Recurrent Neural Networks

- Human brain deals with information streams. Most data is obtained, processed, and generated sequentially.
  - E.g., listening: soundwaves → vocabularies/sentences
  - E.g., action: brain signals/instructions → sequential muscle movements
- Human thoughts have persistence; humans don't start their thinking from scratch every second.
  - As you read this sentence, you understand each word based on your prior knowledge.
- The applications of standard Artificial Neural Networks (and also Convolutional Networks) are limited due to:
  - They only accepted a fixed-size vector as input (e.g., an image) and produce a fixed-size vector as output (e.g., probabilities of different classes).
  - These models use a fixed amount of computational steps (e.g. the number of layers in the model).
- Recurrent Neural Networks (RNNs) are a family of neural networks introduced to **learn sequential data**.
  - Inspired by the temporal-dependent and persistent human thoughts

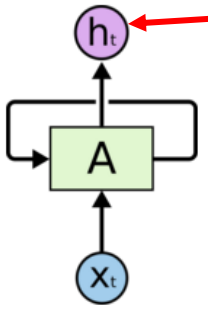
Real-life Sequence Learning Applications

- RNNs can be applied to various type of sequential data to learn the temporal patterns.
  - Time-series data (e.g., stock price) → Prediction, regression
  - Raw sensor data (e.g., signal, voice, handwriting) → Labels or text sequences
  - Text → Label (e.g., sentiment) or text sequence (e.g., translation, summary, answer)
  - Image and video → Text description (e.g., captions, scene interpretation)

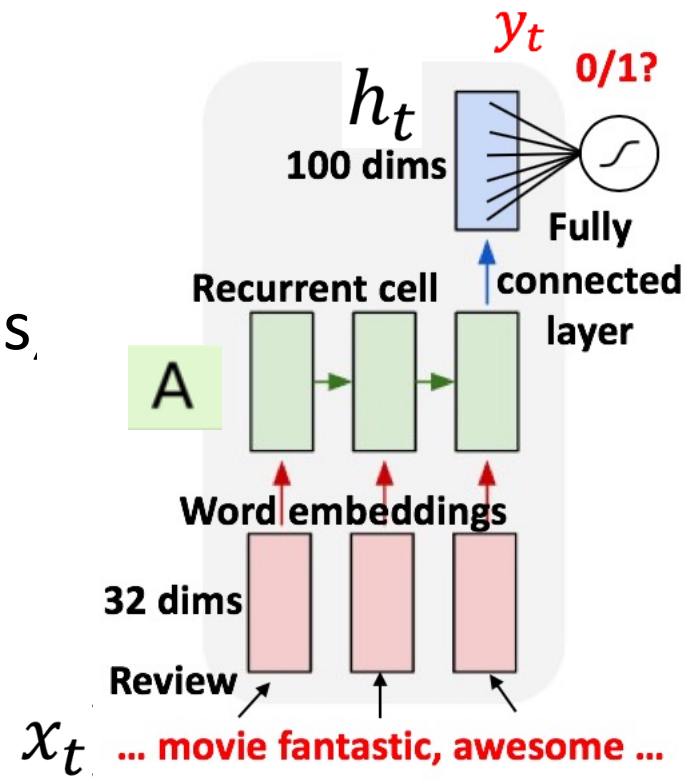
Task	**Input**	**Output**
Activity Recognition (Zhu et al. 2018)	Sensor Signals	Activity Labels
Machine translation (Sutskever et al. 2014)	English text	French text
Question answering (Bordes et al. 2014)	Question	Answer
Speech recognition (Graves et al. 2013)	Voice	Text
Handwriting prediction (Graves 2013)	Handwriting	Text
Opinion mining (Irsoy et al. 2014)	Text	Opinion expression

Recurrent Neural Networks

- Recurrent Neural Networks are networks with loops, allowing information to persist.



Output is to predict a vector h_t, where $output\ y_t = \varphi(h_t)$ at some time steps (t)



Recurrent Neural Networks have loops.

In the above diagram, a chunk of neural network, $A = f_W$, looks at some input x_t and outputs a value h_t. A loop allows information to be passed from one step of the network to the next.

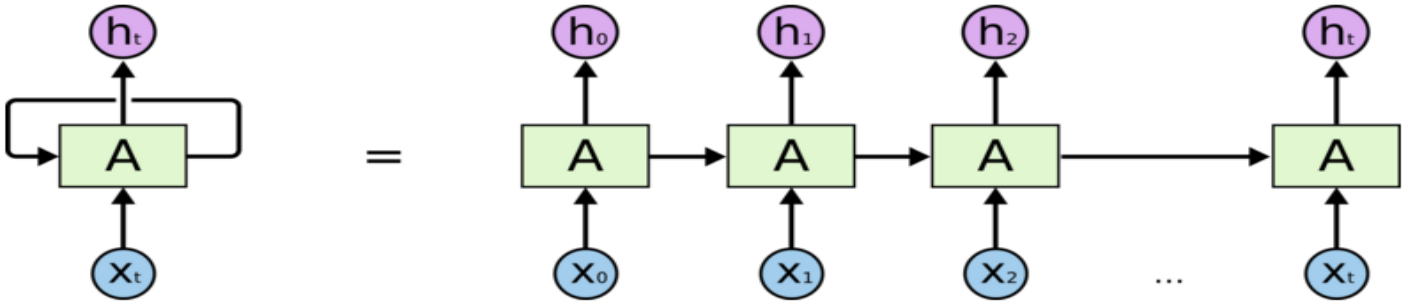
new state — old state

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

function with parameter W — Input vector at some time step

Recurrent Neural Networks

- Unrolling RNN



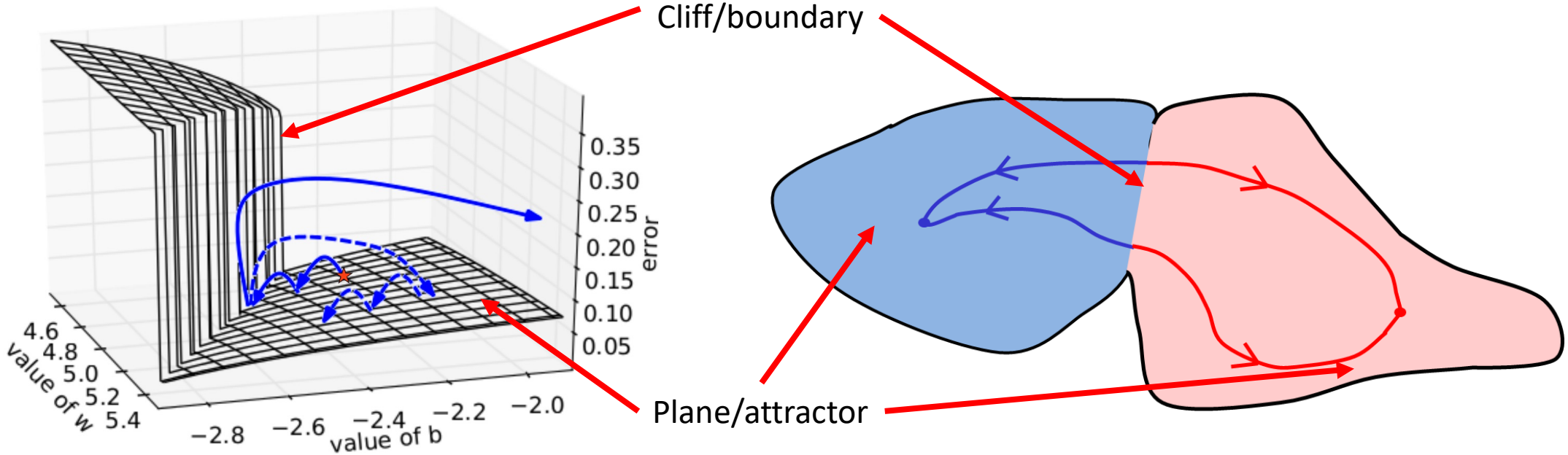
An unrolled recurrent neural network.

