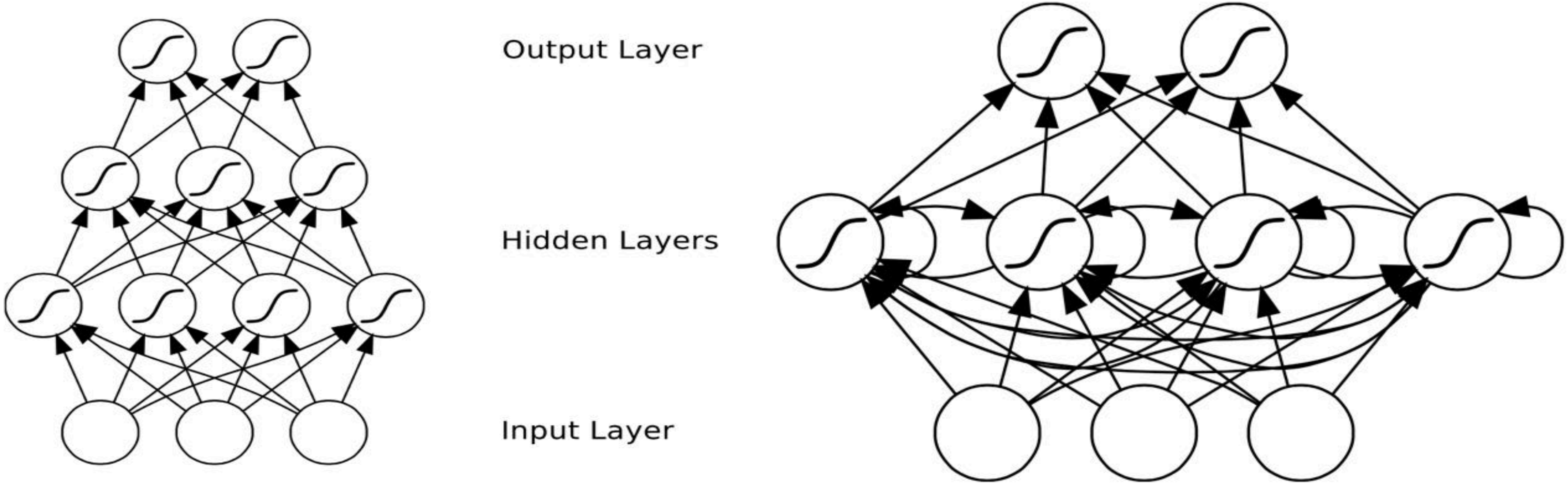
Recurrent Neural Networks

Recurrent Neural Networks

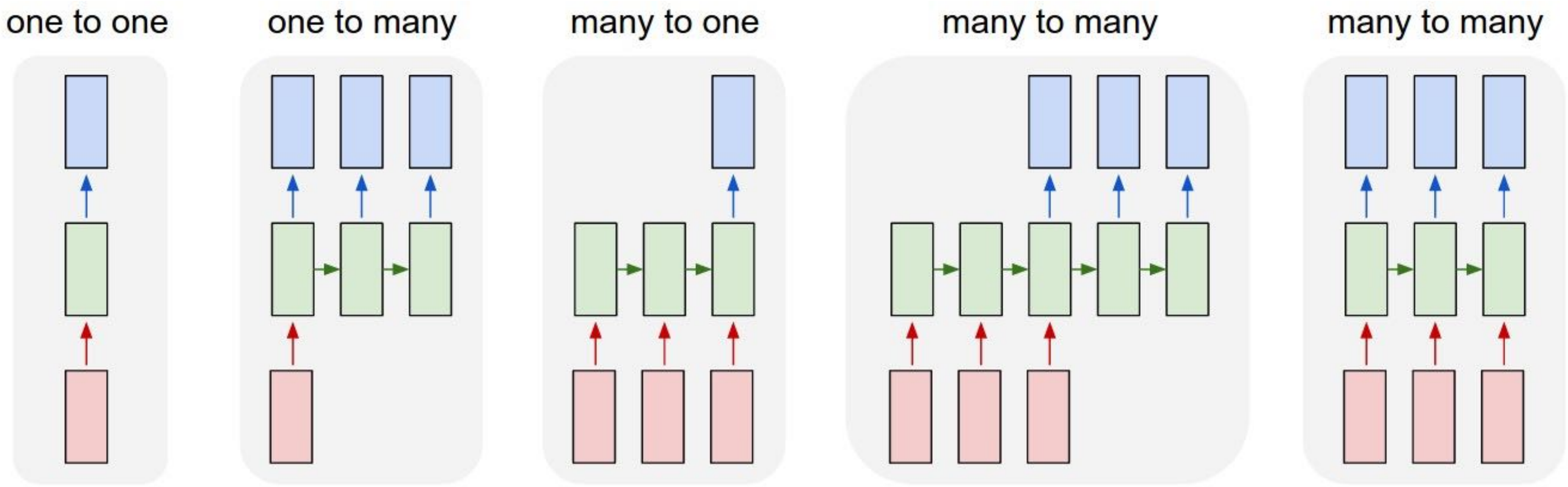
- An MLP can only map from input to output vectors, whereas an RNN can, in principle, map from the entire history of previous inputs to each output.



Multi-layer Perceptron

Recurrent Network

Recurrent Networks offer a lot of flexibility

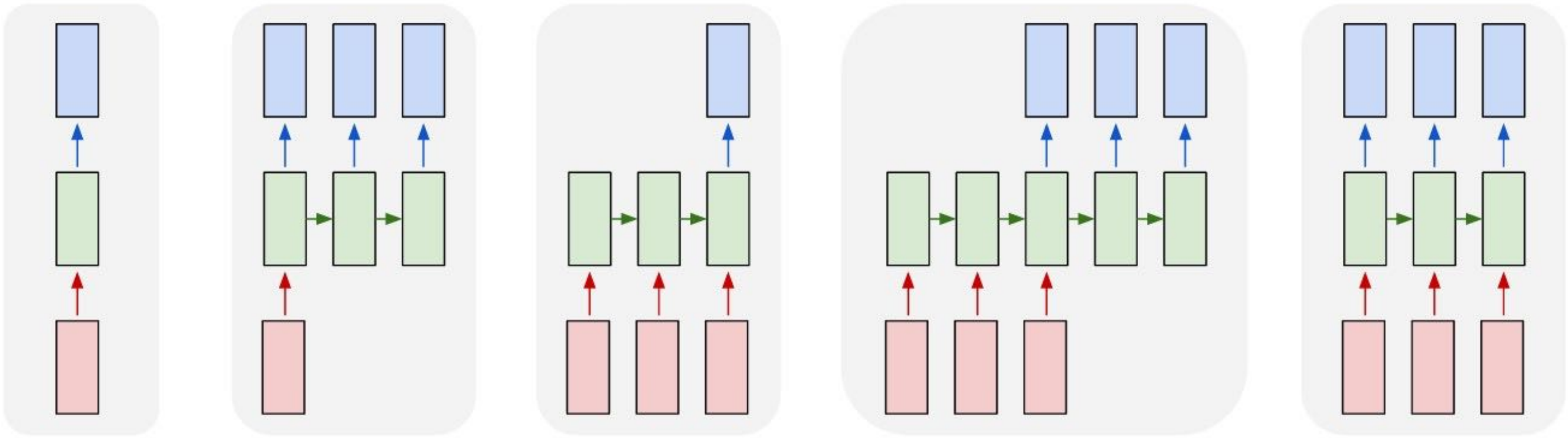


Vanilla Neural Networks

Slide credit: Andrej Karpathy

Recurrent Networks offer a lot of flexibility

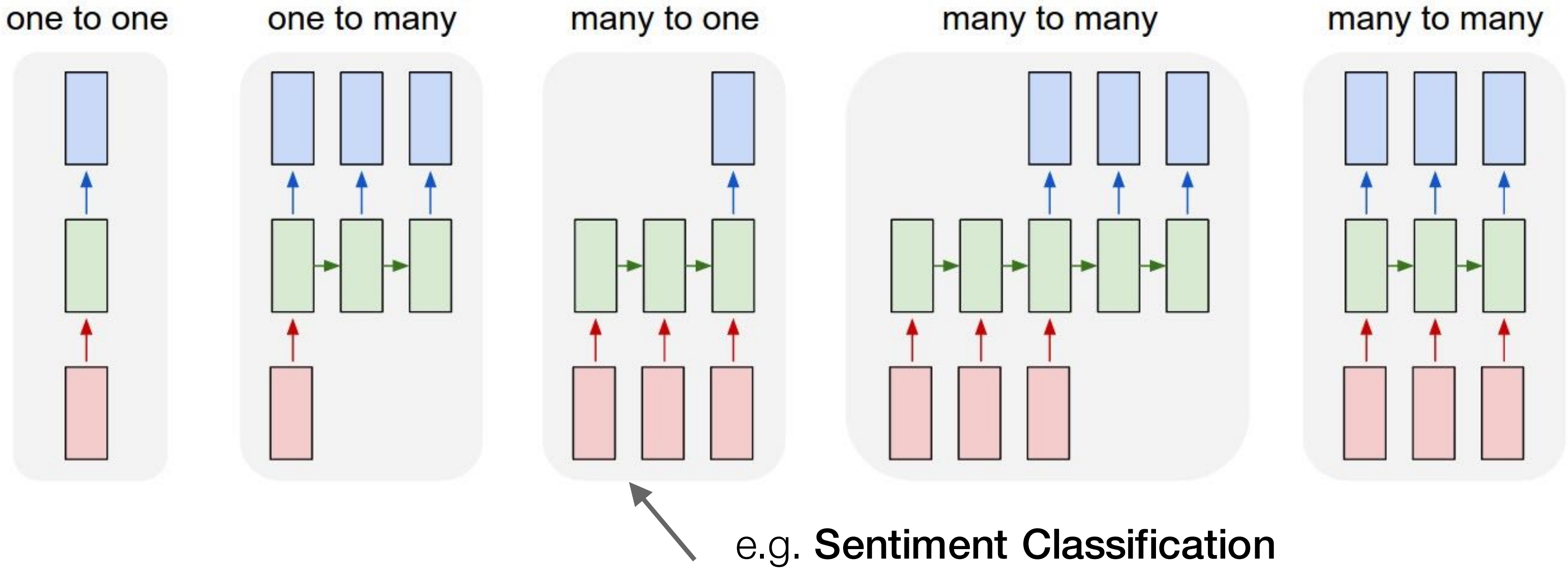
one to one | one to many | many to one | many to many | many to many



