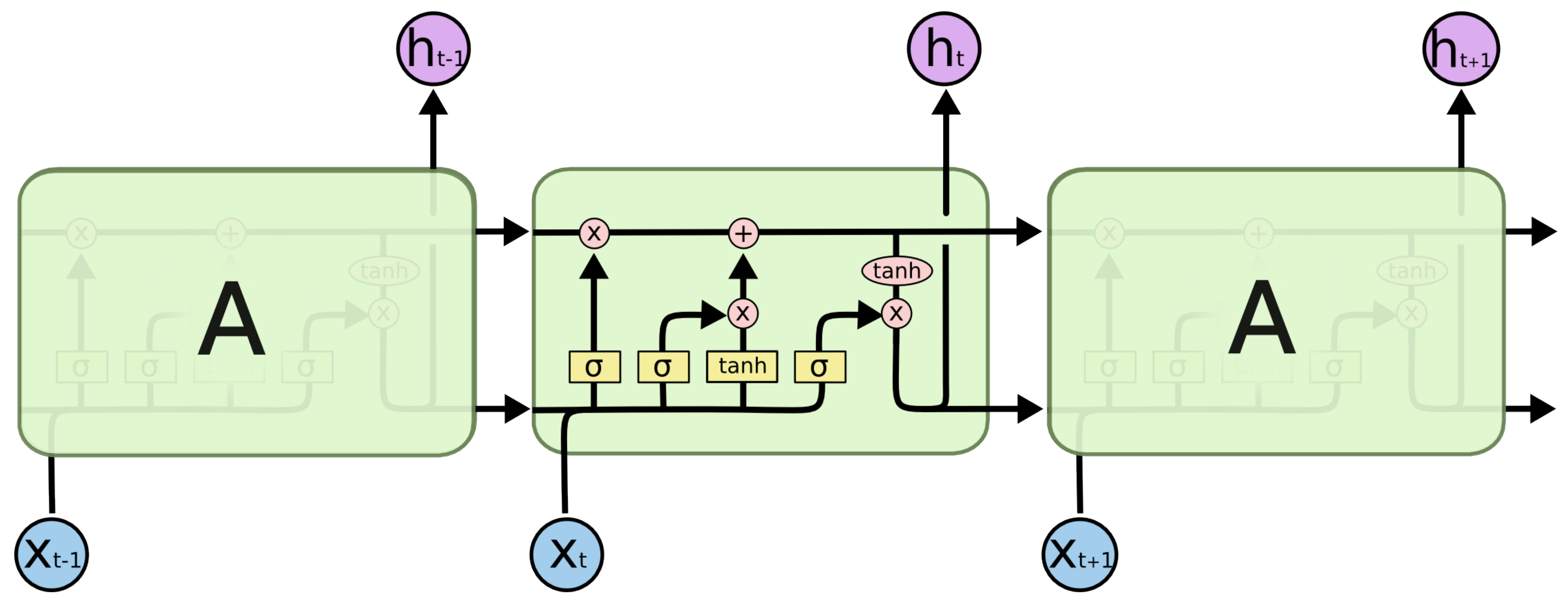
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Long Short Term Memory (LSTM) networks

- Special kind of RNN, capable of learning long-term dependencies



Standard RNN

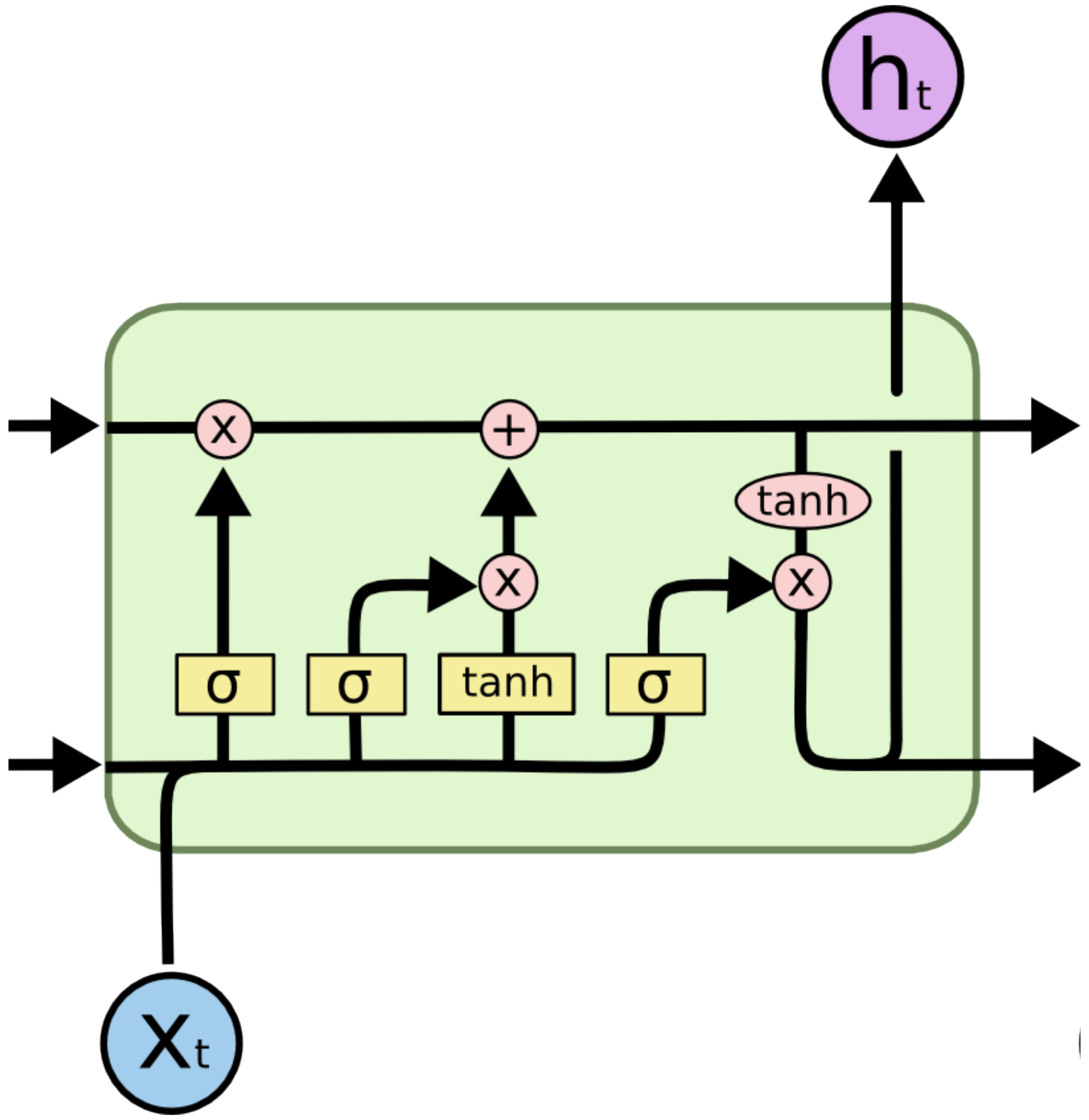


LSTM

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

The Core Idea Behind LSTMs

- Cell state: the horizontal line running through the LSTM cell
- Structures called *gates* carefully regulate addition or removal of information to the cell state
  - composed out of a sigmoid layer and a pointwise multiplication operation
  - sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through
- An LSTM has three of these gates, to protect and control the cell state.



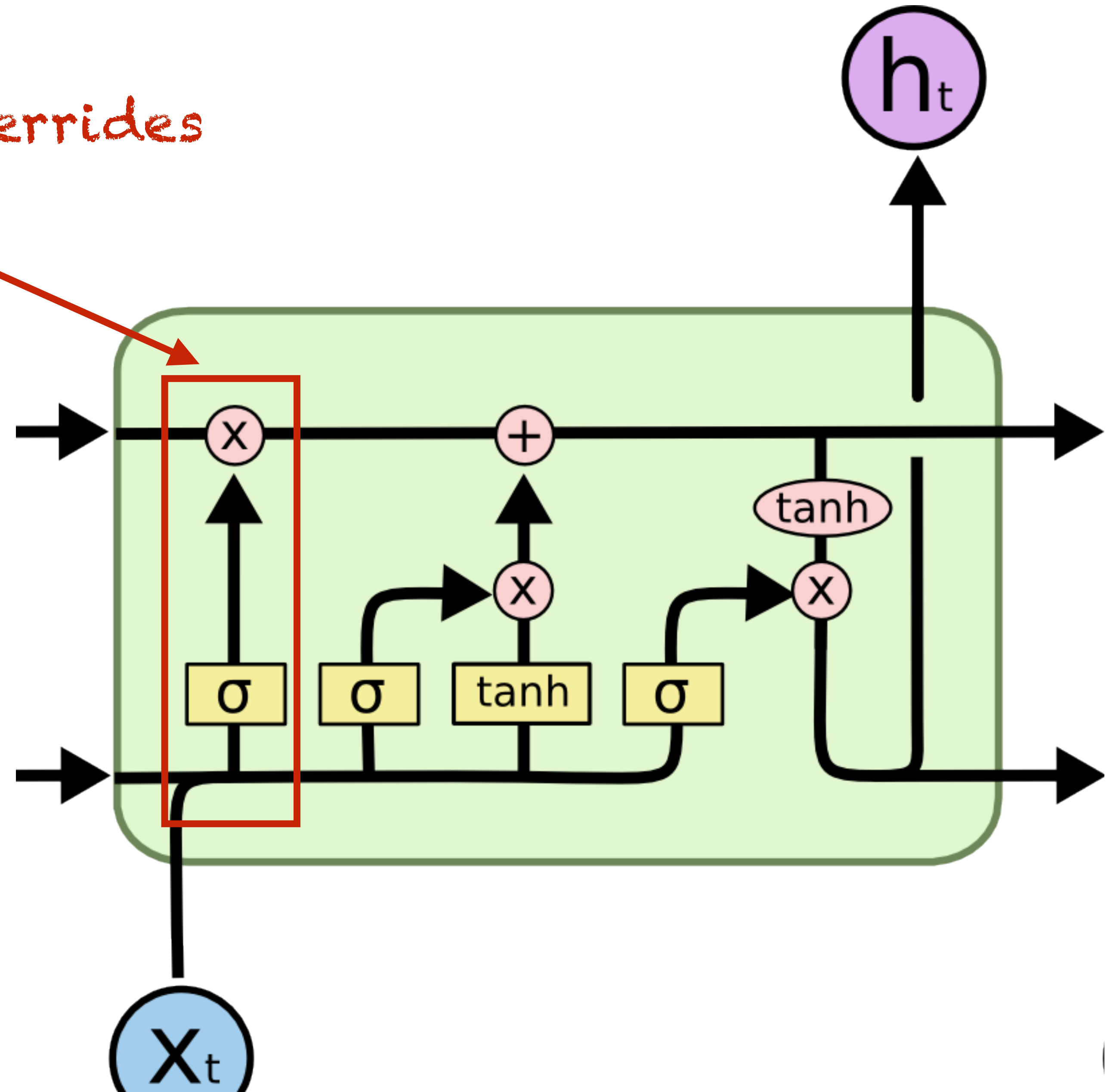
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

The Core Idea Behind LSTMs

- Cell state: the horizontal line running through the LSTM cell
- Structures called *gates* carefully regulate addition or removal of information to the cell state
  - composed out of a sigmoid layer and a pointwise multiplication operation
  - sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through
- An LSTM has three of these gates, to protect and control the cell state.

forget when see something new that overrides

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$



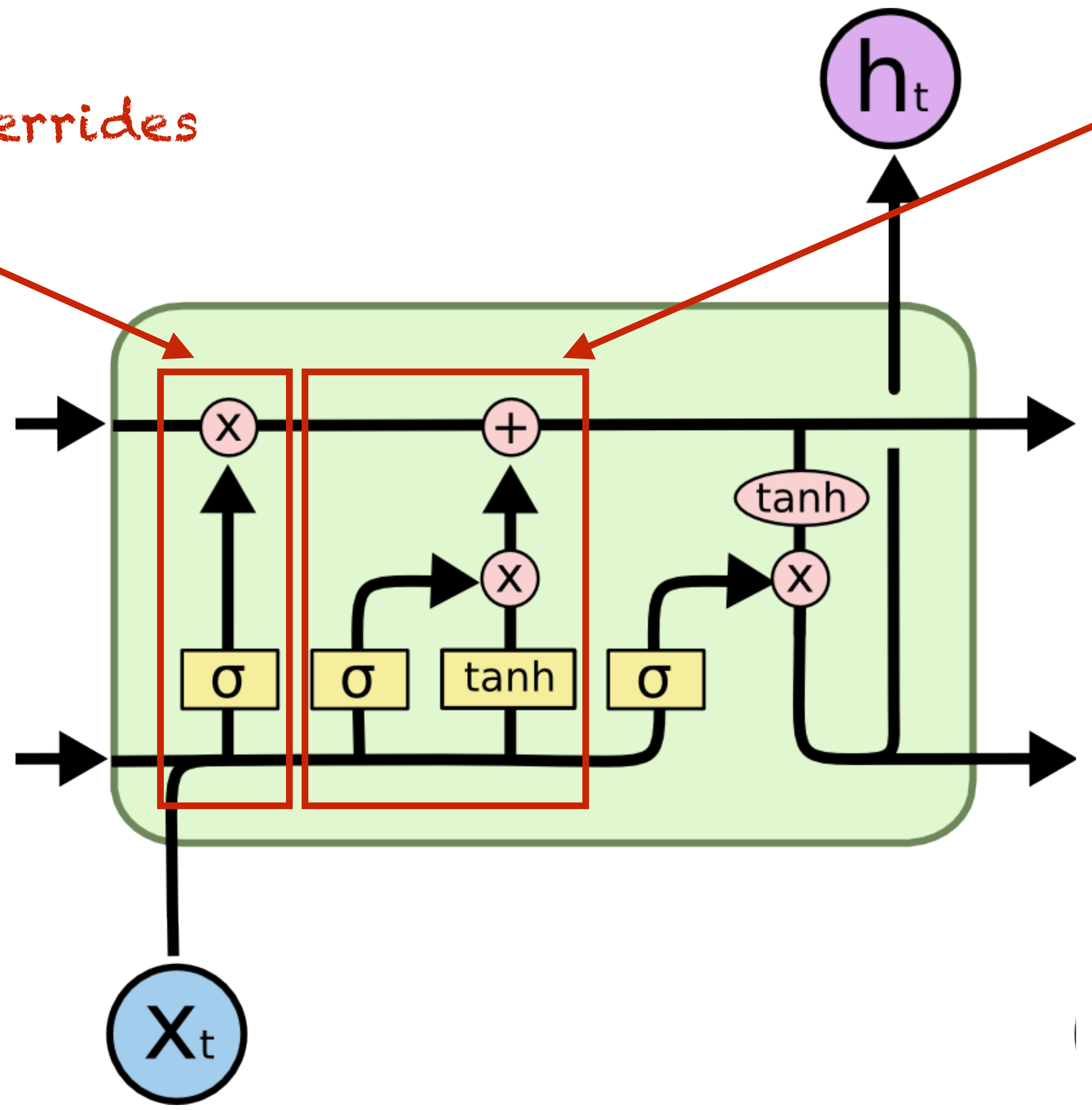
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

The Core Idea Behind LSTMs

- Cell state: the horizontal line running through the LSTM cell
- Structures called *gates* carefully regulate addition or removal of information to the cell state
  - composed out of a sigmoid layer and a pointwise multiplication operation
  - sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through
- An LSTM has three of these gates, to protect and control the cell state.

forget when see something new that overrides

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$



what new information to store

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

The Core Idea Behind LSTMs

- Cell state: the horizontal line running through the LSTM cell
- Structures called *gates* carefully regulate addition or removal of information to the cell state
  - composed out of a sigmoid layer and a pointwise multiplication operation
  - sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through
- An LSTM has three of these gates, to protect and control the cell state.

forget when see something new that overrides

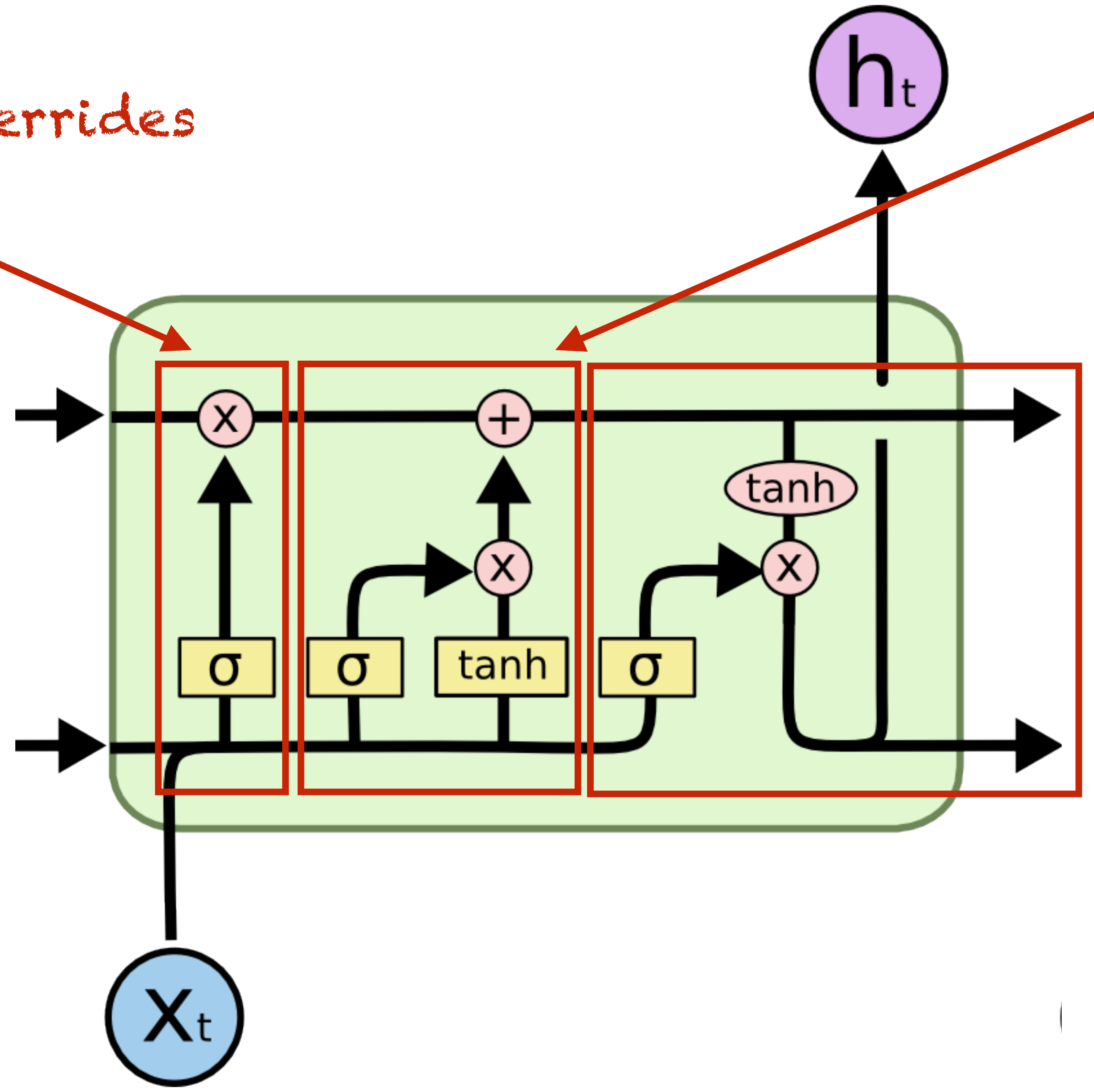
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

what new information to store

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$



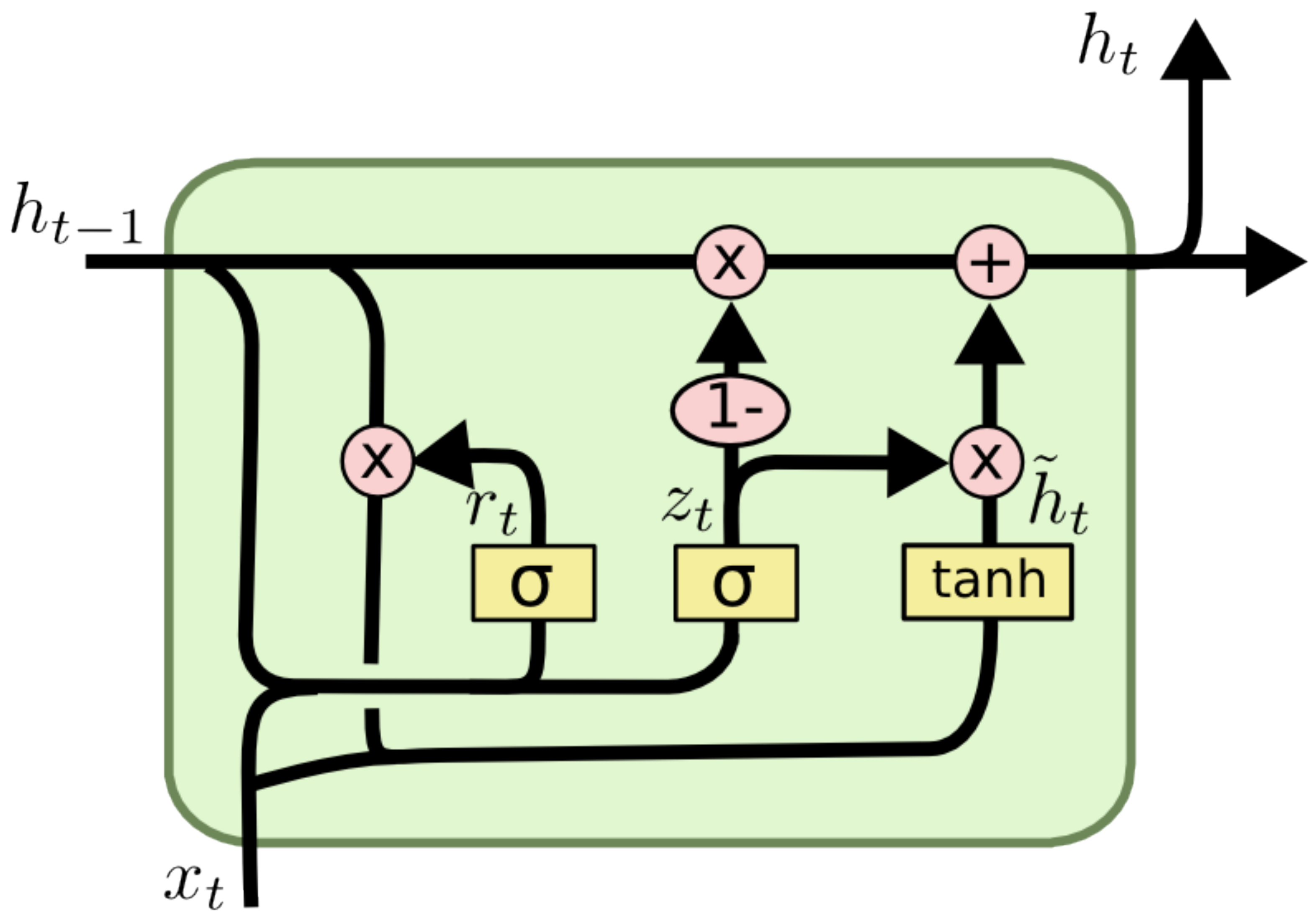
what are we going to output

$$o_t = \sigma\left(W_o \ [h_{t-1}, x_t] \ + \ b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