A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor. The diagram above shows what happens if we **unroll the loop**.

Recurrent Neural Networks

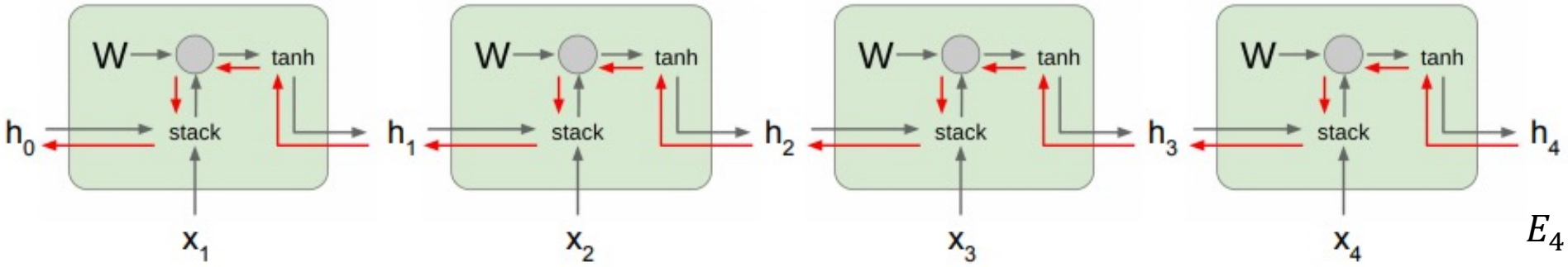
- The recurrent structure of RNNs enables the following characteristics:
  - Specialized for processing a sequence of values $x^{(1)}, \ldots, x^{(\tau)}$
    - Each value $x^{(i)}$ is processed with the **same network *A* that preserves past information**
  - Can scale to much **longer sequences** than would be practical for networks without a recurrent structure
    - Reusing network ***A*** reduces the required amount of parameters in the network
  - Can process **variable-length sequences**
    - The network complexity does not vary when the input length change
  - However, vanilla RNNs suffer from the training difficulty due to **exploding and vanishing gradients**.

Exploding and Vanishing Gradients



- **Exploding:** If we start almost exactly on the boundary (cliff), tiny changes can make a huge difference.
- **Vanishing:** If we start a trajectory within an attractor (plane, flat surface), small changes in where we start make no difference to where we end up.
- Both cases hinder the learning process.

Exploding and Vanishing Gradients



$$E_4 = J(y_4, \widehat{y_4})$$

In vanilla RNNs, computing this gradient involves many factors of W_{hh} (and repeated $\tanh$)*. If we decompose the singular values of the gradient multiplication matrix,

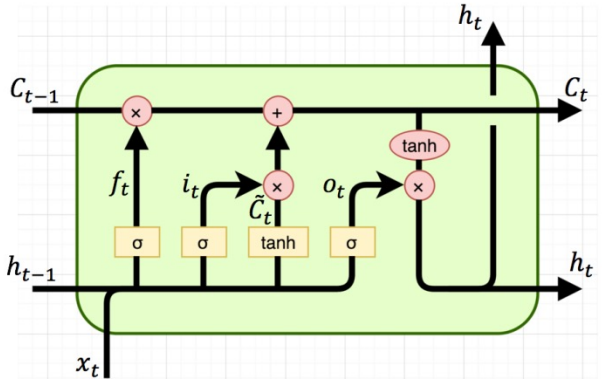
- Largest singular value > 1 → **Exploding gradients**
  - Slight error in the late time steps causes drastic updates in the early time steps → Unstable learning
- Largest singular value < 1 → **Vanishing gradients**
  - Gradients passed to the early time steps is close to 0. → Uninformed correction

* Refer to Bengio et al. (1994) or Goodfellow et al. (2016) for a complete derivation

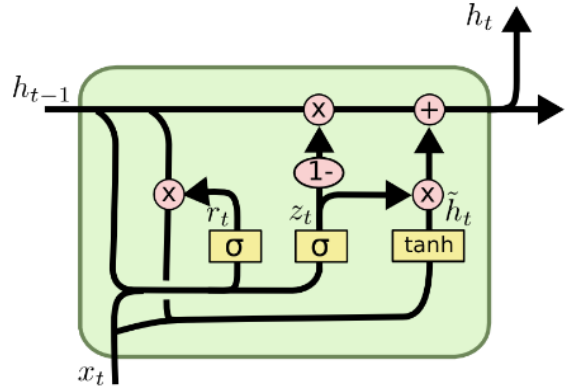
Networks with Memory

- Vanilla RNN operates in a “multiplicative” way (repeated tanh).
- Two recurrent cell designs were proposed and widely adopted:
  - **Long Short-Term Memory (LSTM)** (Hochreiter and Schmidhuber, 1997)
  - Gated Recurrent Unit (GRU) (Cho et al. 2014)
- Both designs process information in an “additive” way with gates to control information flow.
  - **Sigmoid gate outputs numbers between 0 and 1, describing how much of each component should be let through.**

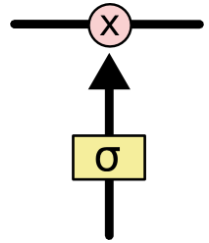
E.g. $f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right) = Sigmoid\left(W_f x_t + U_t h_{t-1} + b_f\right)$



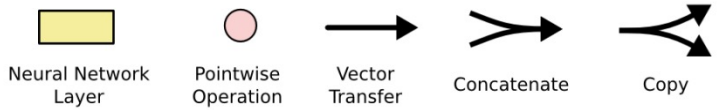
Standard LSTM Cell



GRU Cell

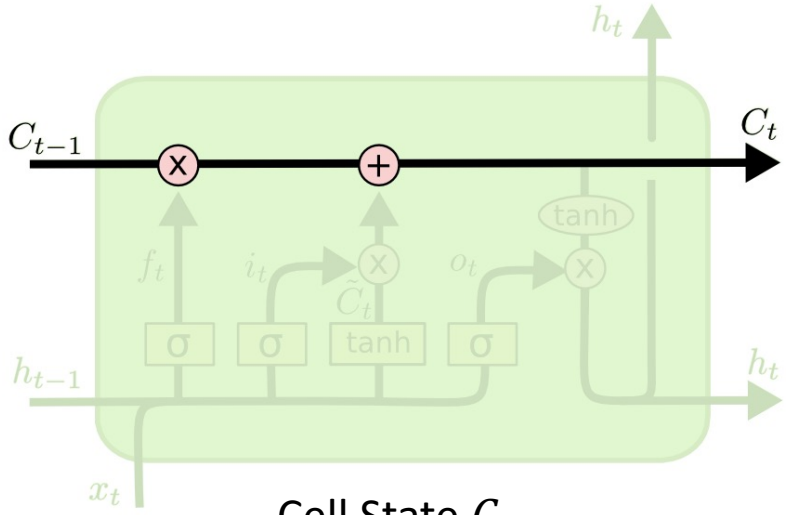


A Sigmoid Gate



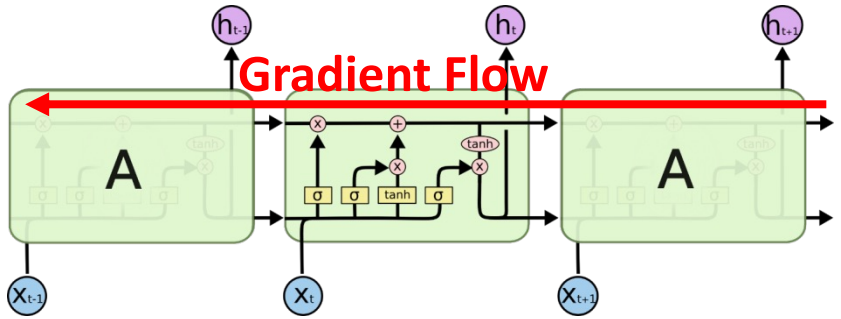
Long Short-Term Memory (LSTM)