e.g. **Image Captioning**
image -> sequence of words

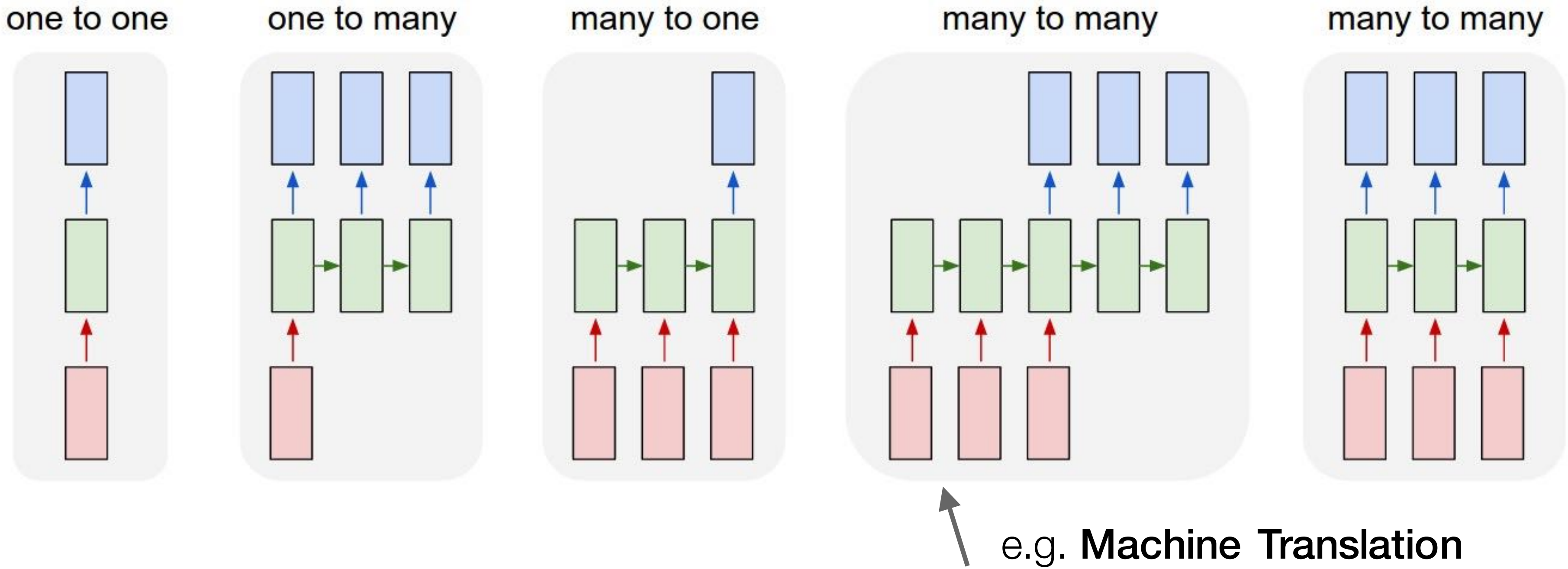
Slide credit: Andrej Karpathy

Recurrent Networks offer a lot of flexibility



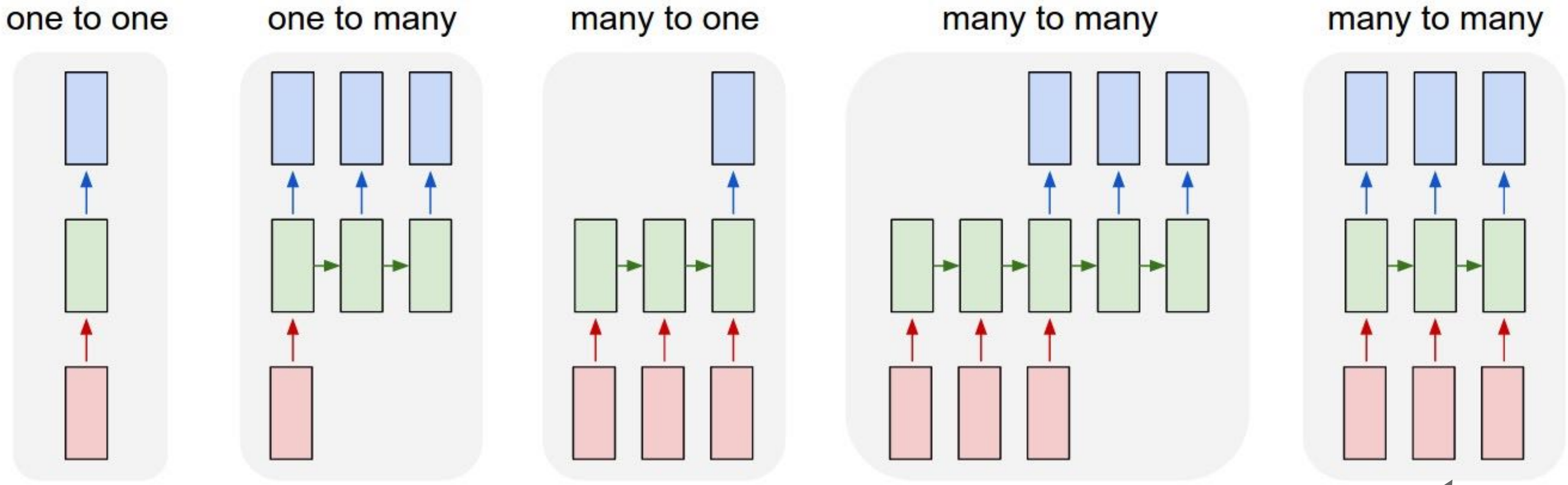
e.g. **Sentiment Classification**
sequence of words -> sentiment

Recurrent Networks offer a lot of flexibility



e.g. **Machine Translation**
seq of words -> seq of words

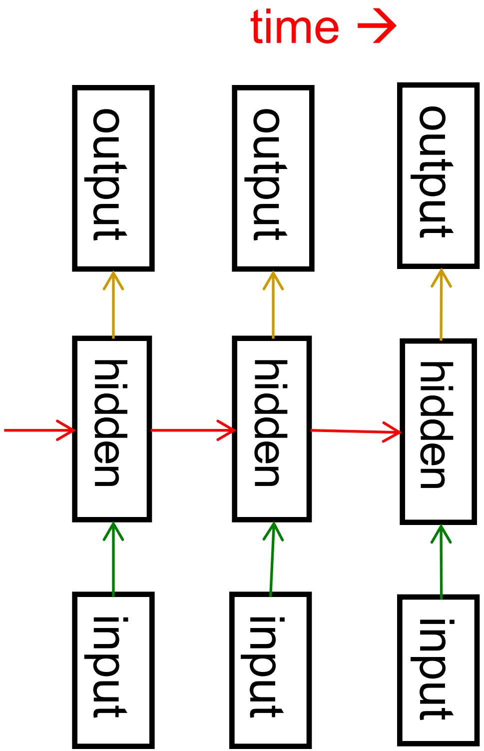
Recurrent Networks offer a lot of flexibility



e.g. **Video classification on frame level**

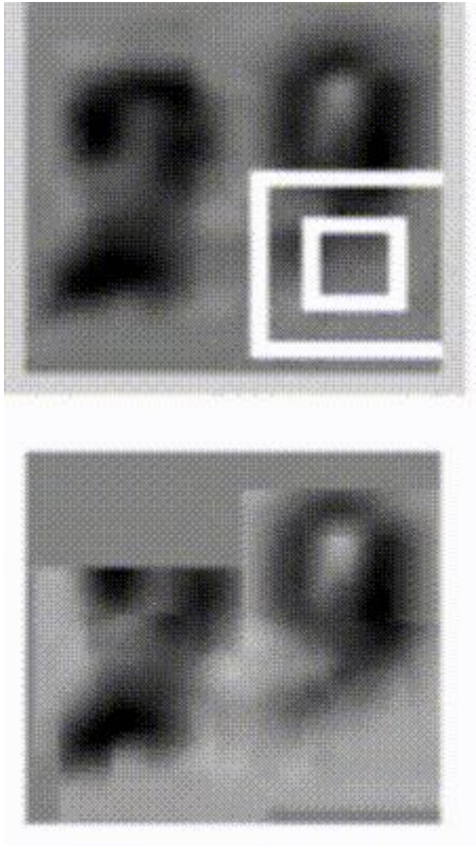
Recurrent neural networks

- RNNs are very powerful, because they combine two properties:
  - Distributed hidden state that allows them to store a lot of information about the past efficiently.
  - Non-linear dynamics that allows them to update their hidden state in complicated ways.
- *With enough neurons and time, RNNs can compute anything that can be computed by your computer.*



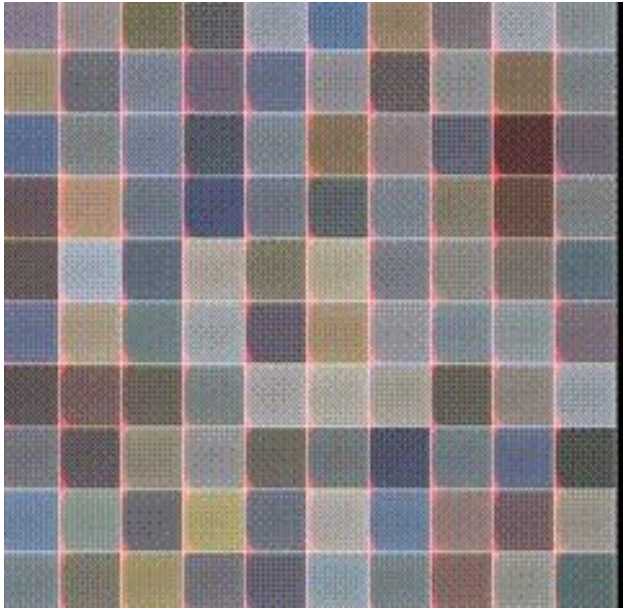
Slide credit: G. Hinton

Sequential Processing of fixed inputs



Multiple Object Recognition with Visual Attention, Ba et al.

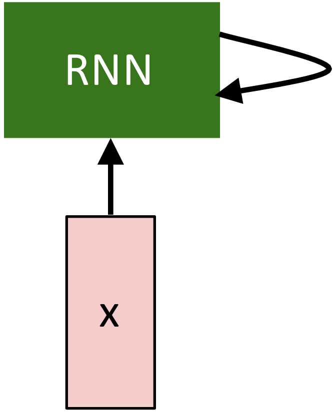
Sequential Processing of fixed outputs



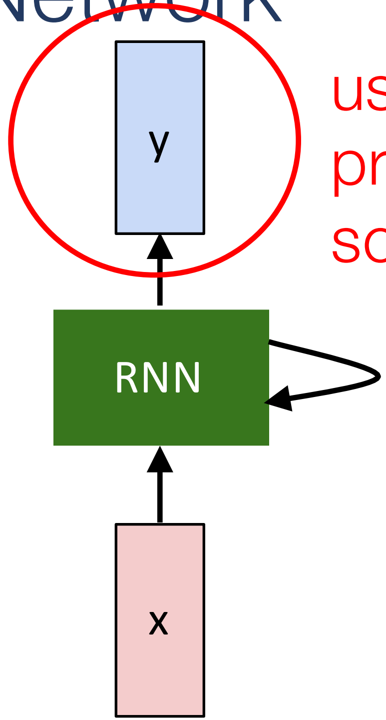
DRAW: A Recurrent Neural Network For Image Generation, Gregor et al.