Many Variants of LSTM

- Most common: Gated Recurrent Unit, or GRU
- It combines the forget and input gates into a single “update gate.”
- It also merges the cell state and hidden state, and makes some other changes.
- The resulting model is simpler than standard LSTM models



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

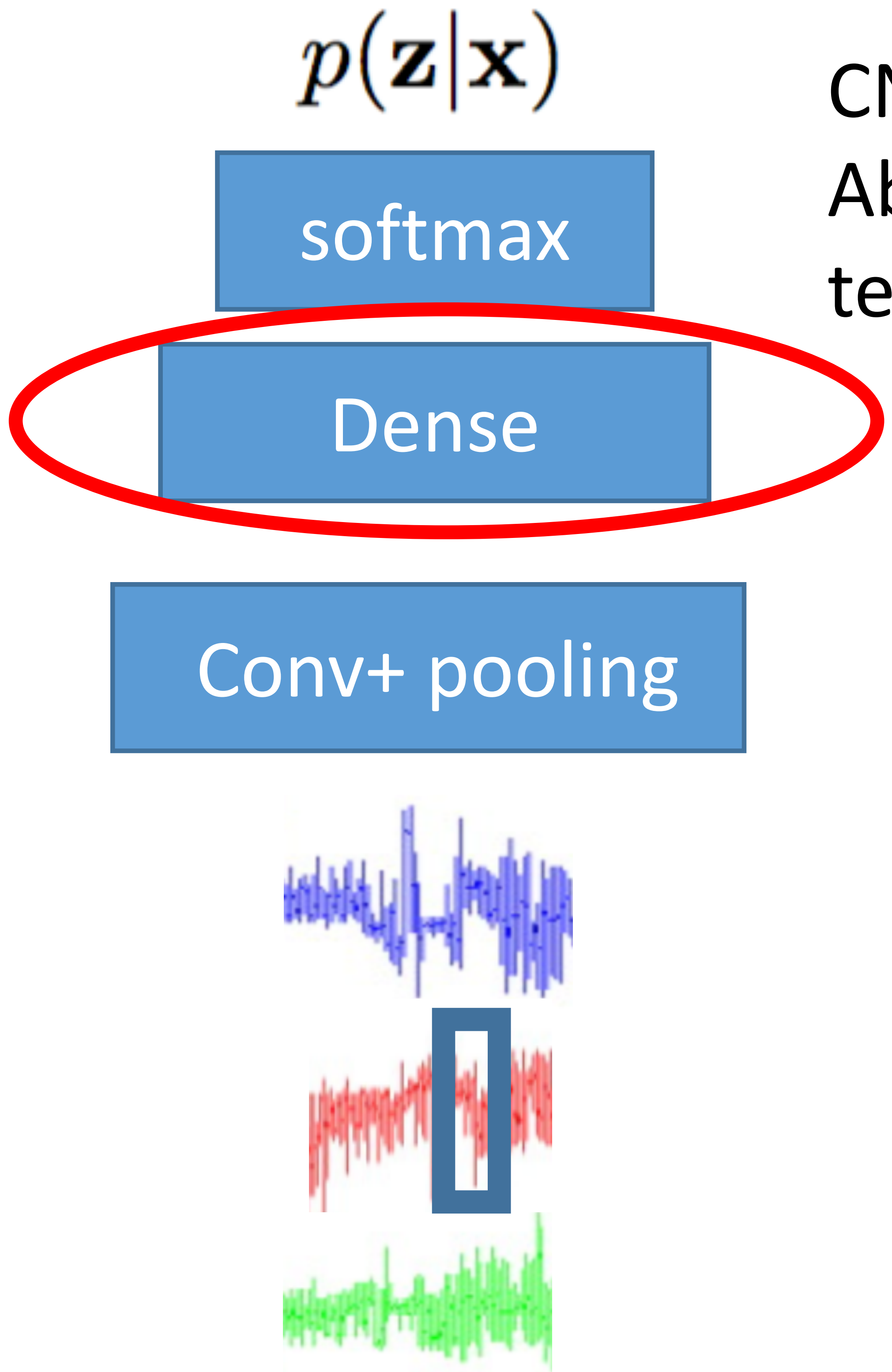
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

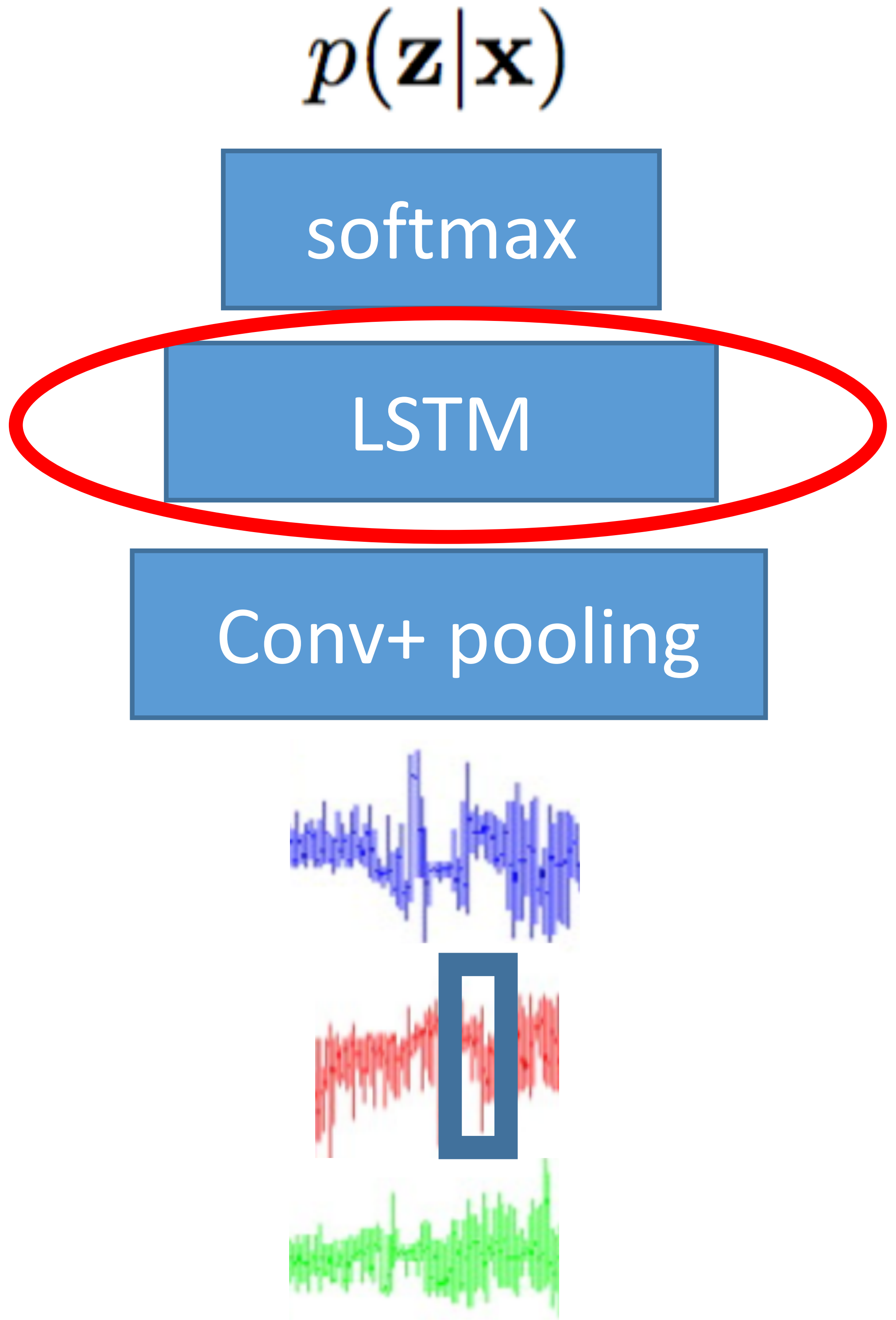
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Combining Conv and LSTM: ConvLSTM



CNN:
Abandoning
temporal structure



ConvLSTM:
Taking advantage of
temporal structure

Stacking RNNs

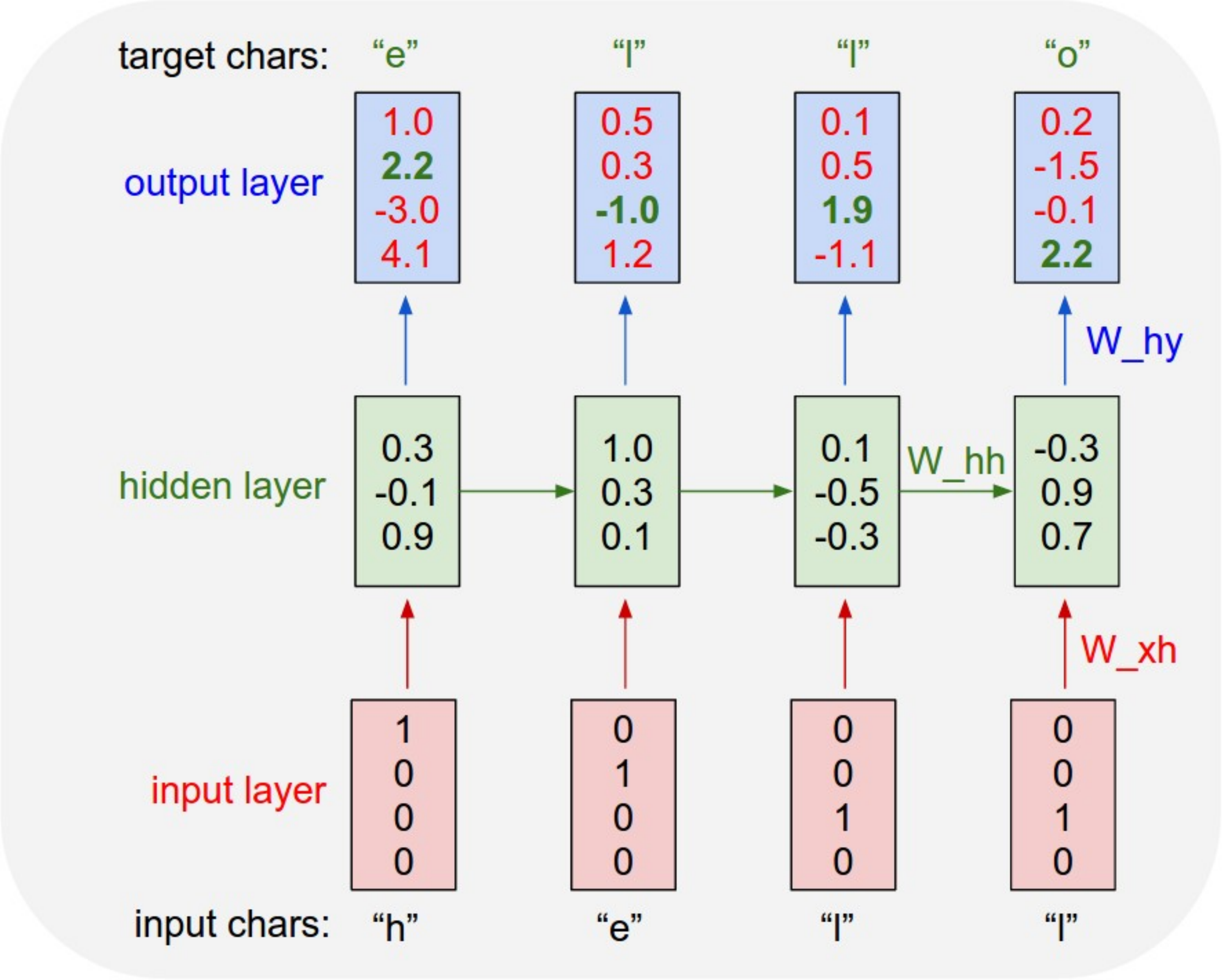
- Just like other types of layers in deep neural networks, one can stack RNNs
- E.g. one can form a two layer RNN

```
y1 = rnn1.step(x)
y = rnn2.step(y1)
```

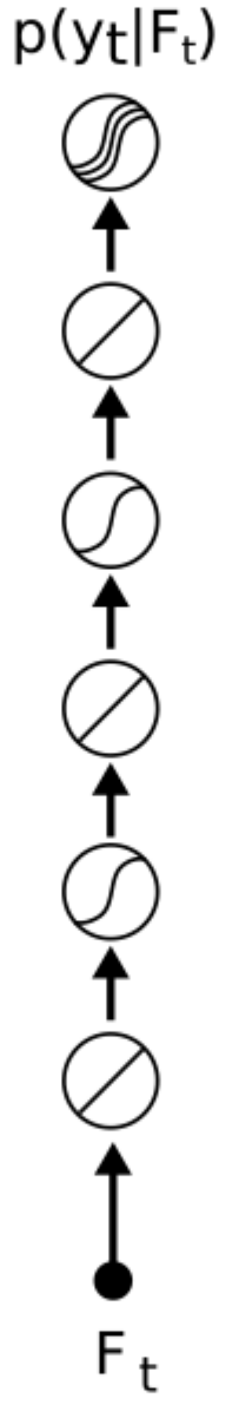

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Using RNNs as Models of Sequences

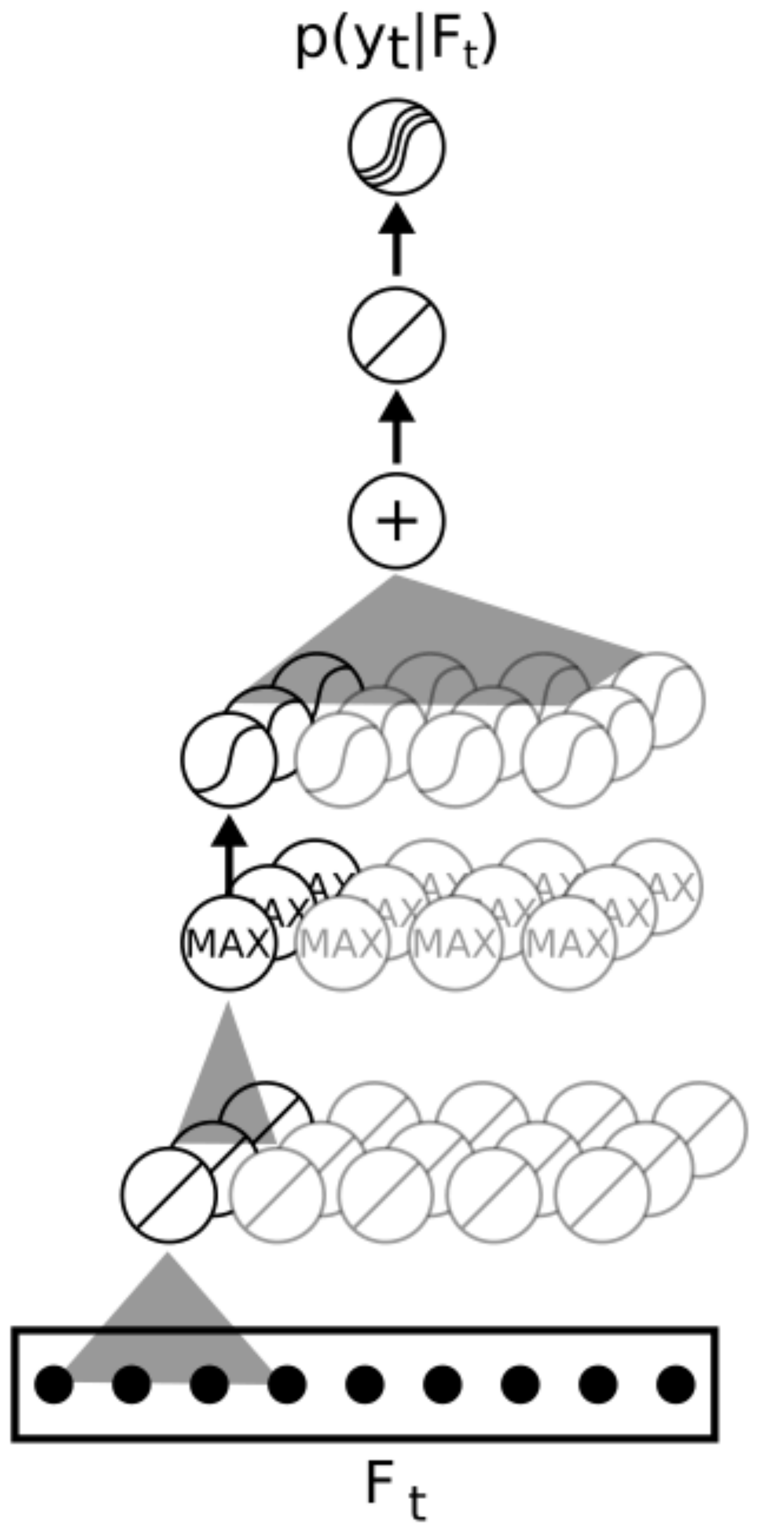
- Give the RNN a large data set of sequences of symbols
  - text, sensor measurements, activity state etc.
- Ask RNN to model the probability distribution of the next symbol in the sequence given a sequence of previous symbol
  - Train using backpropagation
- This will then allow us to generate new sequences one symbol at a time
- Simple example of RNN for character level model of English language
  - https://gist.github.com/karpathy/d4dee566867f8291f086
- Ref: “The Unreasonable Effectiveness of Recurrent Neural Networks”
  - http://karpathy.github.io/2015/05/21/rnn-effectiveness/