- The key to LSTMs is the **cell state**.
  - **Stores information of the past → long-term memory**
  - **Passes along time steps with minor linear interactions → "additive"**
  - Results in an **uninterrupted gradient flow** → errors in the past pertain and impact learning in the future

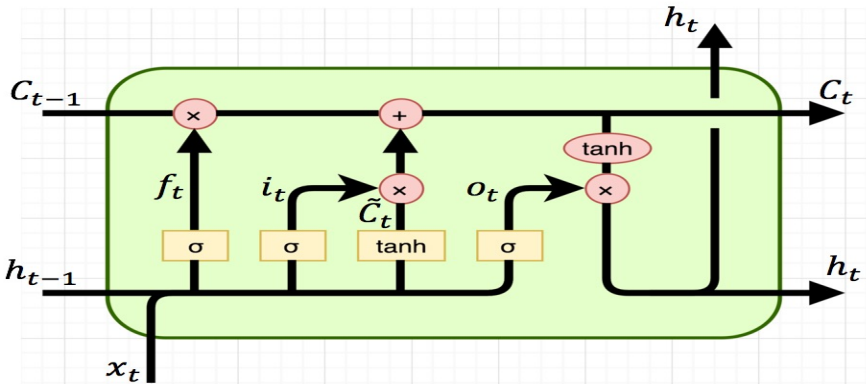


Cell State C_t

- The LSTM cell manipulates input information with three gates.
  - **Input gate** → controls the intake of new information
  - **Forget gate** → determines what part of the cell state to be updated
  - **Output gate** → determines what part of the cell state to output



LSTM: Components & Flow



- LSM unit output
- ***Output gate*** units
- Transformed memory cell contents
- Gated update to memory cell units
- ***Forget gate*** units
- ***Input gate*** units
- Potential *input* to memory cell

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

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

$$\tanh(C_t)$$

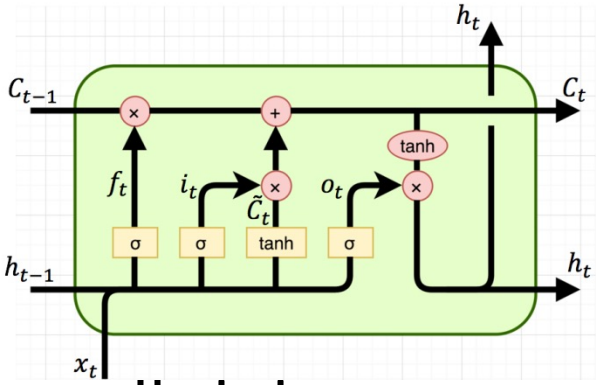
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

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

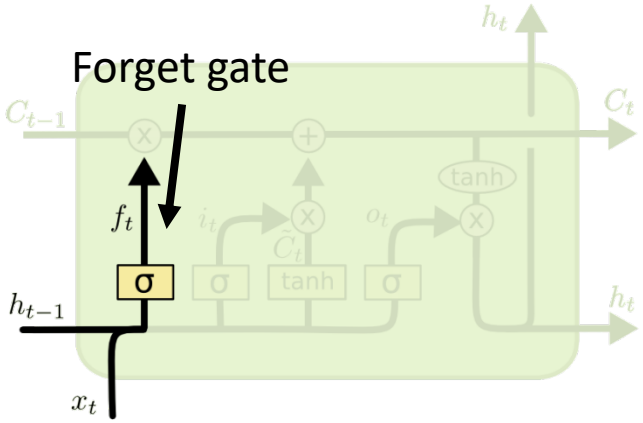
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

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

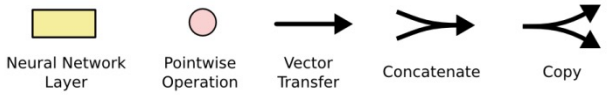
Step-by-step LSTM Walk Through



- **Step 1**: Decide what information to throw away from the cell state (memory) ➔ $f_t * C_{t-1}$
  - The output of the previous state h_{t-1} and the new information x_t jointly determine what to forget
    - h_{t-1} contains selected features from the memory C_{t-1}
  - Forget gate f_t ranges between $[0, 1]$

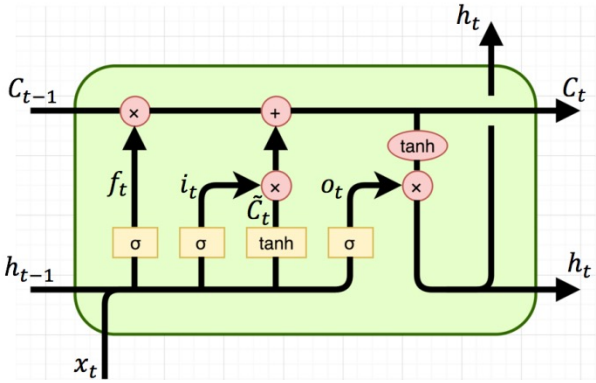


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right)$$



Text processing example:
Cell state may include the <u>gender of the current subject</u> (h_{t-1}). When the model observes a <u>new subject</u> (x_t), it may <u>want to forget</u> ($f_t \rightarrow 0$) the old subject in the <u>memory</u> (C_{t-1}).

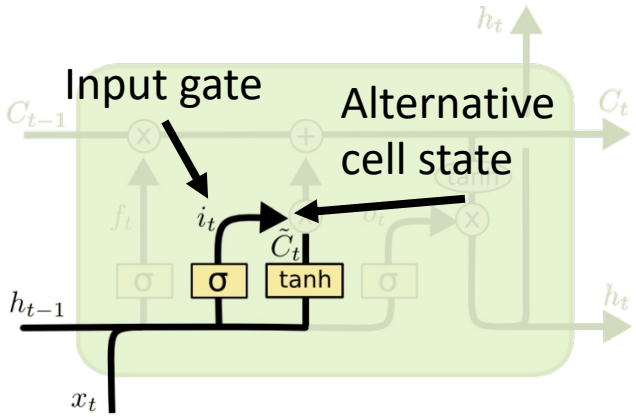
Step-by-step LSTM Walk Through



- **Step 2**: Prepare the updates for the cell state

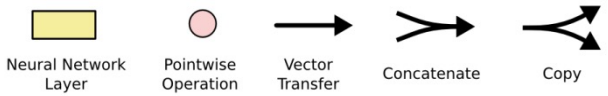
  from input ➔ $i_t * \tilde{C}_t$

  - An alternative cell state $\widetilde{C}_t$ is created from the new information x_t with the guidance of h_{t-1}.
  - Input gate i_t ranges between $[0, 1]$



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

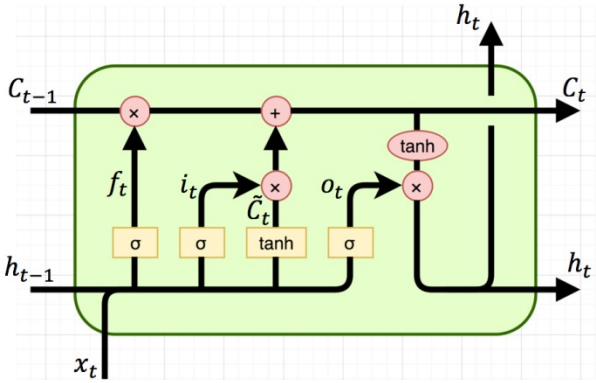
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$



Example:

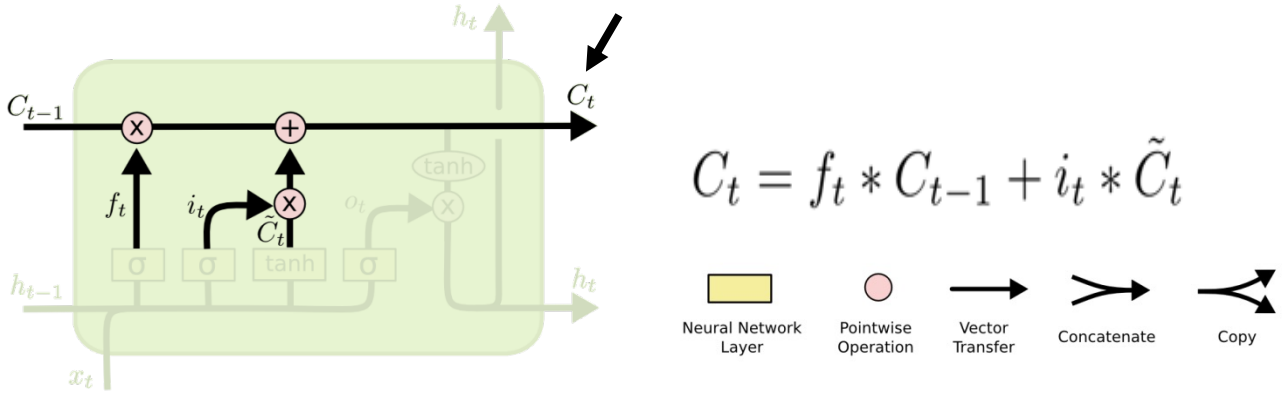
The model may want to add ($i_t \rightarrow 1$) the gender of new subject ($\widetilde{C}_t$) to the cell state to replace the old one it is forgetting.

Step-by-step LSTM Walk Through



- **Step 3**: Update the cell state ➔ $f_t * C_{t-1} + i_t * \tilde{C}_t$
  - The new cell state C_t is comprised of information from the past $f_t * C_{t-1}$ and valuable new information $i_t * \widetilde{C}_t$
  - $*$ denotes elementwise multiplication

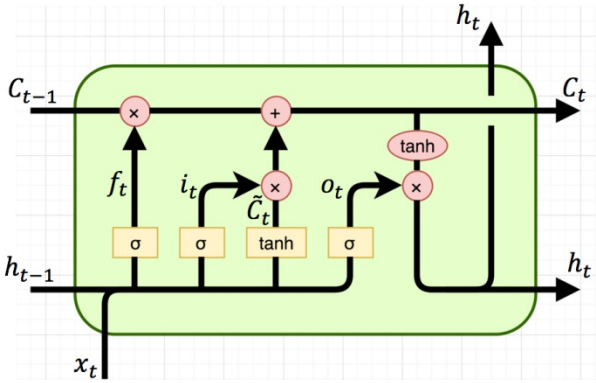
New cell state



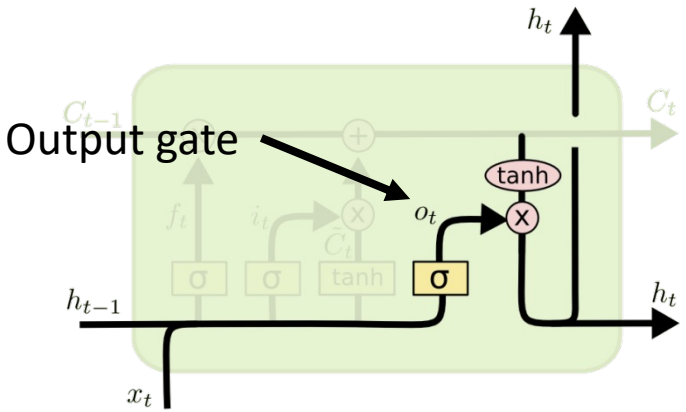
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Example:
The model drops the old gender information ($f_t * C_{t-1}$) and adds new gender information ($i_t * \widetilde{C}_t$) to form the new cell state (C_t).

Step-by-step LSTM Walk Through

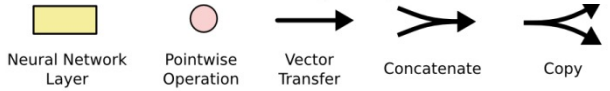


- **Step 4**: Decide the filtered output from the new cell state ➔ $o_t * \tanh(C_t)$
  - $\tanh$ function filters the new cell state to characterize stored information
    - Significant information in C_t → ± 1
    - Minor details → 0
  - Output gate o_t ranges between $[0, 1]$
  - h_t serves as a control signal for the next time step



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

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

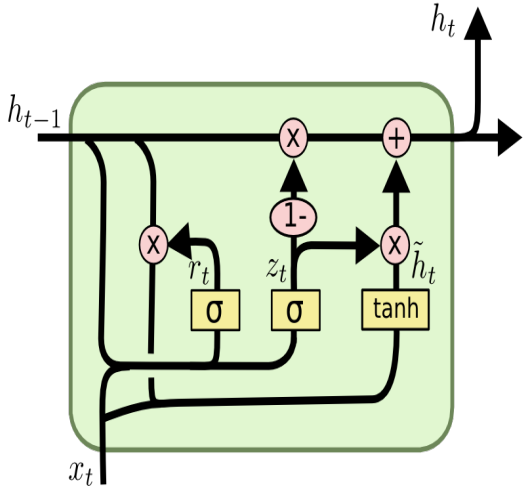


Example:
Since the model just saw a new subject (x_t), it might want to output ($o_t \rightarrow 1$) information relevant to a verb ($\tanh(C_t)$), e.g., singular/plural, in case a verb comes next.

Gated Recurrent Unit (GRU)

- GRU is a variation of LSTM that also adopts the gated design.
- Differences:
  - GRU uses an **update gate z** to substitute the input and forget gates i_t and f_t
  - Combined the cell state C_t and hidden state h_t in LSTM as a single cell state h_t

- GRU obtains similar performance compared to LSTM with fewer parameters and faster convergence. (Cho et al. 2014)



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

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

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

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

Update gate: controls the composition of the new state

Reset gate: determines how much old information is needed in the alternative state $\widetilde{h_t}$

Alternative state: contains new information

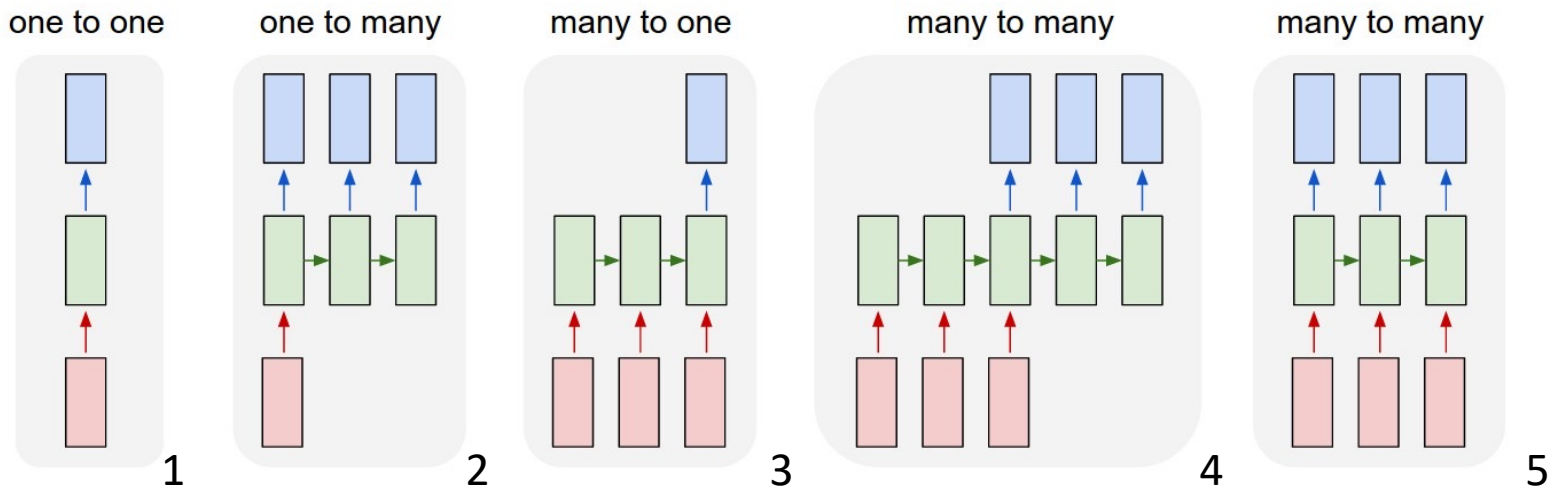
New state: replace selected old information with new information in the new state

Sequence Learning Architectures

- Learning on RNN is more robust when the vanishing/exploding gradient problem is resolved.
  - RNNs can now be applied to different Sequence Learning tasks.