Slide credit: Andrej Karpathy

Recurrent Neural Network

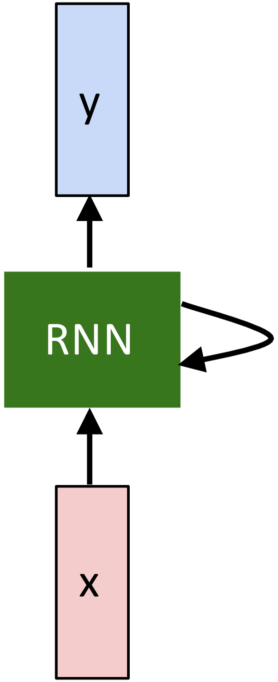


Recurrent Neural Network

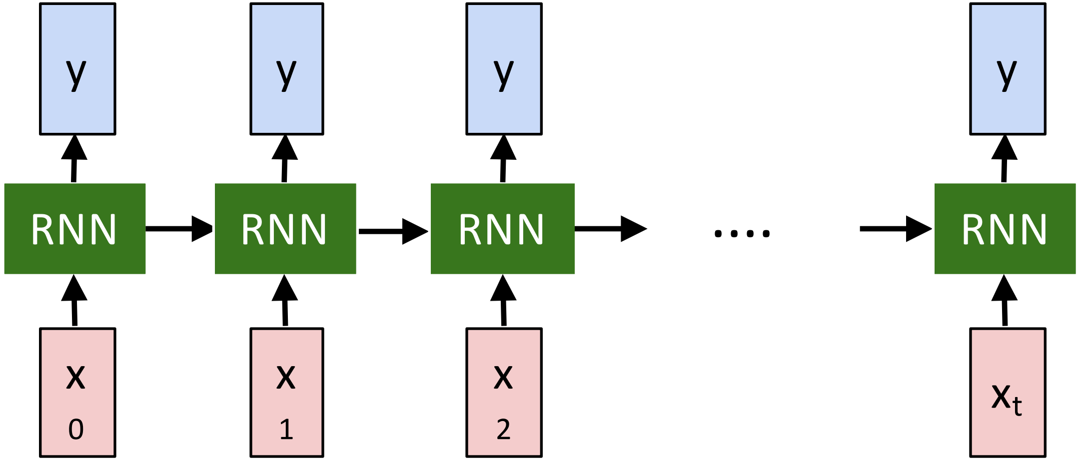


usually want to predict a vector at some time steps

Recurrent Neural Network



Consider what happens when we unroll the loop:



A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor.

Recurrent Neural Network

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

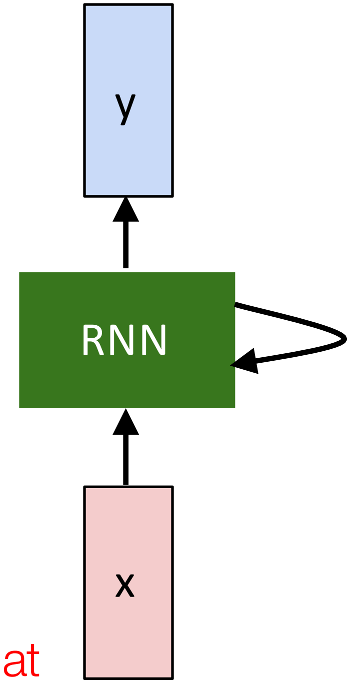
$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

new state

old state

input vector at some time step

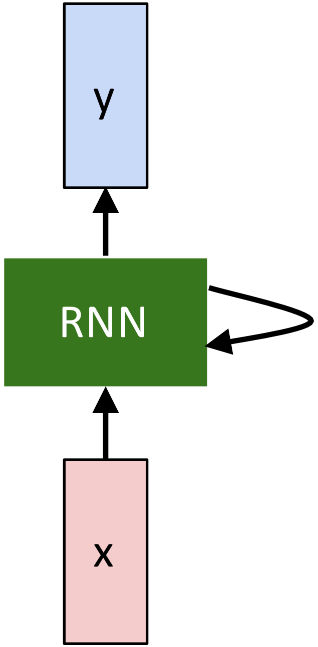
some function with parameters W



Important: the same function and the same set of parameters are used at every time step.

(Vanilla) Recurrent Neural Network

The state consists of a single *“hidden”* vector **h**:



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

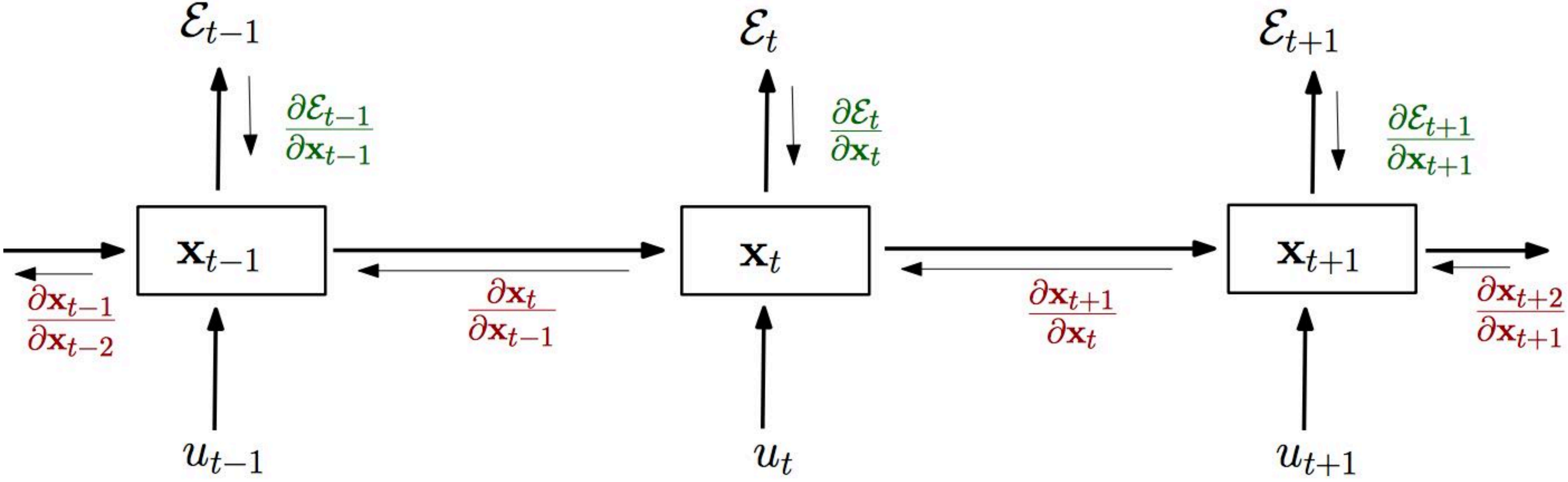
$$\downarrow$$

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

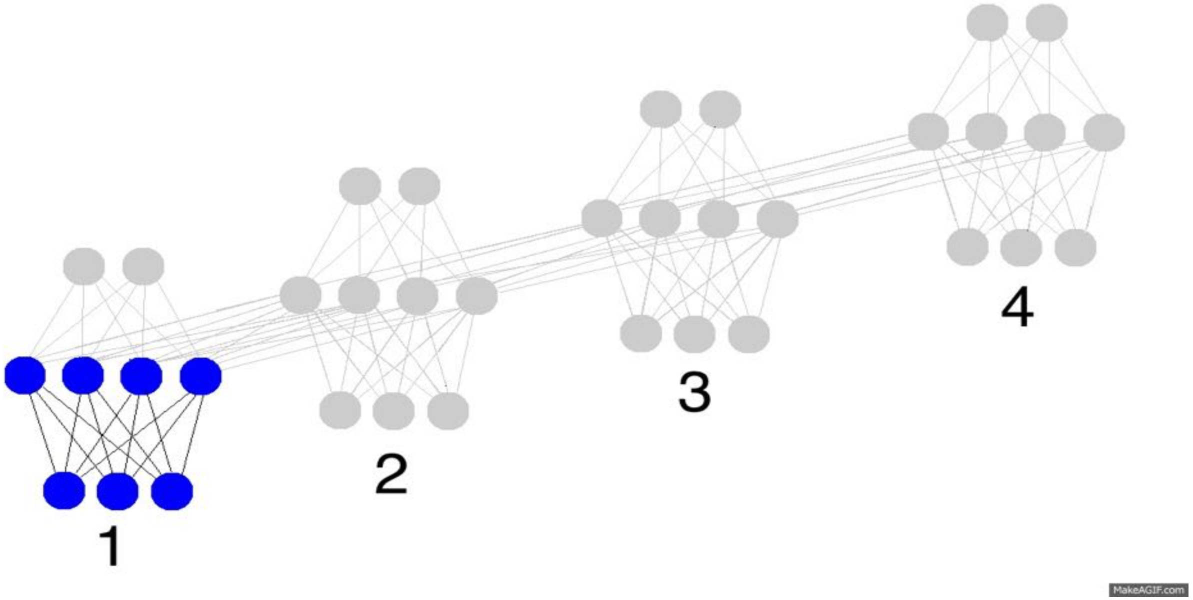
$$y_t = W_{hy}h_t$$

Backpropagation Through Time (BPTT)

- The recurrent model is represented as a multi-layer one (with an unbounded number of layers) and backpropagation is applied on the unrolled model

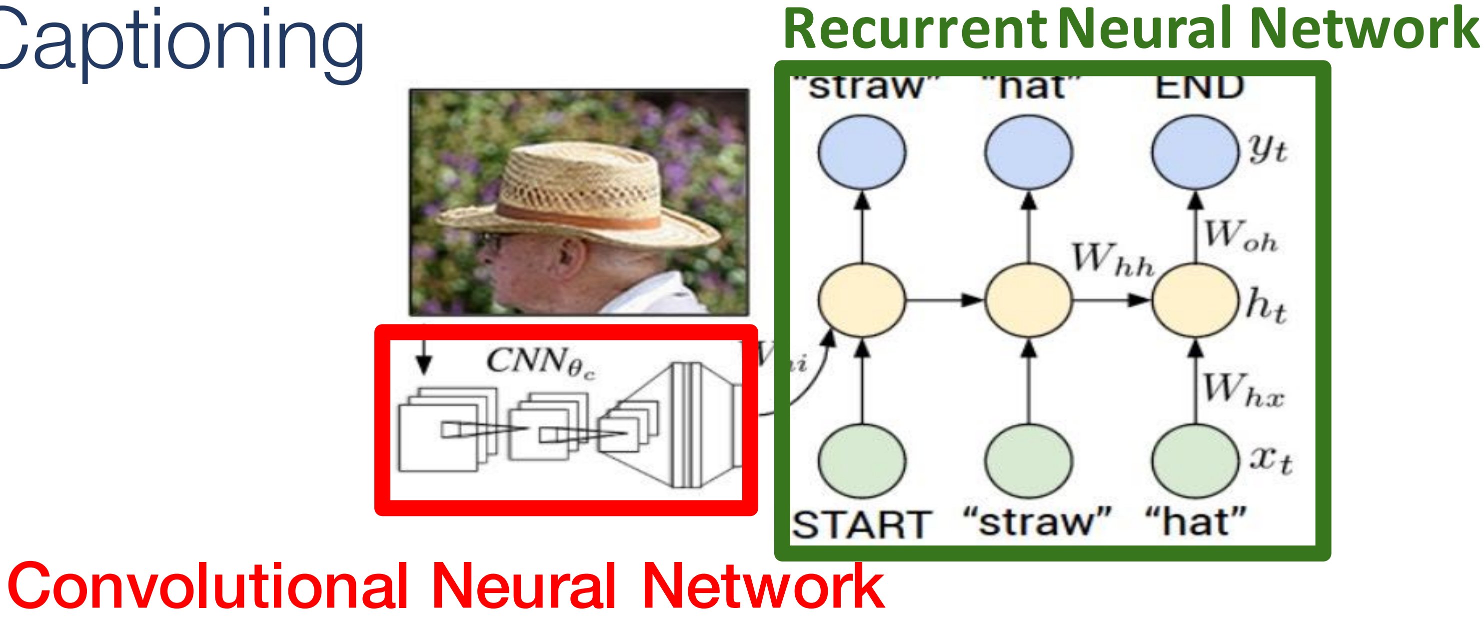


Backpropagation Through Time (BPTT)



Black is the prediction, **errors** are bright yellow, **derivatives** are mustard colored.