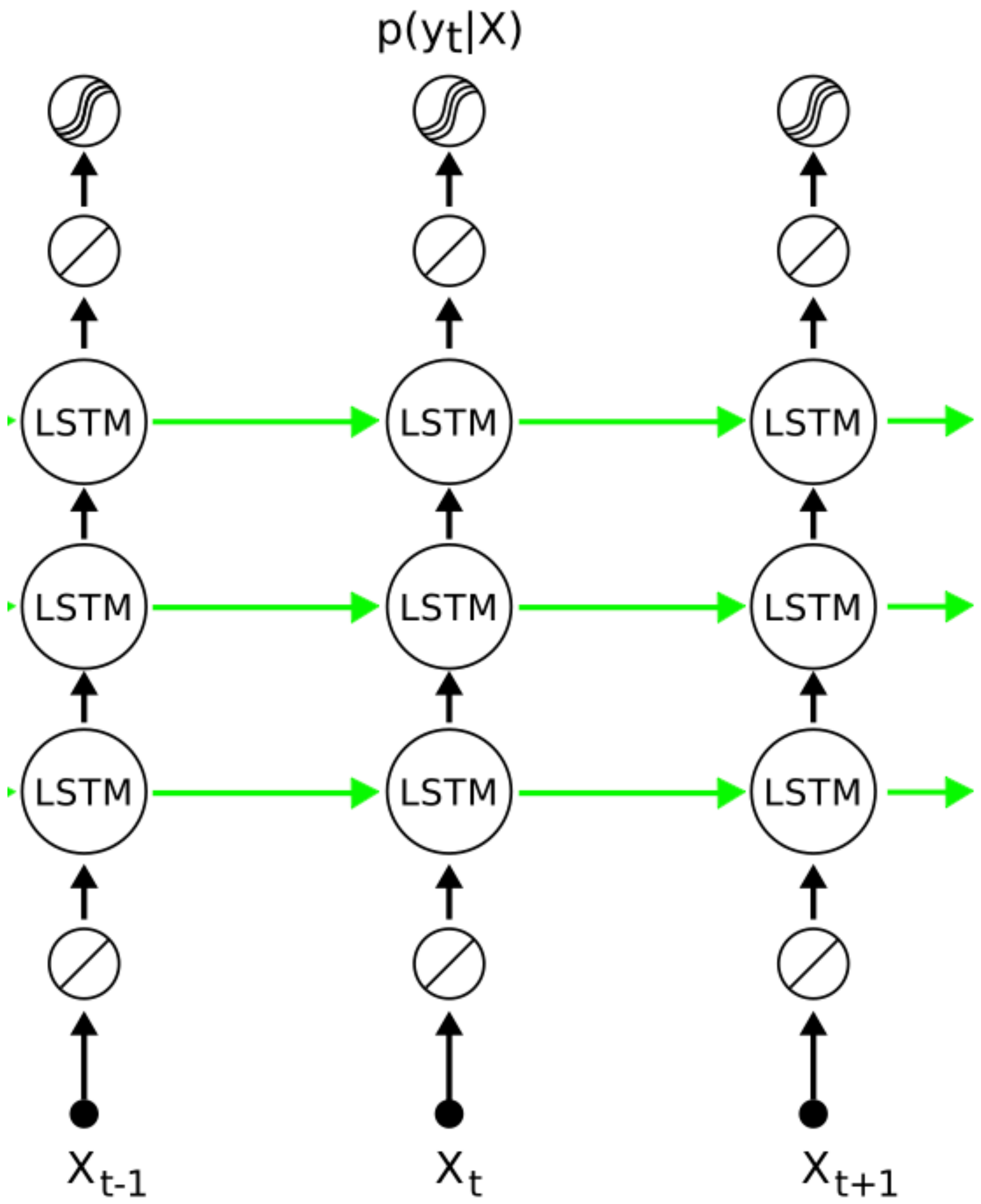
Rich Variety of DNN Models for HAR



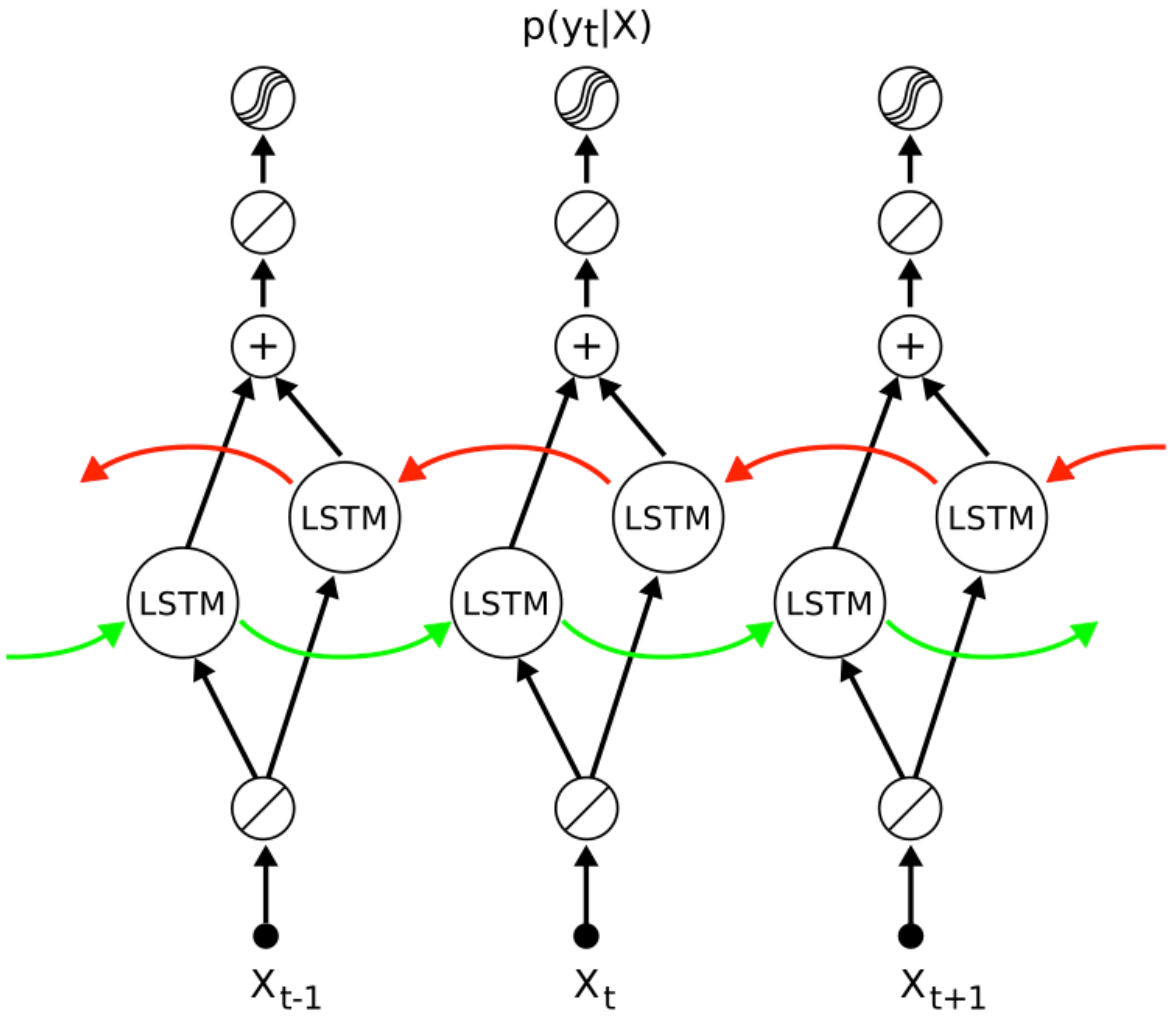
Fully connected feed-forward network with hidden (ReLU) layers



Convolutional networks that contain layers of convolutions and max-pooling, followed by fully-connected layers and a softmax group



LSTM network hidden layers containing LSTM cells and a final softmax layer at the top



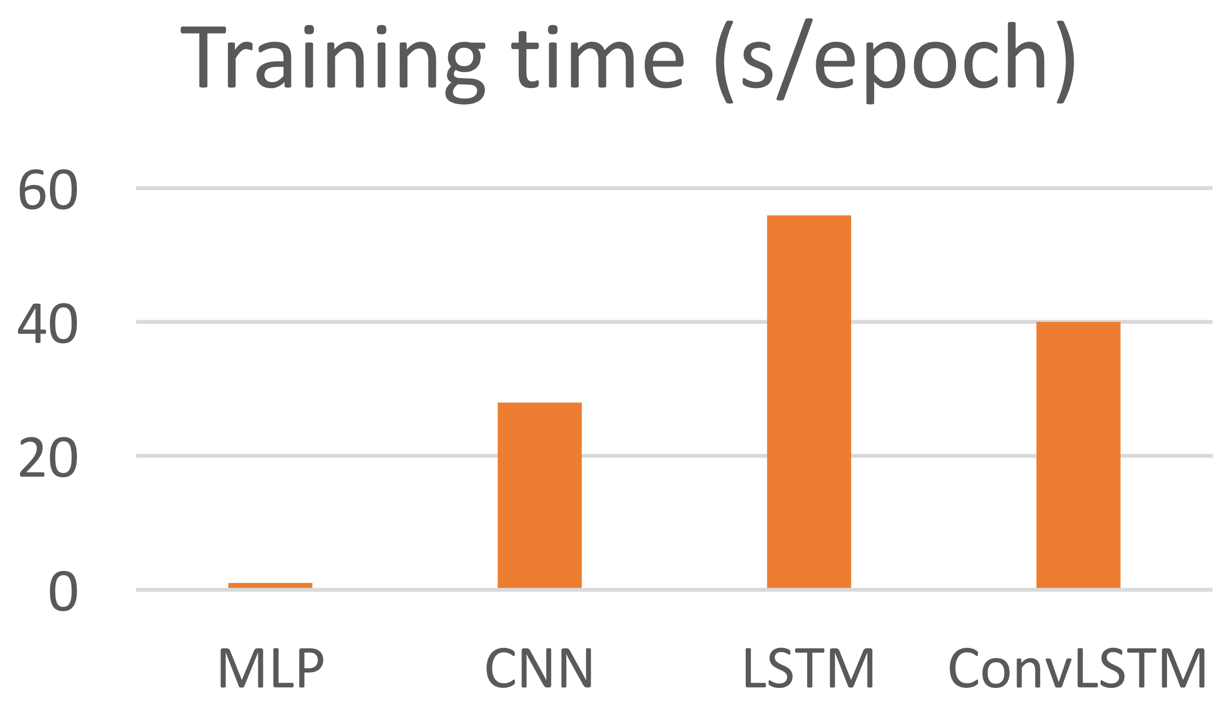
Bi-directional LSTM network with two parallel tracks in both future direction (green) and to the past (red)

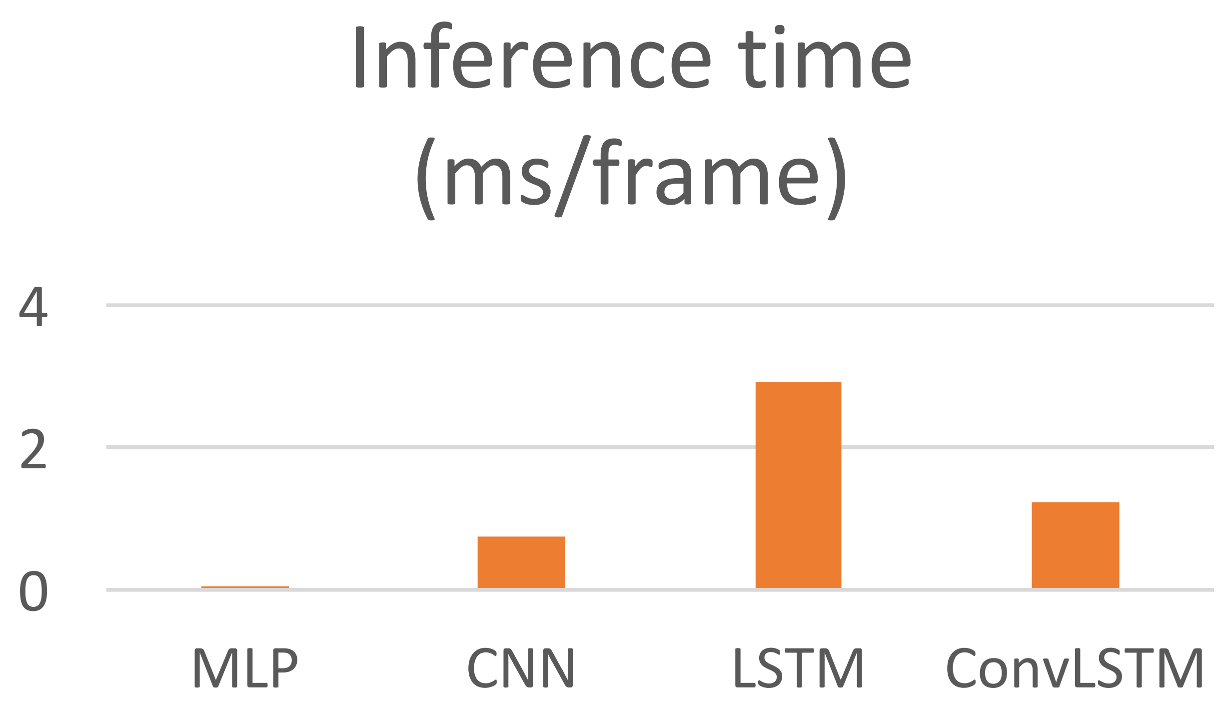
Hammerla, Nils Y., Shane Halloran, and Thomas Plötz. "Deep, convolutional, and recurrent models for human activity recognition using wearables." arXiv preprint arXiv:1604.08880 (2016).

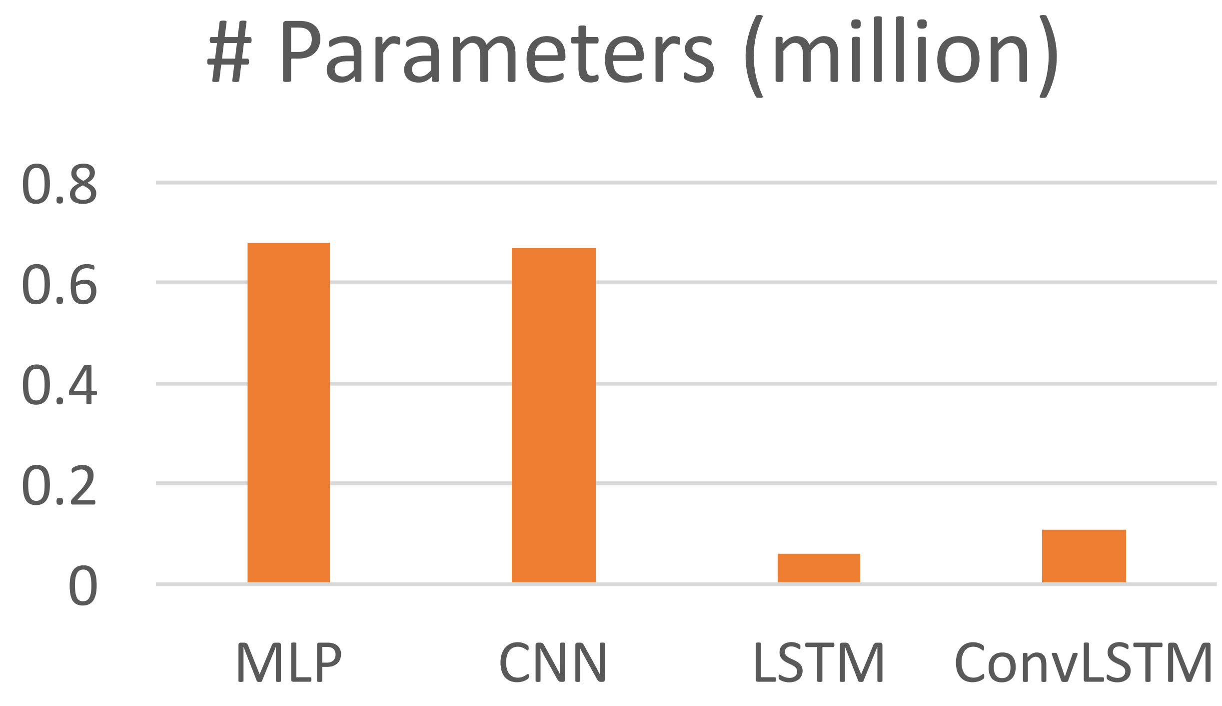
Performance Comparison of DNN Alternatives on HAR

Performance after 3 epochs using CPU …

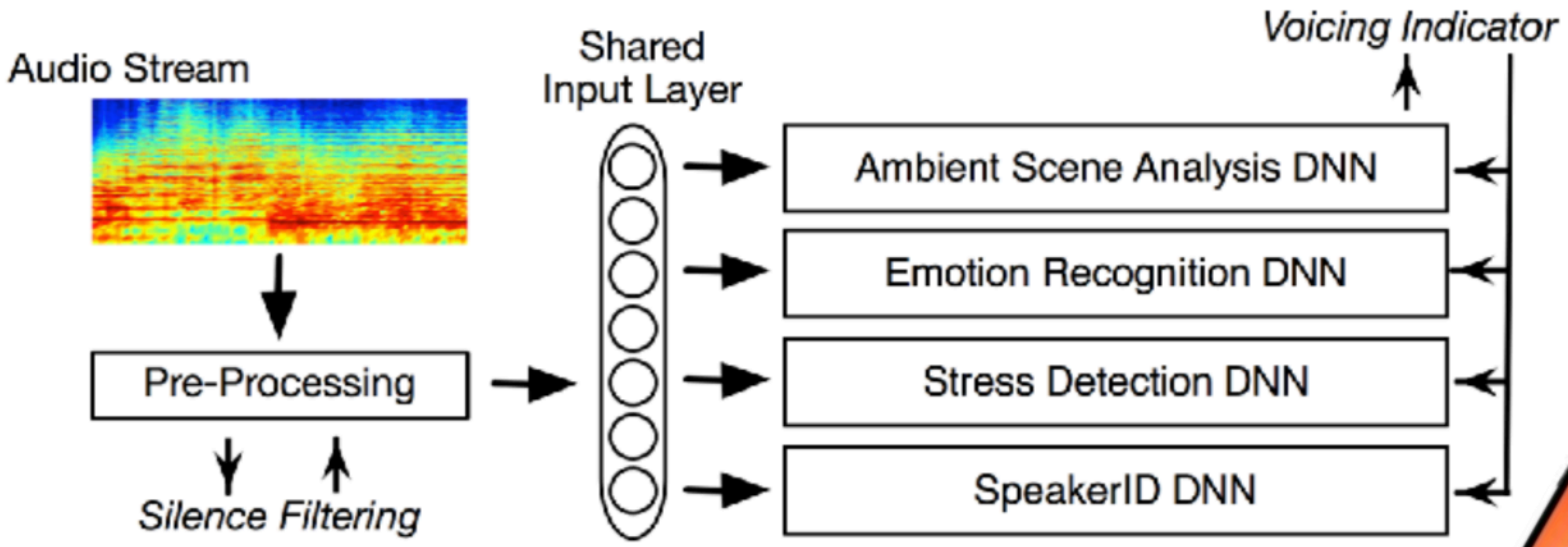
	MLP	**CNN**	**LSTM**	**ConvLSTM**
Accuracy	0.78	0.85	0.78	**0.88**
Mean-f1	0.71	0.84	0.71	**0.87**
Structure	FC128, FC128	C16,P,C16,P, FC32	L64, L64	C16,P,C16,P, L32







Deep Learning Based Audio Analysis: *DeepEar*



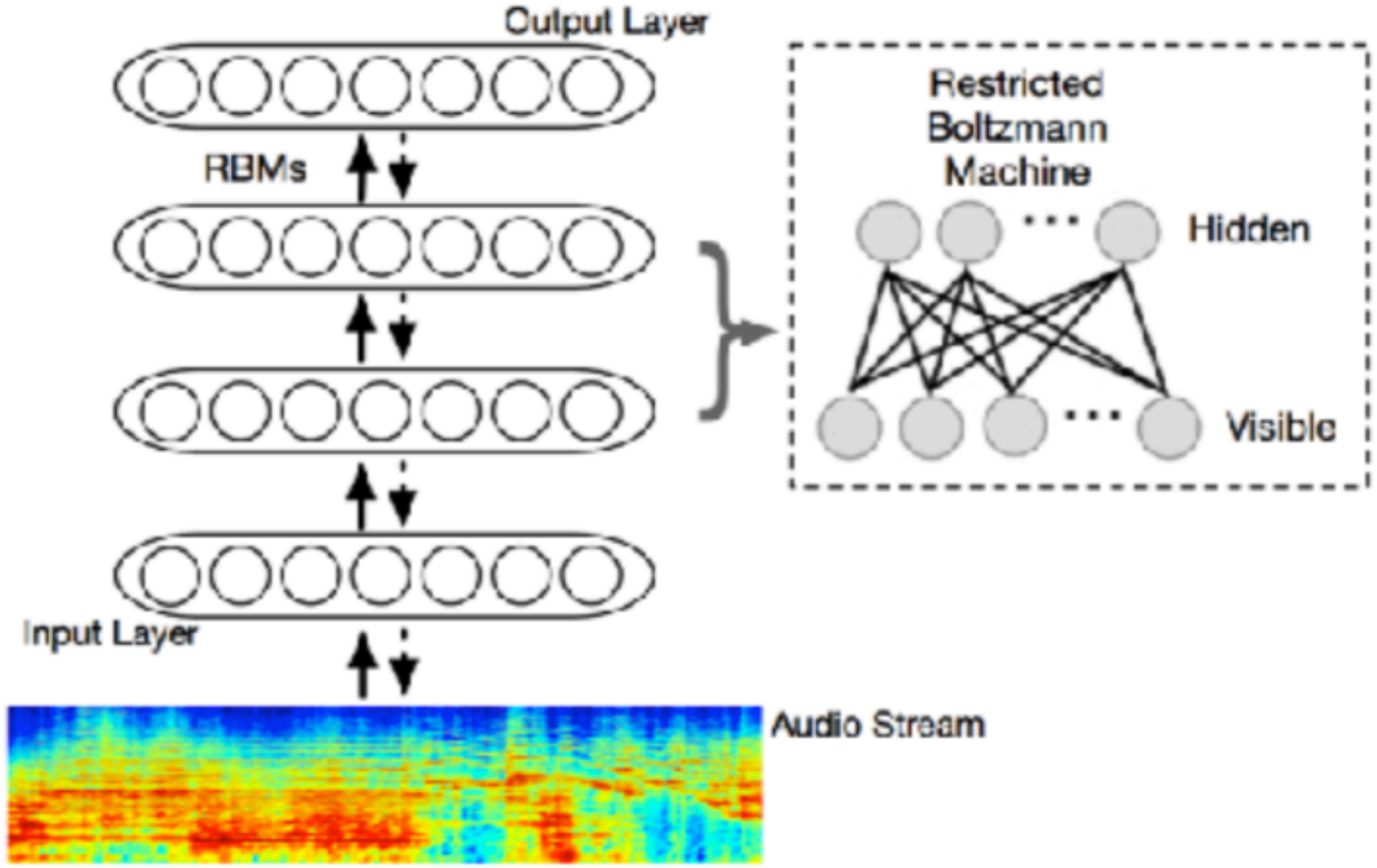
Lane, Nicholas D., Petko Georgiev, and Lorena Qendro. "Deepear: robust smartphone audio sensing in unconstrained acoustic environments using deep learning." In Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing, pp. 283-294. 2015.

