Neural Networks

Recurrent Neural networks, LSDM

(P-ITEEA-0011)

Akos Zarandy
Lecture 9
November 12, 2018

Administrative announces

- Next week 3rd of December at the lecture time
  - Replacement test paper (Pót ZH)
  - Email will be sent through Neptun
- No quizzes on Tuesday in the workclass.
  - It will be done later.
  - The Wednesday and Monday group will have quizzed.
- Consultation will be held in the usual consultation time and place
- Programming test is coming (18th of December)
  - Please practice!

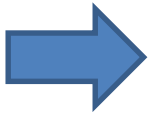
Contents

- How to handle sequential signals with Neural Networks?
- Recurrent Networks
  - Training
  - Examples
  - Vanishing gradient problem
- Long Short Term Memory (LSTM)
  - LSTM versions

Static samples vs Data signal flow

AlexNet could recognize 1000s of images.
ResNet could reach better then human performance.

- Though human can recognize
  - Single letters
  - Single sounds
  - Single tunes
  - Single pictures



- But in real life we handle
  - Texts
  - Speech
  - Music
  - Movies

Story

(temporal analysis of sequential data)

Can feed forward neural networks (perceptrons, conv. nets) solve these problems?

DATA MEMORY

Memory

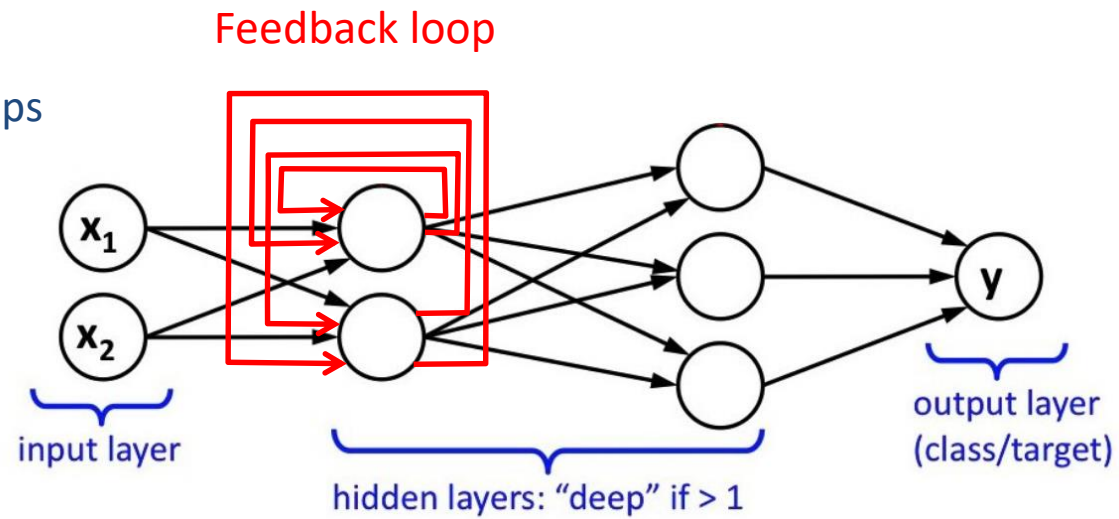
- Our feed-forward nets had so far
  - Program memory (for the weights)
  - Registers
    - For store temporally due to implementation and not matematical resasons
    - Registers were not part of the networks
- After each inferences the net was reset
  - All registers were deleted
  - No information remained in the net after processing an input vector
  - Therefore the order of a test sequence made no difference

Recurrent networks (RNN)

- Unlike traditional neural networks, the output of the RNN depends on the previous inputs
  - State
- RNN contains feedback
- Theoretically:
  - Directed graph with cyclic loops
- From now, time has a role in execution
  - Time steps, delays

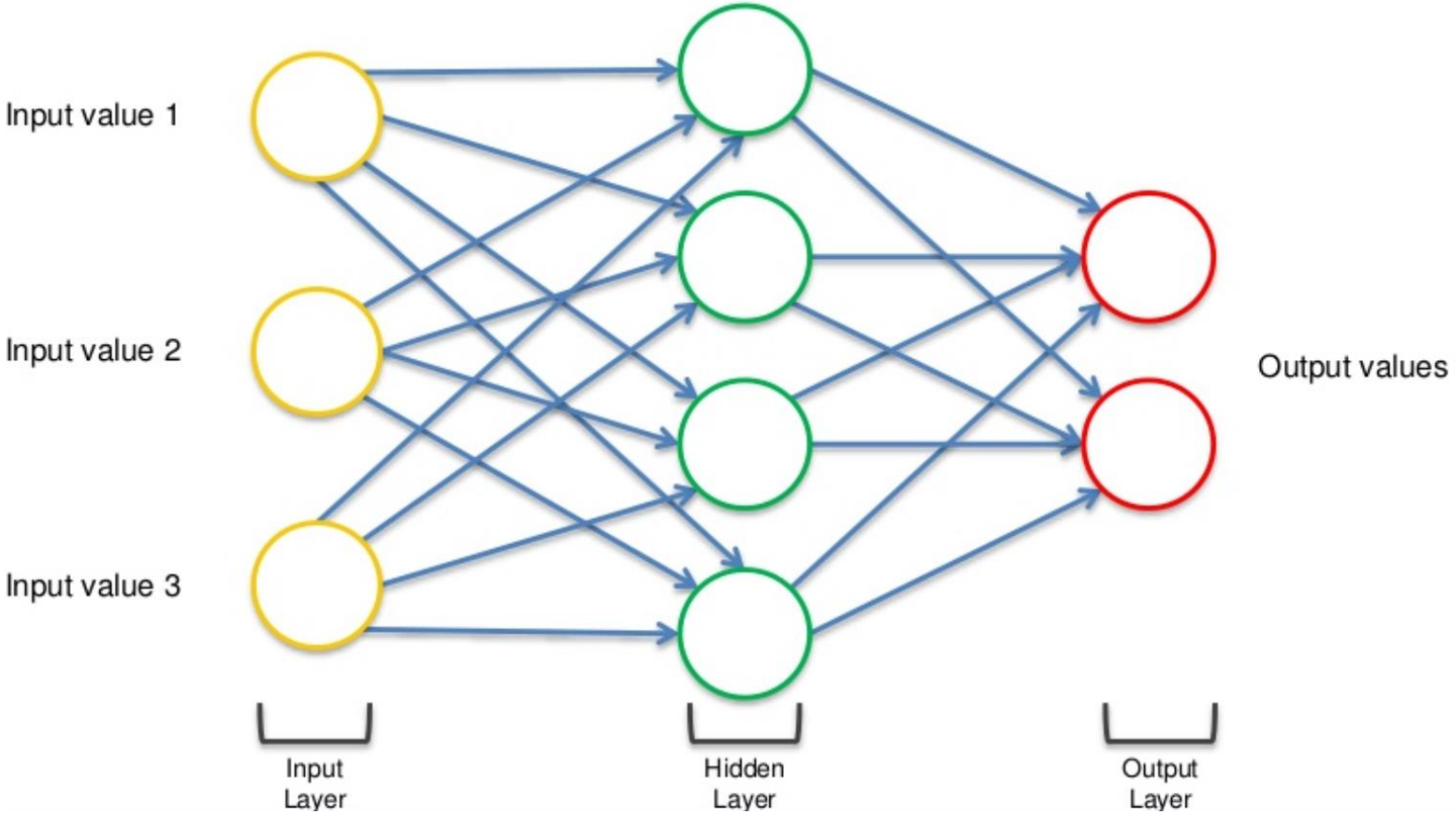
Jürgen lives in Berlin.

He speeks …….



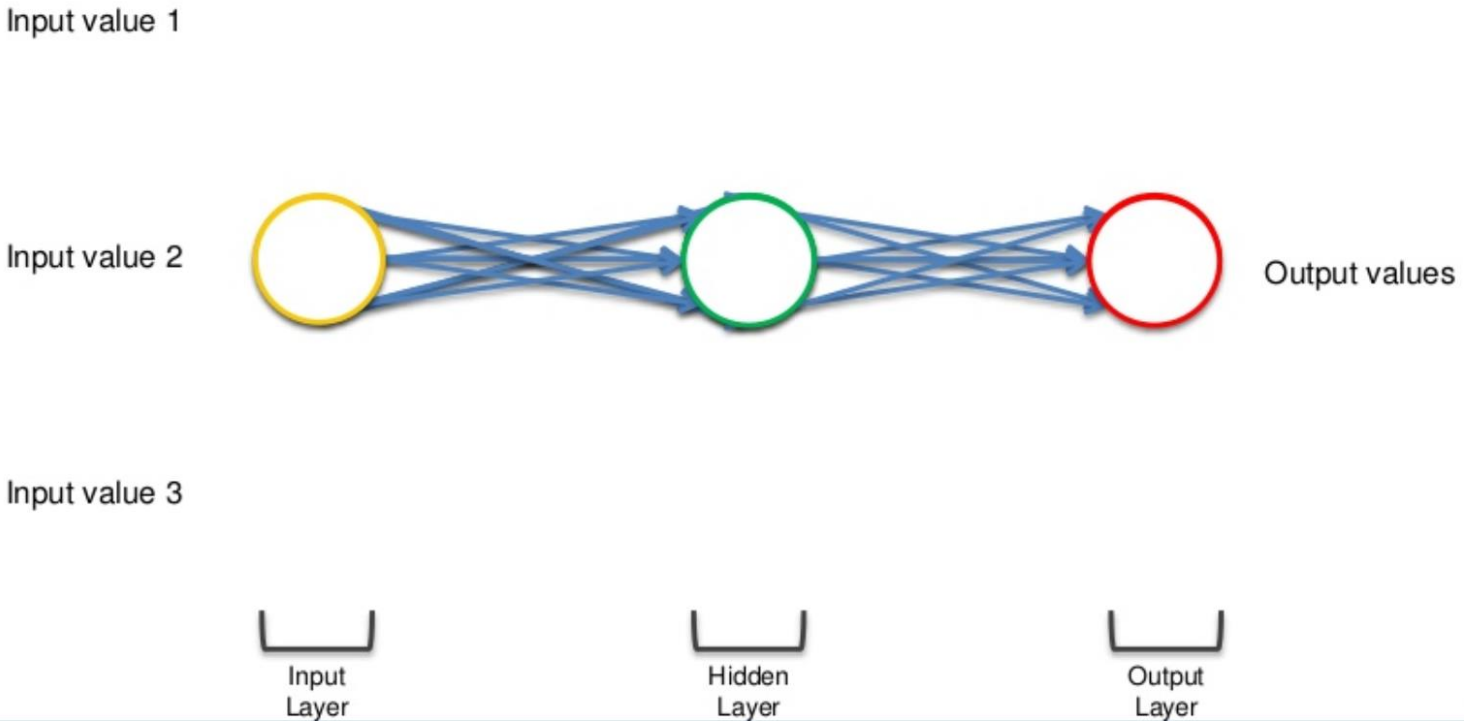
Steps towards vectorized data and parameters

- Weights (multiple arrows)



Steps towards vectorized data and parameters

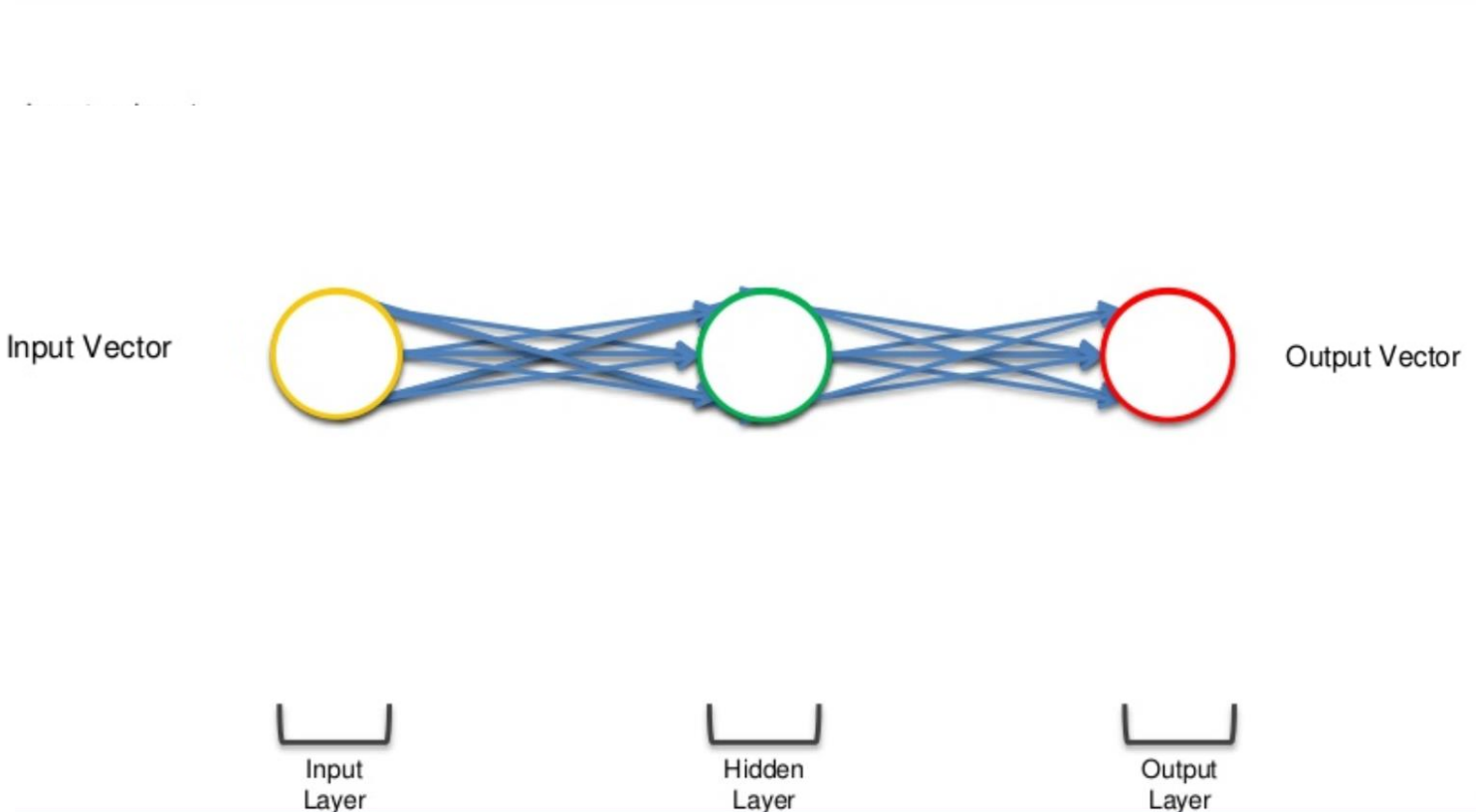
- Weights (multiple arrows)



Steps towards vectorized data and parameters

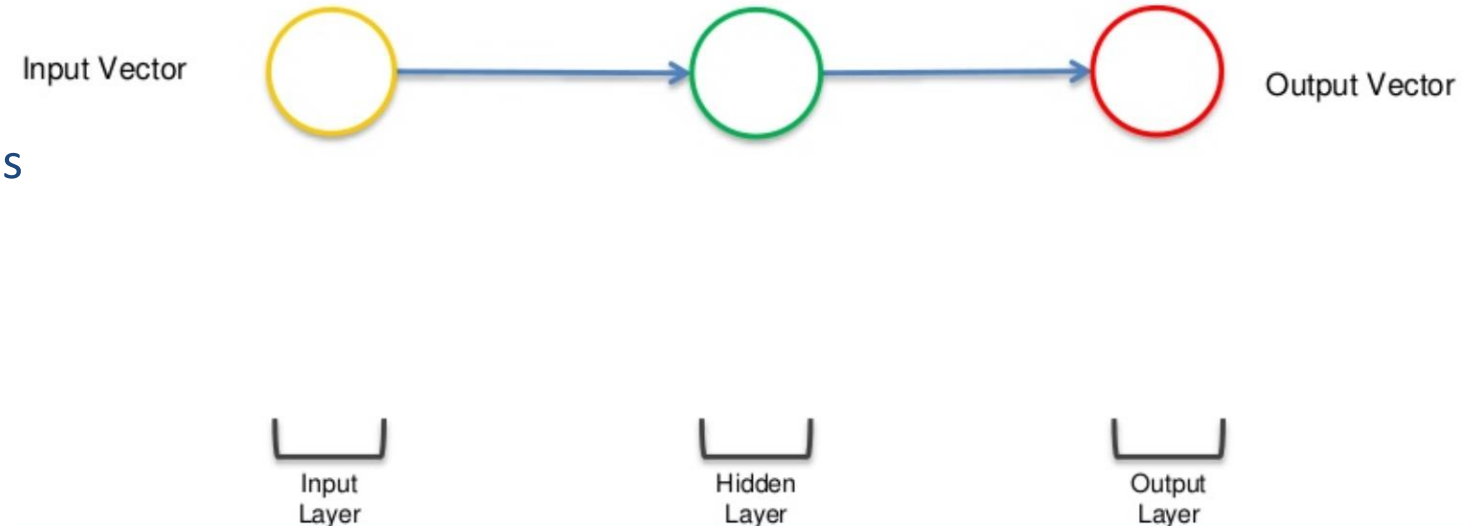
- Weights (multiple arrows)

  replaced with vectors (single arrows)



Steps towards vectorized data and parameters

- Single arrows indicate all interconnections between layers
- w_{ij} matrix matematically



Introducing feedback loop

$$h(0) = \begin{bmatrix} h_1(0) \\ \cdot \\ \cdot \\ h_l(0) \end{bmatrix}$$

$$x(1) = \begin{bmatrix} x_1(1) \\ \cdot \\ \cdot \\ x_k(1) \end{bmatrix}$$

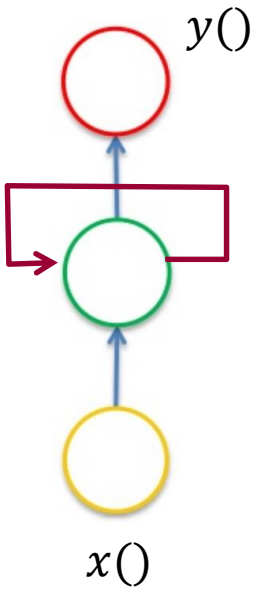
concatenation →

$$c(1) = \begin{bmatrix} h_1(0) \\ \cdot \\ \cdot \\ h_l(0) \\ x_1(1) \\ \cdot \\ \cdot \\ x_k(0) \end{bmatrix}$$

$$h(1) = f\big(h(0), x(1)\big) = \mathbf{W_h} c(1)$$

w: $l \times (k+l)$ *sized weight matrix*

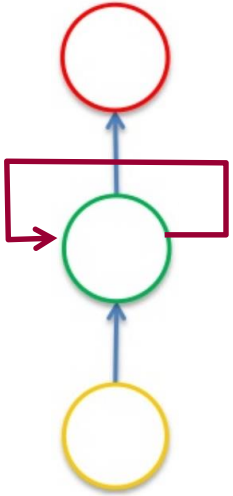
f() can be defined as a more complex function not only a matrix vector multiplication.