- Recurrent NN architecture is flexible to operate over various sequences of vectors.
  - Sequence in the input, the output, or in the most general case both
  - Architecture with one or more RNN layers

Sequence Learning with One RNN Layer

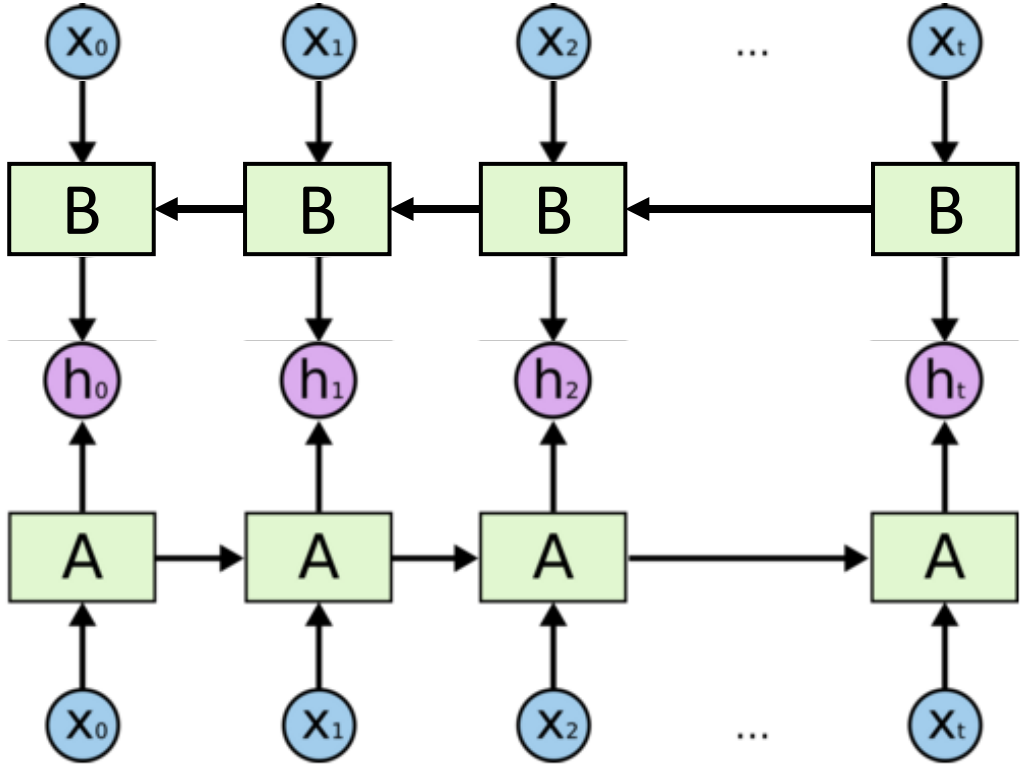


- Each rectangle is a vector and arrows represent functions (e.g. matrix multiply).
- Input vectors are in red, output vectors are in blue and green vectors hold the RNN's state

(1) Standard NN mode without recurrent structure (e.g. **image classification, one label for one image**).
(2) Sequence output (e.g. **image captioning, takes an image and outputs a sentence of words**).
(3) Sequence input (e.g. **sentiment analysis, a sentence is classified as expressing positive or negative sentiment**).
(4) Sequence input and sequence output (e.g. **machine translation, a sentence in English is translated into a sentence in French**).
(5) Synced sequence input and output (e.g. **video classification, label each frame of the video**).

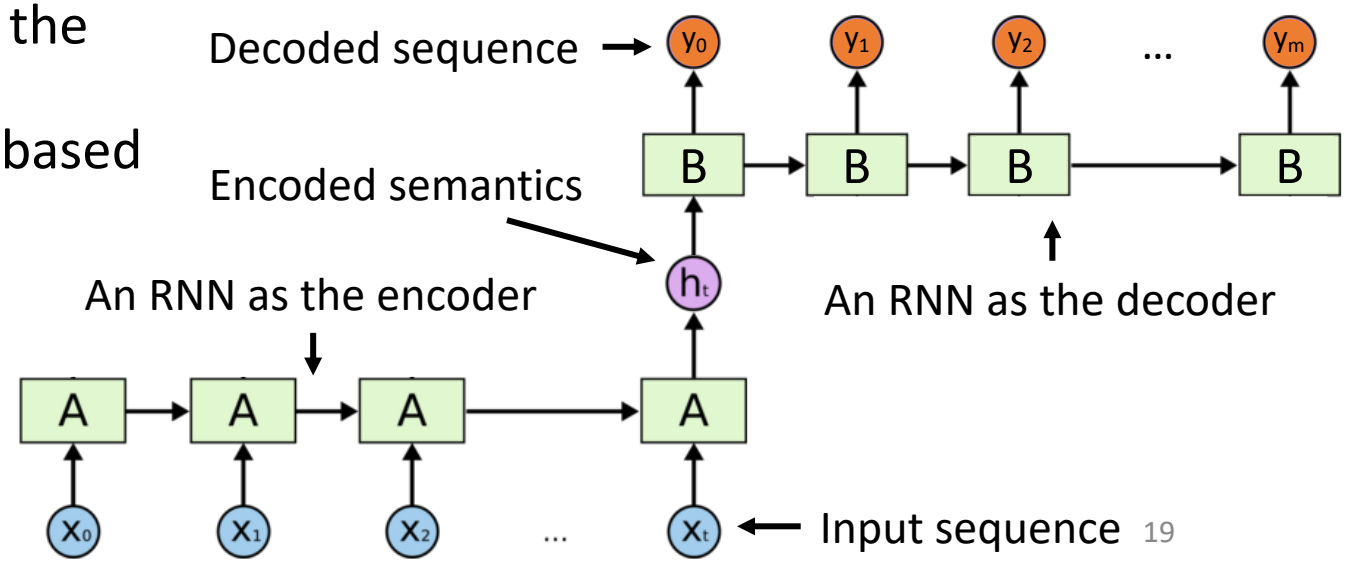
Sequence Learning with Multiple RNN Layers

- Bidirectional RNN
  - Connects two recurrent units (synced many-to-many model) of opposite directions to the same output.
  - Captures forward and backward information from the input sequence
  - Apply to data whose current state (e.g., h_0) can be better determined when given future information (e.g., $x_1, x_2, \ldots, x_t$)
    - E.g., in the sentence “the bank is robbed,” the semantics of “bank” can be determined given the verb “robbed.”



Sequence Learning with Multiple RNN Layers

- **Sequence-to-Sequence (Seq2Seq) model**
  - Developed by Google in 2018 for use in machine translation.
  - Seq2seq turns one sequence into another sequence. It does so by use of a recurrent neural network (RNN) or more often LSTM or GRU to avoid the problem of vanishing gradient.
  - The primary components are one Encoder and one Decoder network. The encoder turns each item into a corresponding hidden vector containing the item and its context. The decoder reverses the process, turning the vector into an output item, using the previous output as the input context.
  - **Encoder RNN**: extract and compress the semantics from the input sequence
  - **Decoder RNN**: generate a sequence based on the input semantics
  - Apply to tasks such as machine translation
    - Similar underlying semantics
    - E.g., “I love you.” to “Je t’aime.”