Image Captioning



- Explain Images with Multimodal Recurrent Neural Networks, Mao et al.
- Deep Visual-Semantic Alignments for Generating Image Descriptions, Karpathy and Fei-Fei
- Show and Tell: A Neural Image Caption Generator, Vinyals et al.
- Long-term Recurrent Convolutional Networks for Visual Recognition and Description, Donahue et al.
- Learning a Recurrent Visual Representation for Image Caption Generation, Chen and Zitnick

test image

Slide credit: Andrej Karpathy

image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
FC-1000
softmax

test image

Slide credit: Andrej Karpathy

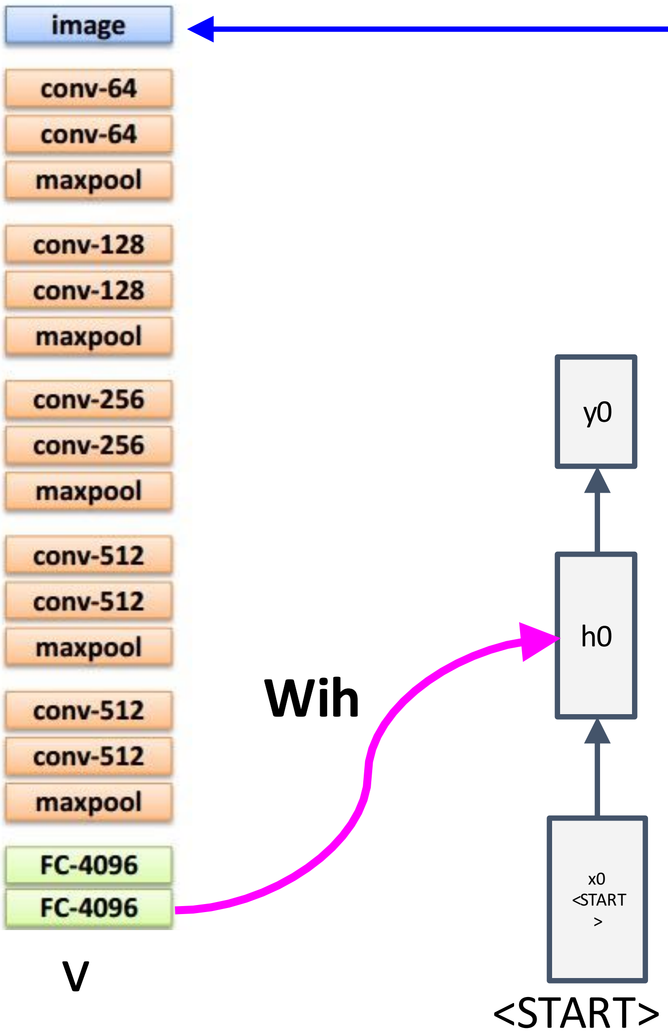
test image

Slide credit: Andrej Karpathy

test image

<START>

Slide credit: Andrej Karpathy



test image

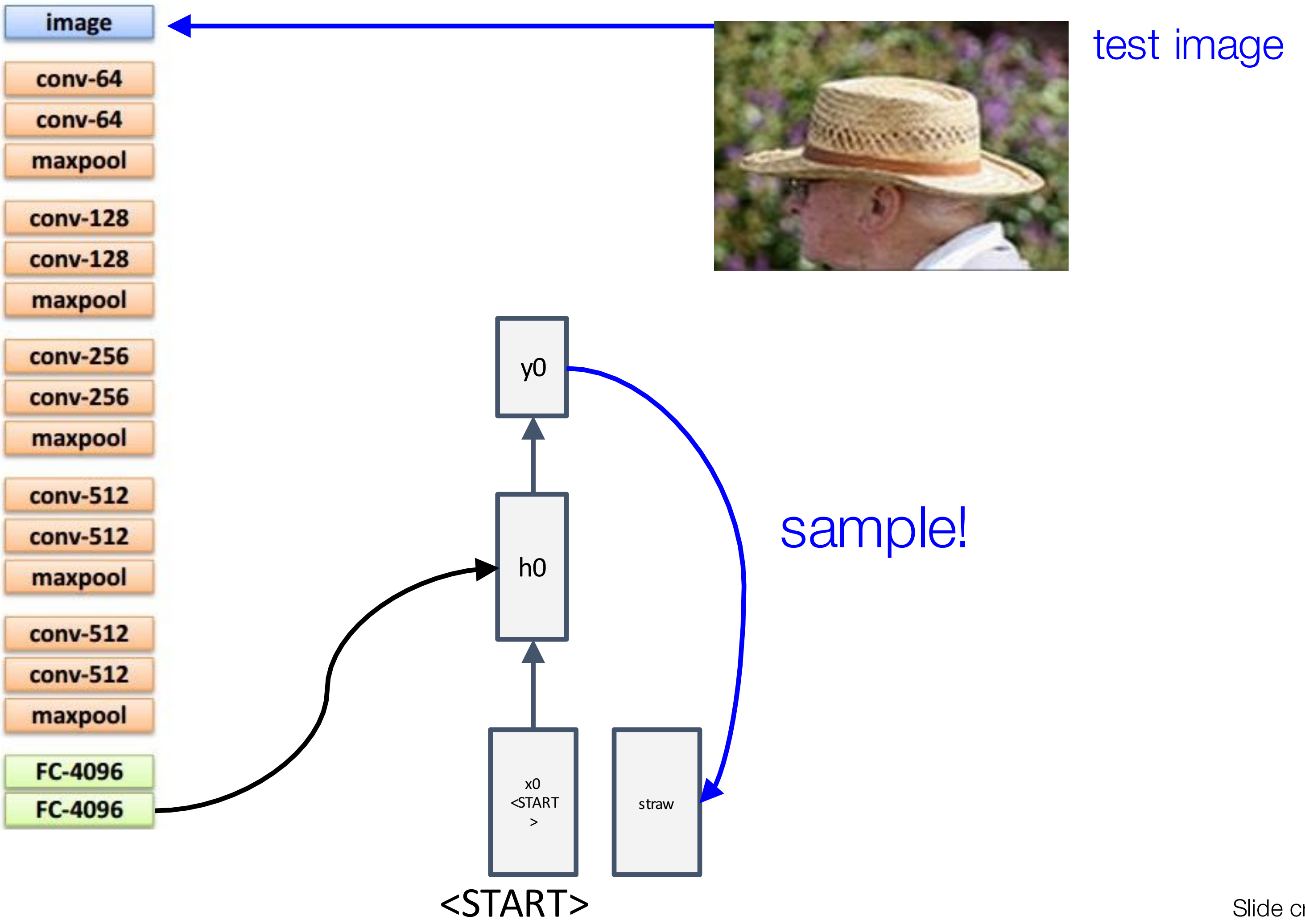
before:

h = tanh(Wxh * x + Whh * h)

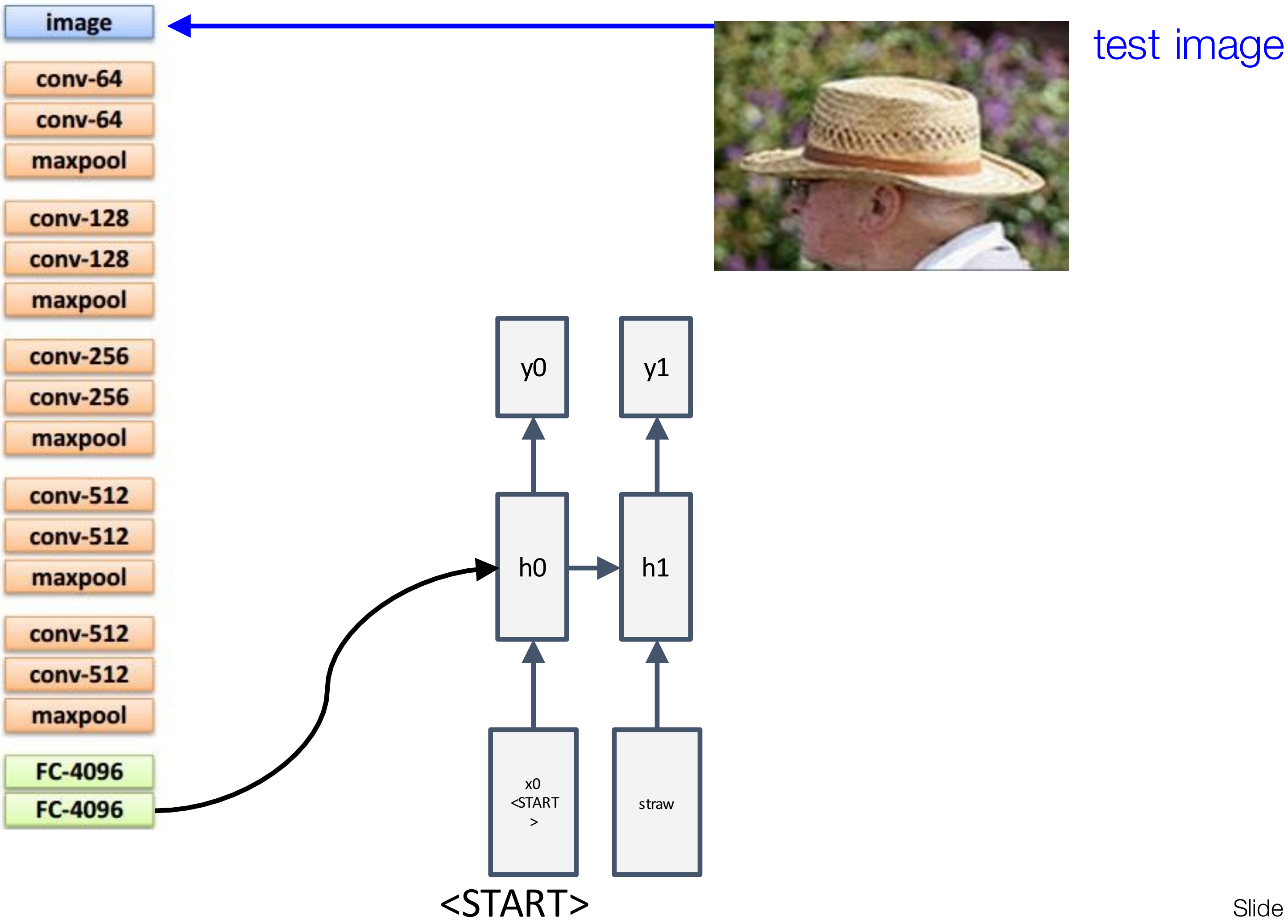
now:

h = tanh(Wxh * x + Whh * h + **Wih * v**)

