Sequential Data

- Sometimes the sequence of data matters.
  - Text generation
  - Stock price prediction

Sequential Data

- Sometimes the sequence of data matters.
  - Text generation
  - Stock price prediction
- The clouds are in the .... ?

Sequential Data

- Sometimes the sequence of data matters.
  - Text generation
  - Stock price prediction
- The clouds are in the .... ?
  - sky

Sequential Data

- Sometimes the sequence of data matters.
  - Text generation
  - Stock price prediction
- The clouds are in the .... ?
  - sky
- Simple solution: N-grams?

Sequential Data

- Sometimes the sequence of data matters.
  - Text generation
  - Stock price prediction
- The clouds are in the .... ?
  - sky
- Simple solution: N-grams?
  - Hard to represent patterns with more than a few words (possible patterns increases exponentially)

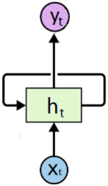
Sequential Data

- Sometimes the sequence of data matters.
  - Text generation
  - Stock price prediction
- The clouds are in the .... ?
  - sky
- Simple solution: N-grams?
  - Hard to represent patterns with more than a few words (possible patterns increases exponentially)
- Simple solution: Neural networks?
  - Fixed input/output size
  - Fixed number of steps

Recurrent Neural Networks

- **Recurrent neural networks (RNNs)** are networks with loops, allowing information to persist [Rumelhart et al., 1986].

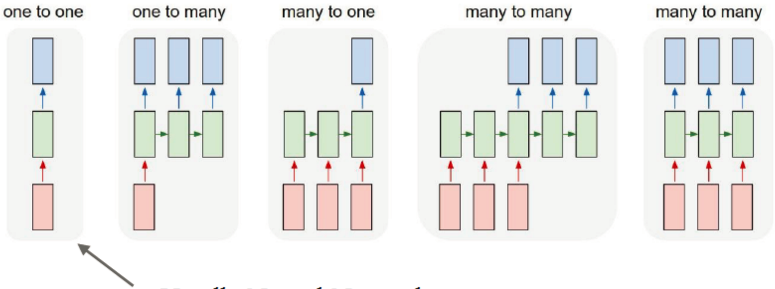
$$\underbrace{\boxed{h_t}}_{\text{new state}} = \underbrace{\boxed{f_W}}_{\text{some function with parameters W}}(\underbrace{\boxed{h_{t-1}}}_{\text{old state}}, \underbrace{\boxed{x_t}}_{\text{input vector at some time step}})$$



- Have **memory** that keeps track of information observed so far
- Maps from the entire history of previous inputs to each output
- Handle sequential data