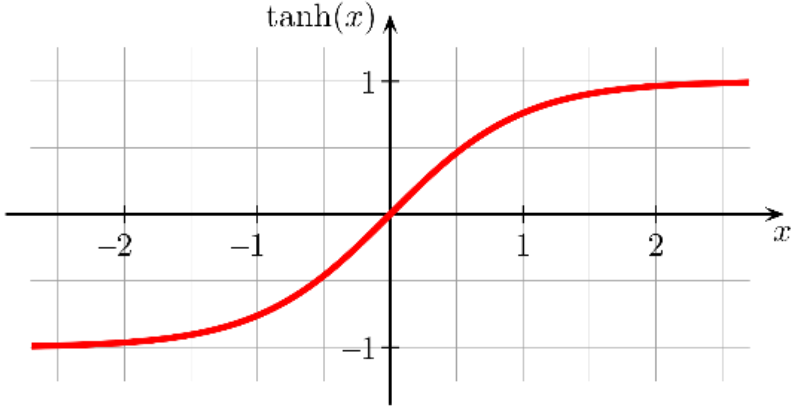


$$h(0) = 0$$

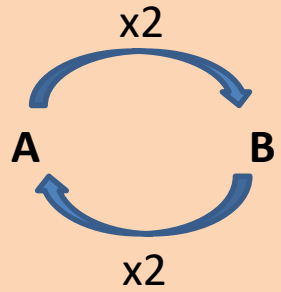
Activation function in feedback loop

- Activation function of the hidden layers is typically hyperbolic tangent
- It avoids large positive feedback
  - Keeps the output between -1 and +1
  - Avoids exploding the loop calculation
  - Gain should be smaller than 1 in the loop!





Positive feedback in a loop:
A *produces more of* ***B*** *which in turn produces more of* ***A.***
It leeds to increase beyond any limit.

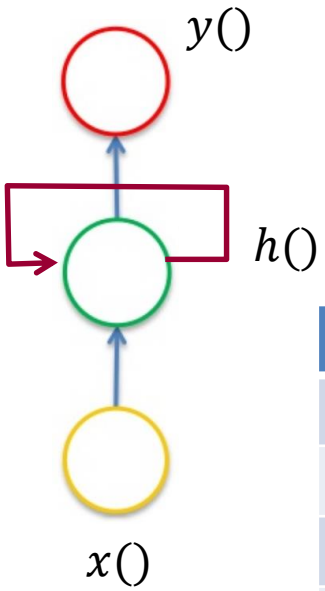


Timing of the RNN

- Discrete time steps are used
- Input vector sequence to apply
- Signals are calculated in a node, when all inputs exist
- State machine

Time	Input	State	output
t=1	$x(1)$	$h(1) = f\big(h(0), x(1)\big)$	$y(1) = g\big(h(1)\big)$
t=2	$x(2)$	$h(2) = f\big(h(1), x(2)\big)$	$y(2) = g\big(h(2)\big)$
t=3	$x(3)$	$h(3) = f\big(h(2), x(3)\big)$	$y(3) = g\big(h(3)\big)$
t=4	$x(4)$	$h(4) = f\big(h(3), x(4)\big)$	$y(4) = g\big(h(4)\big)$
		. . .	

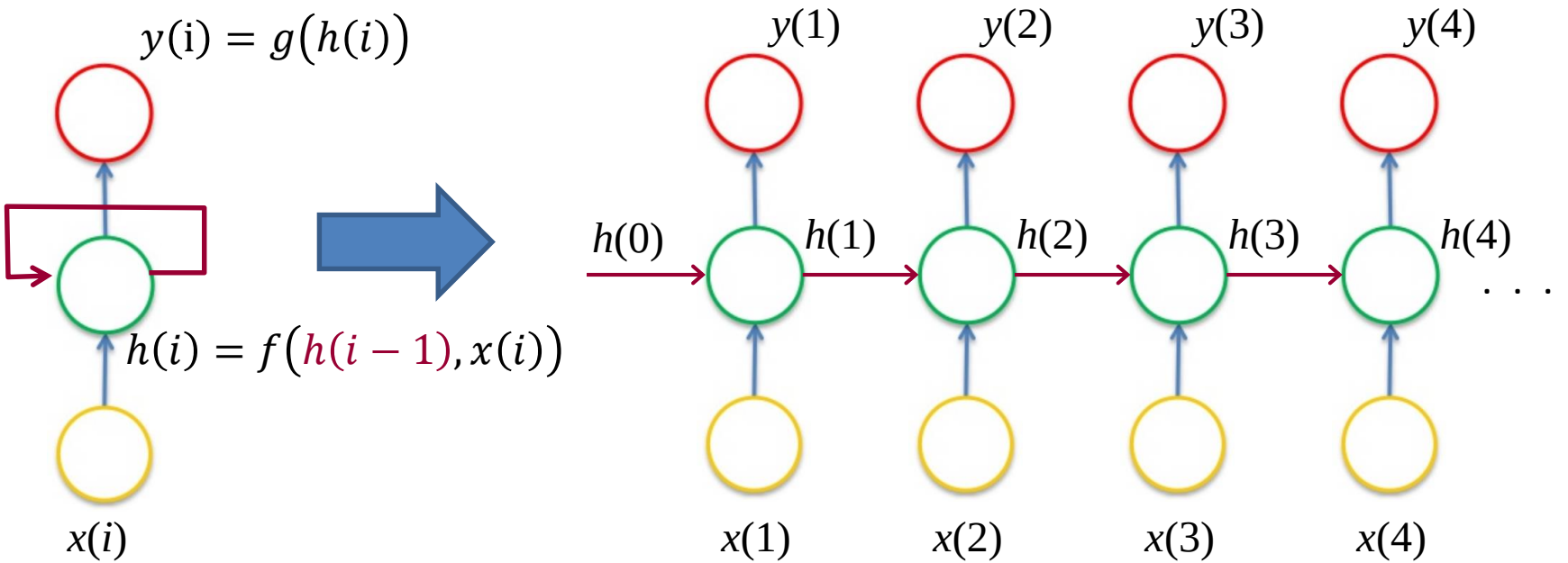
How to calculate back propagation?



$h(0) = 0$

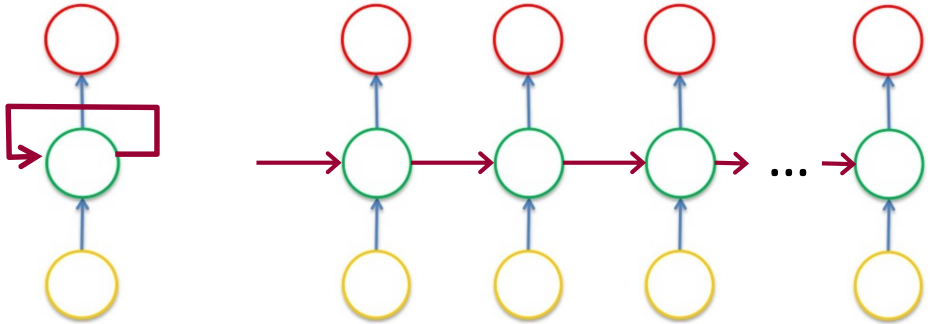
input
$x(1)$
$x(2)$
$x(3)$
$x(4)$
…

Unrolling



Unrolling

- Unrolling generates an acyclic directed graph from the original cyclic directed graph structure
- It generates a final impulse response (FIR) filter from the original infinite impulse response (IIR) filter
- Dynamic behavior

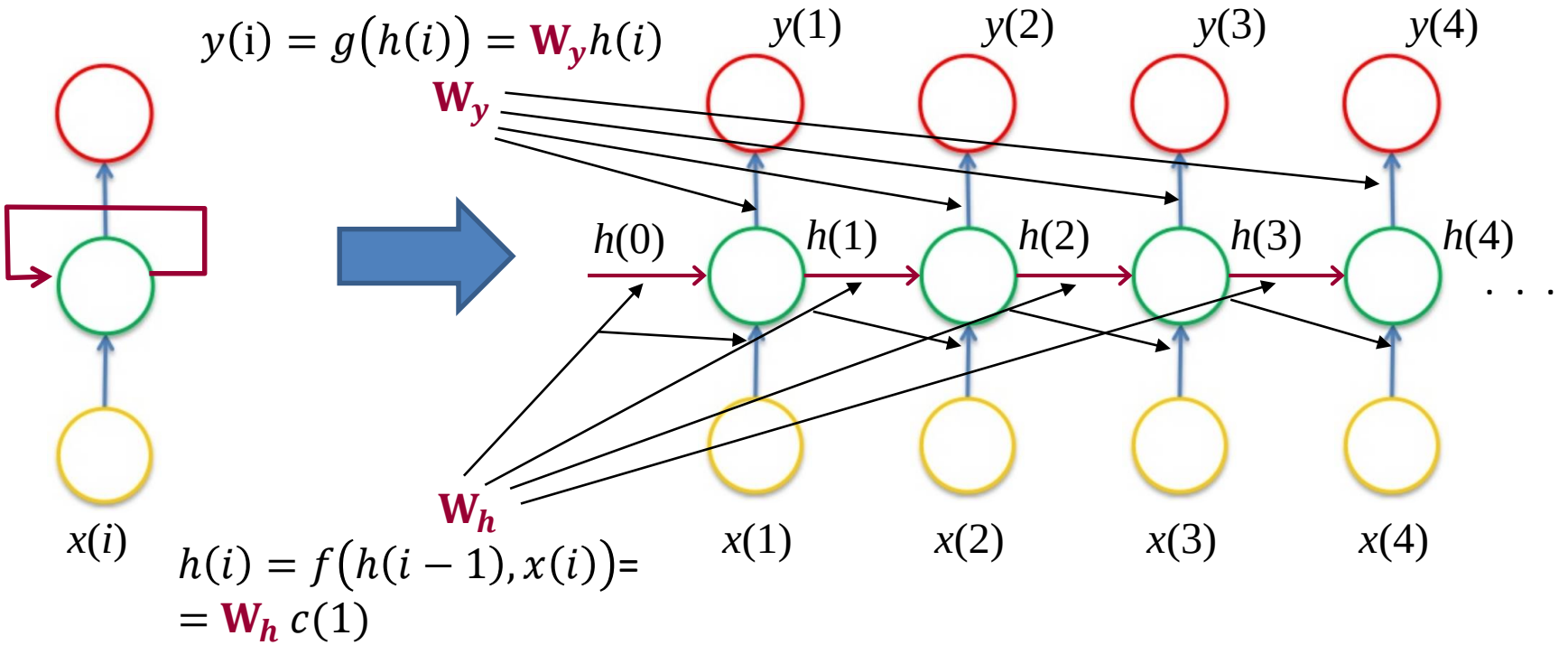


IIR filters may response to any finite length input with a infinite (usually decaying) response, due to their internal loop.

FIR filters response to any finite length input with a final response.

Weight matrix sharing

RNN re-uses the same weight matrix in every unrolled steps.



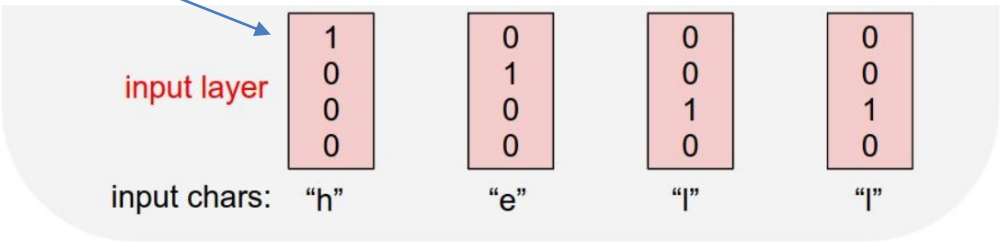
Simple RNN Training Example: Predicting the next letter

Example: Character-level Language Model

Vocabulary: [h,e,l,o]

One-hot encoding

Example training sequence: **“hello”**



http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf

Simple RNN Training Example: Predicting the next letter

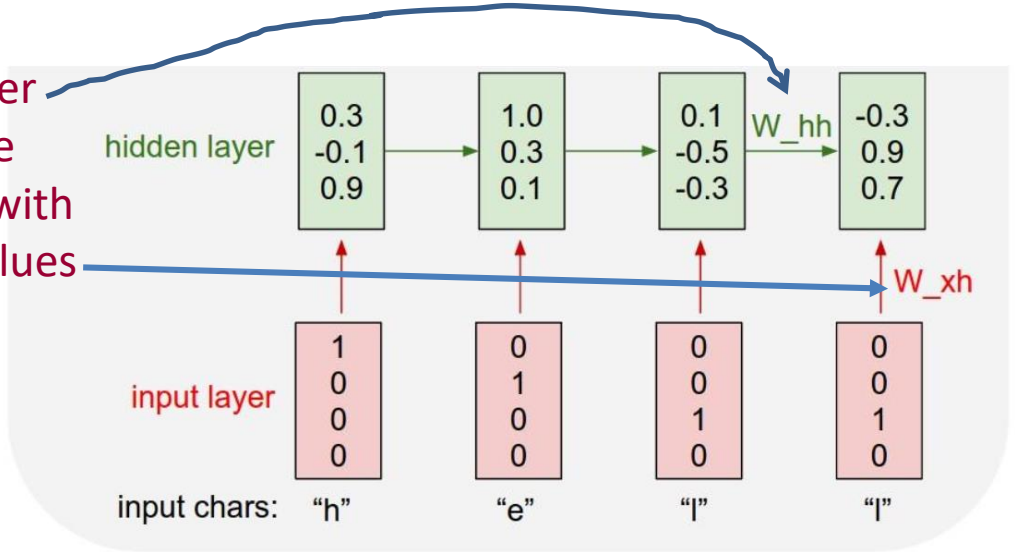
Example: Character-level Language Model

Vocabulary: [h,e,l,o]

Example training sequence: **“hello”**

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

Hidden layer weights are initialized with random values



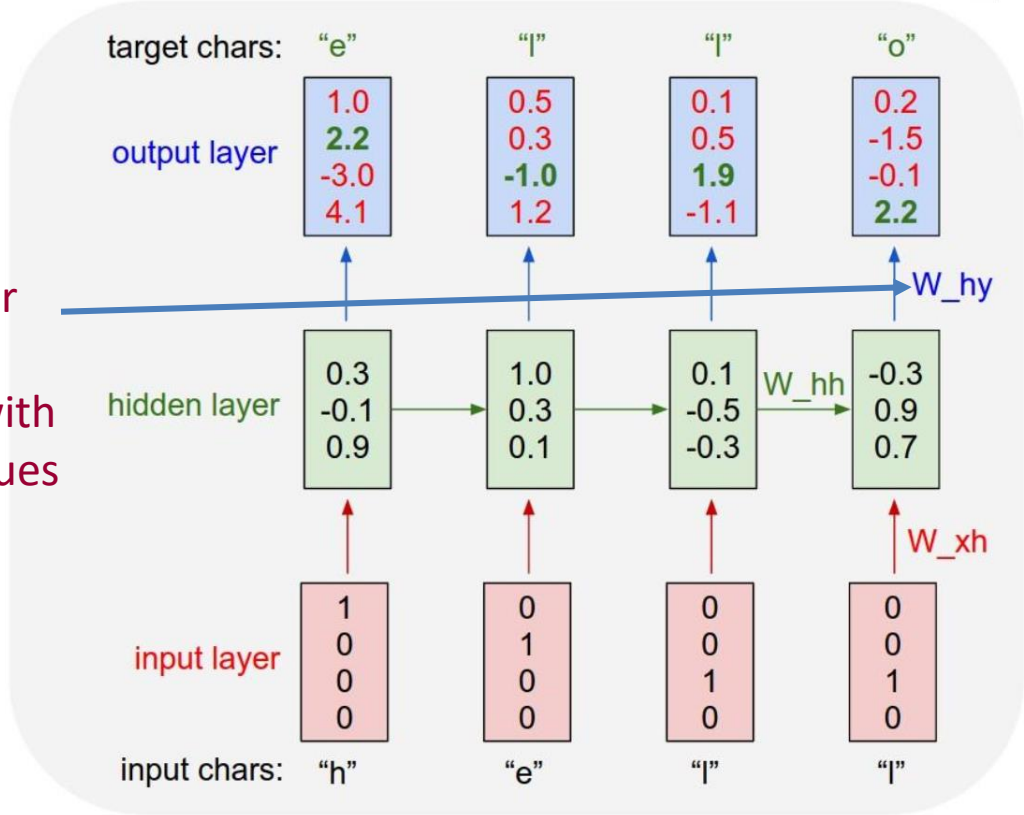
Simple RNN Training Example: Predicting the next letter