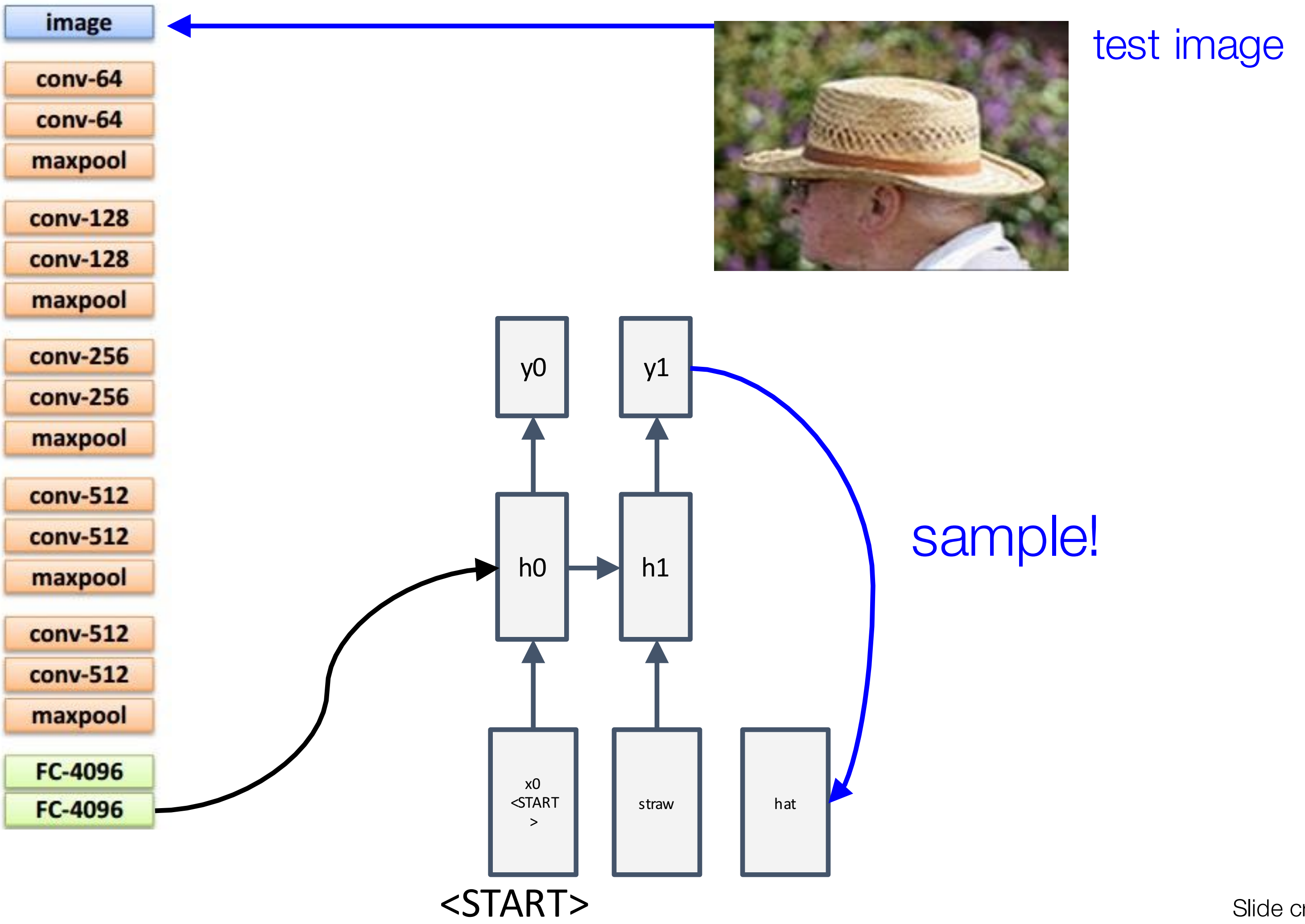
Slide credit: Andrej Karpathy



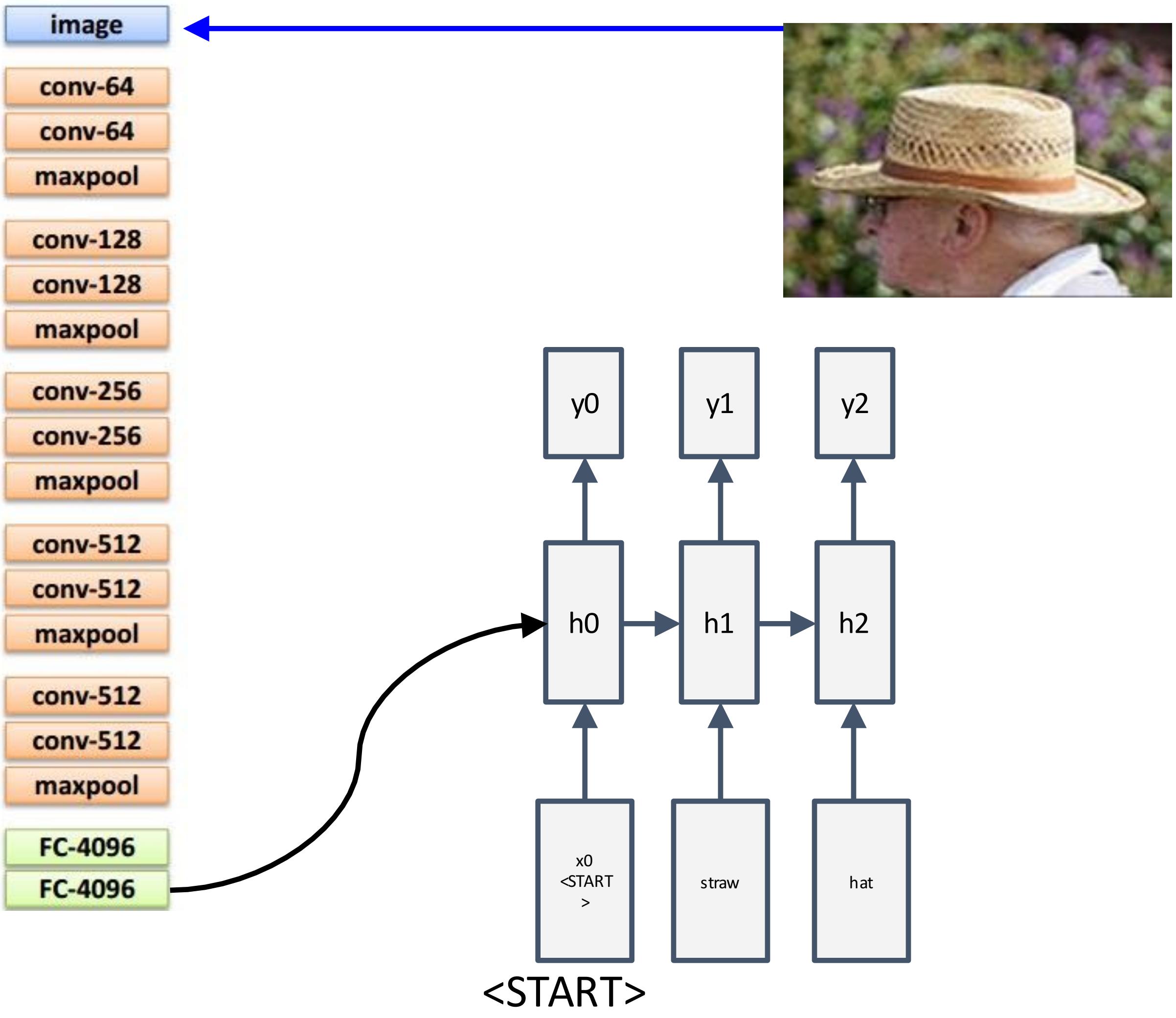
Slide credit: Andrej Karpathy



Slide credit: Andrej Karpathy

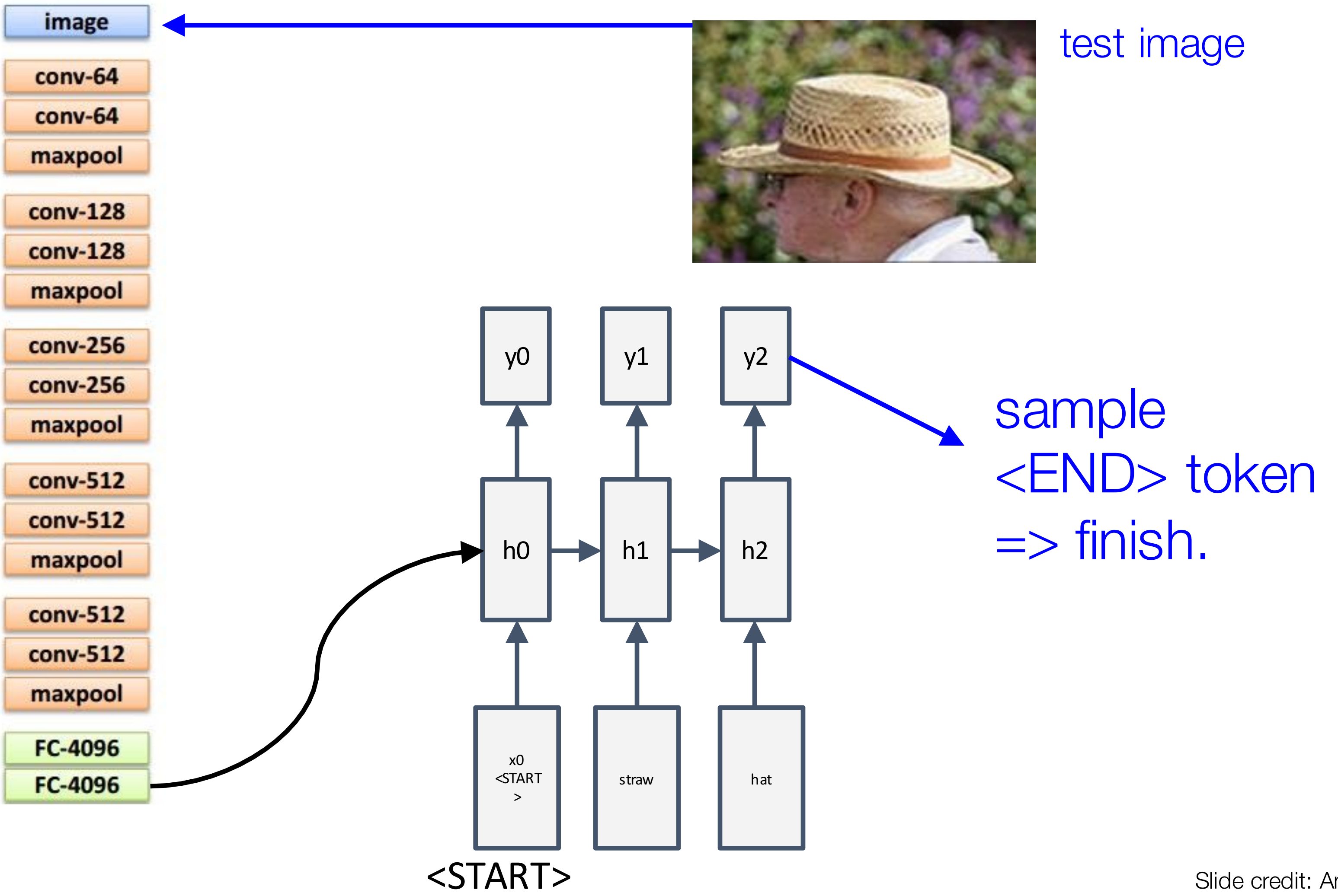


Slide credit: Andrej Karpathy



test image

Slide credit: Andrej Karpathy



Slide credit: Andrej Karpathy

"man in black shirt is playing guitar."

"construction worker in orange safety vest is working on road."

"two young girls are playing with lego toy."

"boy is doing backflip on wakeboard."

Slide credit: Andrej Karpathy

"man in black shirt is playing guitar."

"construction worker in orange safety vest is working on road."

"two young girls are playing with lego toy."

"boy is doing backflip on wakeboard."

"a young boy is holding a baseball bat."