Adapted from: *N. Cinbis and C. Olah*

Recurrent Networks Offer a Lot of Flexibility



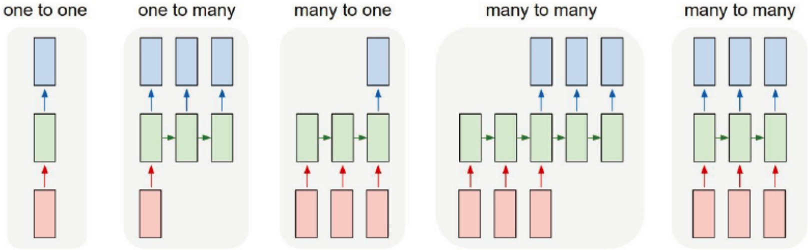
Vanilla Neural Network

fixed-sized input -> fixed-size output

e.g. image classification

Adapted from: *A. Karpathy*

Recurrent Networks Offer a Lot of Flexibility



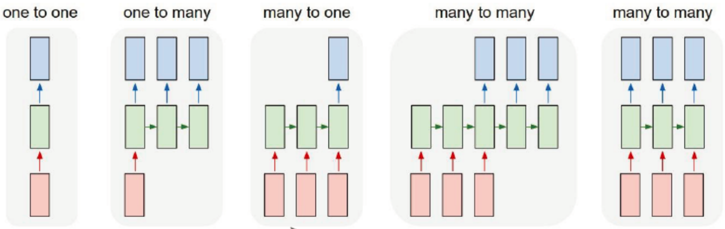
sequence output

e.g. image captioning

image -> sequence of words

Adapted from: *A. Karpathy*

Recurrent Networks Offer a Lot of Flexibility



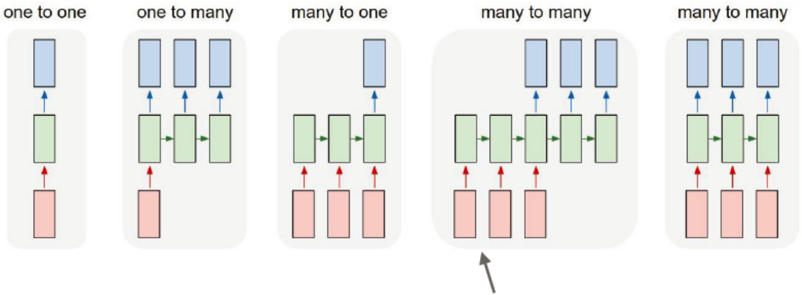
sequence input

e.g. sentiment classification

sequence of words -> sentiment

Adapted from: *A. Karpathy*

Recurrent Networks Offer a Lot of Flexibility

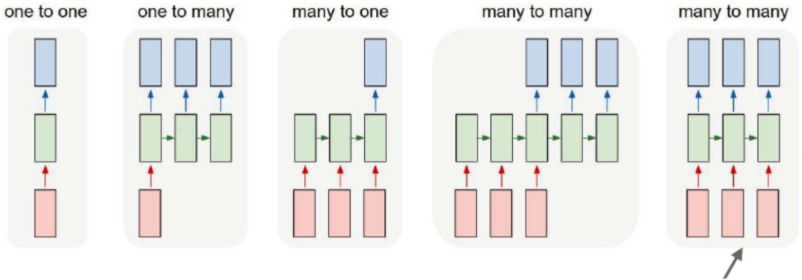


sequence input and sequence output

e.g. machine translation

seq of words -> seq of words

Recurrent Networks Offer a Lot of Flexibility



synced sequence input and output

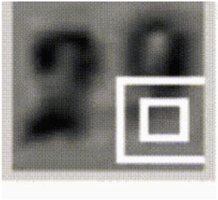
e.g. video classification on frame level

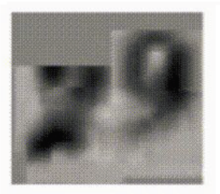
Adapted from: *A. Karpathy*

Sequential Processing in Absence of Sequences

- Even if inputs/outputs are fixed vectors, it is still possible to use RNNs to process them in a sequential manner.

Sequential Processing of fixed inputs





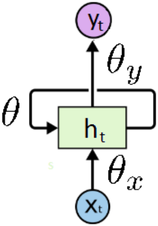
Multiple Object Recognition with Visual Attention, Ba et al.

Adapted from: *Ba et al. 2014*

Recurrent Neural Networks

$$\mathbf{h}_t = \theta\phi(\mathbf{h}_{t-1}) + \theta_x \mathbf{x}_t$$

$$\mathbf{y}_t = \theta_y \phi(\mathbf{h}_t)$$

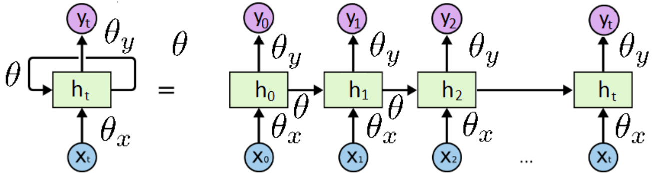


- $\mathbf{x}_t$ is the **input** at time t.
- $\mathbf{h}_t$ is the **hidden state** (memory) at time t.
- $\mathbf{y}_t$ is the **output** at time t.
- θ, θ_x, θ_y are distinct **weights**.
  - weights are the same at all time steps.

Adapted from: *C. Olah*

Recurrent Neural Networks

- RNNs can be thought of as multiple copies of the same network, each passing a message to a successor.