Example: Character-level Language Model

Vocabulary: [h,e,l,o]

Example training sequence: **“hello”**

Output layer weights are initialized with random values

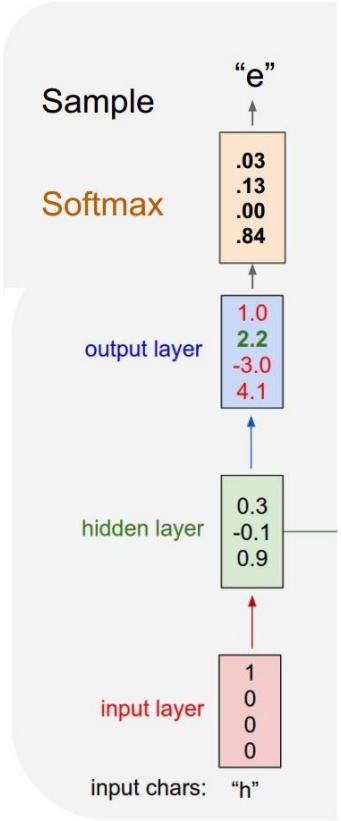


Simple RNN Training Example: Predicting the next letter

Example: Character-level Language Model Sampling

Vocabulary:
[h,e,l,o]

At test-time sample characters one at a time, feed back to model

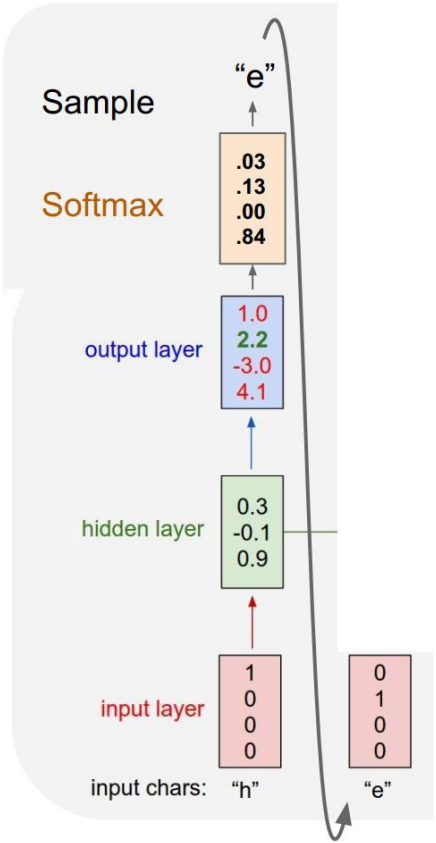


Simple RNN Training Example: Predicting the next letter

Example: Character-level Language Model Sampling

Vocabulary:
[h,e,l,o]

At test-time sample characters one at a time, feed back to model

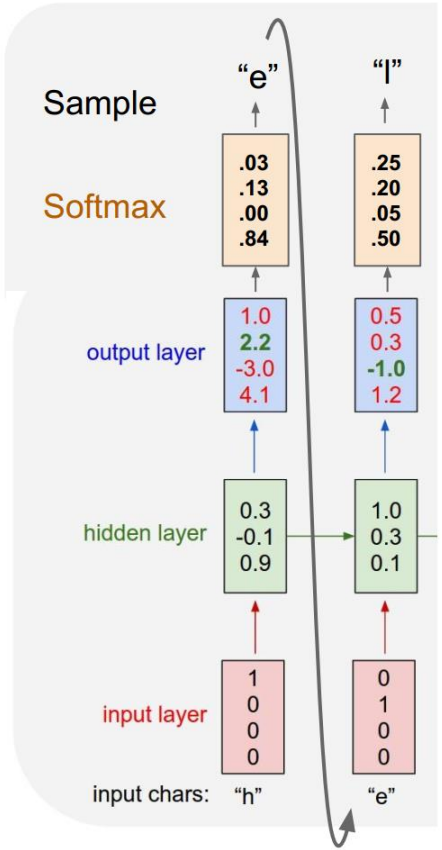


Simple RNN Training Example: Predicting the next letter

Example:
Character-level
Language Model
Sampling

Vocabulary:
[h,e,l,o]

At test-time sample characters one at a time, feed back to model



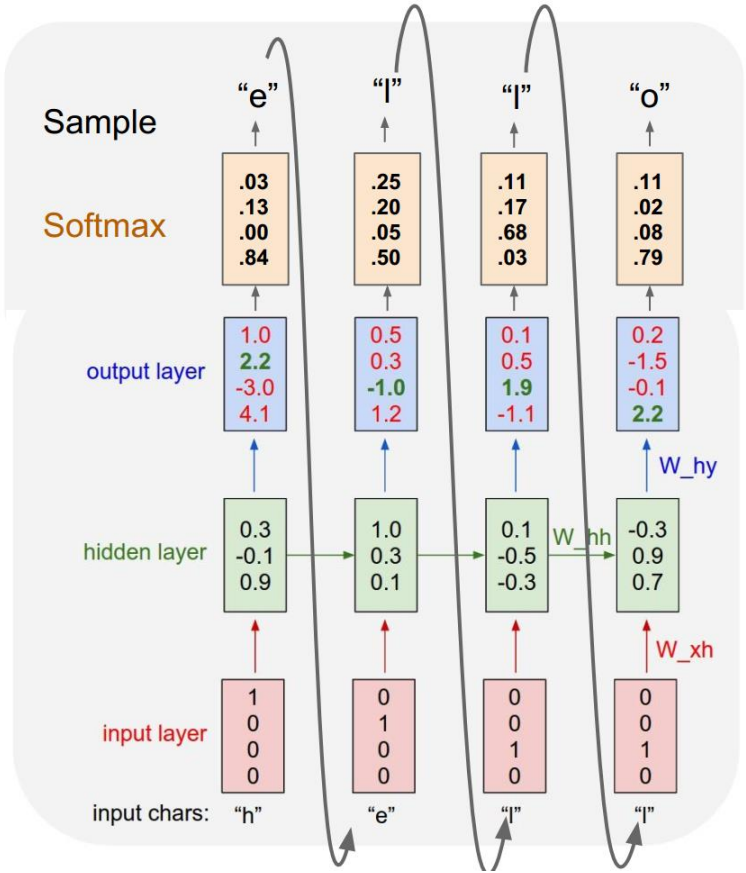
Simple RNN Training Example: Predicting the next letter

Example: Character-level Language Model Sampling

Vocabulary: [h,e,l,o]

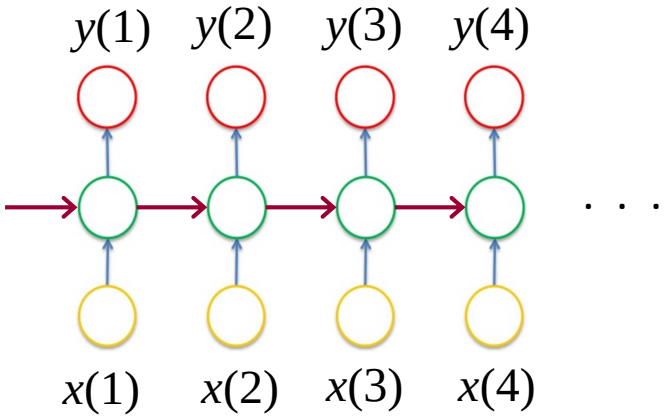
At test-time sample characters one at a time, feed back to model

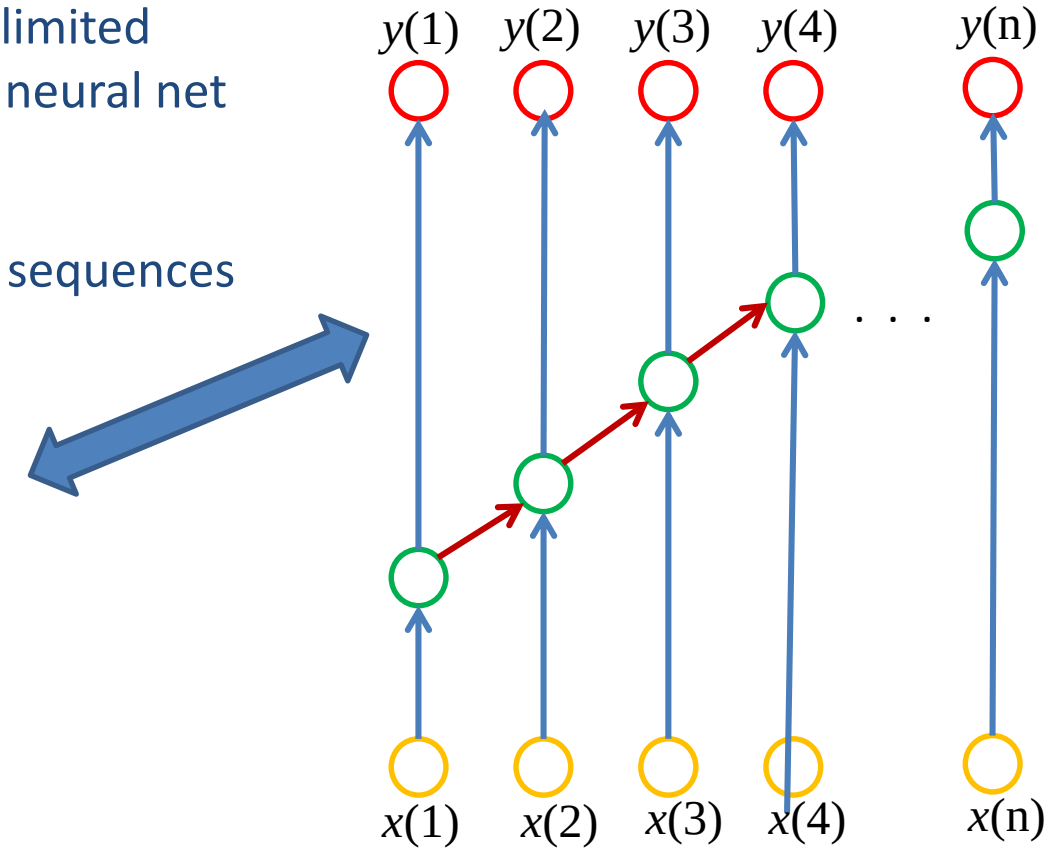
Backpropagation can be started using negative log likelihood cost function



Back propagation through time

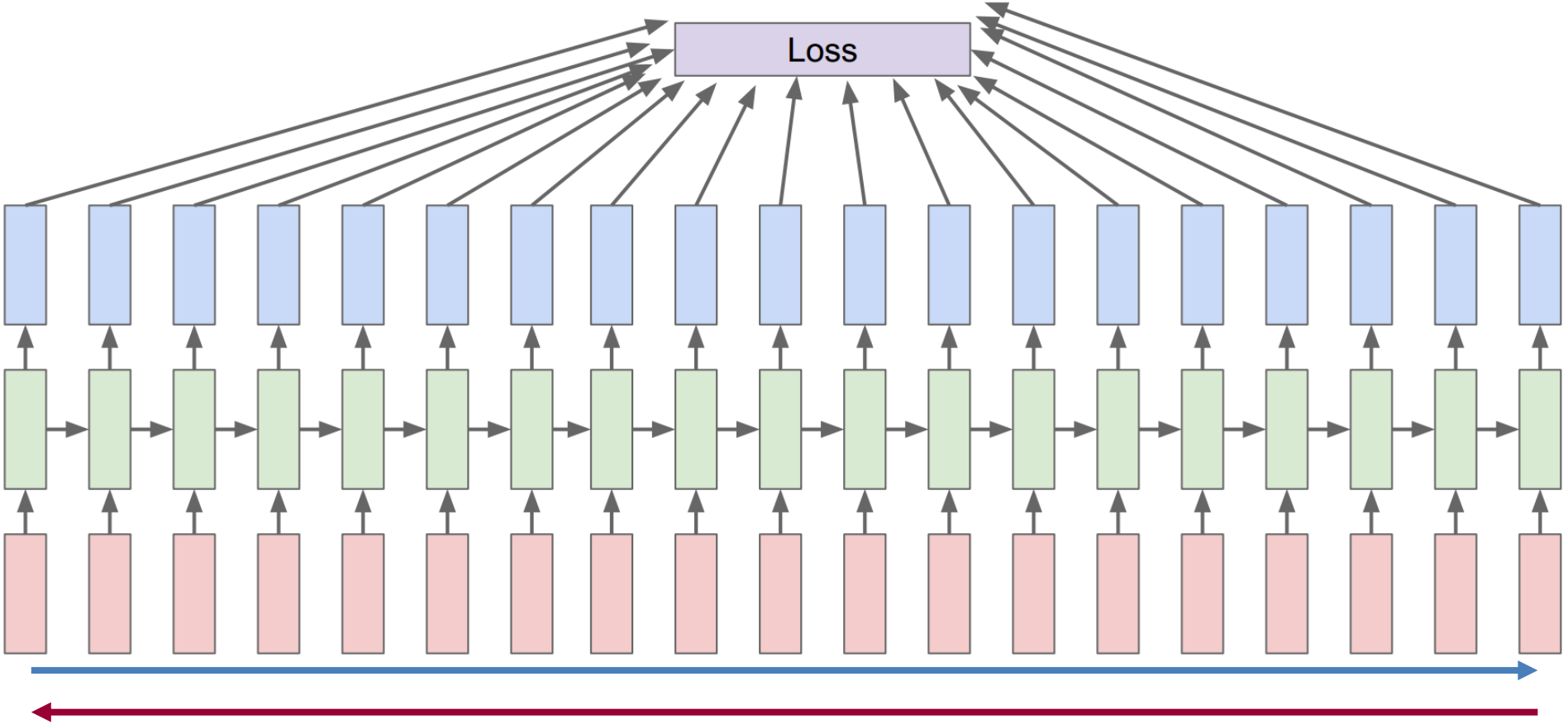
- Assuming that the length of the input vector sequence is limited
- It became a feedforward neural net
- Possible to apply back propagation
- We need multiple vector sequences to train!