DeepEar's Building Block: *Restricted Boltzmann Machines* (RBM)

- RBMs are DNNs with feedback
  - ~ Markov Random Fields

- Components
  - Rectified Linear Units (ReLU)
  - Input layer: Gaussian visible unit
  - Input data: Perceptual Linear Prediction (PLP) features
  - Output: Softmax

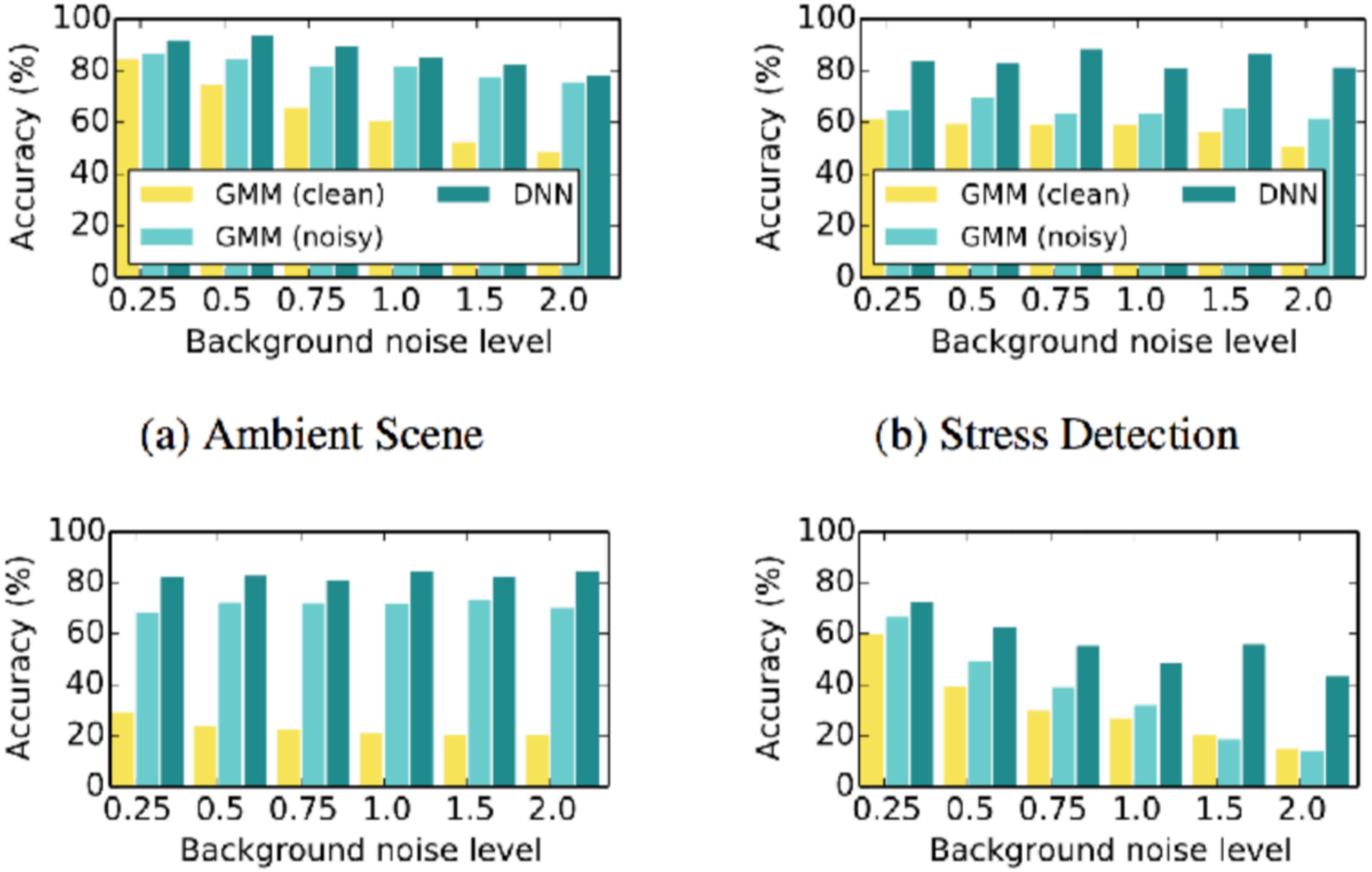
- Unsupervised Pre-Training + Supervised Fine-Tuning



Total Layers	Hidden Layers	Units per Hidden Layer	Total Parameters
5	3	1024	2.3M

Lane, Nicholas D., Petko Georgiev, and Lorena Qendro. "Deepear: robust smartphone audio sensing in unconstrained acoustic environments using deep learning." In Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing, pp. 283-294. 2015.

DeepEar’s Performance



(a) Ambient Scene

(b) Stress Detection

(c) Emotion Recognition

(d) Speaker Identification

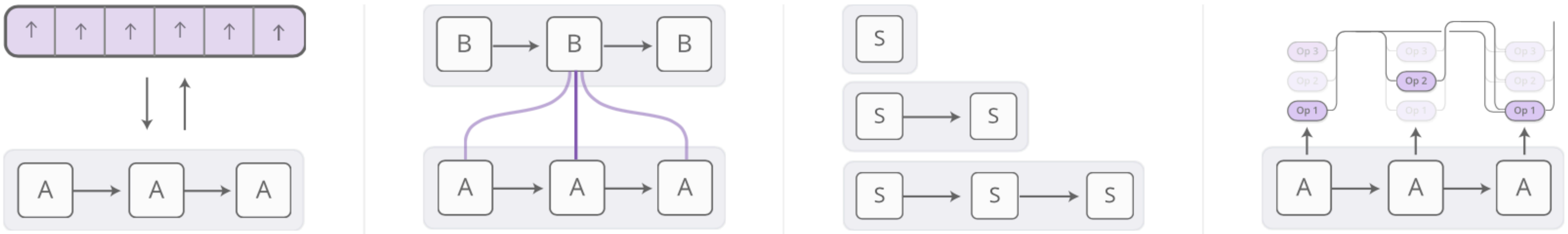
Lane, Nicholas D., Petko Georgiev, and Lorena Qendro. "Deepear: robust smartphone audio sensing in unconstrained acoustic environments using deep learning." In Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing, pp. 283-294. 2015.

Approach 4: RNNs Augmented with Attention

Recap what we have learnt about RNN

- One of the staples of deep learning that allow working with sequences of data
  - text, audio, video, sensors,…
- They can be used to
  - boil a sequence down into a high-level understanding
  - annotate sequences
  - even generate new sequences from scratch
- Basic RNN design struggles with longer sequences, but LSTM can work with these
  - remarkable results in translation, voice recognition, image captioning, HAR, …
- A growing number of attempts to augment RNNs with new properties

Four Directions for Augmenting RNNs



Neural Turing Machines have external memory that they can read and write to.

Attentional Interfaces allow RNNs to focus on parts of their input.

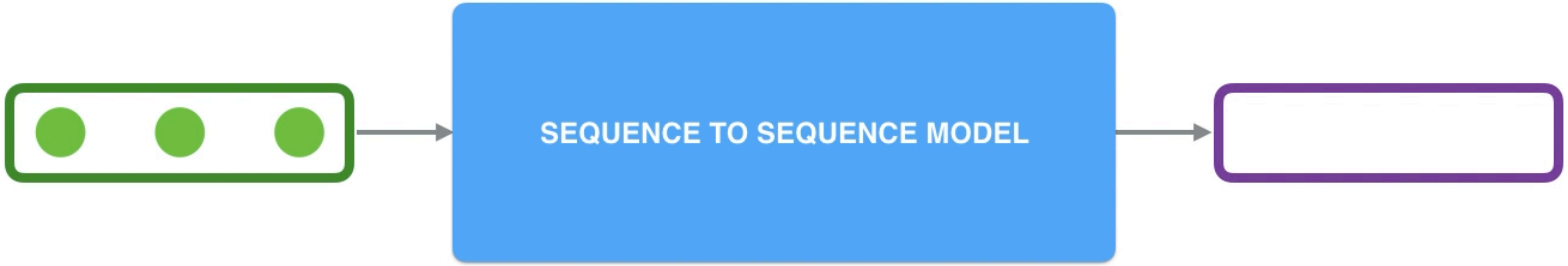
Adaptive Computation Time allows for varying amounts of computation per step.

Neural Programmers can call functions, building programs as they run.

- Individually, these techniques are all potent extensions of RNNs
- Moreover, they can be combined
  - seem to just be points in a broader space
- Rhey all rely on the same underlying trick—something called **attention**—to work.

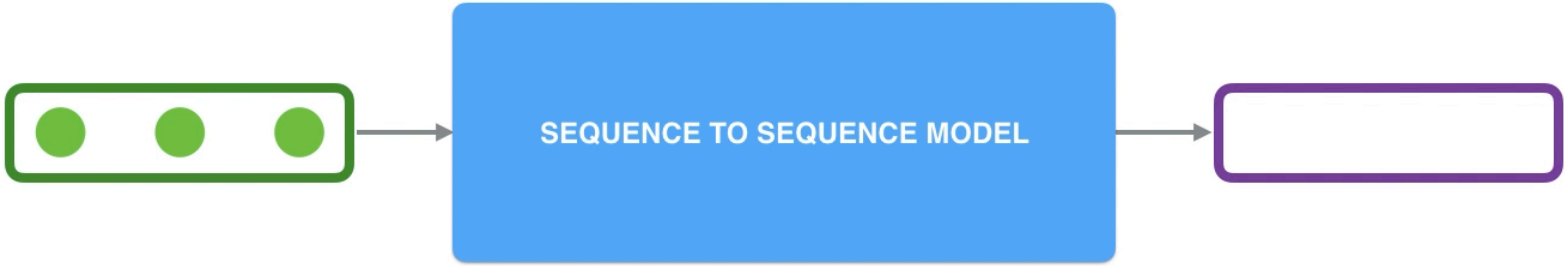
https://distill.pub/2016/augmented-rnns/

Sequence-to-Sequence Model



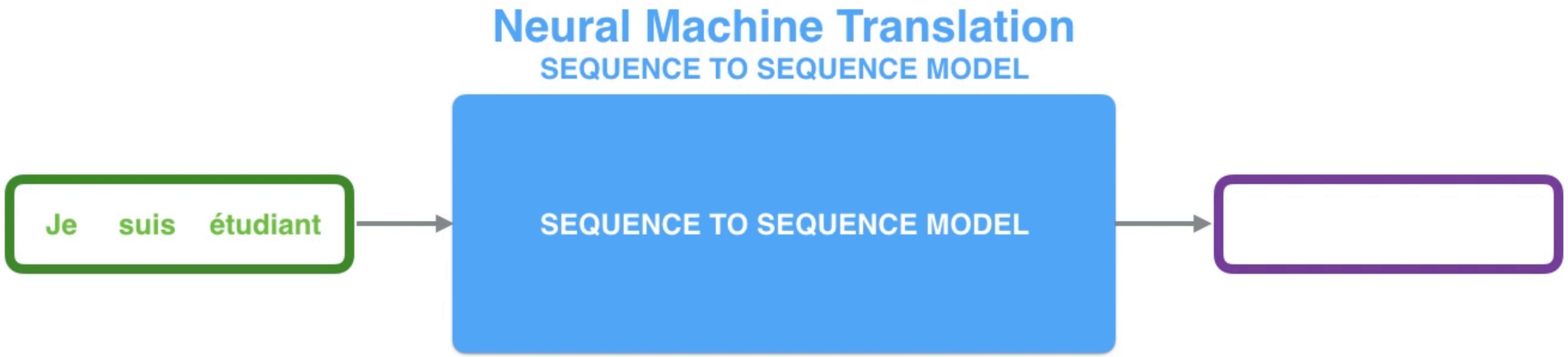
https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

Sequence-to-Sequence Model



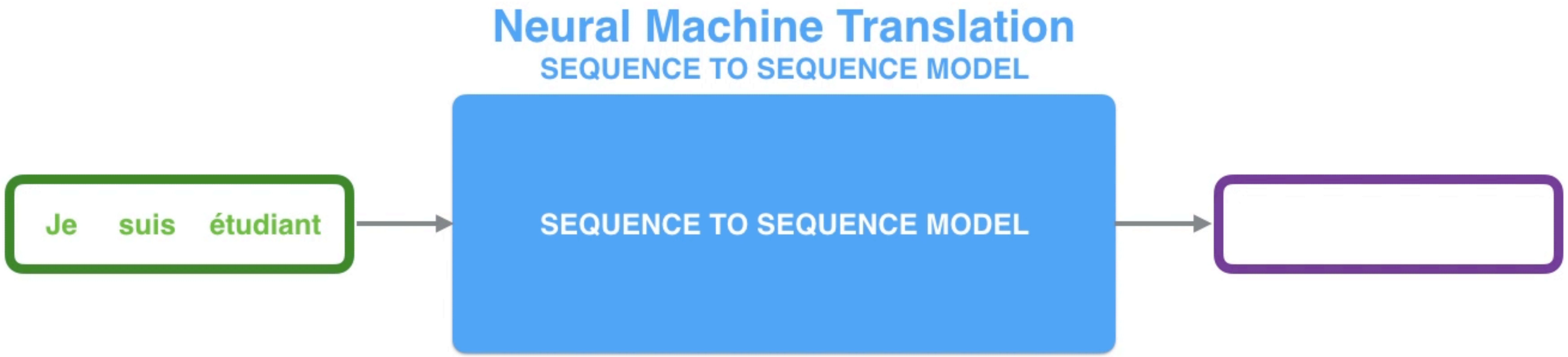
https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

E.g. Neural Machine Translation



https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

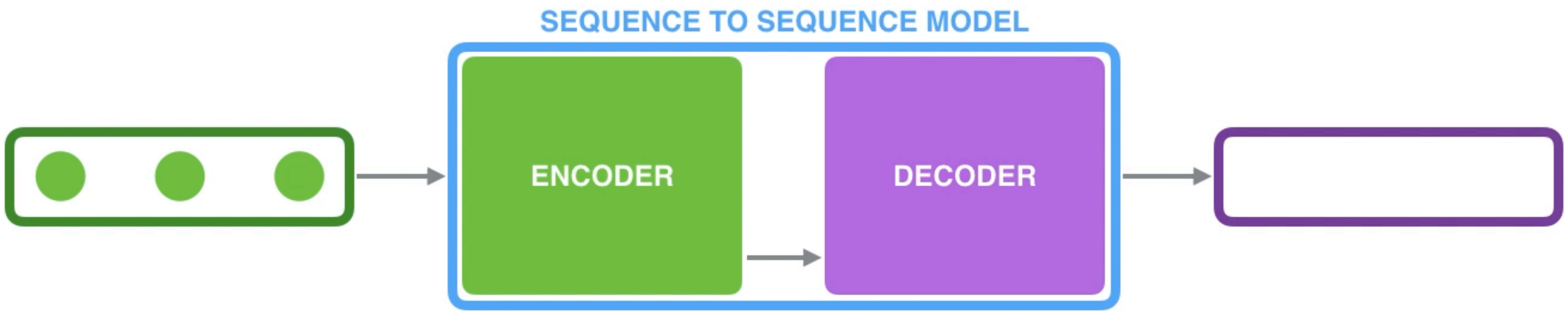
E.g. Neural Machine Translation



https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

Looking under the hood

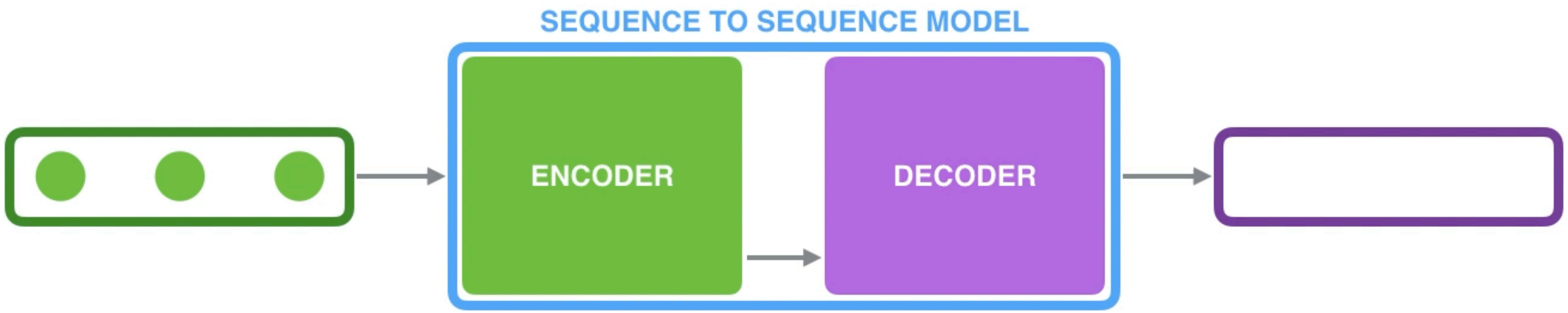
- The model is composed of an encoder and a decoder
- The encoder processes each item in the input sequence, it compiles the information it captures into a vector (called the context)
- After processing the entire input sequence, the encoder sends the context over to the decoder, which begins producing the output sequence item by item



https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

Looking under the hood

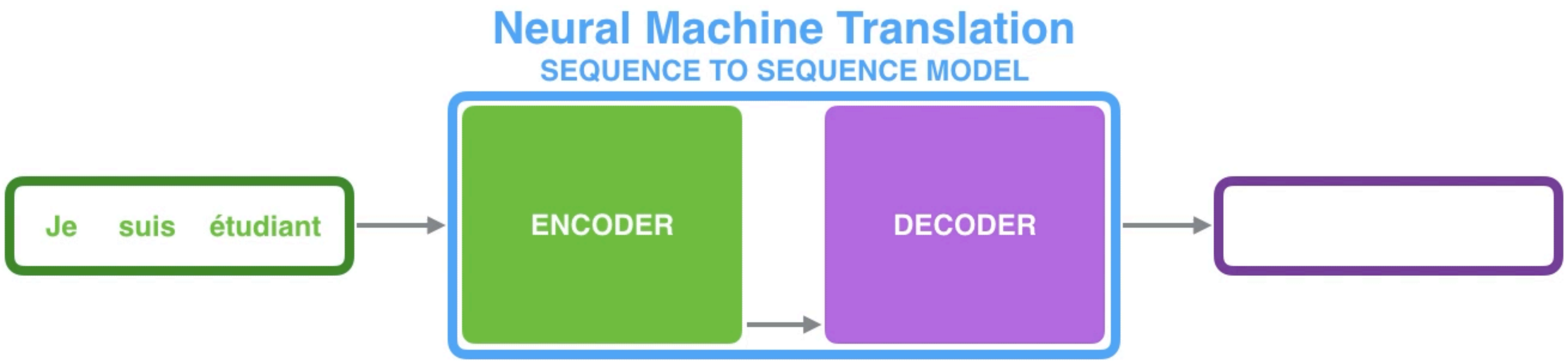
- The model is composed of an encoder and a decoder
- The encoder processes each item in the input sequence, it compiles the information it captures into a vector (called the context)
- After processing the entire input sequence, the encoder sends the context over to the decoder, which begins producing the output sequence item by item



https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

Looking under the hood

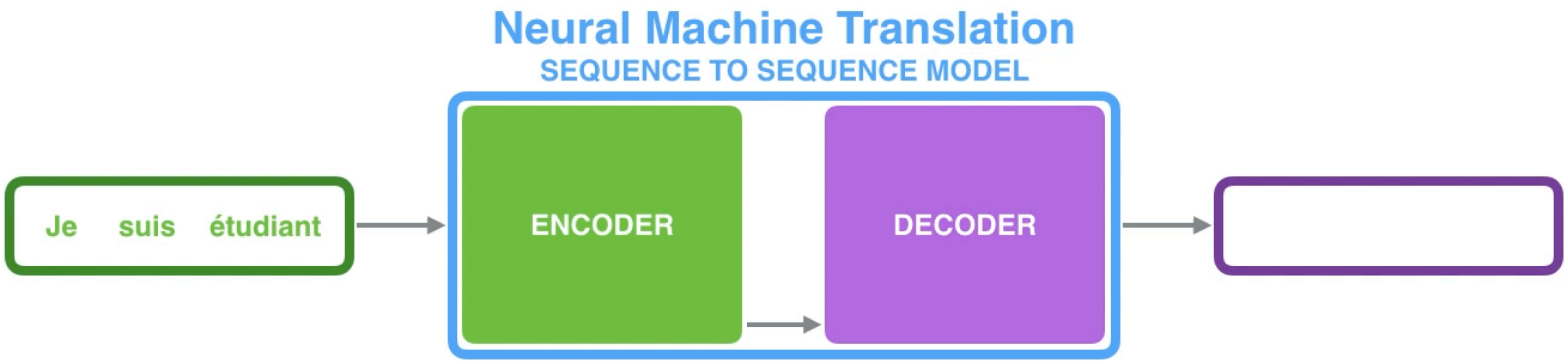
- The model is composed of an encoder and a decoder
- The encoder processes each item in the input sequence, it compiles the information it captures into a vector (called the context)
- After processing the entire input sequence, the encoder sends the context over to the decoder, which begins producing the output sequence item by item
- The same applies in the case of machine translation