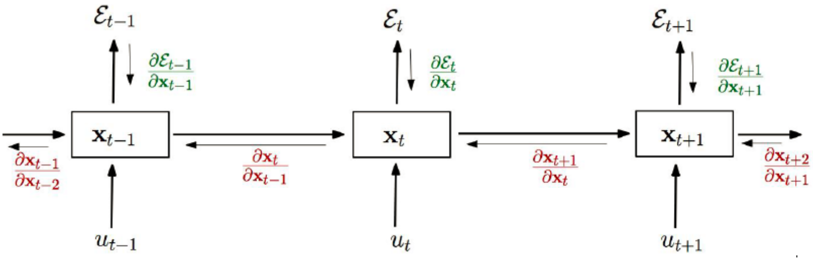


- The same function and the same set of parameters are used at every time step.
  - Are called recurrent because they perform the same task for each input.

Adapted from: *C. Olah*

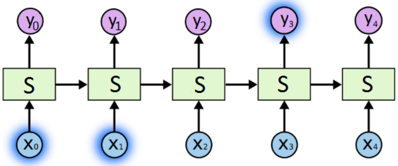
Back-Propagation Through Time (BPTT)

- Using the generalized back-propagation algorithm one can obtain the so-called **Back-Propagation Through Time** algorithm.
- The recurrent model is represented as a multi-layer one (with an unbounded number of layers) and backpropagation is applied on the unrolled model.

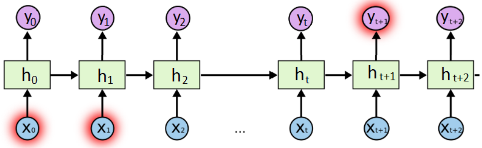


The Problem of Long-term Dependencies

- RNNs connect previous information to present task:
  - may be enough for predicting the next word for ”the clouds are in the sky”



  - may not be enough when more context is needed: ”I grew up in France ... I speak fluent French”



Adapted from: *C. Olah*

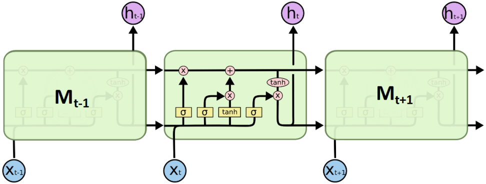
Vanishing/Exploding Gradients

- In RNNs, during the gradient back propagation phase, the gradient signal can end up being multiplied many times.
- If the gradients are large
  - Exploding gradients, learning diverges
  - Solution: clip the gradients to a certain max value.
- If the gradients are small
  - Vanishing gradients, learning very slow or stops
  - Solution: introducing memory via LSTM, GRU, etc.

Adapted from: *N. Cinbis*

Long Short-Term Memory Networks

- **Long Short-Term Memory (LSTM) networks** are RNNs capable of learning **long-term dependencies** [Hochreiter and Schmidhuber, 1997].



- A **memory** cell using logistic and linear units with multiplicative interactions:
  - Information **gets** into the cell whenever its **input gate** is on.
  - Information is **thrown away** from the cell whenever its **forget gate** is off.
  - Information can be **read** from the cell by turning on its **output gate**.

Adapted from: *C. Olah*

LSTM Overview

- We define the LSTM unit at each time step t to be a collection of vectors in $\mathbb{R}^d$:
  - **Memory cell** $\mathbf{c}_t$

    $\widetilde{\mathbf{c}}_t = \text{Tanh}(W_c.[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_c)$ **vector of new candidate values**

    $$\mathbf{c}_t = \mathbf{f}_t * \mathbf{c}_{t-1} + \mathbf{i}_t * \widetilde{\mathbf{c}}_t$$

  - **Forget gate** $\mathbf{f}_t$ in [0, 1]: scales old memory cell value (**reset**)

    $$\mathbf{f}_t = \sigma(W_f.[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_f)$$

  - **Input gate** $\mathbf{i}_t$ in [0, 1]: scales input to memory cell (**write**)

    $$\mathbf{i}_t = \sigma(W_i.[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i)$$

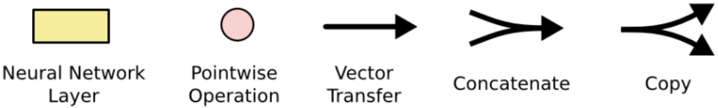
  - **Output gate** $\mathbf{o}_t$ in [0, 1]: scales output from memory cell (**read**)