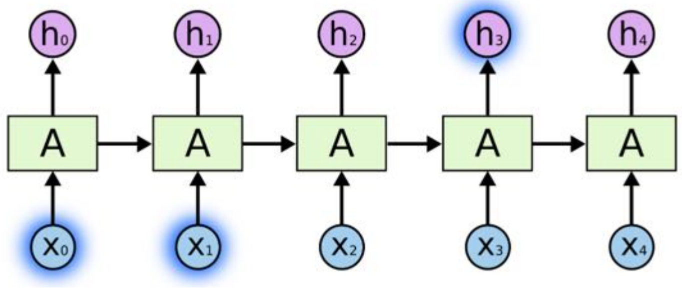
"a cat is sitting on a couch with a remote control."

"a woman holding a teddy bear in front of a mirror."

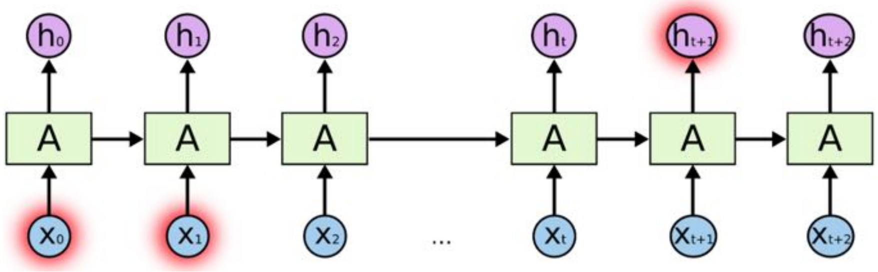
"a horse is standing in the middle of a road."

The problem of long-term dependencies

- (Vanilla) RNNs connect previous information to present task:
- - enough for predicting the next word for "the clouds are in the *sky*"



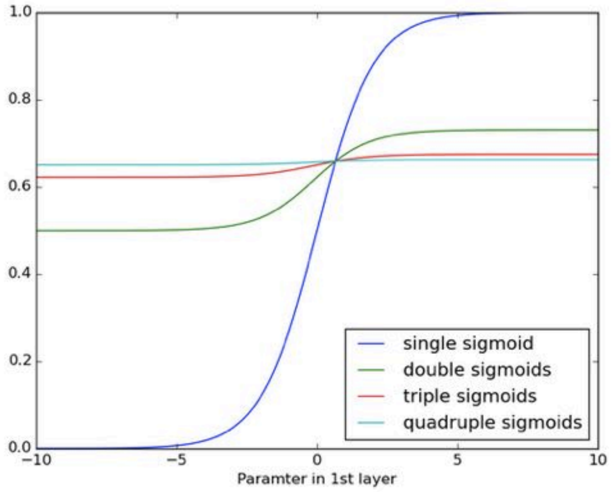
- - may not be enough when more context is needed
- "I grew up in France… I speak fluent *French*."



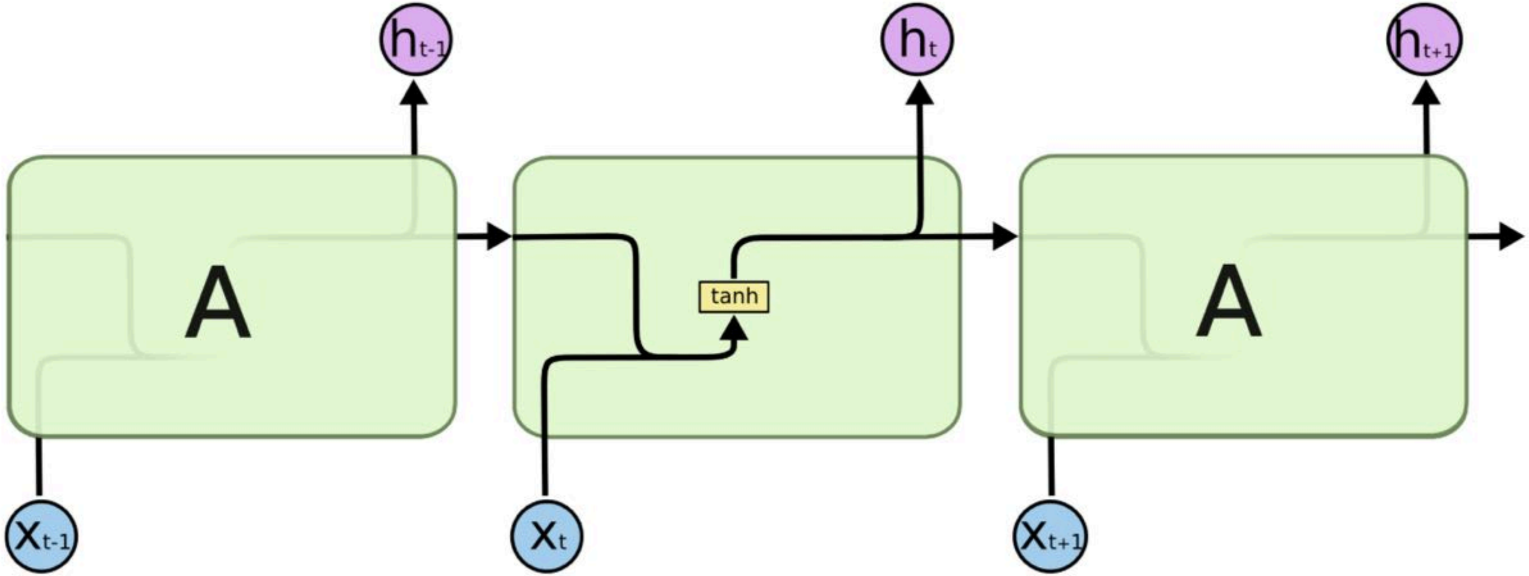
Adapted from: C. Olah

The problem of vanishing gradients

- In a traditional recurrent neural network, during the gradient backpropagation phase, the gradient signal can end up being multiplied a large number of 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.**



All recurrent neural networks have the form of a chain of repeating modules of neural network

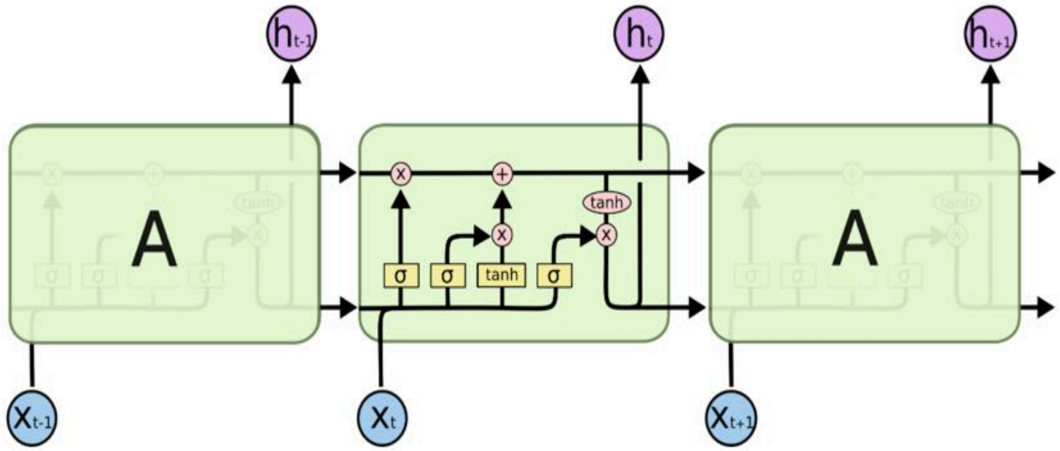


The repeating module in a standard RNN contains a single layer.

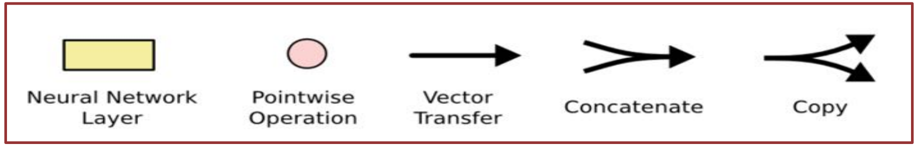
Adapted from: C. Olah

Long Short Term Memory (LSTM)[Hochreiter & Schmidhuber (1997)]

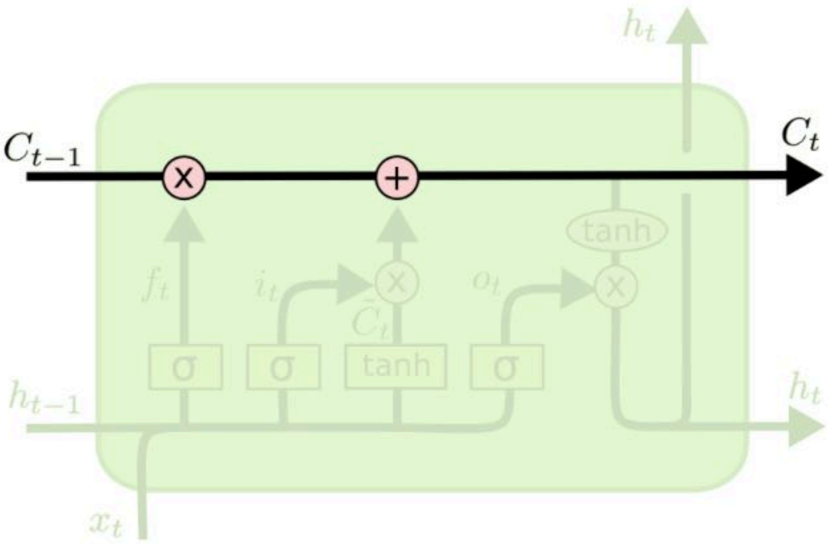
- A memory cell using logistic and linear units with multiplicative interactions:
- Information gets into the cell whenever its **input** gate is on.
- The information stays in the cell so long as its **forget** gate is on.
- Information can be read from the cell by turning on its **output** gate.



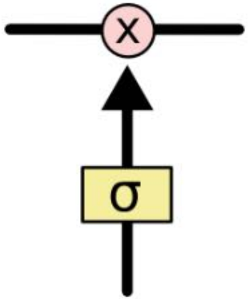
The repeating module in an LSTM contains four interacting layers.



The Core Idea Behind LSTMs : Cell State

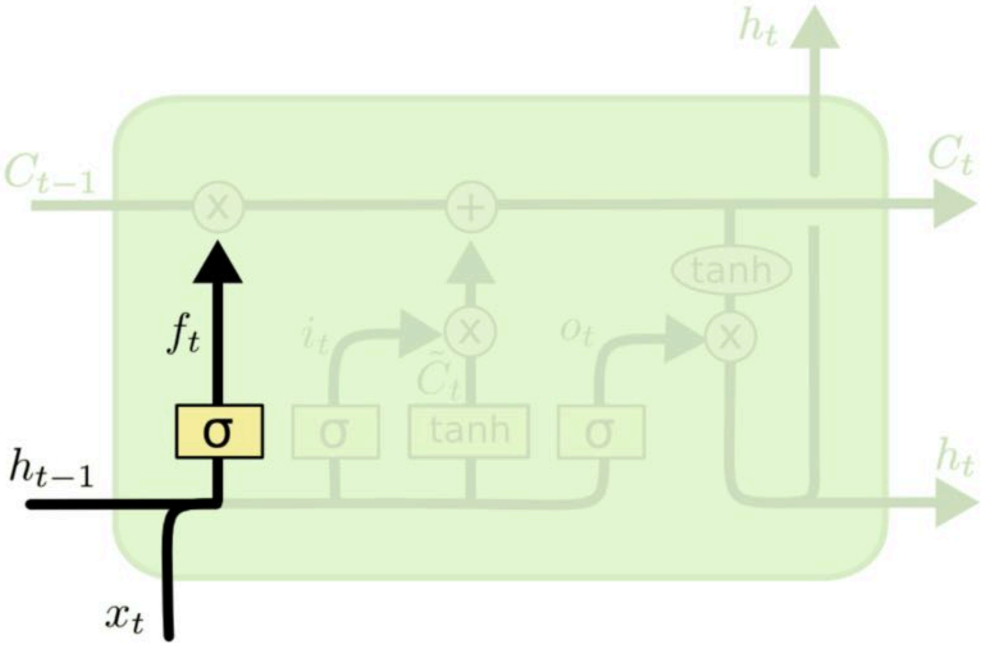


Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation.