https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

Looking under the hood

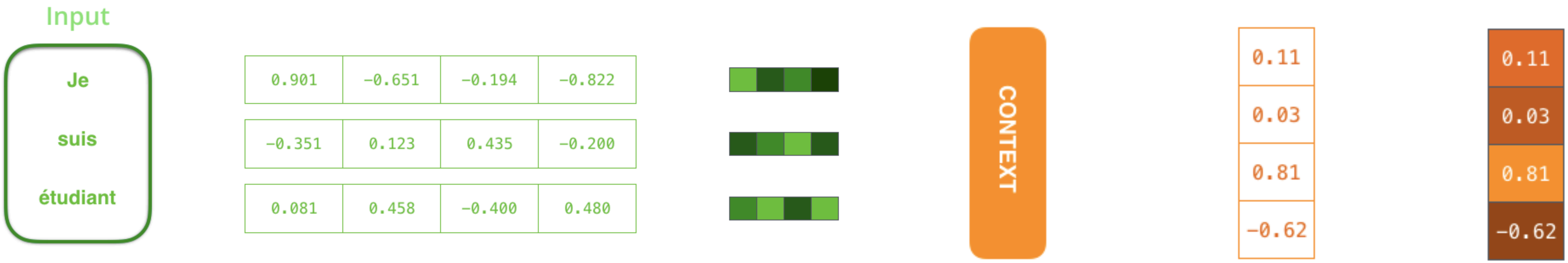
- The model is composed of an encoder and a decoder
- The encoder processes each item in the input sequence, it compiles the information it captures into a vector (called the context)
- After processing the entire input sequence, the encoder sends the context over to the decoder, which begins producing the output sequence item by item
- The same applies in the case of machine translation



https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

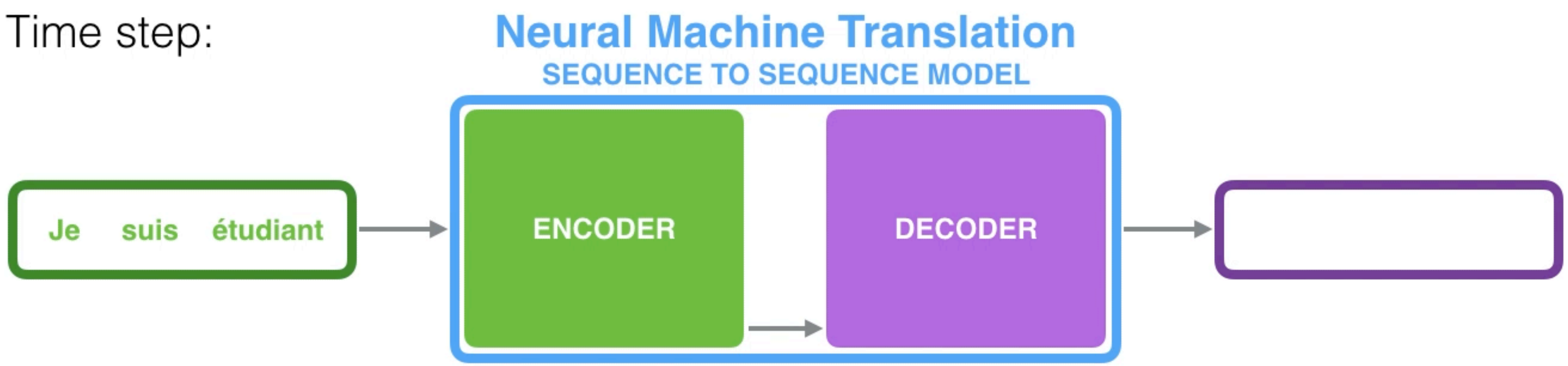
Looking under the hood

- The encoder and the decoder are RNNs
  - Raw inputs (e.g. words, sensor measurements) algorithmically mapped to vectors called embeddings (“x2vec”)
  - Vector spaces that capture a lot of the meaning/semantic information
- The context is a vector whose size is a hyperparameter of the model
  - It is basically the number of hidden units in the encoder RNN
  - in real world applications the context vector may be of a size like 256, 512, or 1024.

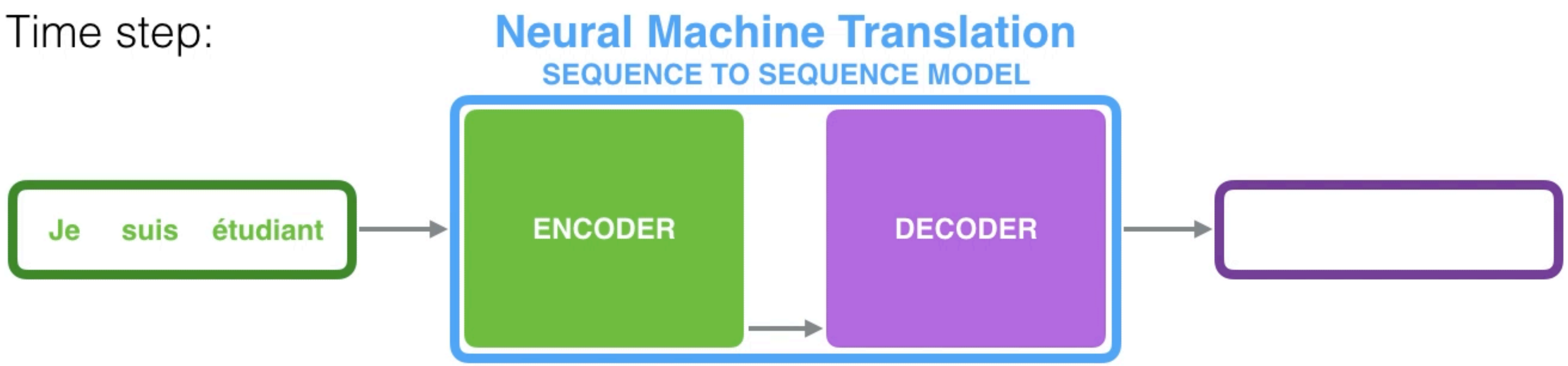


https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

Looking under the hood

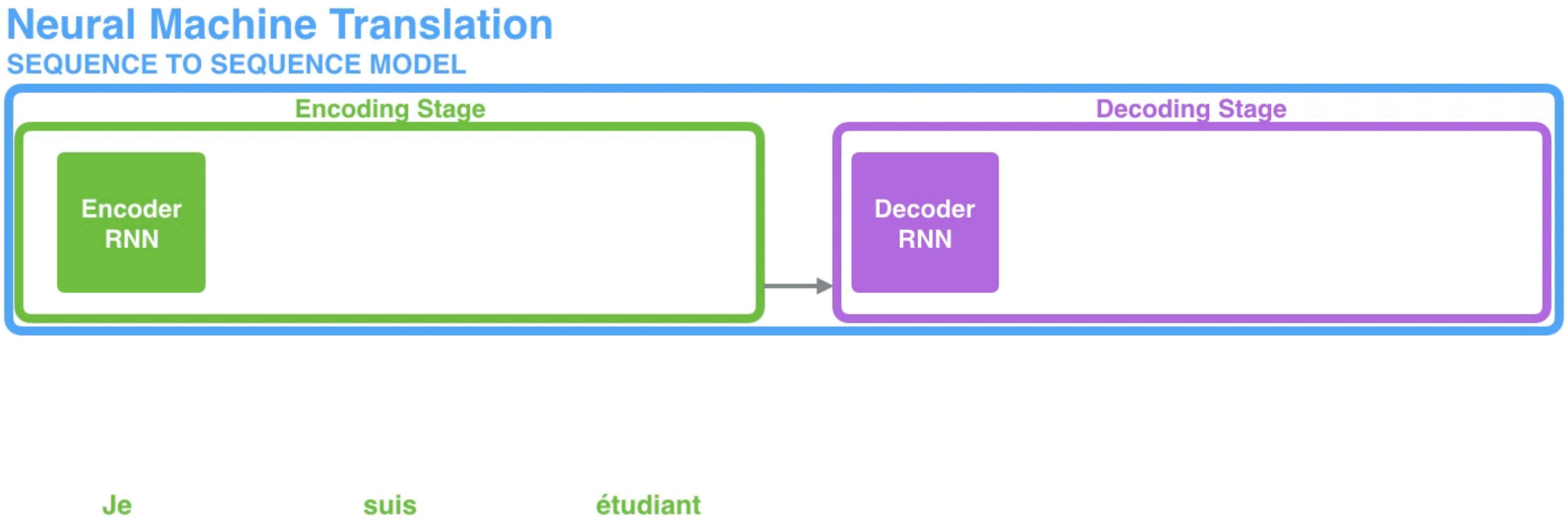


Looking under the hood



Looking under the hood

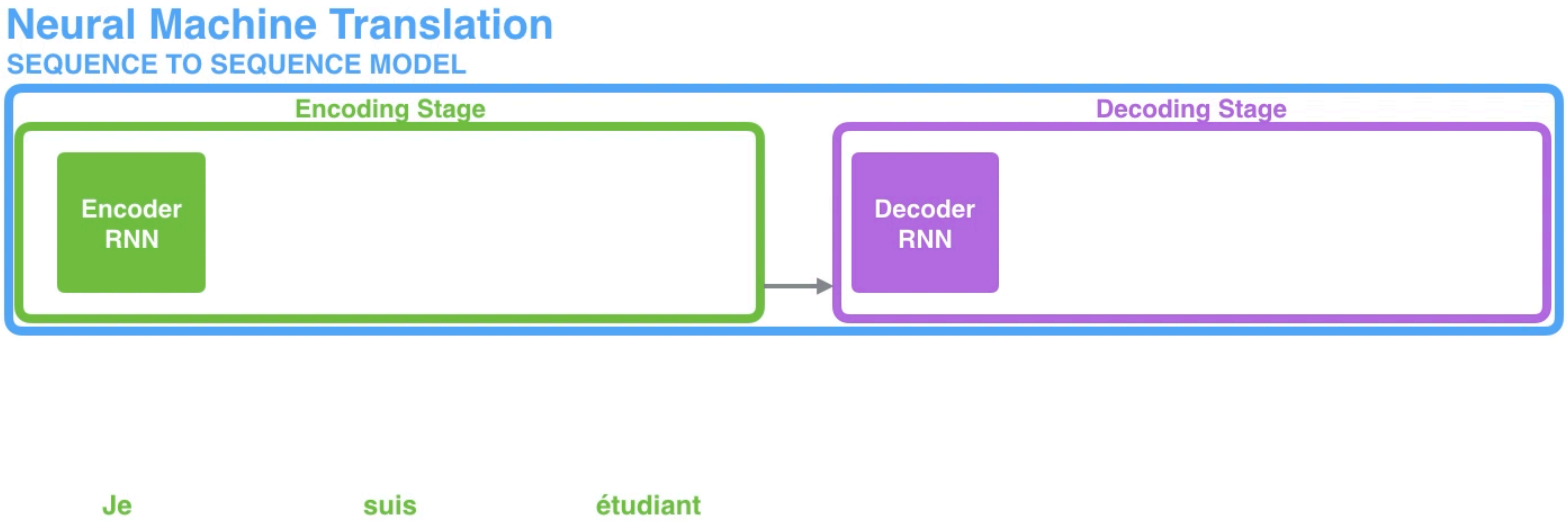
Unrolled View



https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

Looking under the hood

Unrolled View



https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

The Attention Mechanism

- The context vector becomes a bottleneck, making it hard to deal with long sequences
  - Information about the earlier part of the sequence fades
  - Which parts of the sequence are important changes

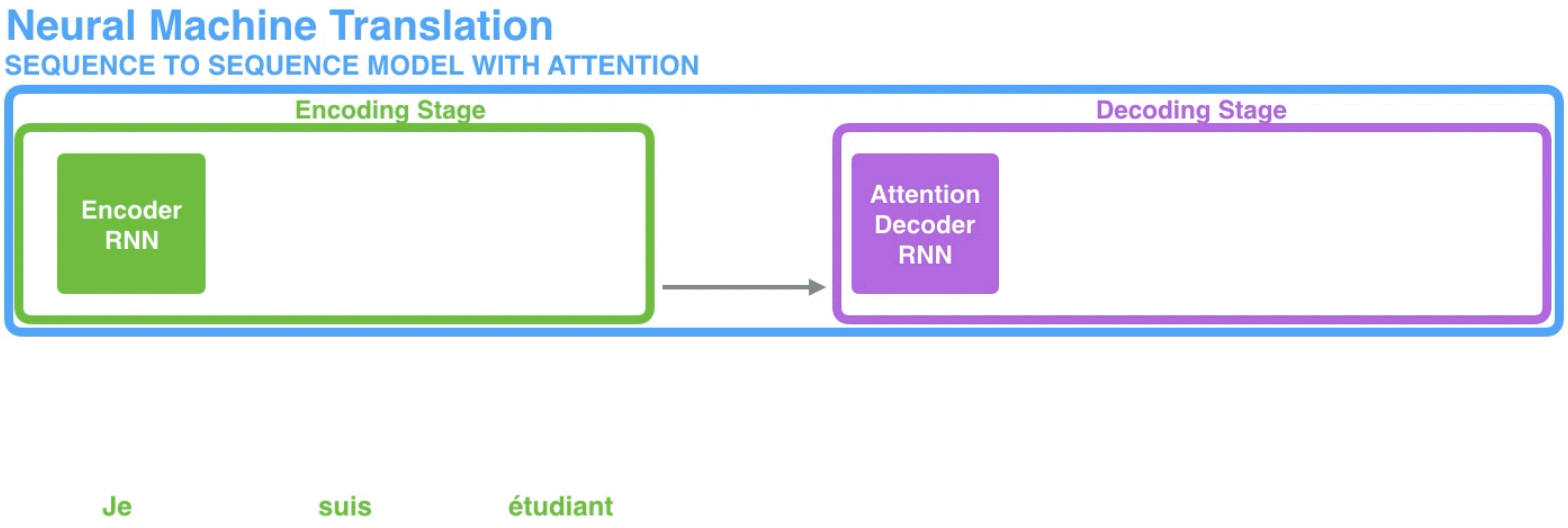
- Attention introduced to allow a model to focus on the relevant parts of the input sequence as needed
  - introduced for machine translation but has broader applicability

- An attention model differs from a classic sequence-to-sequence model in two main ways
  1. The encoder passes a lot more data to the decoder
  2. An attention decoder does an extra step before producing its output

https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

1. Encoder passes a lot more data to the decoder

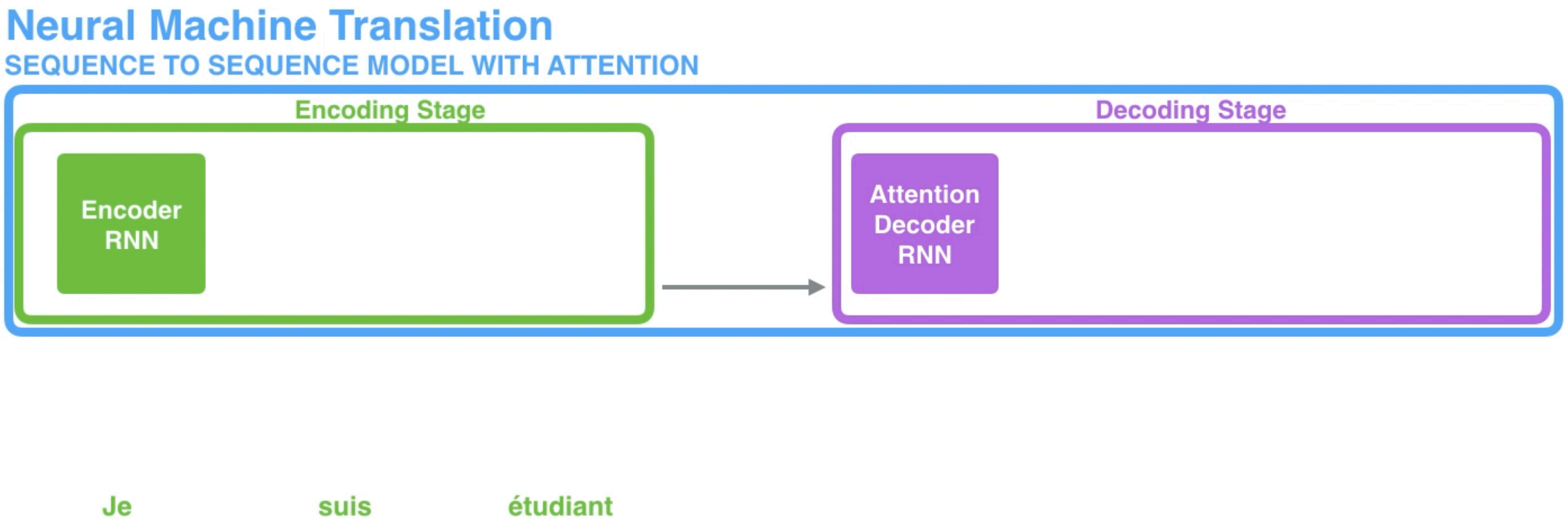
- Instead of passing the last hidden state of the encoding stage, the encoder passes all the hidden states to the decoder



https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

1. Encoder passes a lot more data to the decoder

- Instead of passing the last hidden state of the encoding stage, the encoder passes all the hidden states to the decoder



https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

2. Decoder does an extra step

To focus on the parts of the input that are relevant to a decoding time step, the decoder does the following at each time step:

1. Look at the set of encoder hidden states it received – each encoder hidden state is most associated with a certain word in the input sentence

2. Give each hidden state a score

3. Multiply each hidden state by its softmaxed score, thus amplifying hidden states with high scores, and drowning out hidden states with low scores

Attention at time step 4

https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

2. Decoder does an extra step

To focus on the parts of the input that are relevant to a decoding time step, the decoder does the following at each time step:

1. Look at the set of encoder hidden states it received – each encoder hidden state is most associated with a certain word in the input sentence

2. Give each hidden state a score

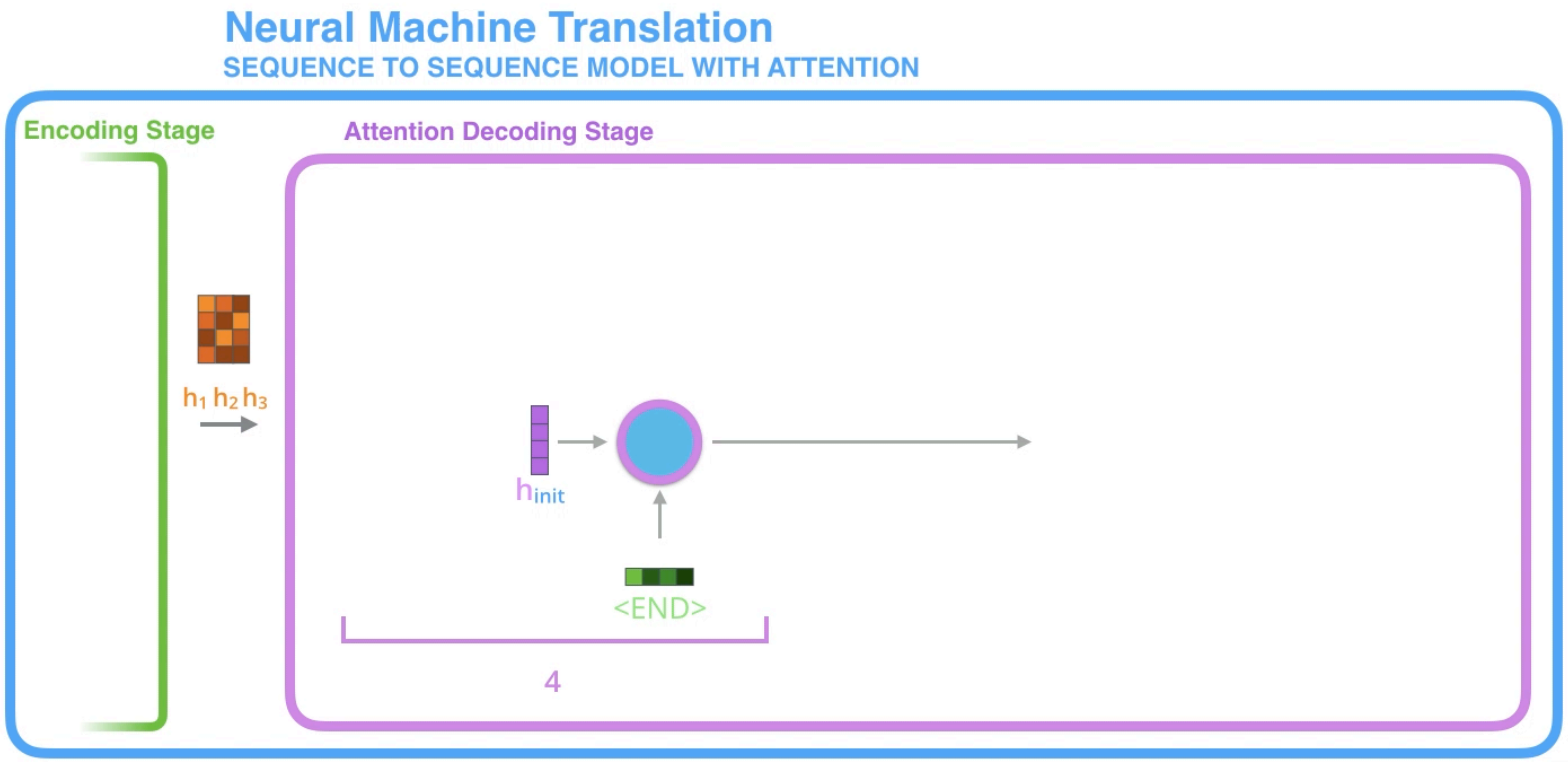
3. Multiply each hidden state by its softmaxed score, thus amplifying hidden states with high scores, and drowning out hidden states with low scores