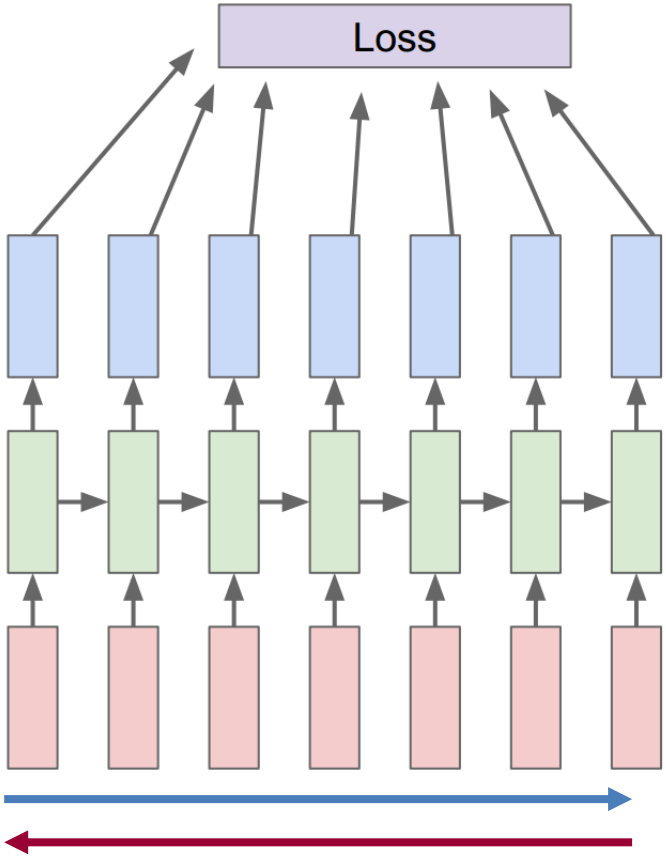


Backpropagation through time

Forward through entire sequence to compute loss, then backward through entire sequence to compute gradient

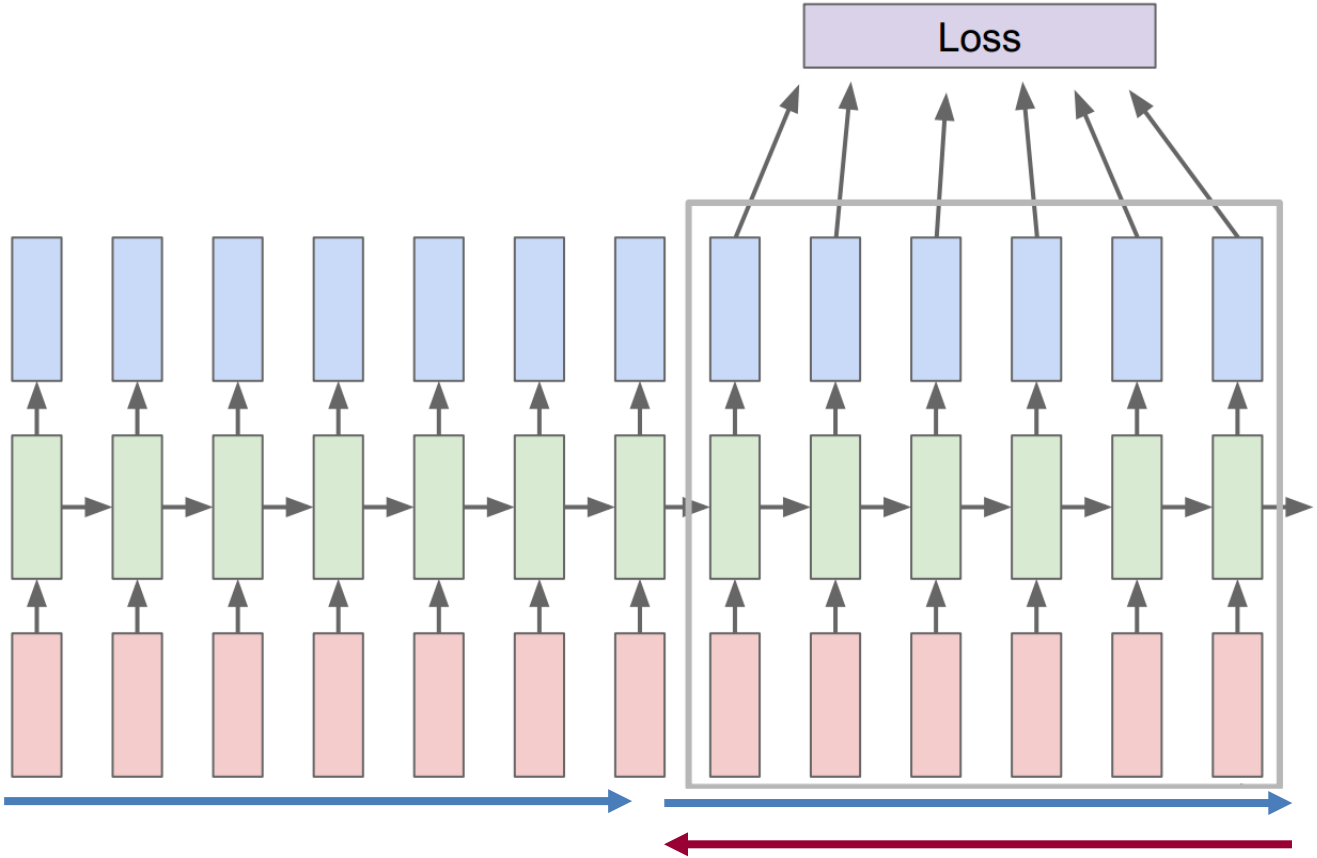


Truncated Backpropagation through time



Run forward and backward through chunks of the sequence instead of whole sequence

Truncated Backpropagation through time



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

Truncated Backpropagation through time

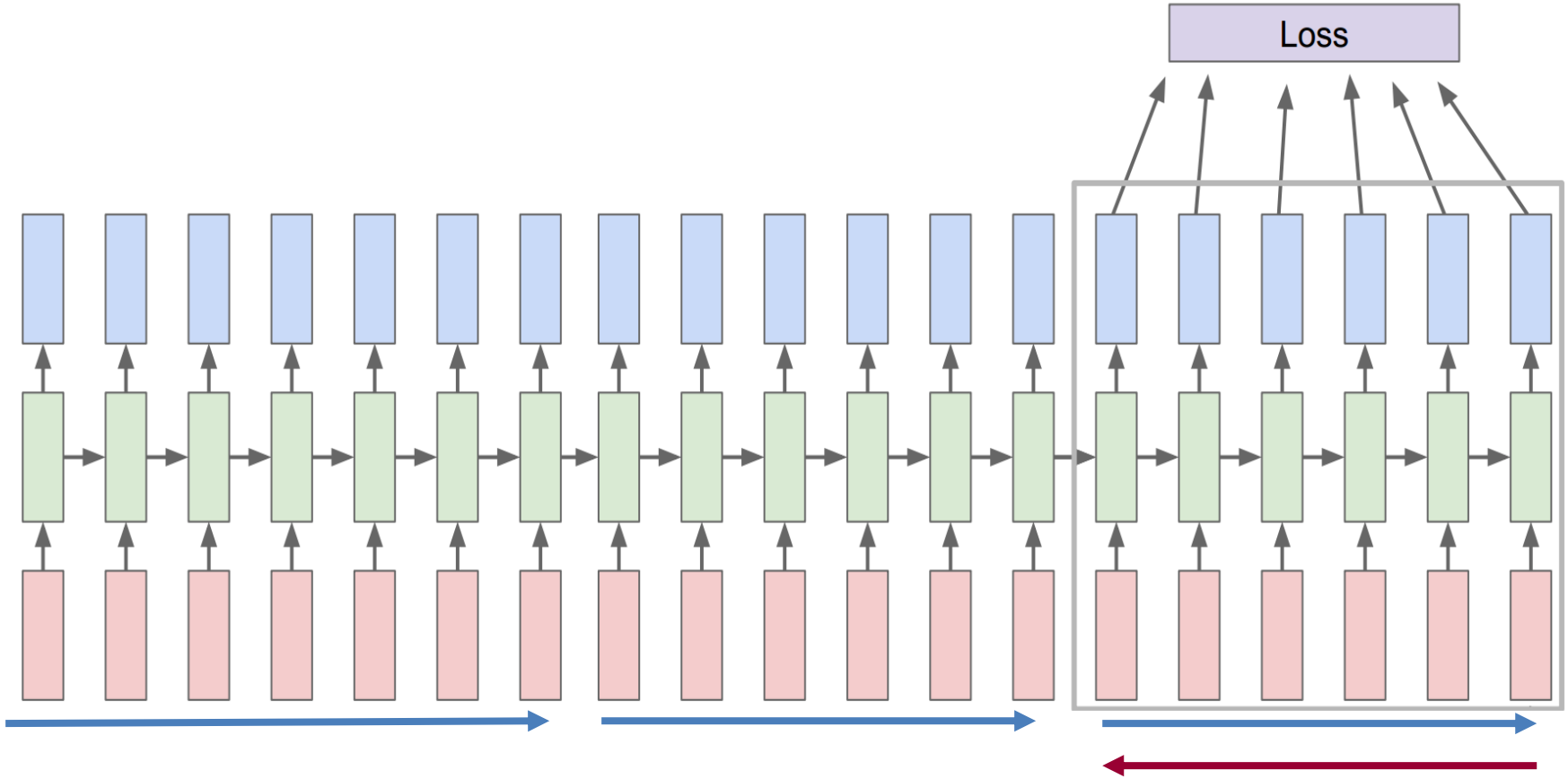


Image captioning example

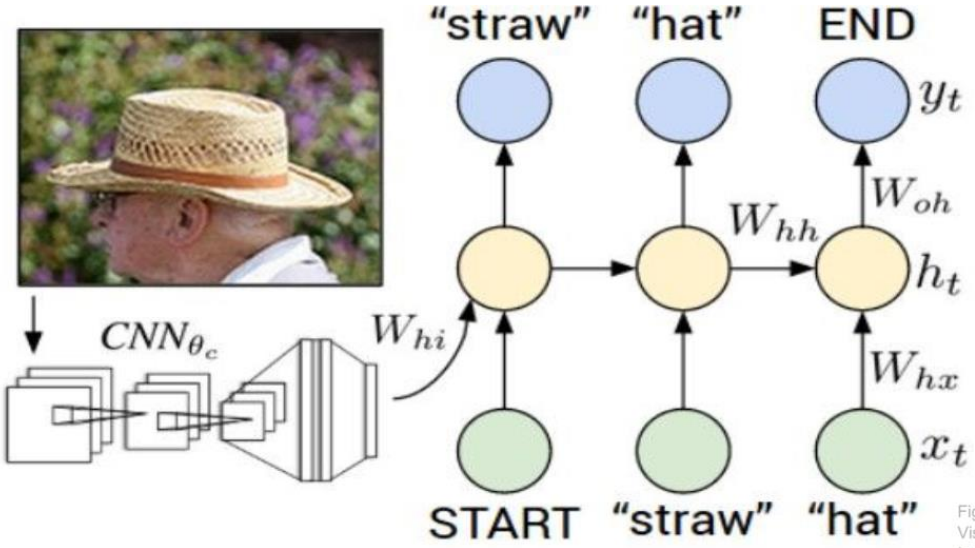
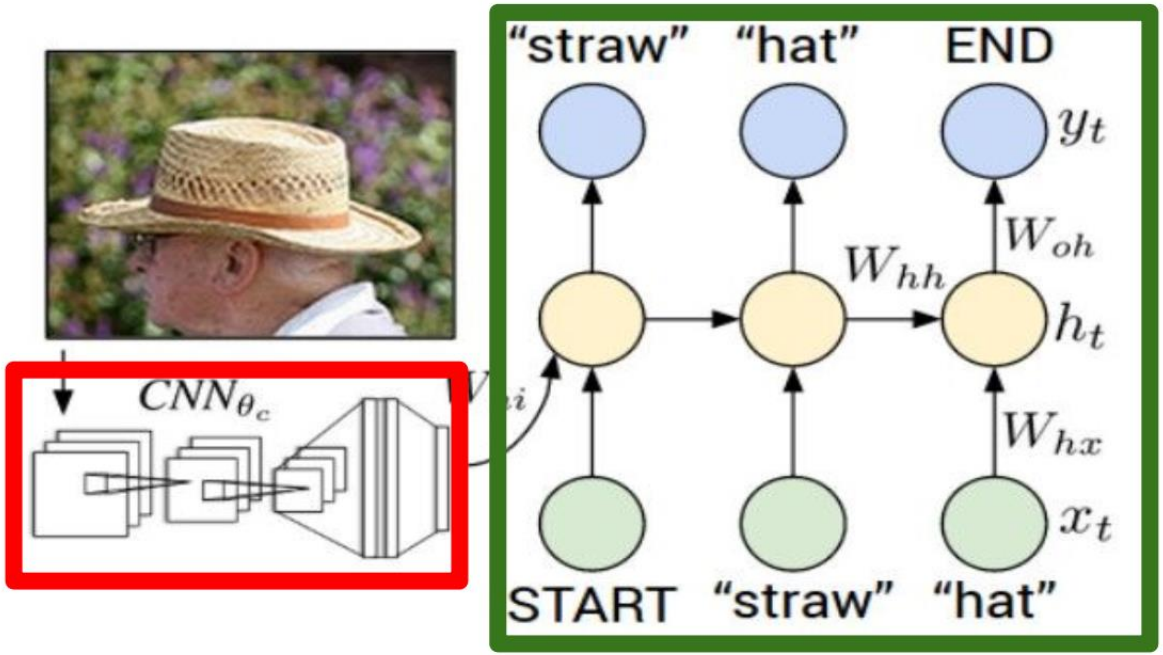


Figure from Karpathy et a, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015; figure copyright IEEE, 2015.
Reproduced for educational purposes.

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

Image captioning example



test image

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

Alexnet: scored 5 best guesses

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

X

test image

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

test image

x0
<STA
RT>

<START>

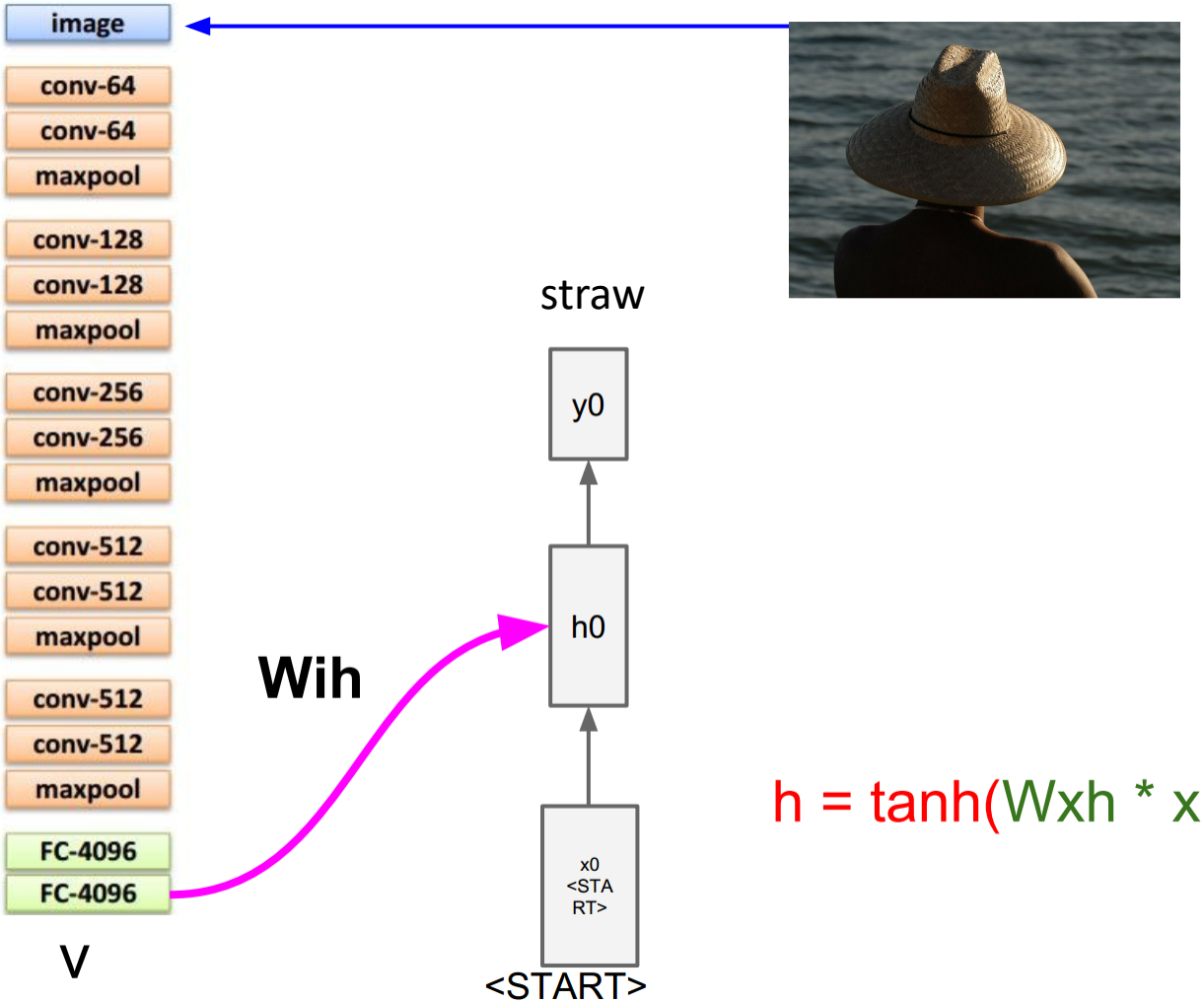


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
V
Wih
straw
y0
h0
x0
<STA
RT>
<START>
test image
h = tanh(Wxh * x + Whh * h + Wih * v)