Implementing RNN in Python using Keras

- Keras provides APIs for the vanilla RNN design, LSTM, and GRU cells:
  - `keras.layers.SimpleRNN(units, activation='tanh', return_sequences=False, …)`
  - `keras.layers.LSTM(units, activation='tanh', recurrent_activation='hard_sigmoid', …)`
  - `keras.layers.GRU(units, activation='tanh', recurrent_activation='hard_sigmoid', …)`

- Critical parameters*
  - units: dimensionality of the output space
  - activation: activation function to use on the recurrent layer, ‘tanh’ by default
  - recurrent_activation: activation function for the gates
  - return_sequences:
    - **`True`** to obtain the output on each time step
    - **`False`** to obtain the last output

*: More can be found on https://keras.io/layers/recurrent/

Implementation Example: Sequence Classification with LSTM

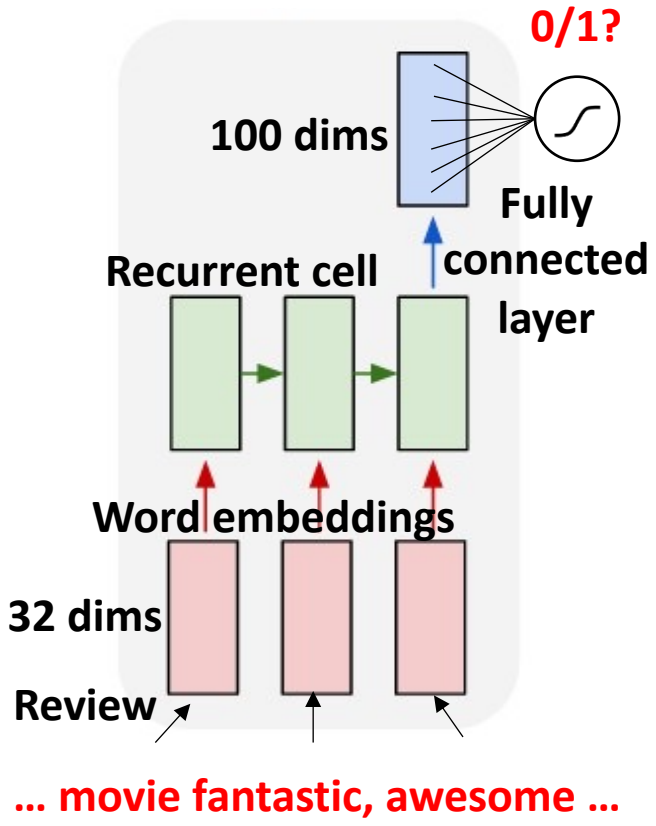
- IMDB movie review sentiment classification problem
  - Classify each movie review into a sentiment class (positive (1) or negative (0))

```
import numpy
from keras.datasets import imdb                    // keras provides this built-in dataset
from keras.models import Sequential
from keras.layers import Dense, LSTM
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
# fix random seed for reproducibility
numpy.random.seed(7)

# load the dataset but only keep the top n words, zero the rest  // data preprocessing
top_words = 5000
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=top_words)
# truncate and pad input sequences
max_review_length = 500
X_train = sequence.pad_sequences(X_train, maxlen=max_review_length)
X_test = sequence.pad_sequences(X_test, maxlen=max_review_length)
```

https://machinelearningmastery.com/sequence-classification-lstm-recurrent-neural-networks-python-keras/
http://ai.stanford.edu/~amaas/data/sentiment/

Implementation Example: Sequence Classification with LSTM (many to one)