Attention at time step 4

https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

Putting it all together

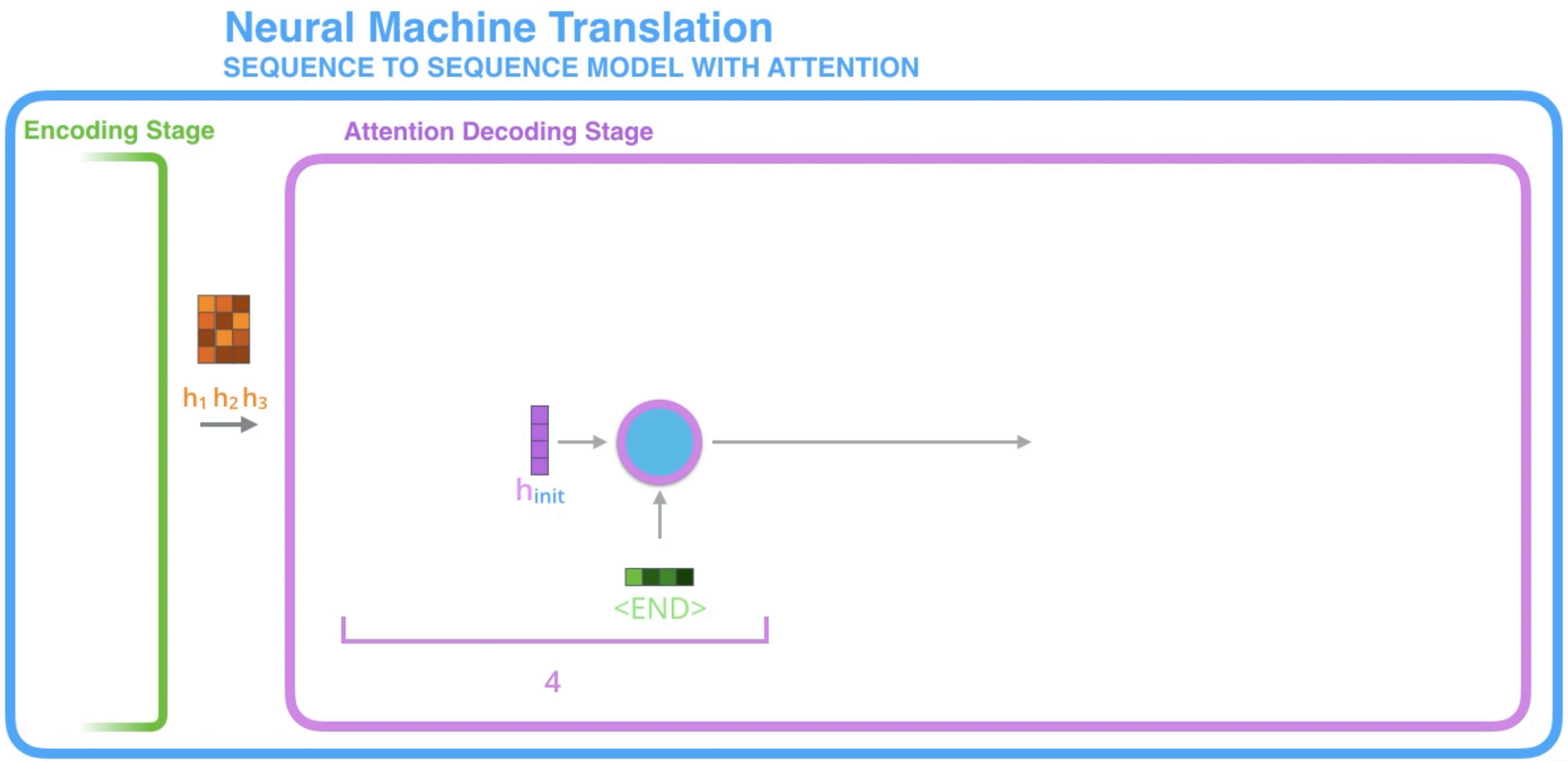
1. The attention decoder RNN takes in the embedding of the <END> token, and an initial decoder hidden state.
2. The RNN processes its inputs, producing an output and a new hidden state vector (h4)
   - The output is discarded.
3. **Attention Step:** We use the encoder hidden states and the h4 vector to calculate a context vector (C4) for this time step.
4. We concatenate h4 and C4 into one vector.
5. We pass this vector through a feedforward neural network
   - one trained jointly with the model
6. The output of the feedforward neural networks indicates the output word of this time step.
7. Repeat for the next time steps



https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

Putting it all together

1. The attention decoder RNN takes in the embedding of the <END> token, and an initial decoder hidden state.
2. The RNN processes its inputs, producing an output and a new hidden state vector (h4)
   - The output is discarded.
3. **Attention Step:** We use the encoder hidden states and the h4 vector to calculate a context vector (C4) for this time step.
4. We concatenate h4 and C4 into one vector.
5. We pass this vector through a feedforward neural network
   - one trained jointly with the model
6. The output of the feedforward neural networks indicates the output word of this time step.
7. Repeat for the next time steps



https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

Approach 5: Transformers

Limitations of RNNs and LSTMs

- Firstly, RNNs are slow, in fact, extremely slow to train
  - often we have to truncate the training using techniques like Truncated Back Propagation In Time

- Secondly and more commonly, RNNs suffer from a problem of vanishing and exploding gradients.
  - the information from the beginning of the sequence gets lost

- LSTMs solves the second problem, but are even slower to train

- Sequential processing does not allow taking advantage of parallel processing in GPUs

- Attention solves some of the flaws of RNNs and LSTMs
  - sequential processing remains :-(

Attention is All You Need

Attention is **all you need**

[PDF] arxiv.org

A Vaswani, N Shazeer, N Parmar, J Uszkoreit… - arXiv preprint arXiv …, 2017 - arxiv.org

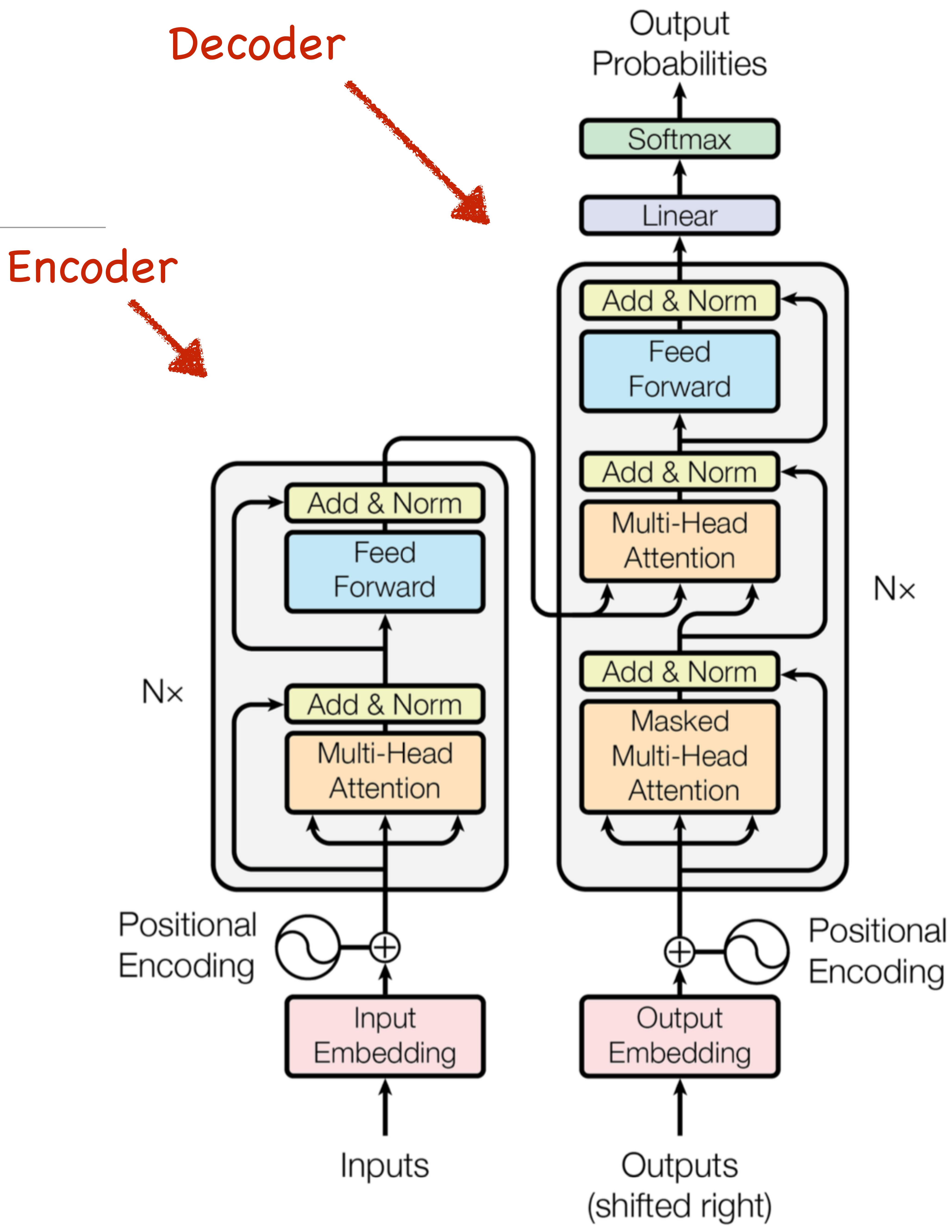
The dominant sequence transduction models are based on complex recurrent or convolutional neural networks in an encoder-decoder configuration. The best performing models also connect the encoder and decoder through an attention mechanism. We …

Cited by 17156 Related articles All 22 versions

- Famous paper that introduced "Transformer" architecture
- Transformer is an architecture for transforming one sequence into another one with the help of two parts (Encoder and Decoder)
- But it does so without any any Recurrent Networks (GRU, LSTM, etc.)
  - an architecture with only attention-mechanism
- The input for the encoder is the whole sequence
- The inputs for the decoder are also the entire sequence (shifted right)

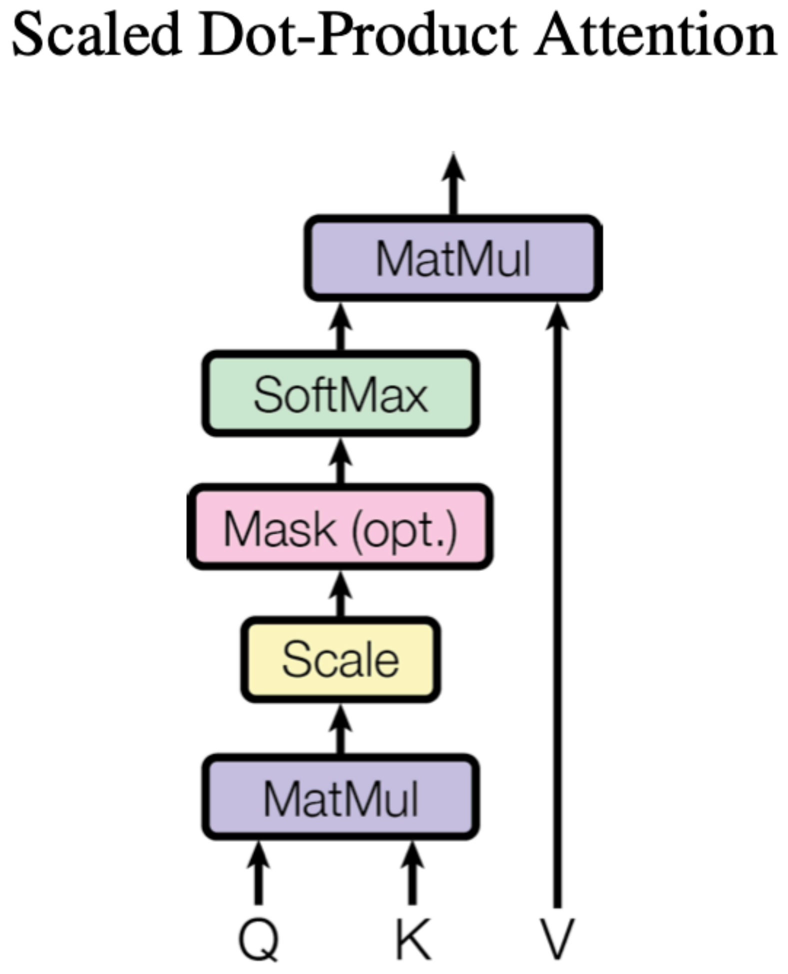
Transformer Model Architecture

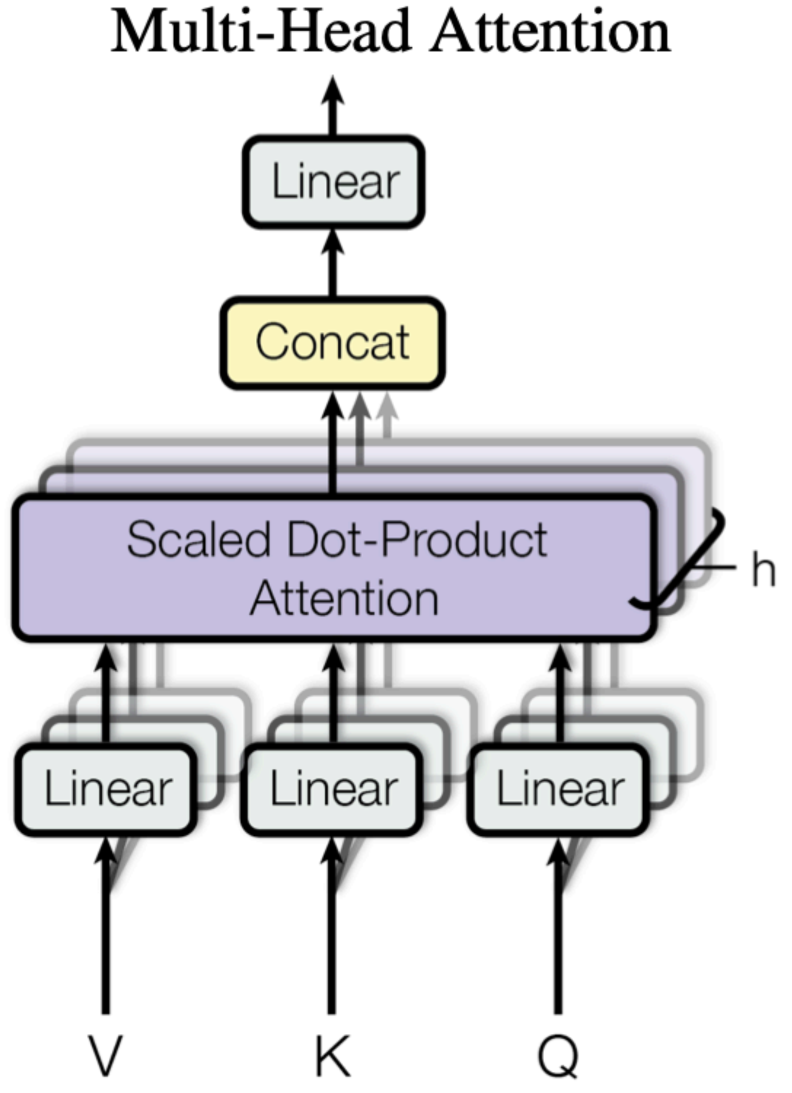
- Encoder and Decoder
  - Composed of modules that can be stacked on top of each other multiple times (Nx in the figure, N=6 in paper)
  - Modules consist mainly of Multi-Head Attention and Feed Forward layers
- inputs and outputs are first embedded into an n-dimensional space
- Important aspect: **positional encoding** of the different symbols in the sequence
  - Gives each symbol in the sequence a relative position
    - since a sequence depends on the order of its elements
  - These positions are added to the embedded representation (n-dimensional vector) of each symbol



https://arxiv.org/pdf/1706.03762.pdf

The Attention Function





- Scaled Dot-Product Attention
  - Operates on queries and keys of dimension d_k and values of dimension d_v
  - Simultaneously on a set of queries packed into a matrix Q
  - The keys and values are also packed together into matrices K and V|

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

- Multi-Head Attention
  - Instead of performing a single attention function with d_{model}-dimensional keys, values and queries, linearly project the queries, keys and values h times with different, learned linear projections to d_k, d_k, and d_v dimensions respectively
  - Attention function performed on each of them in parallel, yielding d_v-dimensional output values, which are concatenated and once again projected to final values
  - Intuition: multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions, while with a single attention head, averaging inhibits this

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \text{head}_2, \ldots, \text{head}_h)W^O$$
$$\text{where head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

https://arxiv.org/pdf/1706.03762.pdf

Positional Encoding

- There is no recurrence and no convolution, and so there is no sense of order of symbols in the sequence
  - One must inject some information about the relative or absolute position of the symbols in the sequence

- Approach: add "positional encodings" to the input embeddings at the bottoms of the encoder and decoder stacks
  - The positional encodings have the same dimension d_{model} as the input embeddings, so that the two can be summed
  - Original Transformer paper uses sine and cosine functions of different frequencies

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}})$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}})$$

where pos is the position and i is the dimension

Note: for any fixed offset k, PE_{pos+k} can be represented as a linear function of PE_{pos}

https://arxiv.org/pdf/1706.03762.pdf

Other Details

- Position-wise feedforward networks (executed in parallel)
  - each of the layers in our encoder and decoder contains a fully connected feed-forward network, which is applied to each position separately and identically

    $$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

  - the linear transformations are the same across different positions, but use different parameters from layer to layer
    - Another way of describing this is as two convolutions with kernel size 1

- Embeddings and Softmax
  - learned embeddings convert the input symbols and output symbols to vectors of dimension d_{model}
  - learned linear transformation and softmax function to convert the decoder output to predicted next-symbol probabilities

- Forcing learning how to predict a symbol at position i given previous symbols at positions $< i$
  - decoder input sequence is shifted right by one (with a special start symbol put in place)
  - applies a mask to the input in the first multi-head attention module to avoid seeing potential 'future' sequence

https://arxiv.org/pdf/1706.03762.pdf

Approach 6: Deep Autoregressive Models

Classical Statistical Model for Time Series: Autoregression

- An autoregressive (AR) model predicts future behavior based on past behavior
  - Useful for forecasting, anomaly detection

- Output variable depends linearly on its own previous values and on a stochastic term
  - $AR(p)$ model (AR model of order p)

$$y_t = c + \sum_{i=1}^{p} w_i y_{t-1} + e_t$$

  where $w_1, w_2, \ldots, w_p$ are parameters of the model, c is a constant, and e_t is white noise
  - Can be viewed as the output of an all-pole infinite impulse response filter whose input is white noise
  - Parameters conteained for necessary for the model to remain wide-sense stationary

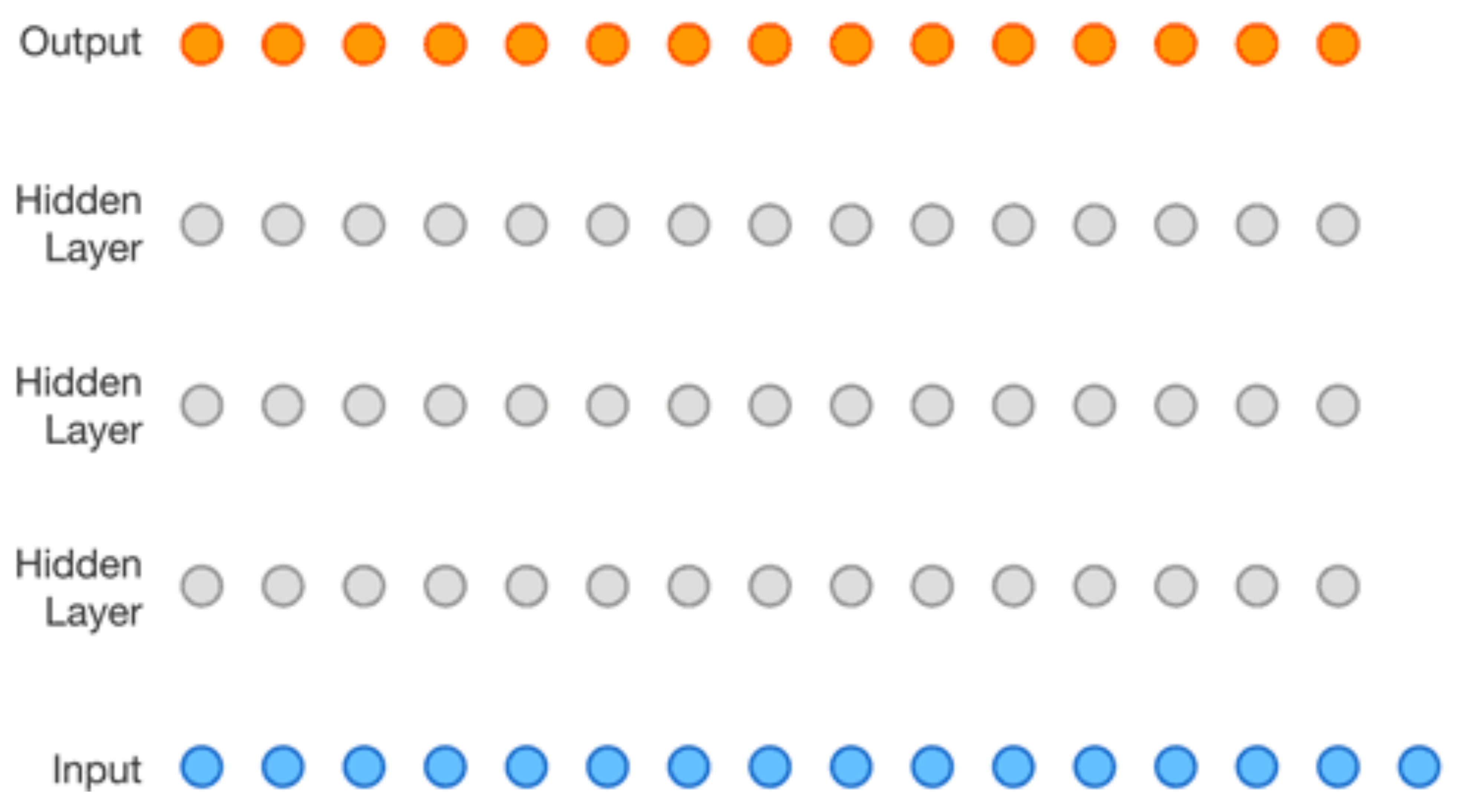
- Advantages: interpretable, analysis tools
- For large p the traditional approach can become impractically slow to train
  - but large p needed for monitoring high-rate data

A Tale of Two Sequence Models

- RNNs are sequence-to-sequence models
  - Expressive model without requiring elaborate features
    - scale well to applications with rich data
  - But, they can be overly complex for typical time series data
    - resulting in a lack of interpretability

- Another model in the class: *Autoregressive Neural Networks*
  - Like an RNN, an autoregressive model's output at time depends on not just on input at the time but also from previous time steps
  - However, unlike an RNN, the inputs from previous timesteps are not provided via some hidden state: they are given as just another input to the model

- Characteristics
  - Bridge statistical and deep learning-based approaches
  - Semi-explicitly incorporate time series dynamics, such as autoregression, trend shifts, and seasonality
  - Scalable, extensible, and interpretable



An autoregressive model is merely a feed-forward model which predicts future values from past values.