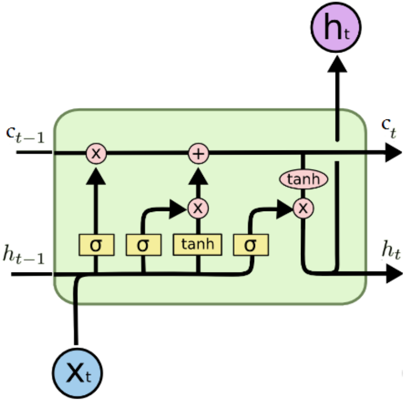
    $$\mathbf{o}_t = \sigma(W_o.[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o)$$

  - **Output** $\mathbf{h}_t$

    $$\mathbf{h}_t = \mathbf{o}_t * \text{Tanh}(\mathbf{c}_t)$$

Notation

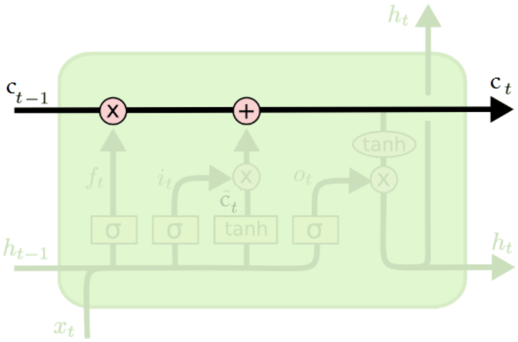




Adapted from: *C. Olah*

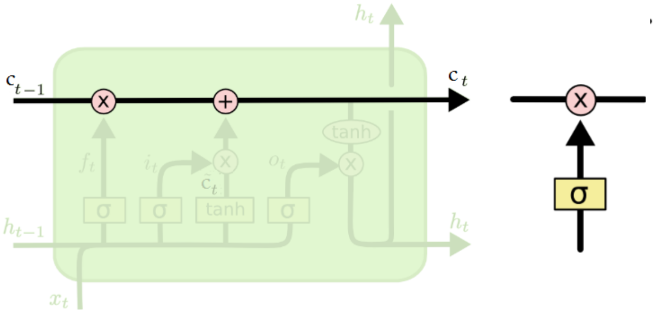
The Core Idea Behind LSTMs: Cell State (Memory Cell)

- Information can flow along the **memory cell unchanged**.
- Information can be **removed** or **written** to the **memory cell**, regulated by gates.



Adapted from: *C. Olah*

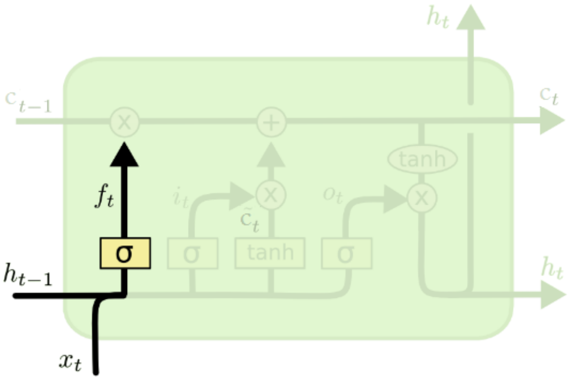
- **Gates** are a way to optionally let information through.
  - A **sigmoid layer** outputs number between 0 and 1, **deciding** how much of each component should be let through.
  - A pointwise multiplication operation applies the decision.



Adapted from: *C. Olah*

Forget Gate

- A **sigmoid** layer, **forget gate**, **decides** which values of the **memory cell** to **reset**.

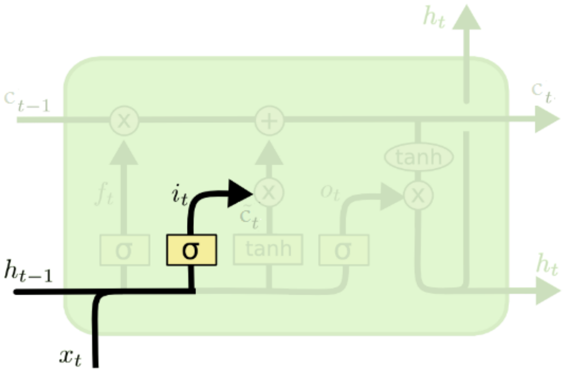


$$\mathbf{f}_t = \sigma(W_f.[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_f)$$

Adapted from: *C. Olah*

Input Gate

- A **sigmoid** layer, **input gate**, **decides** which values of the **memory cell** to **write** to.