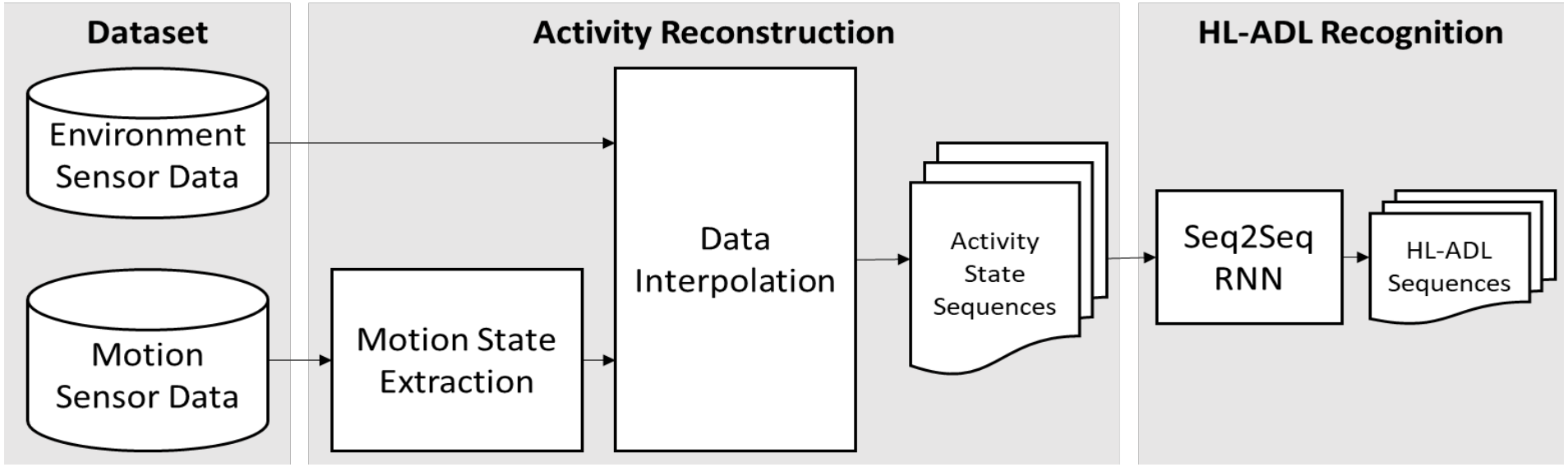
```
# define word embedding size
embedding_vecor_length = 32

# create a sequential model
model = Sequential()
# transform input sequence into word embedding representation
model.add(Embedding(top_words, embedding_vecor_length,
input_length=max_review_length))
# generate a 100-dimention output, can be replaced by GRU or SimpleRNN
recurrent cells
model.add(LSTM(100))
# a fully connected layer weight the 100-dimention output to one node for
prediction
model.add(Dense(1, activation='sigmoid’))
# use binary_crossentropy as the loss function for the classification
model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
print(model.summary())

# model training (for 3 epochs)
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=3,
batch_size=64)

# Final evaluation of the model
scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```



RNN Applications: High-level Activities of Daily Living (HL-ADL) Recognition

- Activities of Daily Living (ADLs) are introduced to evaluate the self-care ability of senior citizens (Williams 2014).
- Sensor-based home monitoring systems
  - **Environment:** Placed in the environment to capture changes (e.g., on/off, motion)
  - **Wearable:** Attached to the body to measure movements and physiological signals
- Sensors sample at 10 Hz → 0.8 million records per day
- Machine learning (esp. deep learning) needed to recognize ADLs from the large amount of data
  - Mid-level (ML) ADLs: gestures, gait, etc.
  - **High-level (HL) ADLs: preparing food, medical intake, senior care, etc.**
- Research Objective: develop a universal ADL recognition framework to extract HL-ADLs from raw sensor data

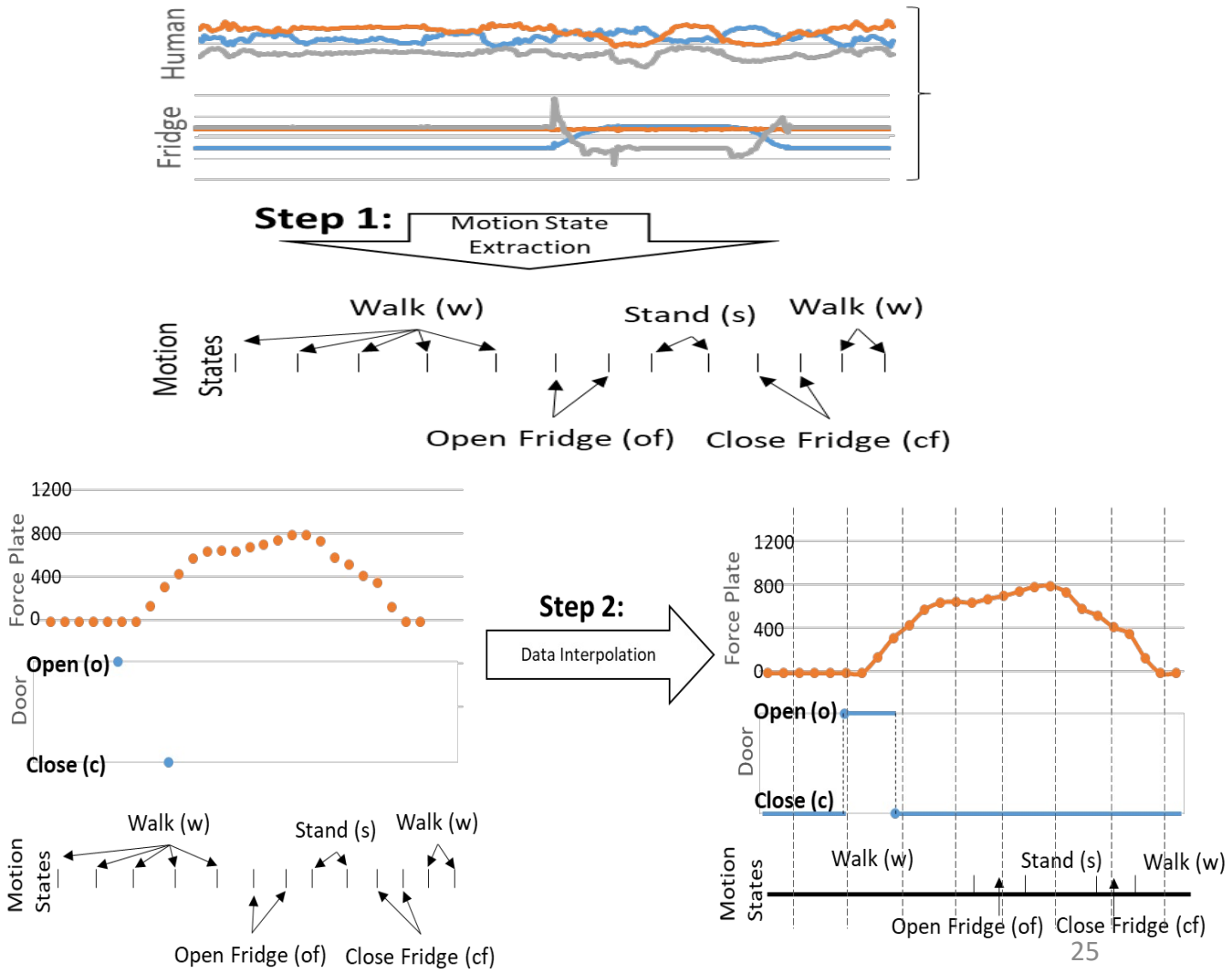
RNN Applications: Seq2Seq-ADL Research Design



- Intuition: Recognizing the HL-ADLs from the sensor data is similar to captioning/translation. We can generate ADL label sequence with a Seq2Seq model for the input data. The underlying semantics are similar.
- A Seq2Seq model is designed to extract HL-ADLs from the activity state sequence

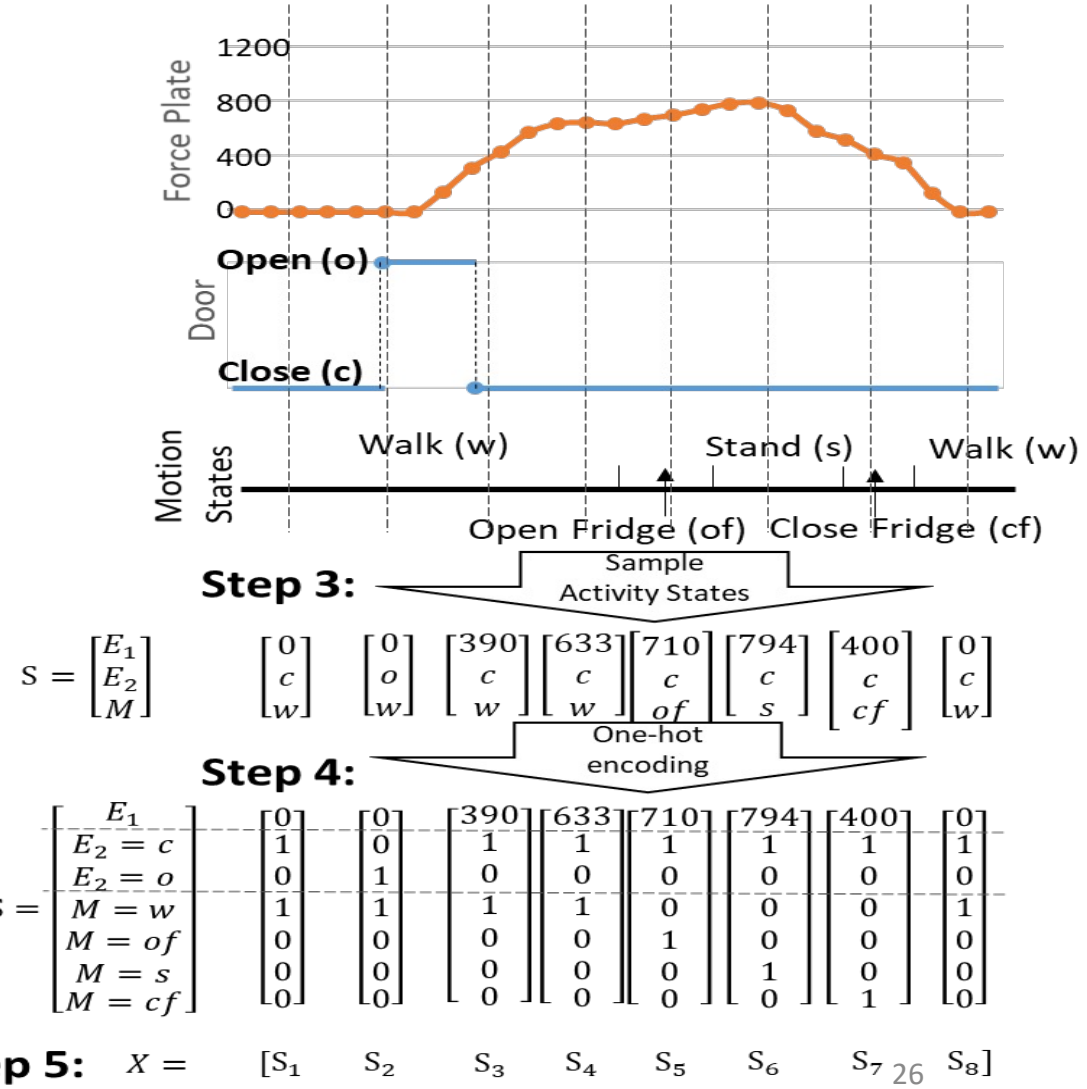
RNN Applications: Activity Reconstruction

- Objective: create temporally aligned activity representation from different data sources
- Four sensors for demonstration:
  - a force plate → pressure on the floor
  - a door on/off sensor → open (o) and close (c) states
  - a human motion sensor
  - object motion sensor attached on the fridge
- **Step 1.** Extract discrete motion states from motion sensor data with a state-of-the-art gesture recognition model
- **Steps 2.** Interpolate discrete data from each sensor



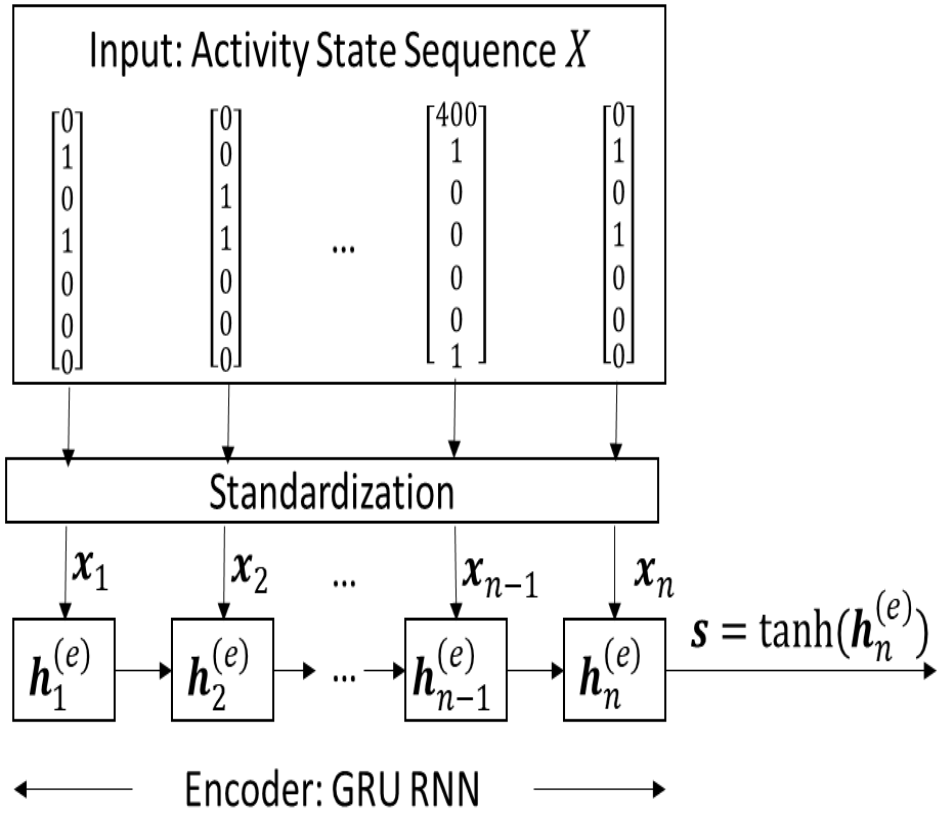
RNN Applications: Activity Reconstruction

- **Steps 3.** Sample each data stream at same timestamps to construct the Activity State representations
  - Temporally aligned observations
- **Steps 4.** Encode the Activity States S_i
  - Encode categorical (discrete) values using one-hot encoding
- **Step 5.** Organize the states vector in a matrix X
  - Data matrix X aggregates temporally aligned sensor observation sequences to represent the activity.