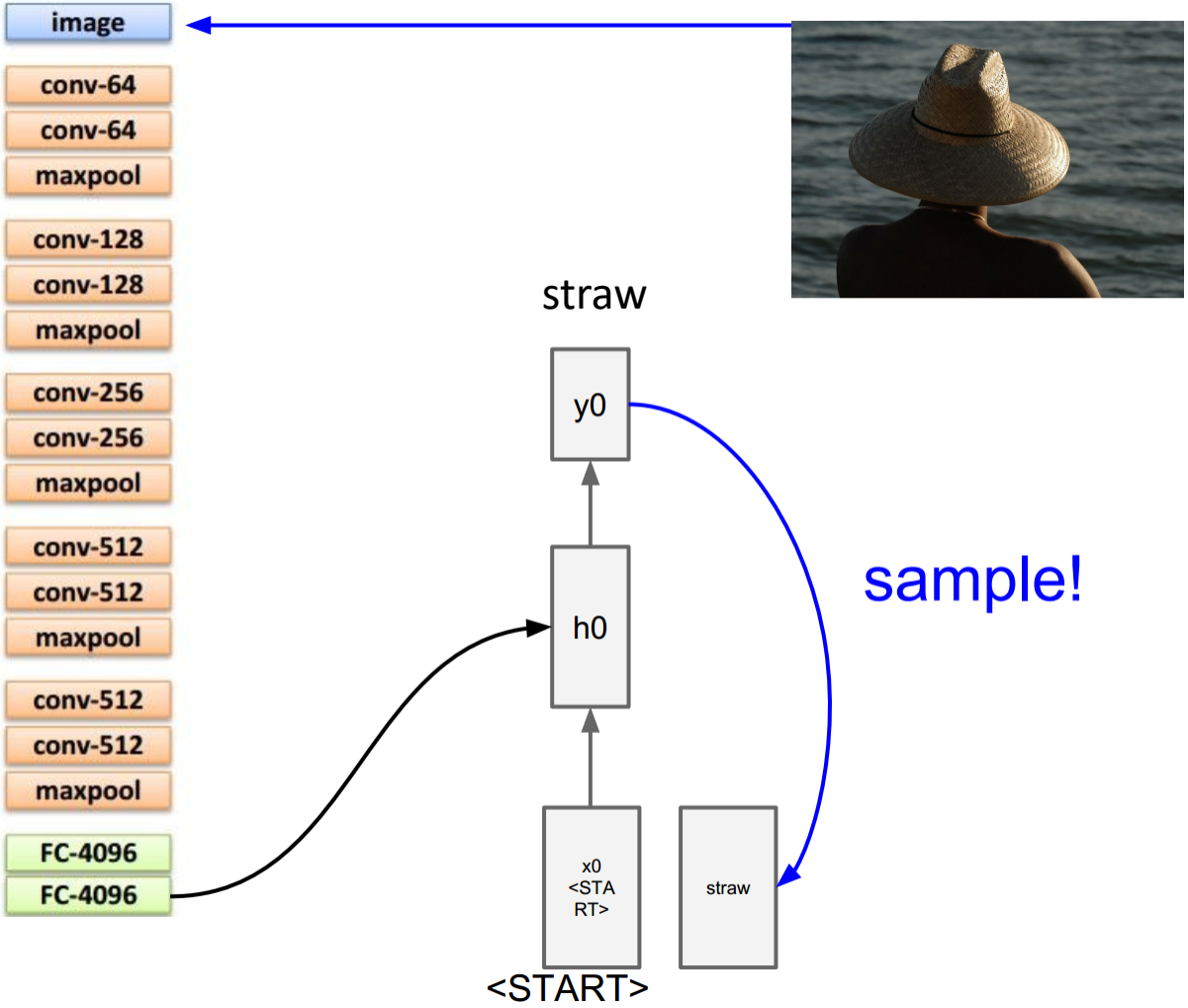


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
test image
straw
y0
h0
x0
<STA
RT>
straw
<START>
sample!

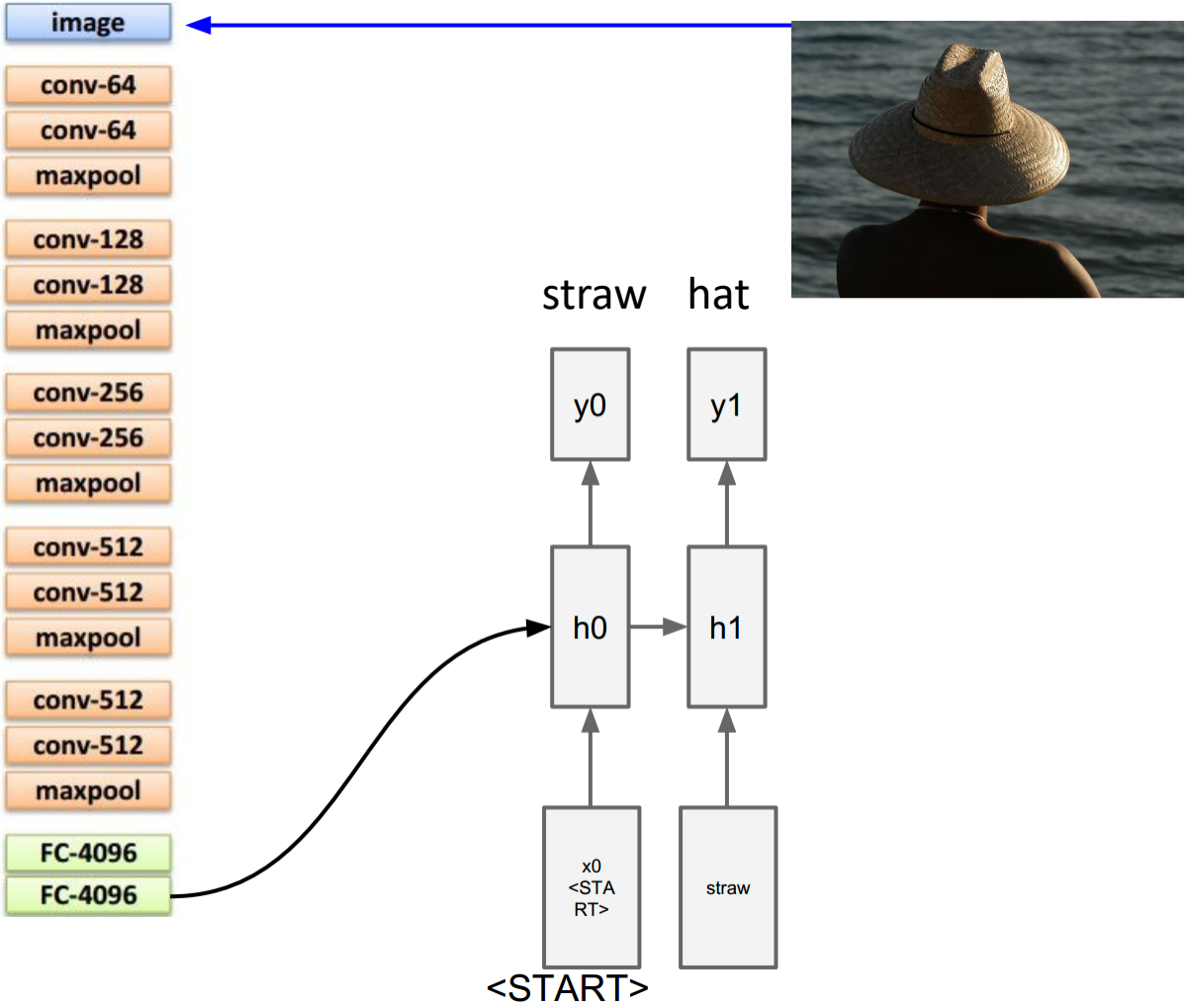


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
test image
straw
hat
y0
y1
h0
h1
x0
<STA
RT>
straw
<START>

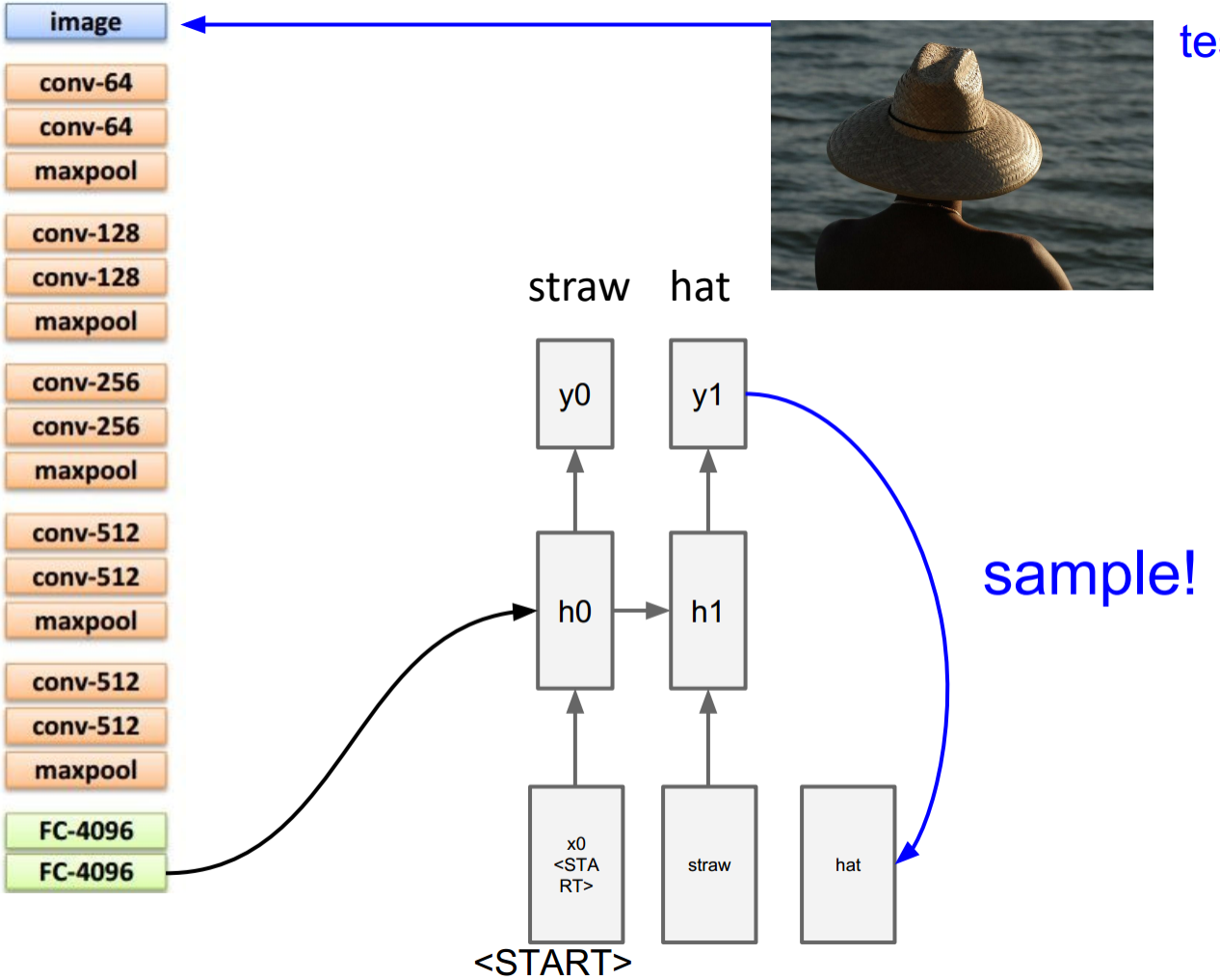


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
test image
straw
hat
y0
y1
h0
h1
sample!
x0
<STA
RT>
straw
hat
<START>

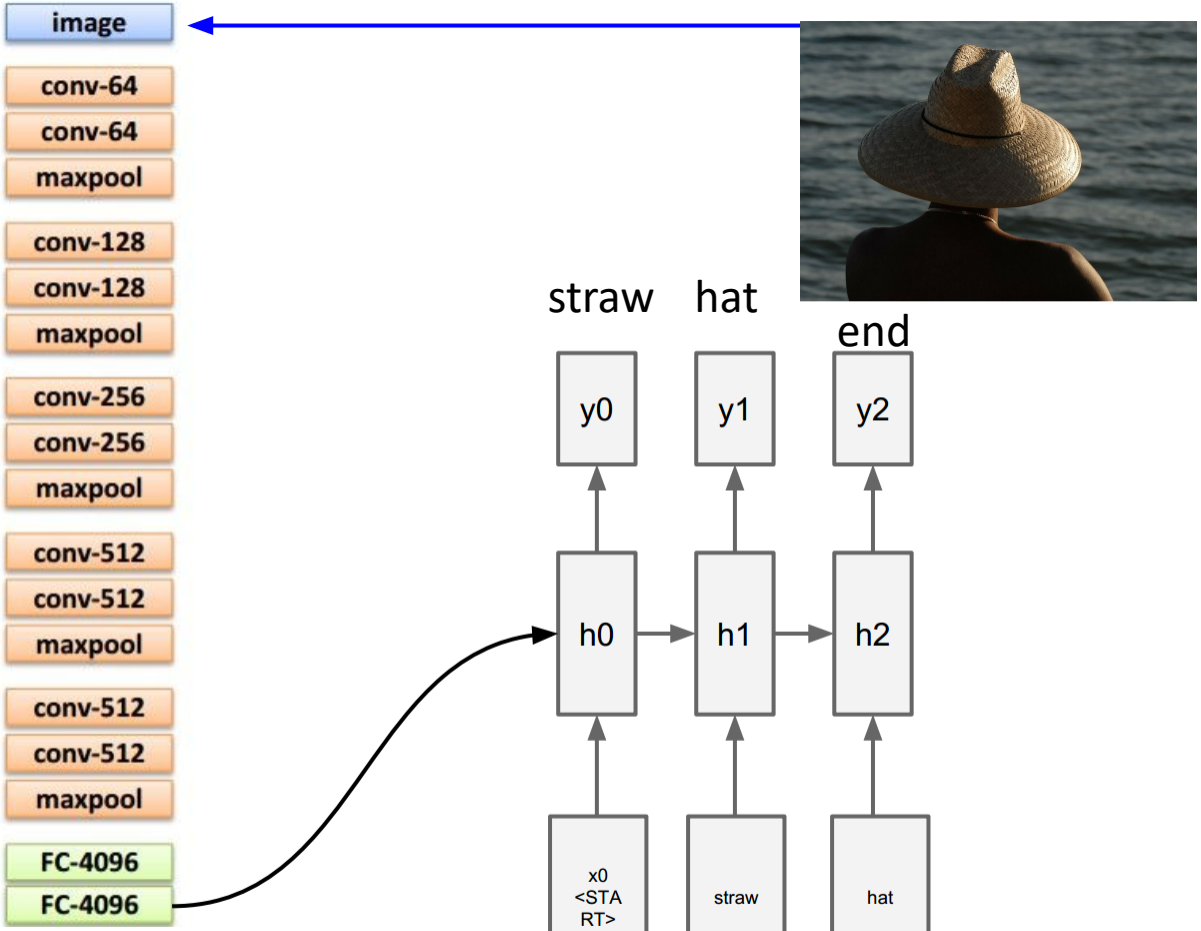


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
test image
straw
hat
end
y0
y1
y2
h0
h1
h2
x0 <STA RT>
straw
hat
<START>

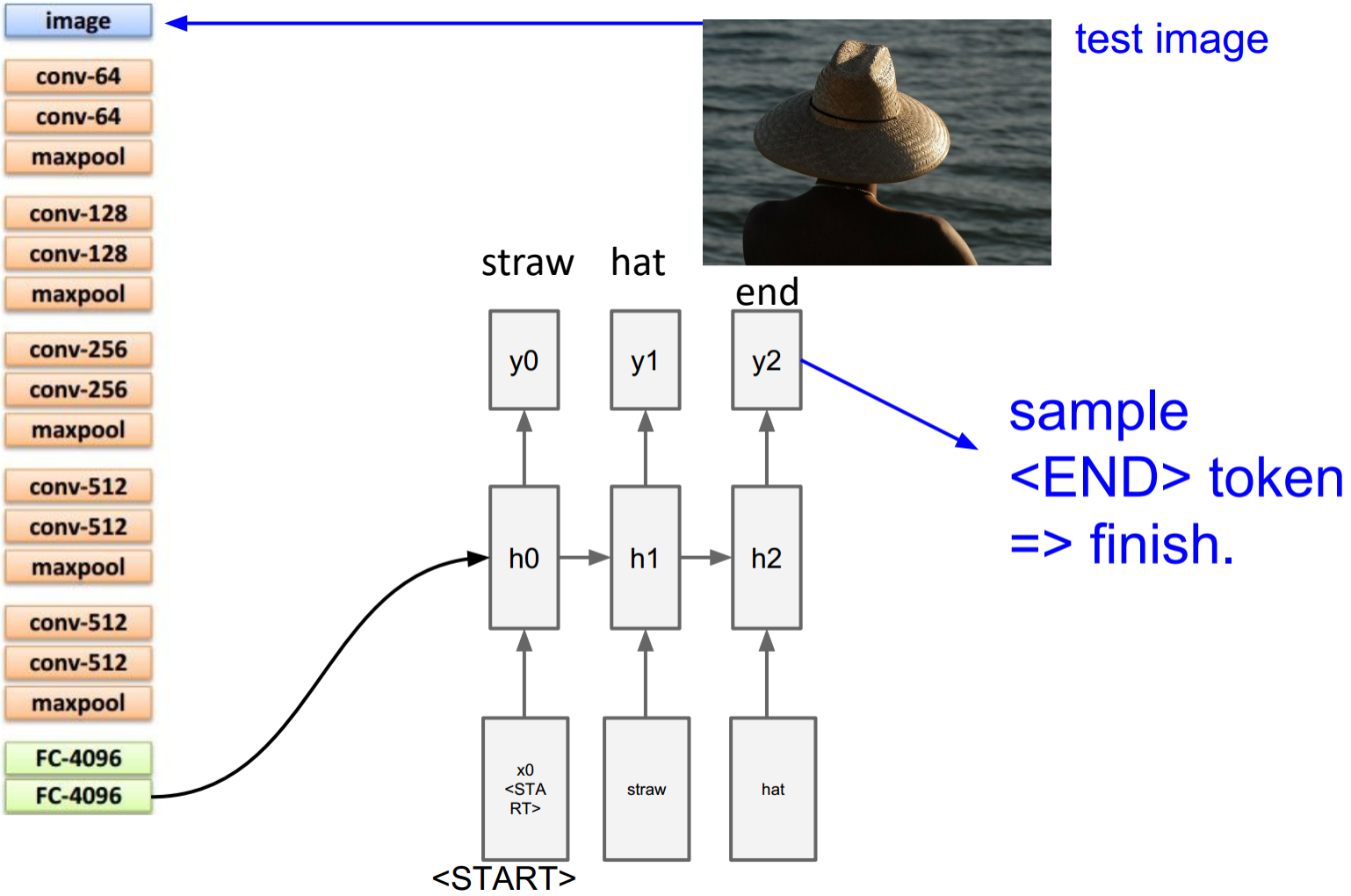


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
test image
straw
hat
end
y0
y1
y2
h0
h1
h2
x0 <STA RT>
straw
hat
<START>
sample <END> token => finish.