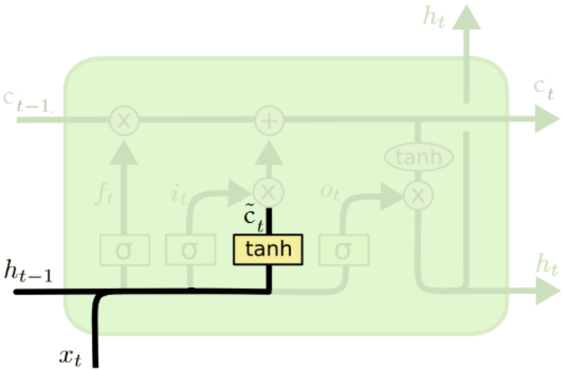


$$\mathbf{i}_t = \sigma(W_i.[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i)$$

Adapted from: *C. Olah*

Vector of New Candidate Values

- A **Tanh** layer creates a **vector of new candidate values** $\widetilde{\mathbf{c}}_t$ to **write** to the **memory cell**.

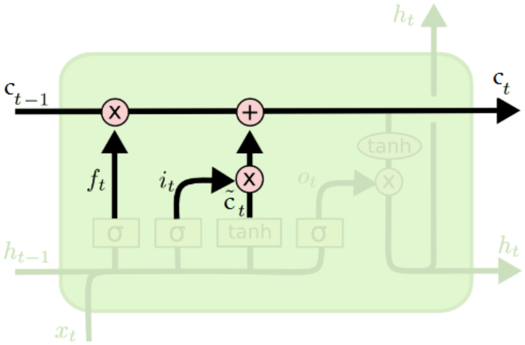


$$\widetilde{\mathbf{c}}_t = \text{Tanh}(W_c.[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_c)$$

Adapted from: *C. Olah*

Memory Cell Update

- The previous steps decided which values of the **memory cell** to **reset** and **overwrite**.
- Now the LSTM **applies the decisions** to the **memory cell**.

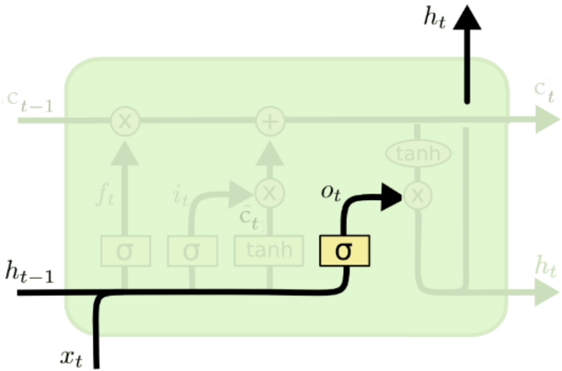


$$\mathbf{c}_t = \mathbf{f}_t * \mathbf{c}_{t-1} + \mathbf{i}_t * \widetilde{\mathbf{c}_t}$$

Adapted from: *C. Olah*

Output Gate

- A **sigmoid** layer, **output gate**, **decides** which values of the **memory cell** to **output**.

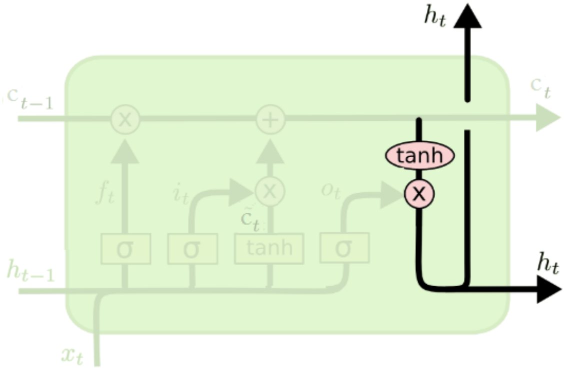


$$\mathbf{o}_t = \sigma(W_o.[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o)$$

Adapted from: *C. Olah*

Output Update

- The **memory cell** goes through **Tanh** and is multiplied by the **output gate**.