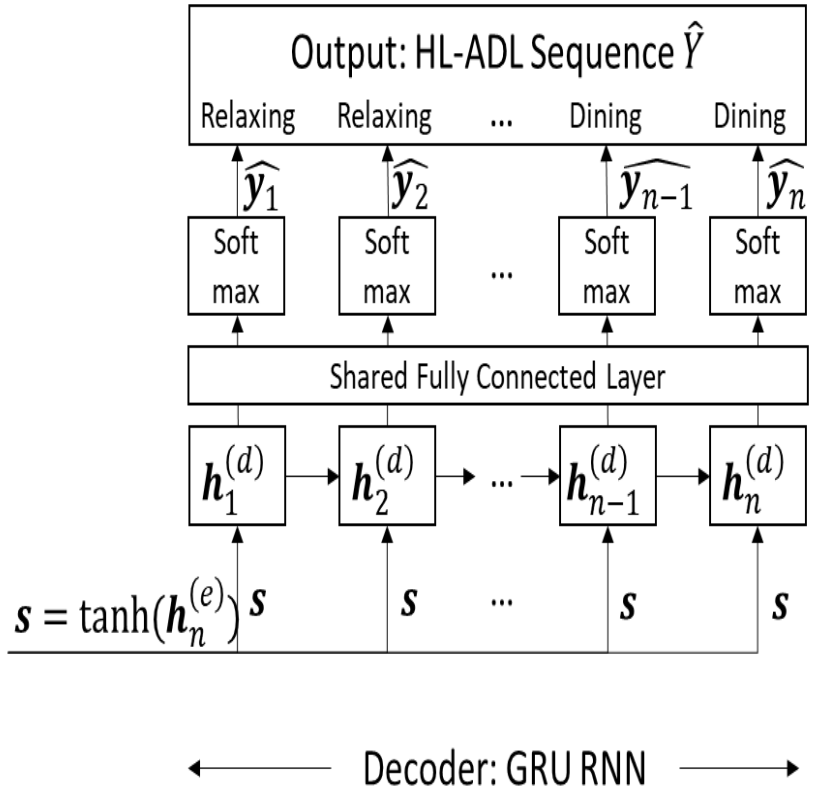
RNN Applications: Seq2Seq – Encoder

- The encoder network takes the Activity State Sequence X as the input to generate the activity semantics vector s.
- The encoder network adopts GRU recurrent cells to learn temporal patterns.
  - Each hidden state $h_i^{(e)}$ depends on the input x_i and the previous state $h_{i-1}^{(e)}$.
- s is expected to be a condensed representation for human/object motions, object usages, and their temporal patterns during the period.



RNN Applications: Seq2Seq – Decoder

- The decoder network takes the encoded activity semantics vector s to generate HL-ADL label for each input vector.
- The decoder network also adopts GRU recurrent cells to interpret the temporal patterns.
- Multi-class classification
  - Softmax → probability distribution over output classes
  - Categorical cross-entropy loss



RNN Applications: Activities of Daily Living (ADL) Recognition using S2S_GRU

- Our proposed S2S_GRU model is evaluated on two different datasets.
- Our model is more accurate and flexible in adjusting to different real-life HL-ADL recognition applications.

Recall of S2S_GRU against benchmarks on a wearable/environment motion sensor dataset

Recall (%)	Relaxing	Coffee time	Early morning	Cleanup	Sandwich time
S2S_GRU	61.7	66.5	72.1	82.6	79.1
HMM	48.8	19.5***	6.4***	17.4***	40.9***
S2S_LSTM	60.0	54.9**	75.9	83.2	59.9***

Recall of S2S_GRU against benchmarks on a environment sensor dataset

Recall (%)	Filling pillbox	Watch DVD	Water plants	Answer the phone	Prepare gift card	Prepare soup	Cleaning	Choose outfit
S2S_GRU	88.6	79.0	58.5	80.8	92.1	71.1	85.0	58.0
HMM	33.2***	31.6***	16.8***	22.0***	29.9***	26.4***	20.4***	22.5***
S2S_LSTM	86.1*	77.8	64.3	74.9*	90.8*	69.6	82.5	49.5**

*: p-value<0.05, **: p-value<0.01, ***: p-value<0.001

Summary

- LSTM and GRU are RNNs that retain past information and update with a gated design.
  - The “additive” structure avoids vanishing gradient problem.
- RNNs allow flexible architecture designs to adapt to different sequence learning requirements.
- RNNs have broad real-life applications.
  - Text processing, machine translation, signal extraction/recognition, image captioning
  - Mobile health analytics, activity of daily living, senior care

Important References

- Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, *5*(2), 157-166.
- Goodfellow, I., Bengio, Y., Courville, A., & Bengio, Y. (2016). *Deep learning* (Vol. 1). Cambridge: MIT press.
- Graves, A. (2012). *Supervised sequence labelling with recurrent neural networks*. *https://www.cs.toronto.edu/~graves/preprint.pdf*
- Jozefowicz, R., Zaremba, W., & Sutskever, I. (2015, June). An empirical exploration of recurrent network architectures. In *International Conference on Machine Learning* (pp. 2342-2350).
- Lipton, Z. C., Berkowitz, J., & Elkan, C. (2015). A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019*.
- Salehinejad, H., Baarbe, J., Sankar, S., Barfett, J., Colak, E., & Valaee, S. (2017). Recent Advances in Recurrent Neural Networks. *arXiv preprint arXiv:1801.01078*.