Image captioning Example: Results

Captions generated using neuraltalk2
All images are CC0 Public domain:
cat suitcase, cat tree, dog, bear,
surfers, tennis, giraffe, motorcycle

A cat sitting on a suitcase on the floor

A cat is sitting on a tree branch

A dog is running in the grass with a frisbee

A white teddy bear sitting in the grass

Two people walking on the beach with surfboards

A tennis player in action on the court

Two giraffes standing in a grassy field

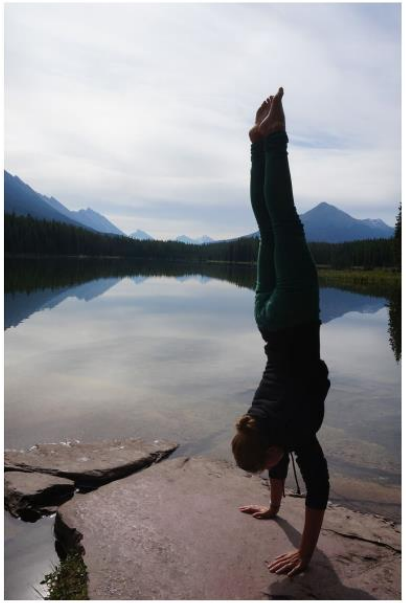
A man riding a dirt bike on a dirt track

Image captioning: Failure cases

Captions generated using neuraltalk2
All images are CC0 Public domain: fur coat, handstand, spider web, baseball

A woman is holding a cat in her hand

A person holding a computer mouse on a desk



A woman standing on a beach holding a surfboard

A bird is perched on a tree branch

A man in a baseball uniform throwing a ball

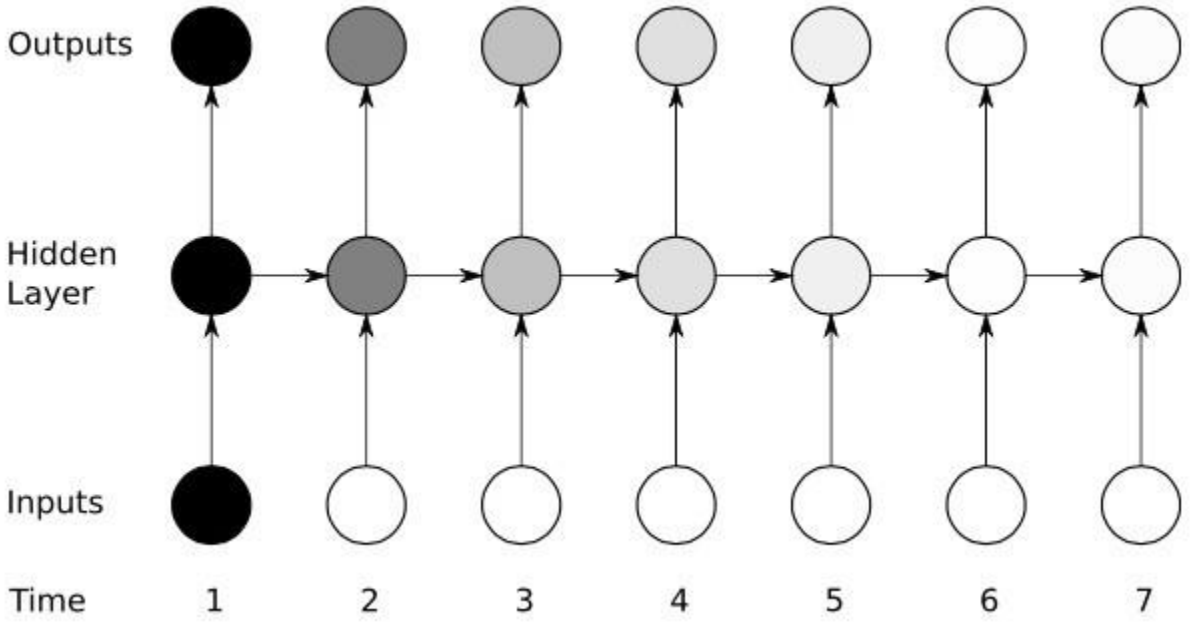
Problem

- What happens if the input sequence is too long?

Vanishing gradient!

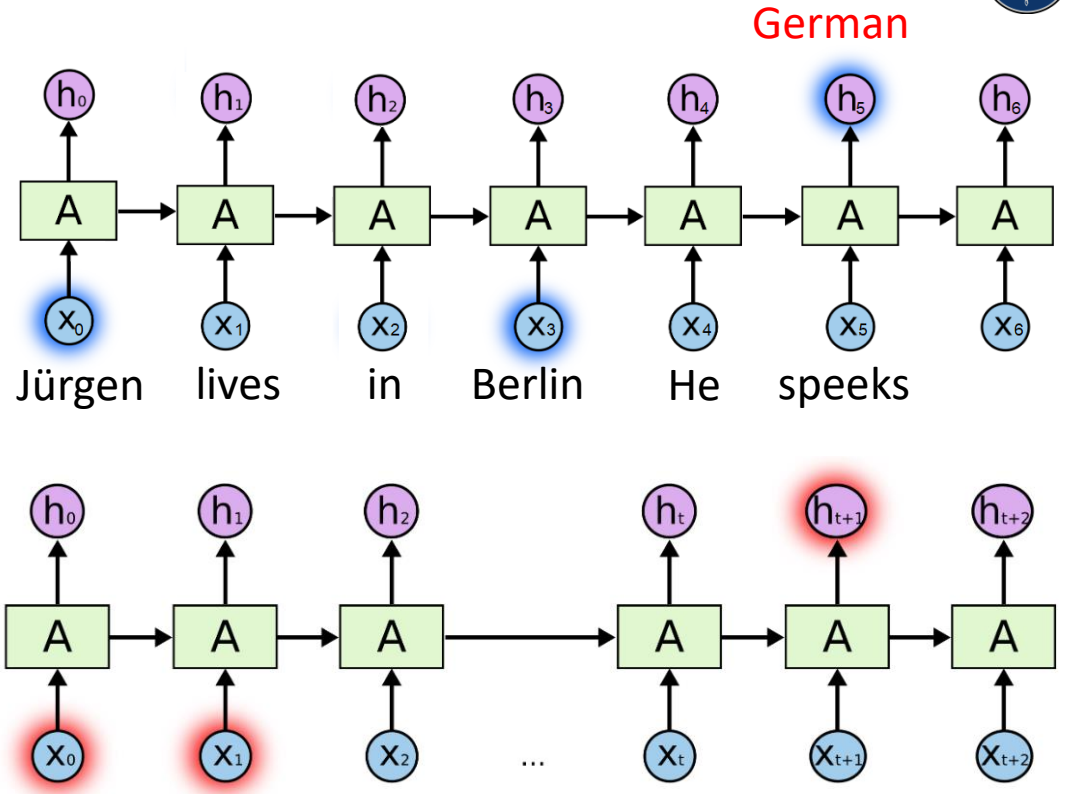
Vanishing Gradient Problem

- In case of long input vector sequencies, the old vectors has a strongly fading effect in inference phase
- In training phase, the stacked gradient functions will be very small



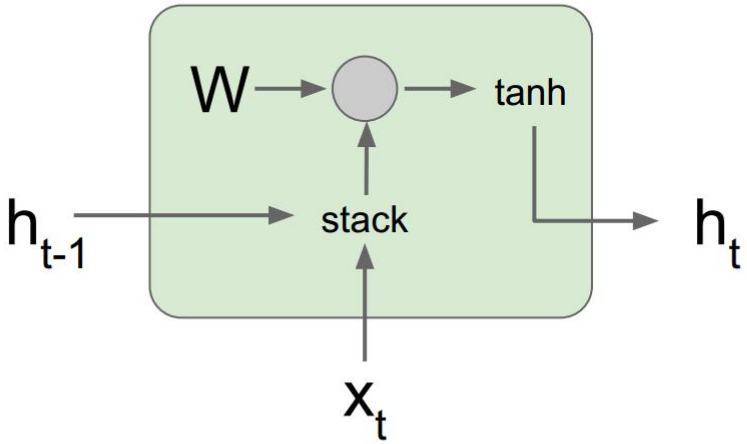
Practical problem of long term dependences

- Consider a network which predicts the next word in a text
  - If the information needed to predict is close, it can be successfully trained
  - If required information is far, the training will be difficult



RNN Gradient flow

Bengio et al, “Learning long-term dependencies with gradient descent is difficult”, IEEE Transactions on Neural Networks, 1994
Pascanu et al, “On the difficulty of training recurrent neural networks”, ICML 2013

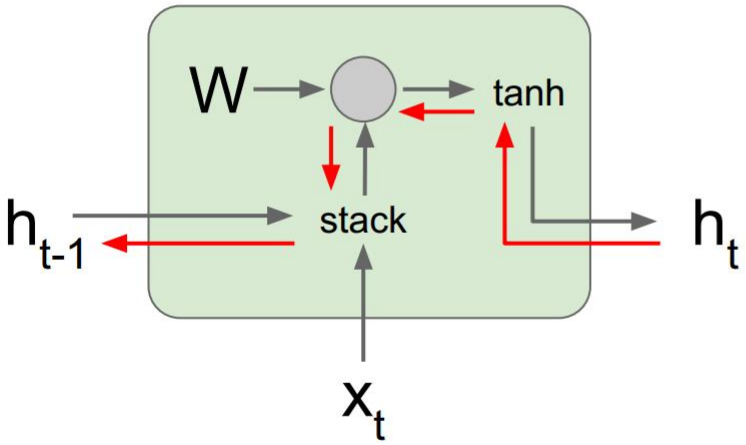


$$
\begin{aligned}
h_t &= \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \\
&= \tanh\left(\begin{pmatrix} W_{hh} & W_{hx} \end{pmatrix} \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right) \\
&= \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right)
\end{aligned}
$$

RNN Gradient flow

Bengio et al, “Learning long-term dependencies with gradient descent is difficult”, IEEE Transactions on Neural Networks, 1994
Pascanu et al, “On the difficulty of training recurrent neural networks”, ICML 2013

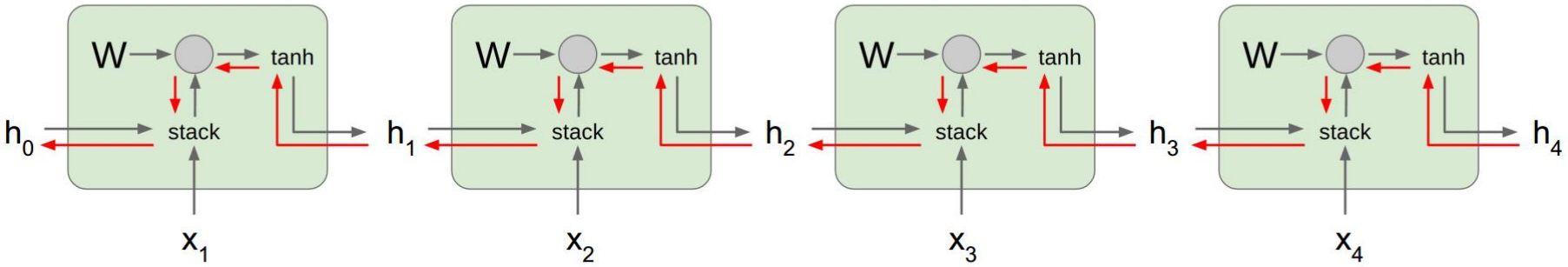
Backpropagation from h_t to h_{t-1} multiplies by W (actually W_{hh}^T)



$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$
$$= \tanh\left((W_{hh} \quad W_{hx})\begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$
$$= \tanh\left(W\begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

RNN Gradient flow

Bengio et al, “Learning long-term dependencies with gradient descent is difficult”, IEEE Transactions on Neural Networks, 1994
Pascanu et al, “On the difficulty of training recurrent neural networks”, ICML 2013



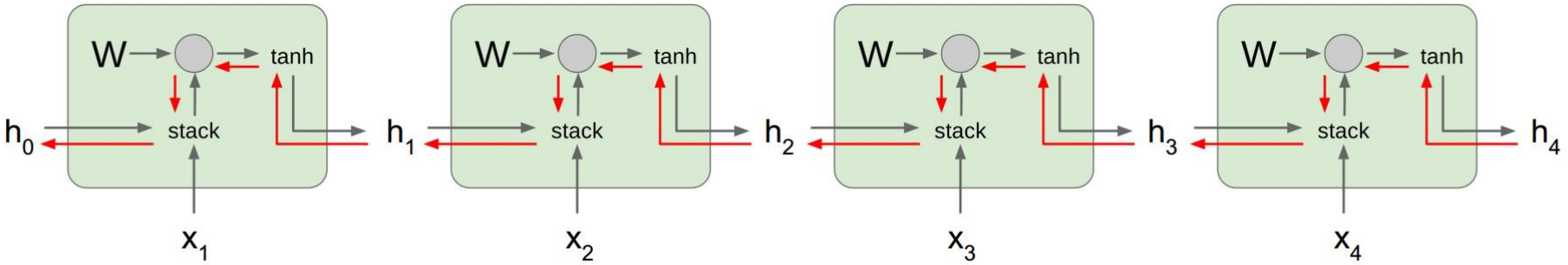
Computing gradient of h_0 involves many factors of W (and repeated tanh)

Largest singular value > 1: **Exploding gradients** → **Gradient clipping**: Scale gradient if its norm is too big

Largest singular value < 1: **Vanishing gradients**

RNN Gradient flow

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



Computing gradient of h_0 involves many factors of W (and repeated tanh)

Largest singular value > 1:
Exploding gradients

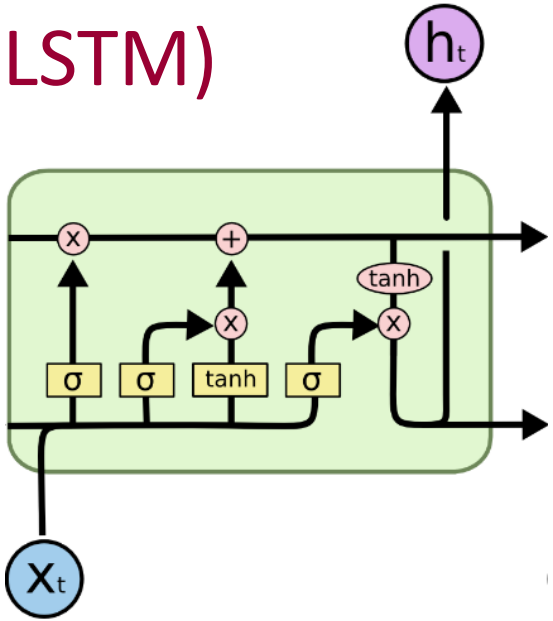
Largest singular value < 1:
Vanishing gradients

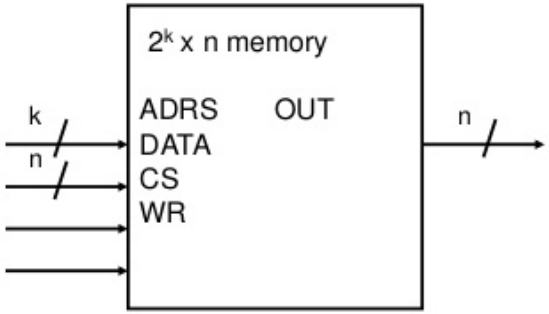
→ Change RNN architecture

Introduction of Long Short Term Memory (LSTM)

Long Short Term Memory (LSTM)

- Was originally introduced Hochreiter & Schmidhuber (1997)
- Idea:
  - To be able to learn long term dependences
  - Collects data when the input is considered to be relevant
  - Keeps it as long as it considers to be important
  - Technique:
    - Handle the state as a memory with minor modifications
      - No matrix multiplication
      - No tanh
      - Apply memory handling kind signals
        - » data in, data out, write, enable