$$\mathbf{h}_t = \mathbf{o}_t * \text{Tanh}(\mathbf{c}_t)$$

Adapted from: *C. Olah*

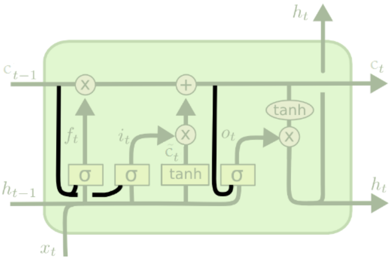
Variants on LSTM

- Gate layers look at the memory cell [Gers and Schmidhuber, 2000].

$$\mathbf{f}_t = \sigma(W_f.[\mathbf{c}_{t-1}, \mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_f)$$

$$\mathbf{i}_t = \sigma(W_i.[\mathbf{c}_{t-1}, \mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i)$$

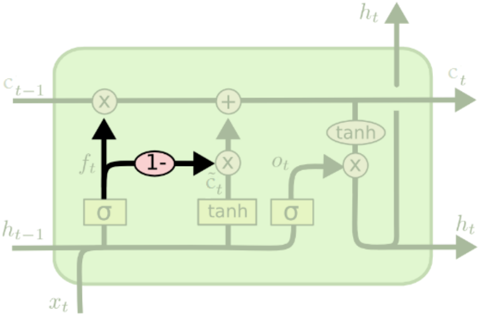
$$\mathbf{o}_t = \sigma(W_o.[\mathbf{c}_{t-1}, \mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o)$$



Adapted from: *C. Olah*

Variants on LSTM

- Use coupled **forget** and **input** gates. Instead of separately deciding what to **forget** and what to **add**, make those decisions together.



$$\mathbf{c}_t = \mathbf{f}_t * \mathbf{c}_{t-1} + (1 - \mathbf{f}_t) * \widetilde{\mathbf{c}}_t$$

Adapted from: *C. Olah*

Variants on LSTM

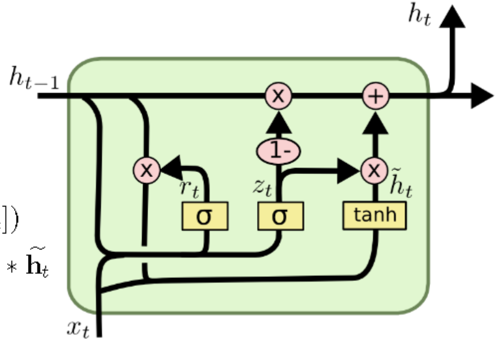
- Gated Recurrent Unit (GRU) [Cho et al., 2014]:
  - Combine the **forget** and **input** gates into a single **update** gate.
  - **Merge the memory cell and the hidden state**.
  - ...

$$z_t = \sigma(W_z.[\mathbf{h}_{t-1}, \mathbf{x}_t])$$

$$r_t = \sigma(W_r.[\mathbf{h}_{t-1}, \mathbf{x}_t])$$

$$\widetilde{\mathbf{h}}_t = \text{Tanh}(W.[r_t * \mathbf{h}_{t-1}, \mathbf{x}_t])$$

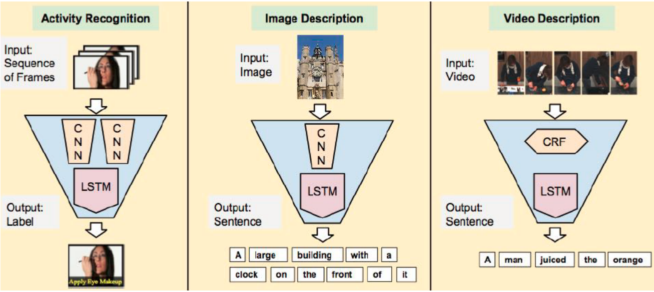
$$\mathbf{h}_t = (1 - z_t) * \mathbf{h}_{t-1} + (z_t) * \widetilde{\mathbf{h}}_t$$



Adapted from: *C. Olah*

Applications

- Cursive handwriting recognition
  - https://www.youtube.com/watch?v=mLxsbWAYIpw
- Translation
  - Translate any signal to another signal, e.g., translate English to French, translate image to image caption, and songs to lyrics.
- Visual sequence tasks



Adapted from: *Jeff Donahue et al. CVPR15*