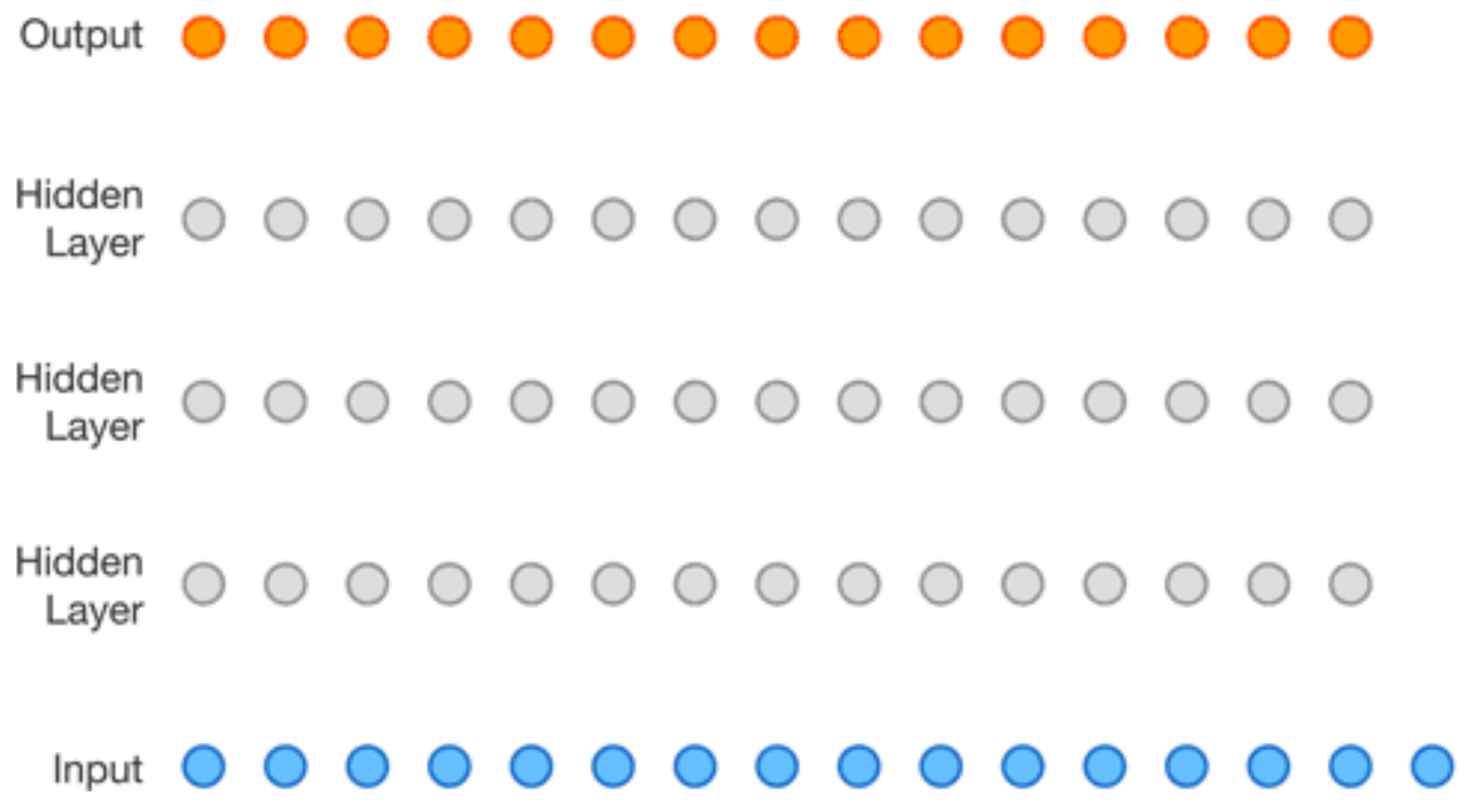
https://eigenfoo.xyz/deep-autoregressive-models/

A Tale of Two Sequence Models

- RNNs are sequence-to-sequence models
  - Expressive model without requiring elaborate features
    - scale well to applications with rich data
  - But, they can be overly complex for typical time series data
    - resulting in a lack of interpretability

- Another model in the class: *Autoregressive Neural Networks*
  - Like an RNN, an autoregressive model's output at time depends on not just on input at the time but also from previous time steps
  - However, unlike an RNN, the inputs from previous timesteps are not provided via some hidden state: they are given as just another input to the model

- Characteristics
  - Bridge statistical and deep learning-based approaches
  - Semi-explicitly incorporate time series dynamics, such as autoregression, trend shifts, and seasonality
  - Scalable, extensible, and interpretable



An autoregressive model is merely a feed-forward model which predicts future values from past values.

https://eigenfoo.xyz/deep-autoregressive-models/

The Bargain with Autoregressive Neural Networks

Parallelization

Trainability

Inference Speed

- **Pros:** One can have stable, parallel and easy-to-optimize training, faster inference computations (with some caveats), and completely do away with the fickleness of truncated backpropagation through time
- **Cons:** Provided you are willing to accept a model that (by design) cannot have infinite memory. Also, in some cases such as long sequences autoregressive inference can be a bottleneck and require clever engineering.
- Very much like FIR (Finite Impulse Response) vs IIR (Infinite Impulse Response) filters

Feed-Forward Models Can Outperform Recurrent Models

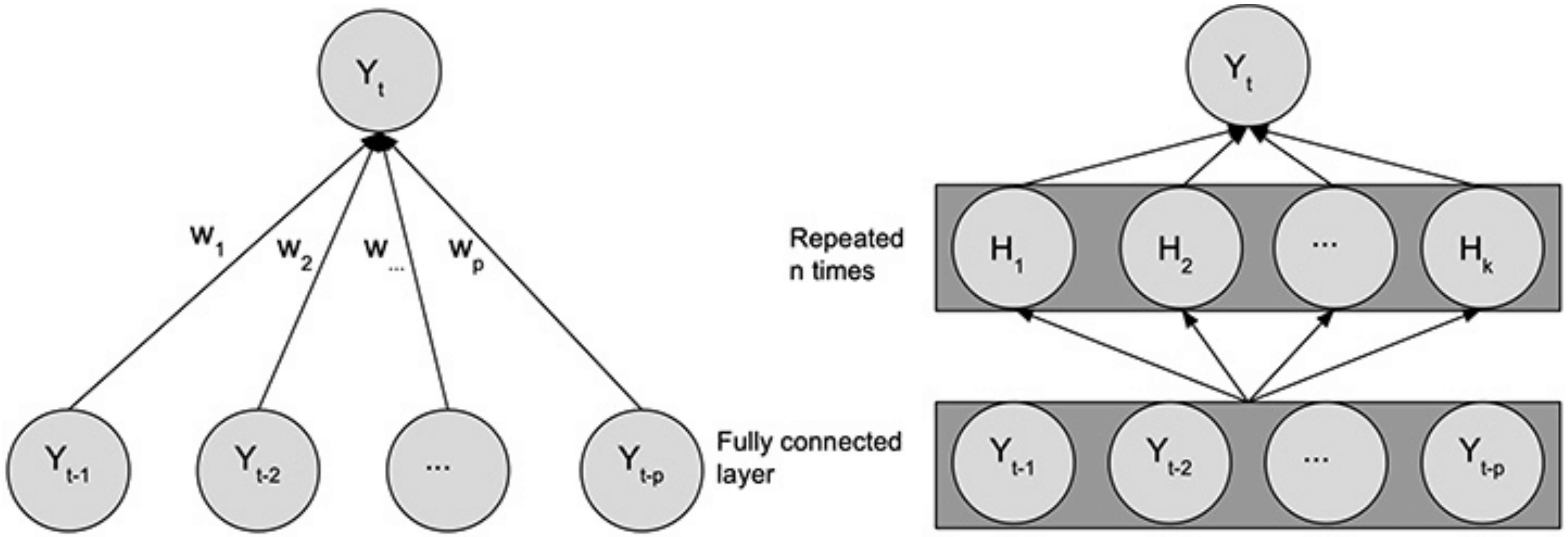
- It would appear that trainability and parallelization for feed-forward models comes at the price of reduced accuracy

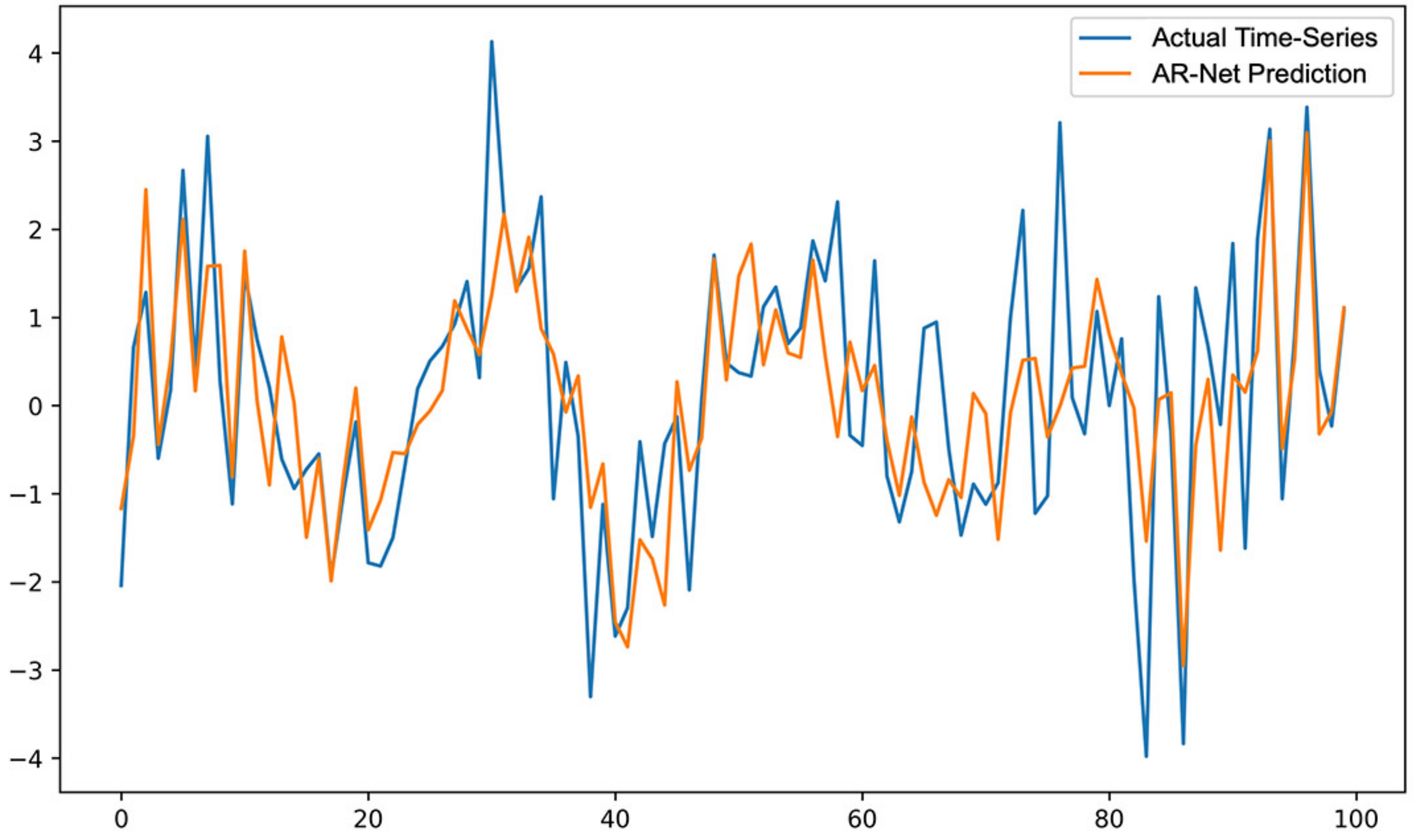
- However, research has shown that feed-forward networks can actually achieve the same accuracies as their recurrent counterparts on many tasks

- Why?
  - The unlimited context offered by recurrent models is not strictly necessary for modeling.
  - The "infinite memory" advantage of RNNs is largely absent in practice and that are effectively feedforward
    - either because truncated backpropagation through time cannot learn long patterns
    - or may be because models trainable by gradient descent cannot have long-term memory
  - Research suggests that if the recurrent model is stable (meaning the gradients can not explode), then the model can be well-approximated by a feed-forward network for the purposes of both inference and training

- References
  - When Recurrent Models Don't Need to be Recurrent
    http://www.offconvex.org/2018/07/27/approximating-recurrent/
  - Bai, Shaojie, J. Zico Kolter, and Vladlen Koltun. "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling." arXiv preprint arXiv:1803.01271 (2018).
    https://arxiv.org/pdf/1803.01271.pdf

Example: Facebook's AR-Net





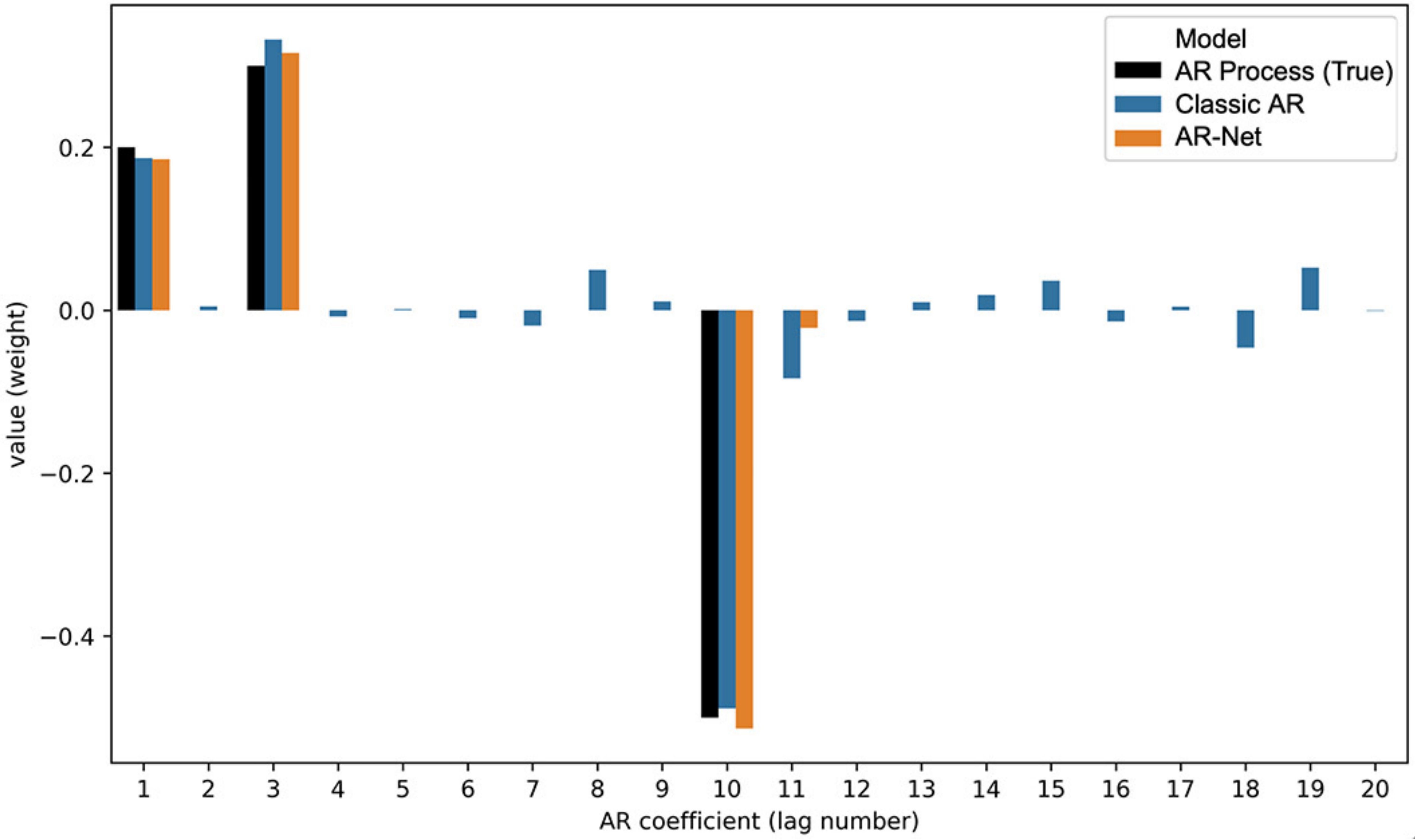
Left: AR-equivalent neural network without hidden layers (simplest form of AR-Net).
Right: AR-inspired neural network with n hidden layers (general AR-Net).