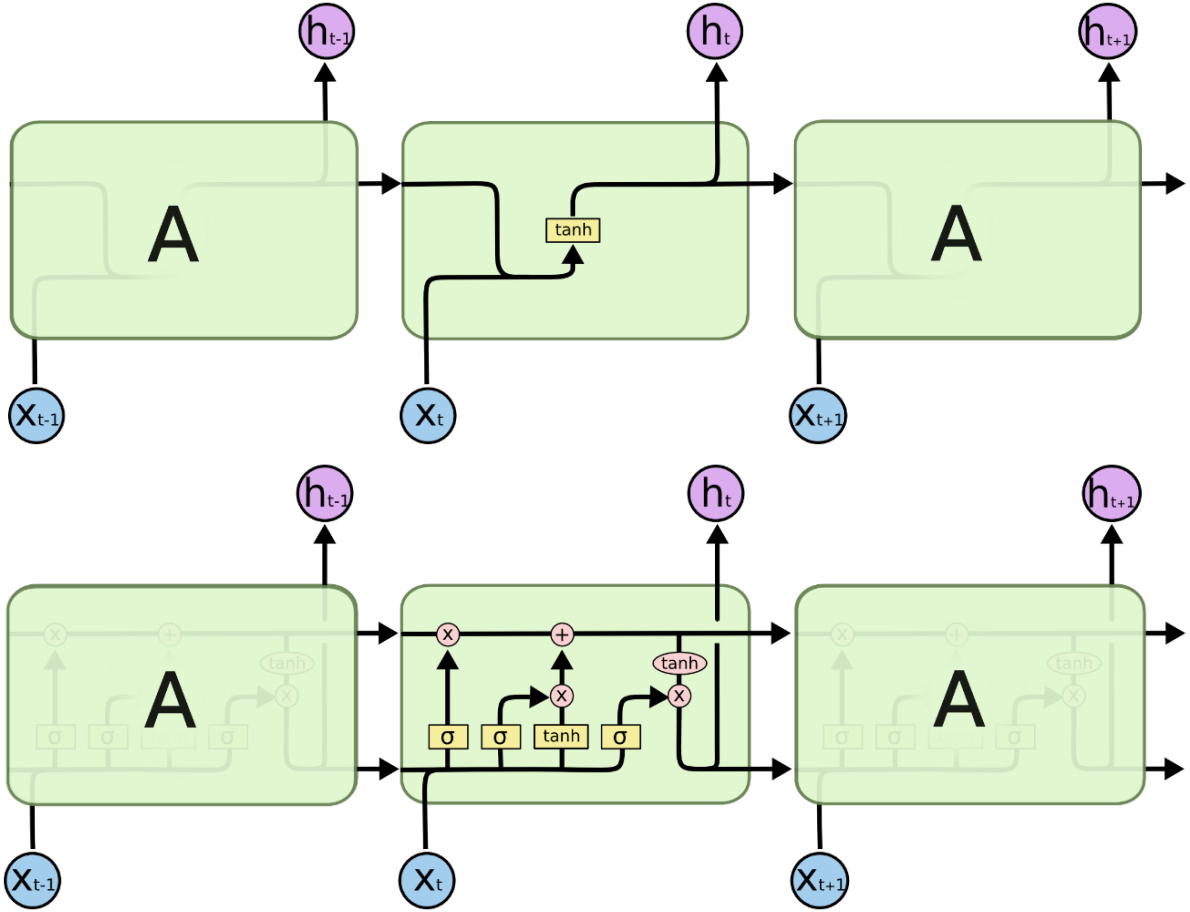




http://colah.github.io/posts/2015-08-Understanding-LSTMs/

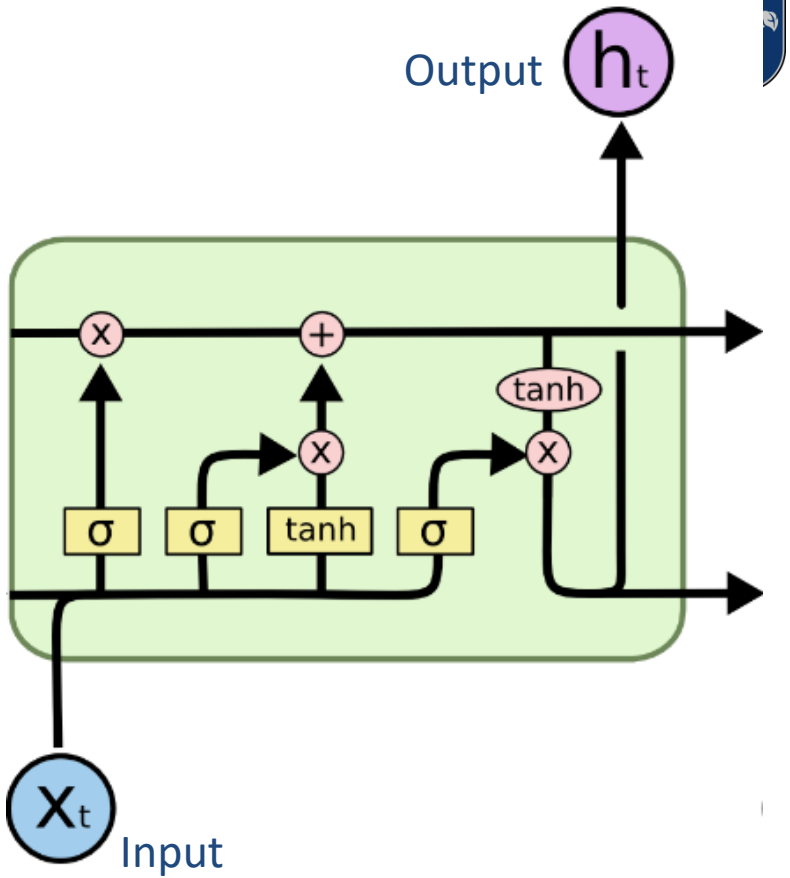
Derivation of LSTM

- Repeating module in Normal RNN
  - concatenates the input and the state
  - A neural network with tanh output and repeats the result
- LSTM
  - Uses the state as a memory
  - Uses 4 neural nets to control the memory
    - Forget, Input, W, Output



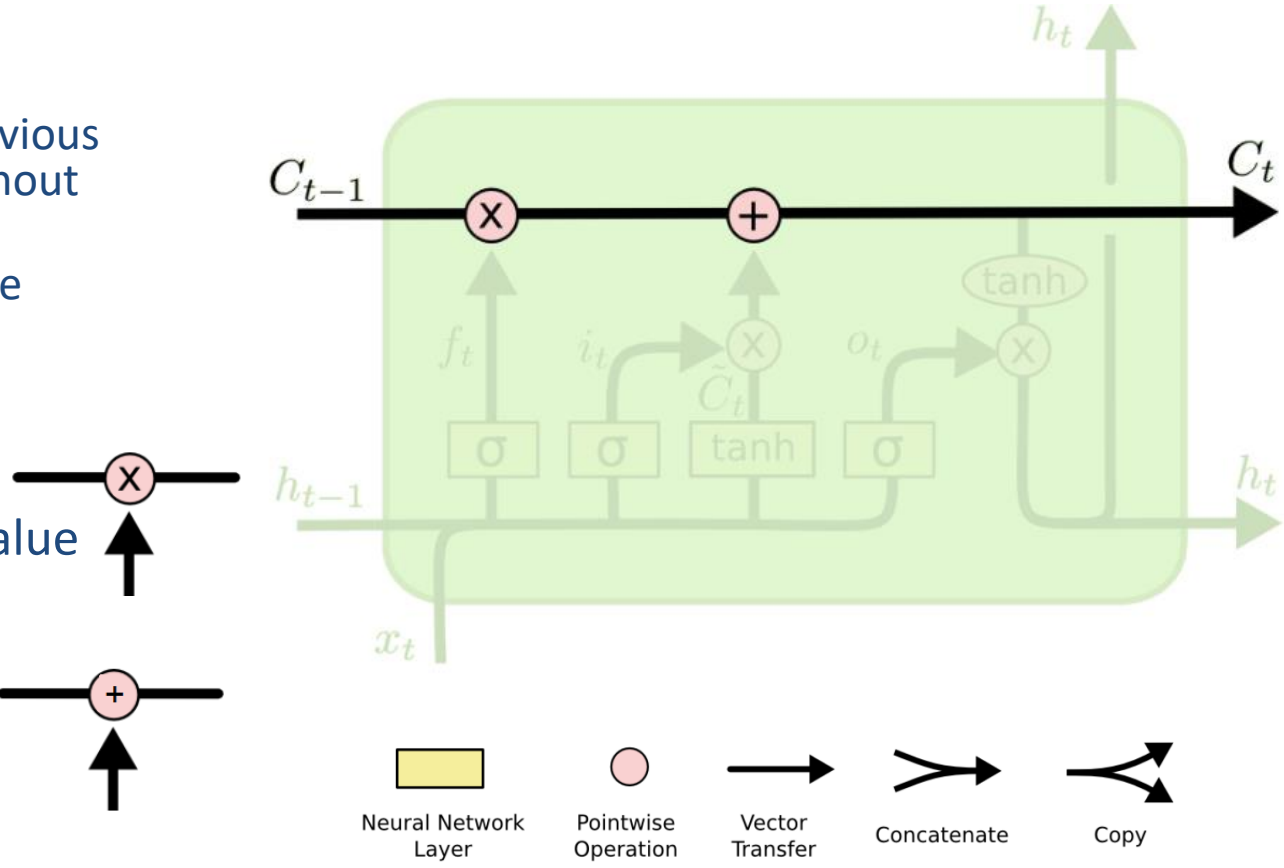
Components of LSTM I

- All wires represents vector
  - Vector transfer
  - Vector concatenation
  - Vector copy
- Neural nets with (yellow boxes)
  - **Multi-layer NN** with *tanh* activation function used for update value calculation
  - **Multi-layer NN** with *logistic* activation function (sigmoid) used for value selection (kind of addressing)
- Pointwise operation (pink circles)
  - Pointwise multifaction
  - Pointwise addition



Components of LSTM II

- State of the LSTM
  - This is the actual memory,
  - It can pass the previous values with or without update
  - Represented by the upper black line
  - Indicated with C_t

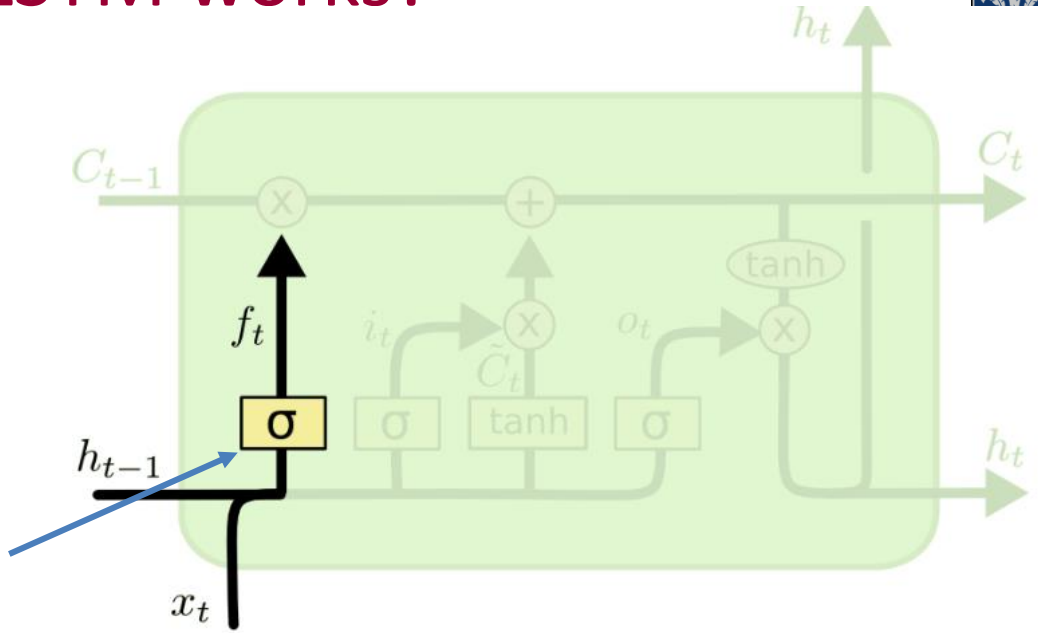


- Old content can be removed value-by-value

- New content can be added

How LSTM works?

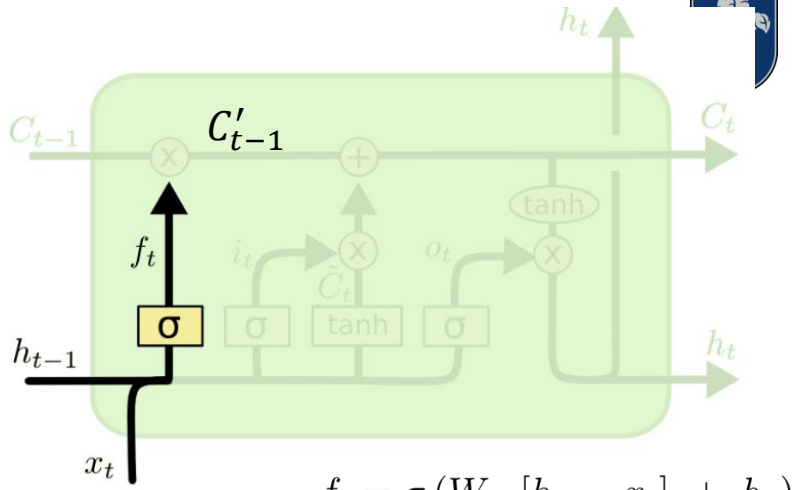
- Step 1
  - Combines input and previous output (concatenation)
  - Selects which values to forget
    - Sort of addressing
    - Done by the **"Forget Gate"**
    - Neural net with sigmoid output



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

Updating state memory (Example)

- Input: "James"
- **Forget Neural** network figures out:
  - Analyzes the concatenated vector
  - Name, Subject of a sentence, Male
- Selects which values to forget and how much
  - Position and weight
- Task:
  - Update gender of the subject (forget the old value)
  - Gender might be represented with a variable
    - c_1: value proportional with the probability that the subject is a male
    - c_2: represents weather
  - Calculate the forget factor of the gender memories
    - 0 completely get rid of it
    - 1 keep the previous value
    - 0 .. 1 partial forget
    - Adressing and suppressing!!!



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

c_1: subject's gender

f1: forget factor of c_1

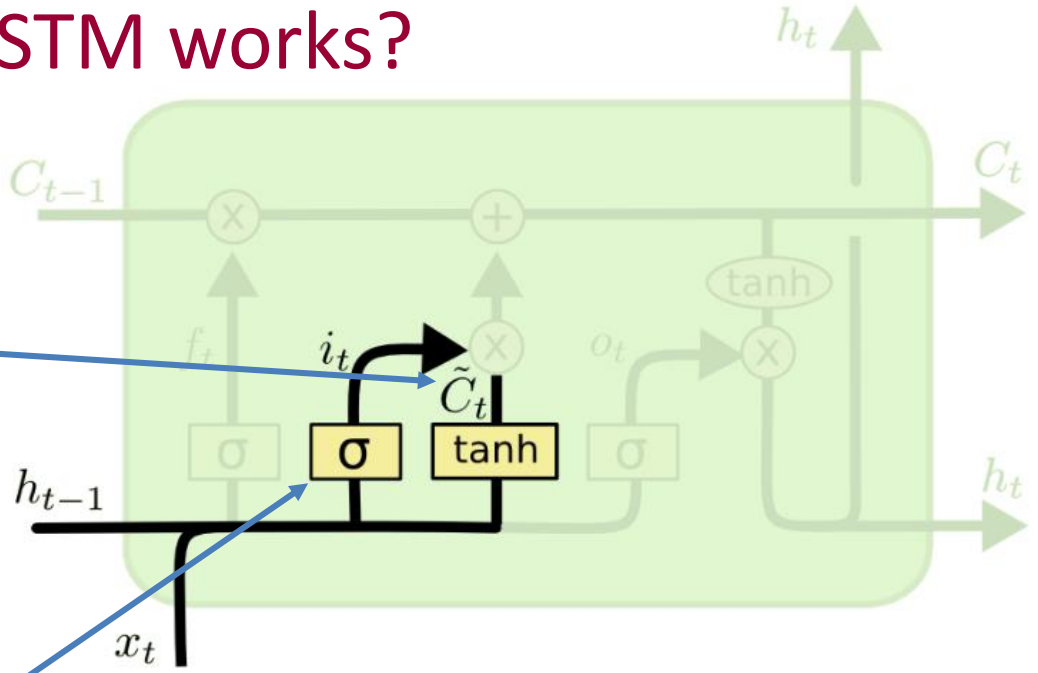
c1 value after partial forget

$$C_{t-1} = \begin{bmatrix} -0.5 \\ 0.2 \\ \vdots \end{bmatrix} \qquad f_t = \begin{bmatrix} 0.1 \\ 1 \\ \vdots \end{bmatrix} \qquad C'_{t-1} = \begin{bmatrix} -0.05 \\ 0.2 \\ \vdots \end{bmatrix}$$

Not to forget c_2

How LSTM works?