An LSTM has three of these gates, to protect and control the cell state.

LSTM : Forget gate



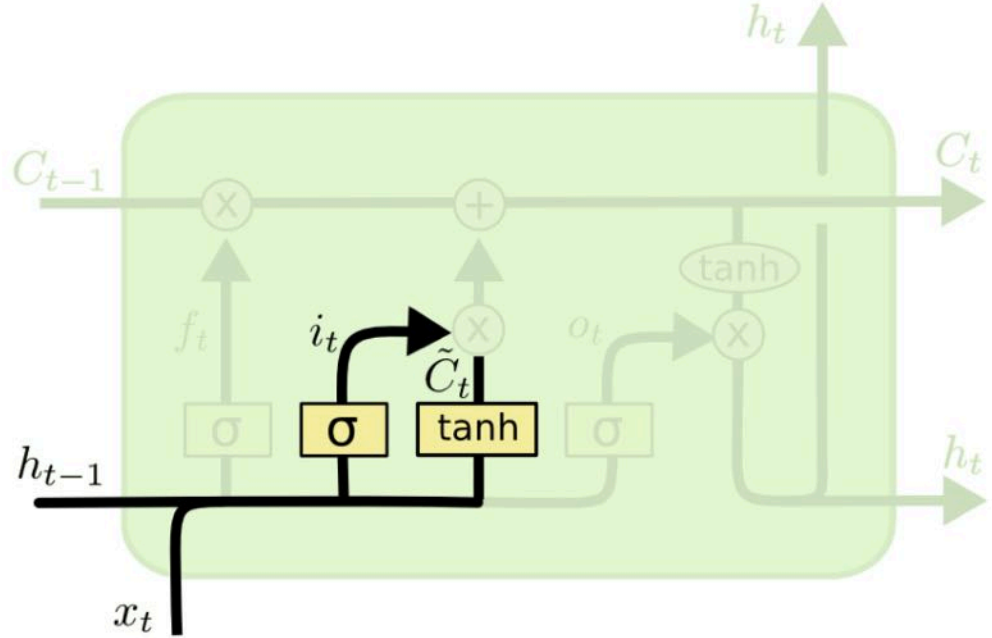
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

It looks at h_{t-1} and x_t and outputs a number between 0 and 1 for each number in the cell state C_{t-1}.

A 1 represents **completely keep this** while a 0 represents **completely get rid of this.**

LSTM : Input gate and Cell State

The next step is to decide what new information we're going to store in the cell state.



a sigmoid layer called the **input gate layer** decides which values we'll update.

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

a tanh layer creates a vector of new candidate values, that could be added to the state.

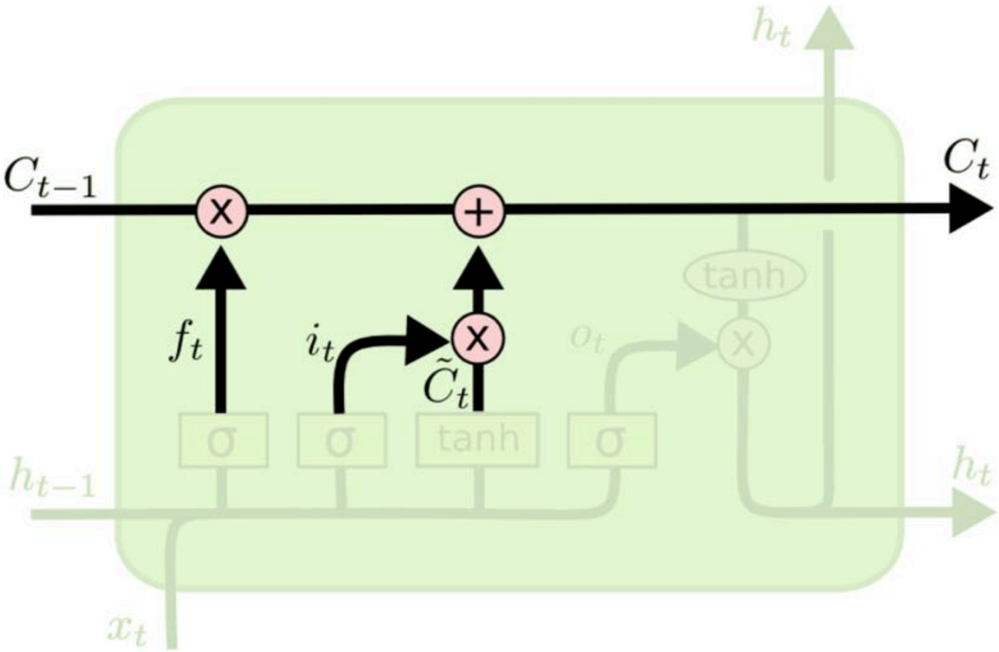
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

Adapted from: C. Olah

LSTM : Input gate and Cell State

It's now time to update the old cell state into the new cell state.

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

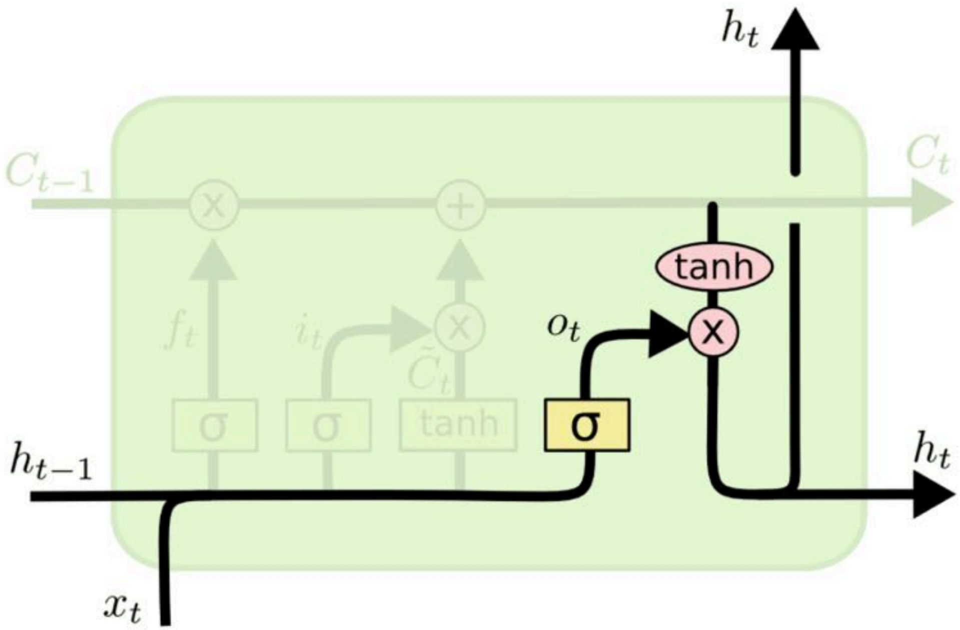


We multiply the old state by ft forgetting the things we decided to forget earlier.

Then, we add the new candidate values, scaled by how much we decided to update each state value.

LSTM : Output

Finally, we need to decide what we're going to output.



First, we run a sigmoid layer which decides what parts of the cell state we're going to output.

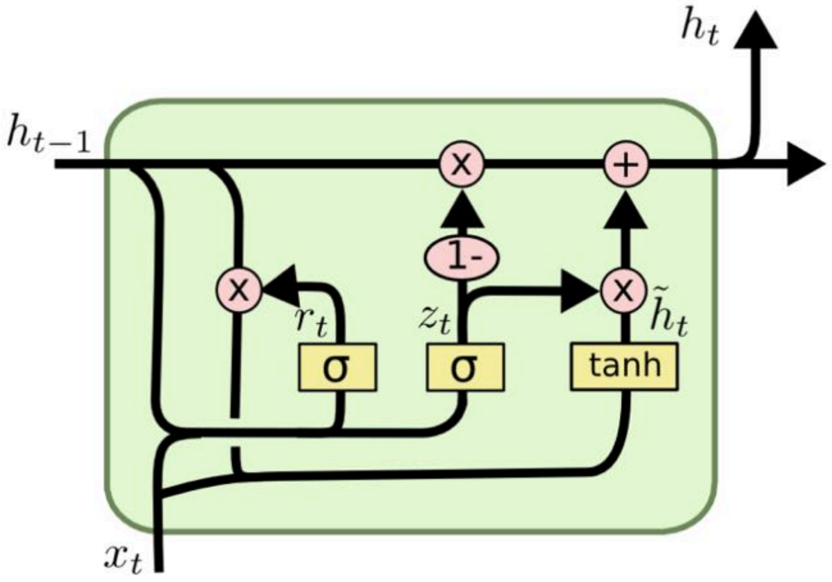
$$o_t = \sigma\left(W_o \left[h_{t-1}, x_t\right] + b_o\right)$$

Then, we put the cell state through tanh (to push the values to be between -1 and 1) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.

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

LSTM variants : Gated Recurrent Unit (GRU)

- Introduced by Cho et al. (2014) It combines the forget and input gates into a single "update gate." It also merges the cell state and hidden state, and



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

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

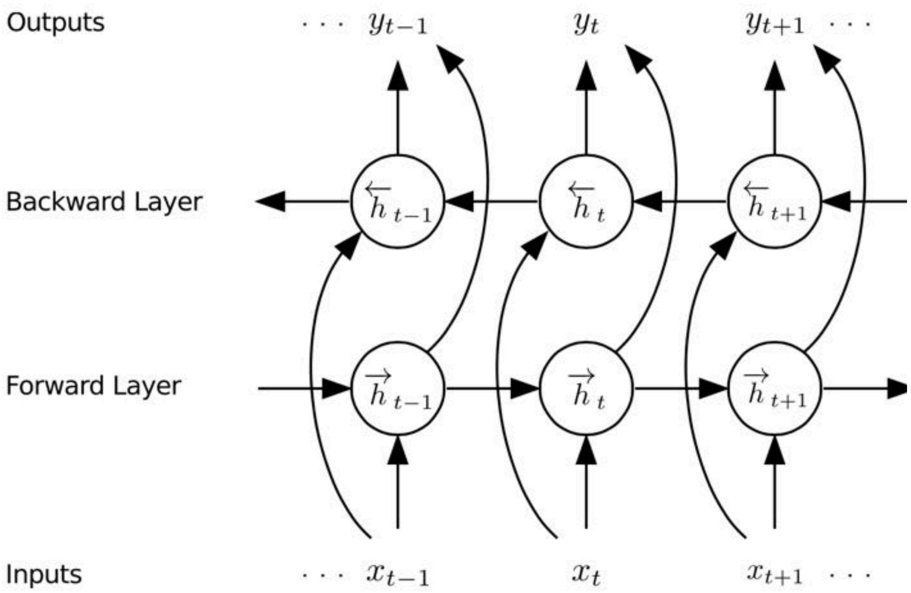
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