- Two advantages over its traditional counterpart
  - scales well to large orders, making it possible to estimate long-range dependencies
    - important in high-resolution monitoring applications
  - automatically selects and estimates the important coefficients of a sparse AR process
    - achieve this by introducing a small regularization factor of the learned weights
    - eliminating the need to know the true order of the AR process
- Can be expanded to include any arbitrary number of hidden layers but at less interpretability

https://ai.facebook.com/blog/ar-net-a-simple-autoregressive-neural-network-for-time-series/ & https://arxiv.org/pdf/1911.12436.pdf

Facebook's AR-Net: Learns only relevant weights

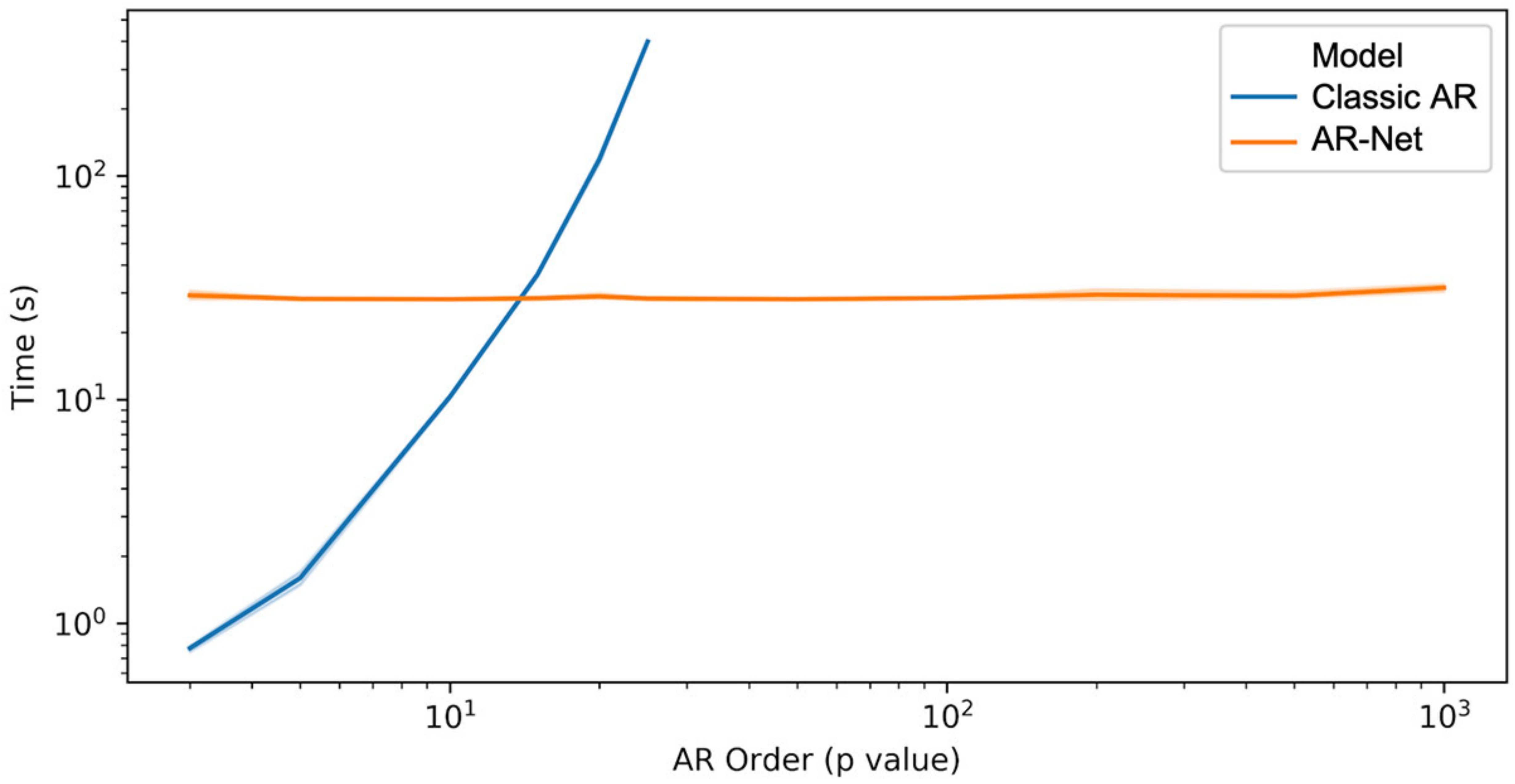
- If the order is unknown, AR-Net automatically learns the relevant weights, even if the underlying data is generated by a noisy and extremely sparse AR process
  - It achieves this by introducing a small regularization factor of the learned weights
  - In such a sparse setting, AR-Net outperforms classic AR



AR-Net effectively learns the sparse weights, setting the irrelevant weights to zero. Classic AR overestimates the irrelevant weights. Fitted on data generated by a noisy AR-3 process with sparsity (lags 1, 3, and 10 are non-zero).

https://ai.facebook.com/blog/ar-net-a-simple-autoregressive-neural-network-for-time-series/ & https://arxiv.org/pdf/1911.12436.pdf

Facebook's AR-Net: Computational Performance of Learning

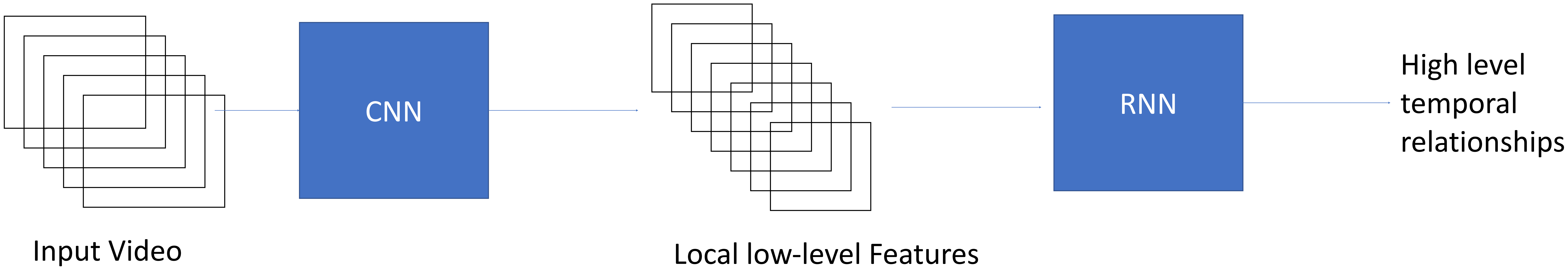


- Shows that modeling time series with neural networks can be just as interpretable as doing so using classical statistical methods
- Computationally tractable and simple for the practitioner to fit a sparse AR model of a high order
  - This makes it possible to model temporal data without having to determine the true order of the underlying AR process, allowing the model to automatically learn accurate long-range dependencies without overfitting

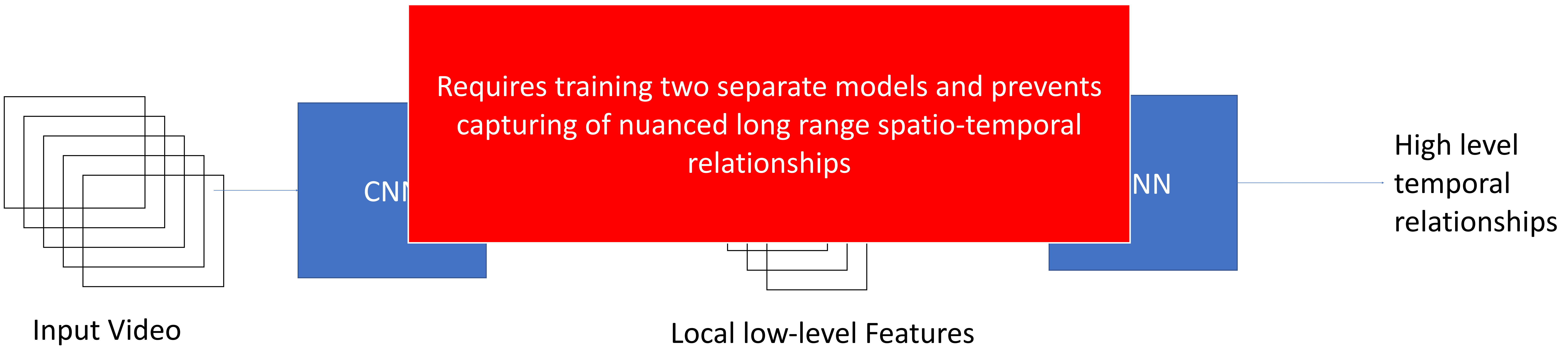
https://ai.facebook.com/blog/ar-net-a-simple-autoregressive-neural-network-for-time-series/ & https://arxiv.org/pdf/1911.12436.pdf

Approach 7: Time Convolution Networks

Video-based Action Segmentation



Video-based Action Segmentation



Time Convolution Networks (TCNs)

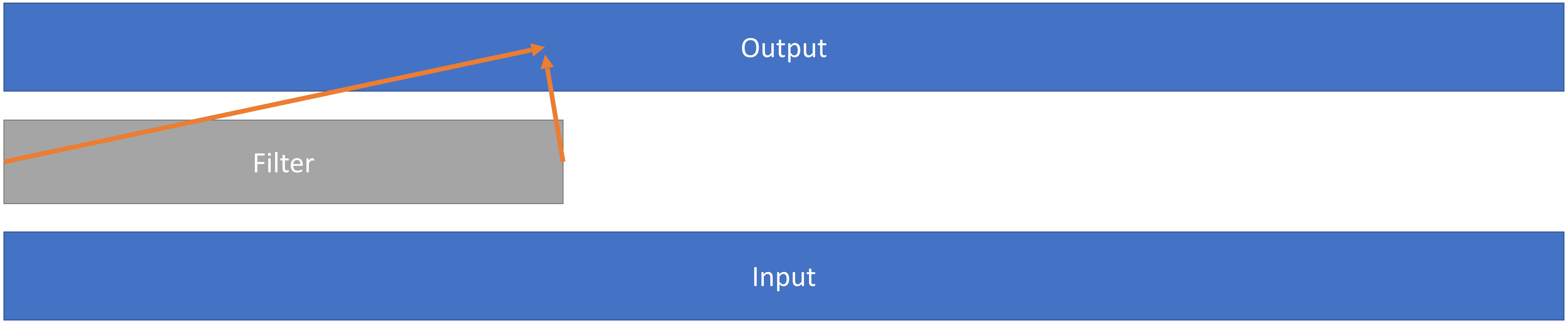
The architecture can take a sequence of any length and map it to an output sequence of the same length, just as with an RNNs.

The convolutions in the architecture are **causal**, meaning that there is no information “leakage” from future to past.

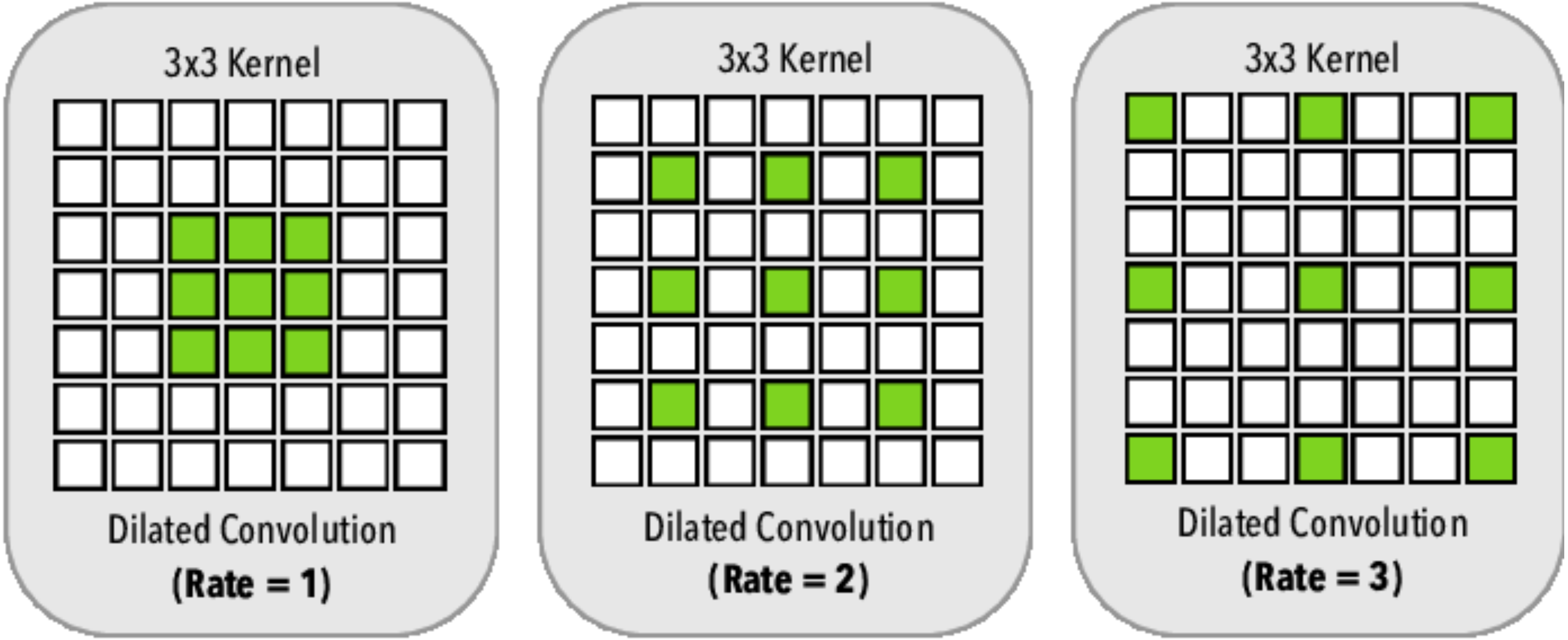
Take less time to train than RNN and are more memory efficient

Causal Convolutions

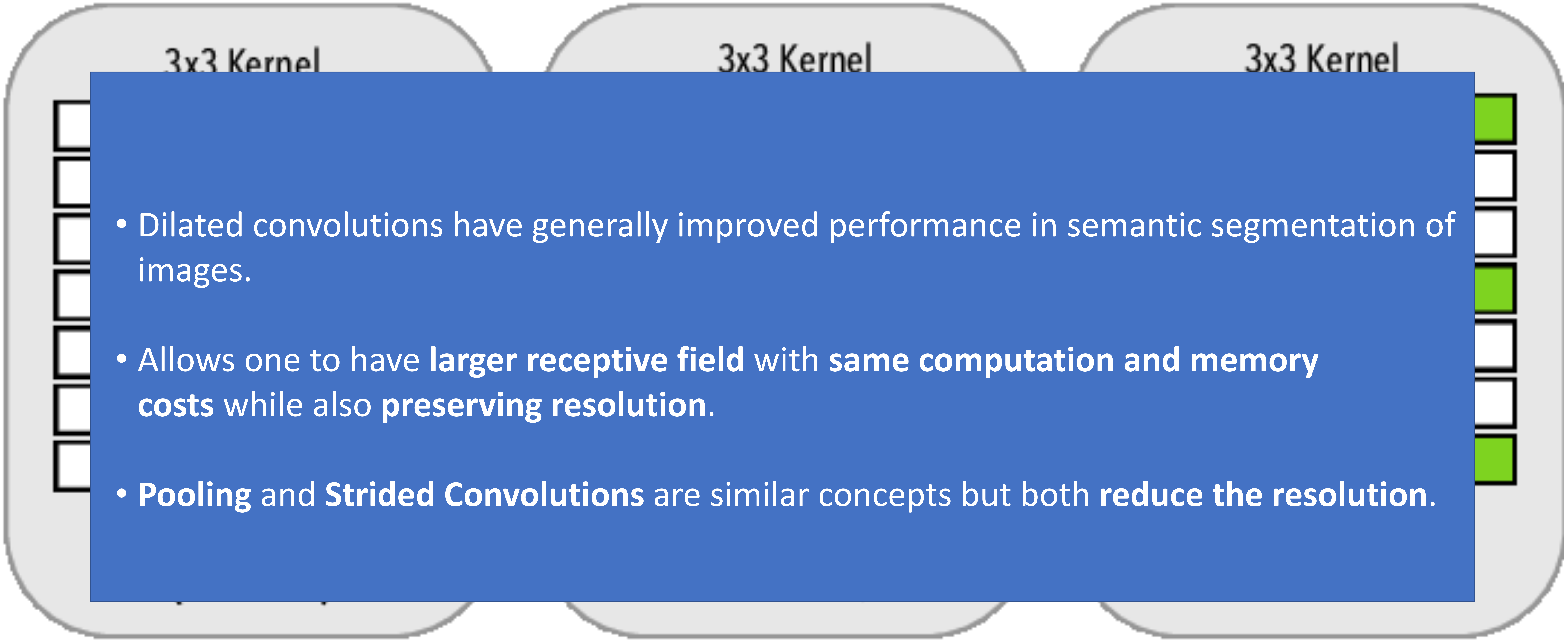
- The output at time t depends solely on current and past elements of the input.



Dilated Convolutions



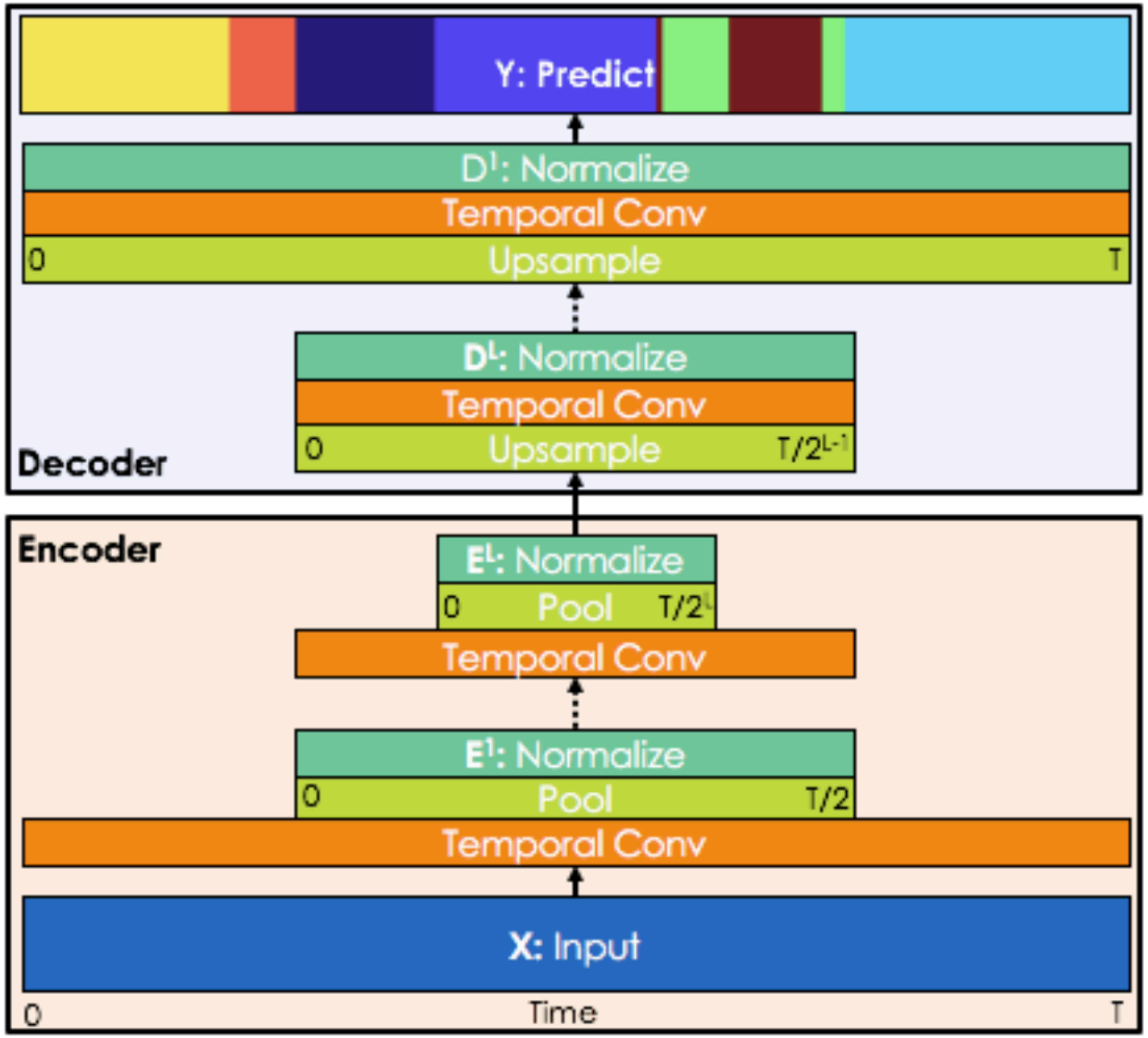
Dilated Convolutions



- Dilated convolutions have generally improved performance in semantic segmentation of images.
- Allows one to have **larger receptive field** with **same computation and memory costs** while also **preserving resolution**.
- **Pooling** and **Strided Convolutions** are similar concepts but both **reduce the resolution**.

What does a TCN use?

- **1D convolutions**
  - Captures how features at lower levels change over time
- **Pooling**
  - Efficient computation of long-range temporal patterns
- **Channel-wise normalization**
  - Robustness towards varying environmental conditions



Temporal Encoder-Decoder network hierarchically models actions from video or other time series data

evolve over the course of an action. The filters for each layer are parameterized by tensor $W^{(l)} \in \mathbb{R}^{F_l \times d \times F_{l-1}}$ and biases $b^{(l)} \in \mathbb{R}^{F_l}$, where $l \in \{1, \dots, L\}$ is the layer index and d is the filter duration. For the l-th layer of the encoder, the i-th component of the (unnormalized) activation $\hat{E}_t^{(l)} \in \mathbb{R}^{F_l}$ is a function of the incoming (normalized) activation matrix $E^{(l-1)} \in \mathbb{R}^{F_{l-1} \times T_{l-1}}$ from the previous layer

$$\hat{E}_{i,t}^{(l)} = f\Big(b_i^{(l)} + \sum_{t'=1}^{d} \langle W_{i,t',\cdot}^{(l)}, E_{\cdot,t+d-t'}^{(l-1)} \rangle\Big) \tag{1}$$

Max pooling is applied with width 2 across time (in 1D) such that $T_l = \frac{1}{2} T_{l-1}$.[1] Pooling enables us to efficiently compute activations over a long period of time.

Descriptors (TDD) [10]. We normalize the pooled activation vector $\hat{E}_t^{(l)}$ by the highest response at that time step, $m = \max_i \hat{E}_{i,t}^{(l)}$, with some small $\epsilon = 1\text{E-}5$ such that

$$E_t^{(l)} = \frac{1}{m + \epsilon} \hat{E}_t^{(l)}. \tag{2}$$

50 Salads ("eval" setup)

Sensor-based	**Edit**	**Acc**
[9] LC-SC-CRF	50.2	77.8
LSTM	54.5	73.3
TCN	**65.6**	**82.0**
Video-based	**Edit**	**Acc**
[8] VGG	7.6	38.3
[8] IDT	16.8	54.3
[8] Seg-ST-CNN	**62.0**	72.0
Spatial CNN	28.4	68.6
ST-CNN	55.5	74.2
TCN	61.1	**74.4**

GTEA

Video-based	**Edit**	**Acc**
[3] Hand-crafted	-	47.7
[16] EgoNet	-	57.6
[16] TDD	-	59.5
[16] EgoNet+TDD	-	**68.5**
Spatial CNN	36.6	56.1
ST-CNN	53.4	64.5
TCN	**58.8**	66.1

JIGSAWS

Sensor-based	**Edit**	**Acc**
[2] LSTM	75.3	80.5
[9] LC-SC-CRF	76.8	**83.4**
[2] Bidir LSTM	81.1	83.3
[17] SD-SDL	83.3	78.6
TCN	**85.8**	79.6
Vision-based	**Edit**	**Acc**
[19] MsM-CRF	-	71.7
[8] IDT	8.5	53.9
[8] VGG	24.3	45.9
[8] Seg-ST-CNN	66.6	74.7
Spatial CNN	37.7	74.0
ST-CNN	68.0	77.7
TCN	**83.1**	**81.4**

Table 1. Results on 50 Salads, Georgia Tech Egocentric Activities, and JHU-ISI Gesture and Skill Assessment Working Set. Notes: (1) Results using VGG and Improved Dense Trajectories (IDT) were intentionally computed without a temporal component for ablative analysis, hence their low edit scores. (2) We re-computed [9] using the author's public code to be consistent with the setup of [14].