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

Bi-directional Recurrent Neural Networks (BRNN)

- BRNNs process the data in both directions with two separate hidden layers:

- *Forward hidden sequence*: iterates from t=T:1
- *Backward hidden sequence*: iterates from t=1:T



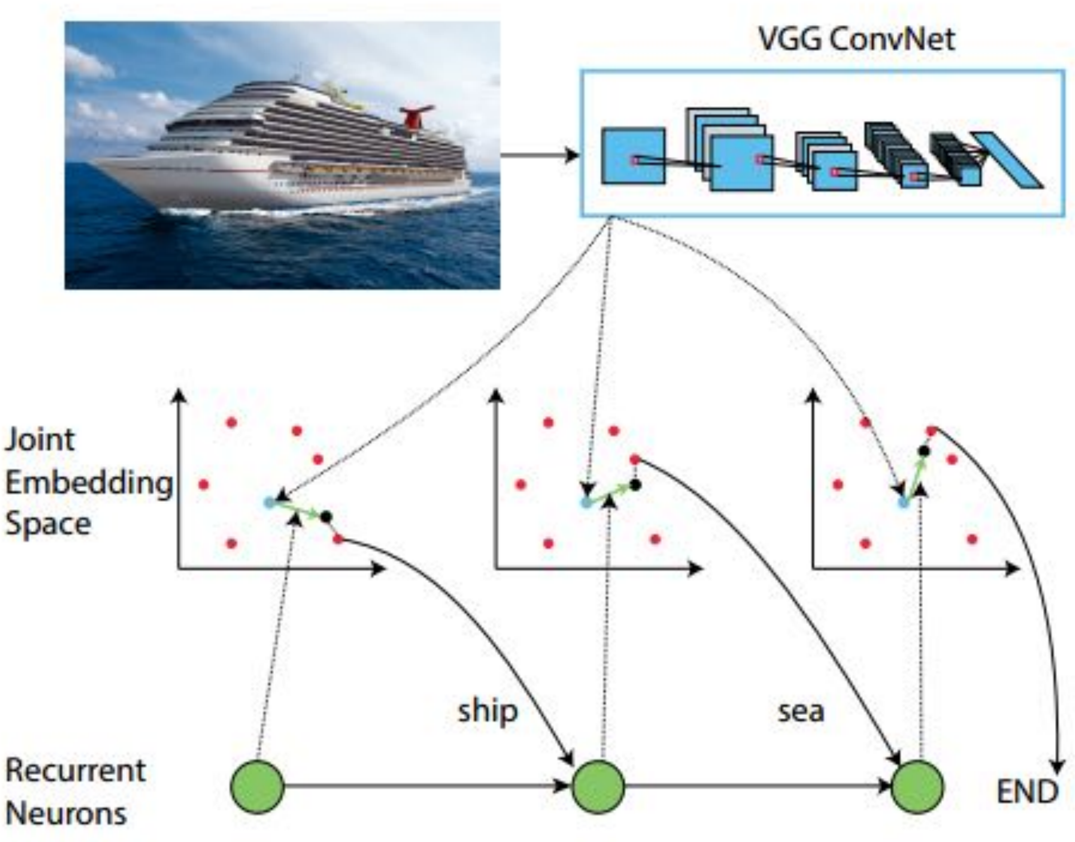
$$\overrightarrow{h}_t = \mathcal{H}\left(W_{x\overrightarrow{h}} x_t + W_{\overrightarrow{h}\overrightarrow{h}} \overrightarrow{h}_{t-1} + b_{\overrightarrow{h}}\right)$$

$$\overleftarrow{h}_t = \mathcal{H}\left(W_{x\overleftarrow{h}} x_t + W_{\overleftarrow{h}\overleftarrow{h}} \overleftarrow{h}_{t+1} + b_{\overleftarrow{h}}\right)$$

$$y_t = W_{\overrightarrow{h}y} \overrightarrow{h}_t + W_{\overleftarrow{h}y} \overleftarrow{h}_t + b_o$$

Adapted from: A. Graves

Applications : Multi-label image classification

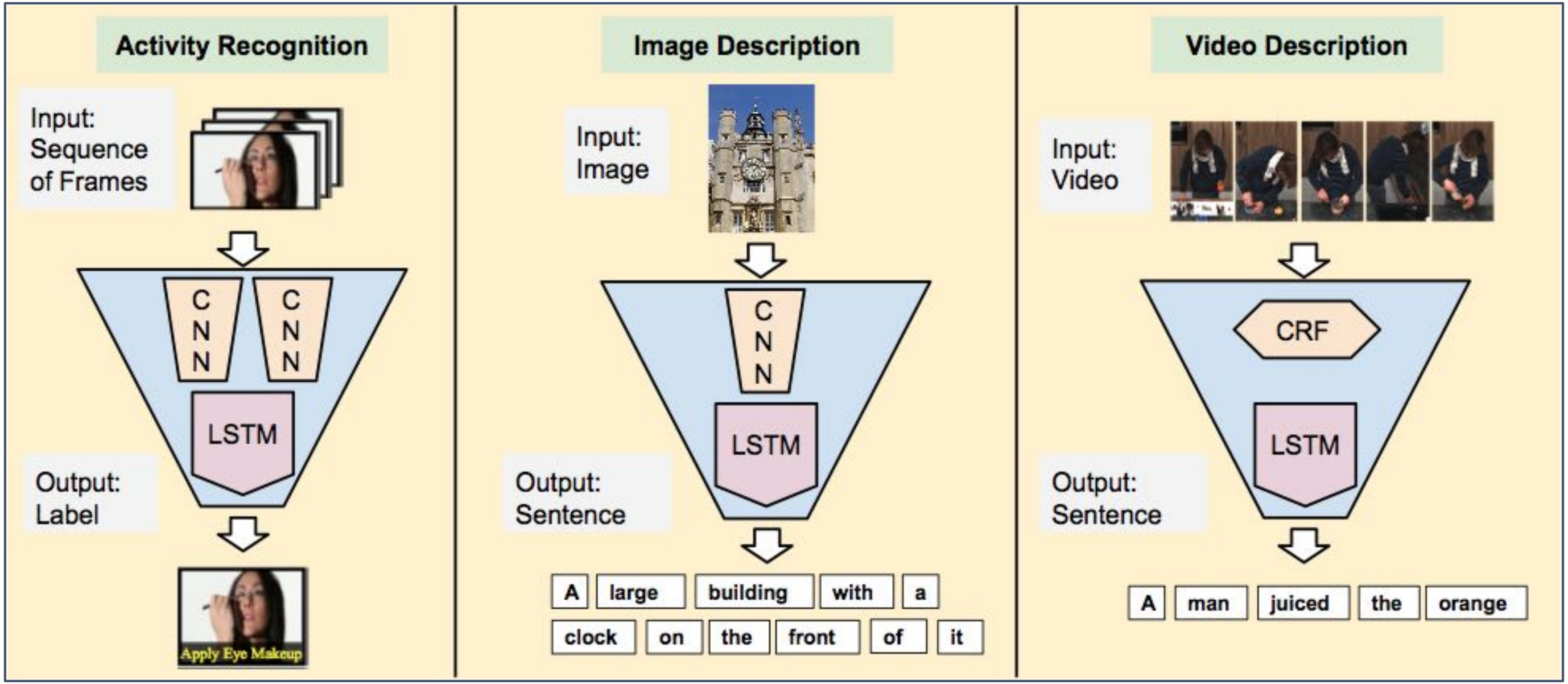


Wang et al CVPR 2016

Applications : Segmentation

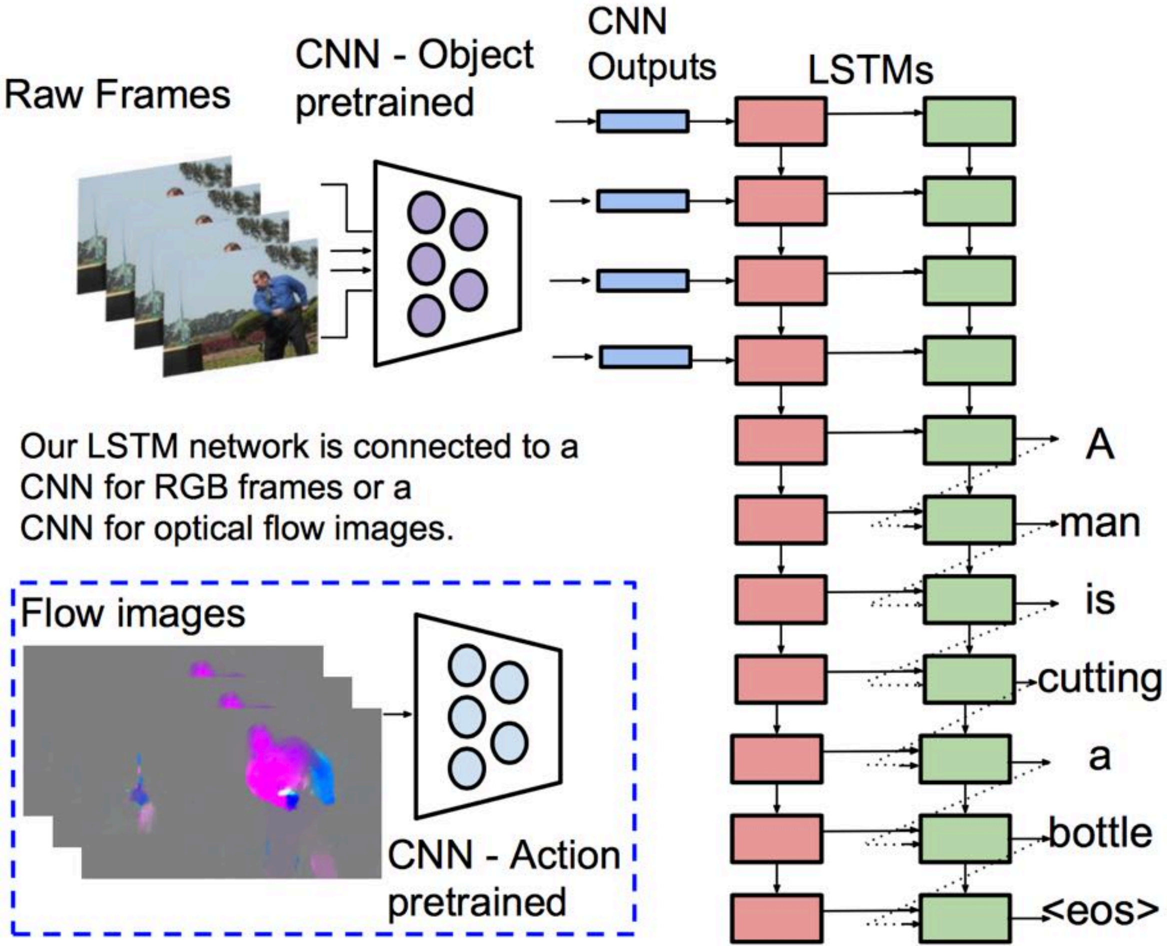
Zheng et al ICCV 2015

Applications: Visual Sequence Tasks



Jeff Donahue et al. CVPR'15

Applications : Videos to Natural Text



Venugopalan et al. ICCV 2015