- Step 2
  - Calculation of the state update
    - Done by the **"Cell Network"**
    - Not yet the new value, only the update value
    - Neural Net with *tanh*
  - Selection of the state values to be updates (Addressing)
    - Done by the **"Input Gate"**
    - Neural Net with sigmoid
  -

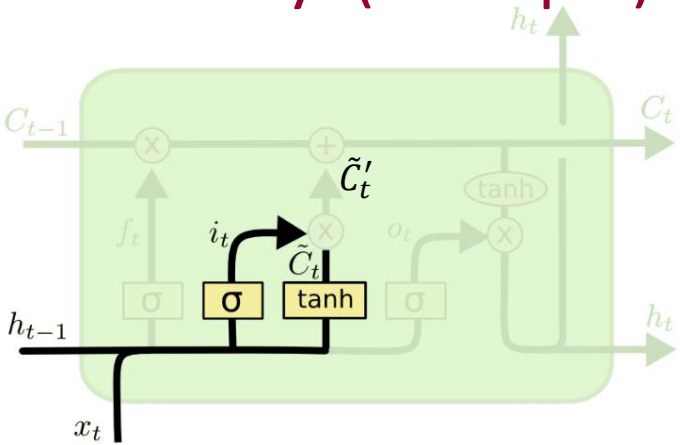


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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

Updating state memory (Example)

- Input: “James”
- Input Gate figures out:
  - Analyze the concatenated vector
  - Select which values to update (ENABLE!!!)
  - Calculate the update weights
- Cell Network calculates:
  - The update values
- Task:
  - Update gender of the subject (calculate the update value)
  - Gender might be represented with a variable
    - c_1: value proportional with the probability that the gender is male
    - c_2: represents weather
  - Calculate the update factor of the gender memories
    - 0 not to update
    - 1 fully update
    - 0 .. 1 partial update
    - ADRESSING!!!



James

c_1: subject gender estimate value

f1: update factor of c_1

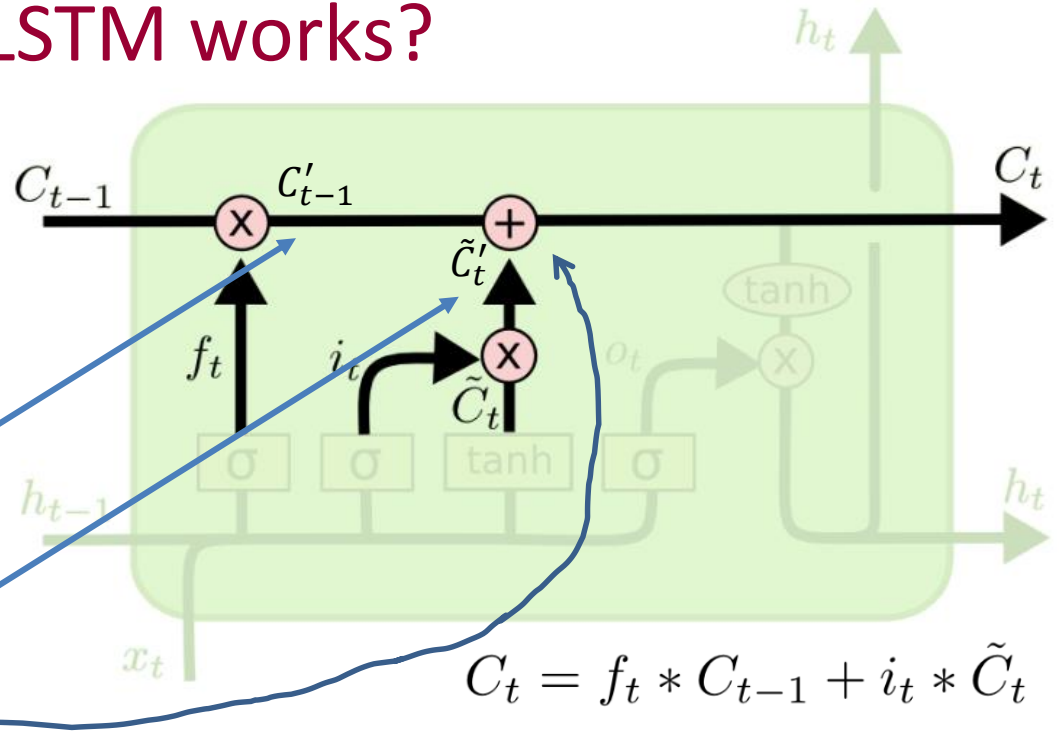
c1 update value

$$\tilde{C}_t = \begin{bmatrix} 0.9 \\ -0.75 \\ \vdots \end{bmatrix} \quad i_t = \begin{bmatrix} 0.8 \\ 0 \\ \vdots \end{bmatrix} \quad \tilde{C}'_t = \begin{bmatrix} 0.72 \\ 0 \\ \vdots \end{bmatrix}$$

Not to modify c_2

How LSTM works?

- Step 3
  - Calculation of the state update
    - The old state
      - With the forgotten values in the vector
    - And the state update
      - With update vector
    - **Are added up**



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

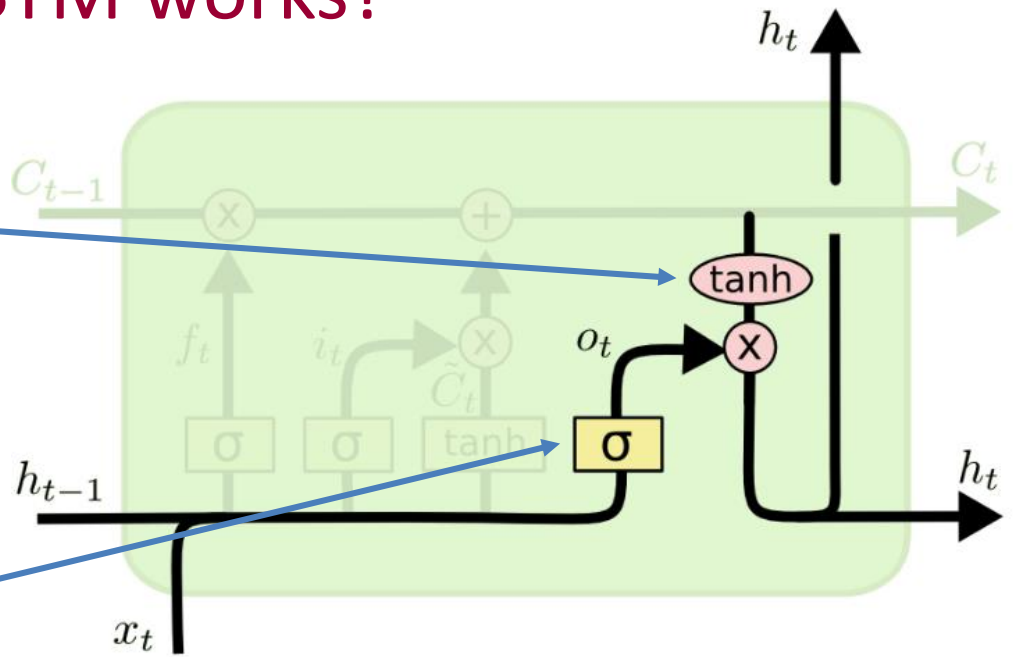
$$C_t = C'_{t-1} + \tilde{C}'_t = \begin{bmatrix} -0.05 \\ 0.2 \\ \vdots \end{bmatrix} + \begin{bmatrix} 0.72 \\ 0 \\ \vdots \end{bmatrix} = \begin{bmatrix} 0.67 \\ 0.2 \\ \vdots \end{bmatrix}$$

c_1: subject gender's estimate value update

c_2: (weather) unchaged

How LSTM works?

- Step 4
  - Apply activation function to the output
    - Squeeze the values between -1 and +1
    - Done by *tanh* activation function
  - Selection of the new output values (Addressing)
    - Done by the **"Output Gate"**
    - Not all the state value is released in each step
    - **Output Gate** decides which values are relevant in this step



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

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

Output vector can be sparse

- Output gate might enables
  - All values of C_t
  - Fraction of C_t (sparse)
  - None of C_t
- C_t can be sparse

$$tanh(C_t) = \begin{bmatrix} 0.2 \\ 0.98 \\ -0.97 \\ -0.1 \\ 0.98 \\ 0.8 \\ 0.2 \\ 0.3 \\ -0.99 \\ 0.8 \\ 0.7 \end{bmatrix}$$

Values are bounded

$$o_t = \begin{bmatrix} 0.01 \\ 0.85 \\ 0.75 \\ 0.1 \\ 0.2 \\ 0.8 \\ 0.1 \\ 0.1 \\ 0.02 \\ 0.9 \\ 0.8 \end{bmatrix}$$

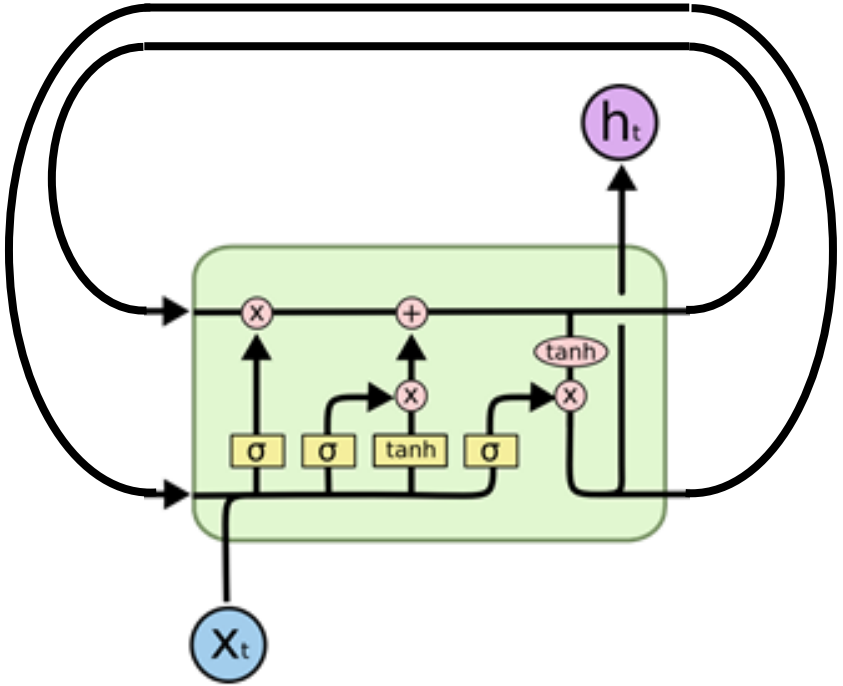
Enabling factor: Enabled values are red

$$h_t = o_t * tanh(C_t) = \begin{bmatrix} 0.002 \\ 0.83 \\ -0.73 \\ -0.01 \\ 0.2 \\ 0.64 \\ 0.02 \\ 0.03 \\ -0.02 \\ 0.72 \\ 0.63 \end{bmatrix}$$

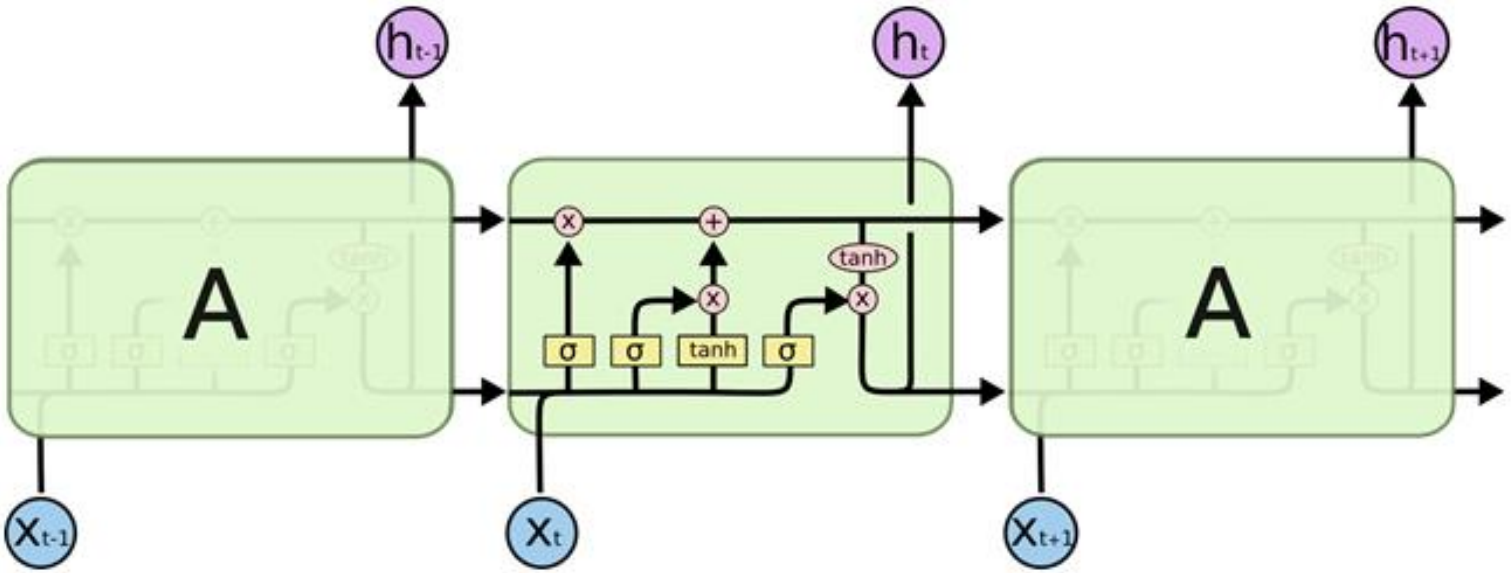
Output vector: Enabled values are red Disabled values (gray) will appear on the output, but with reduced values

LSTM network

- General form of an LSTM network

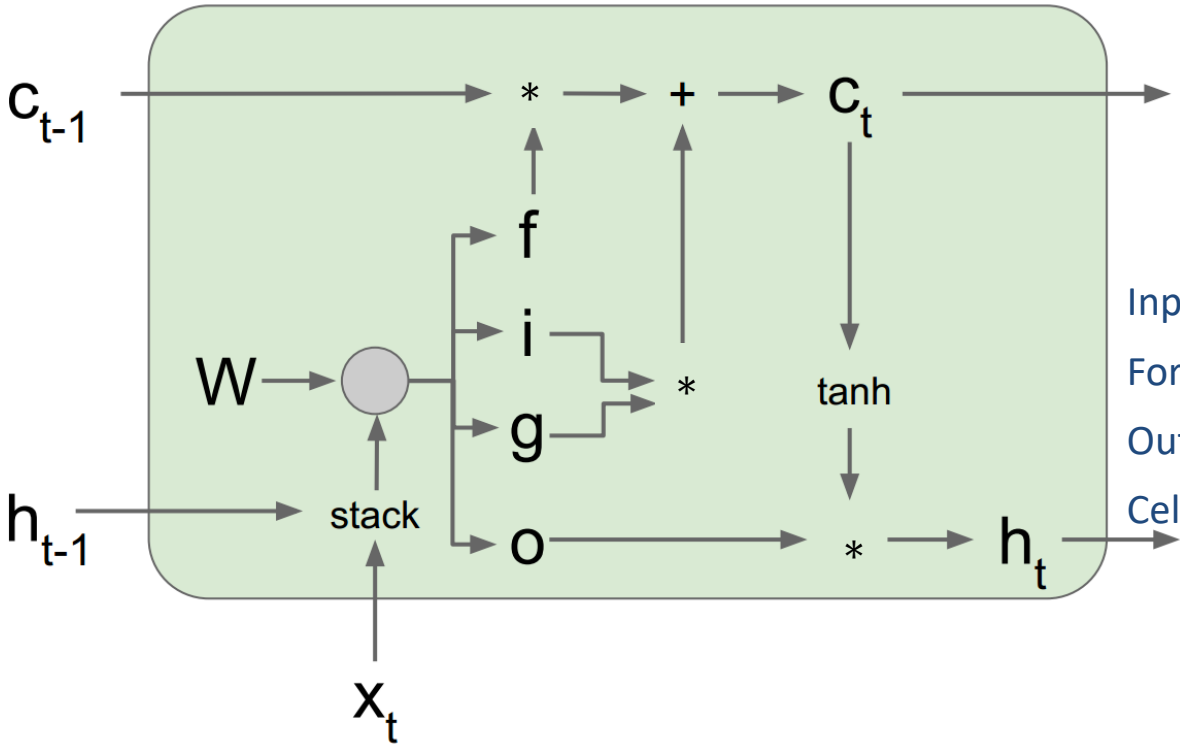


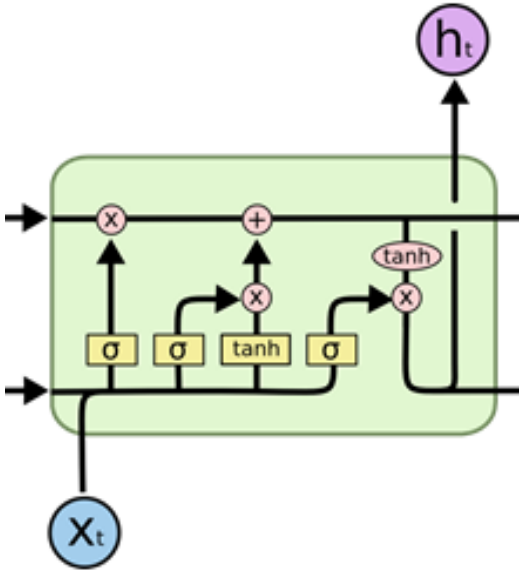
Unrolling LSTM network



Gradient calculation in LSTM

Reformulating equations





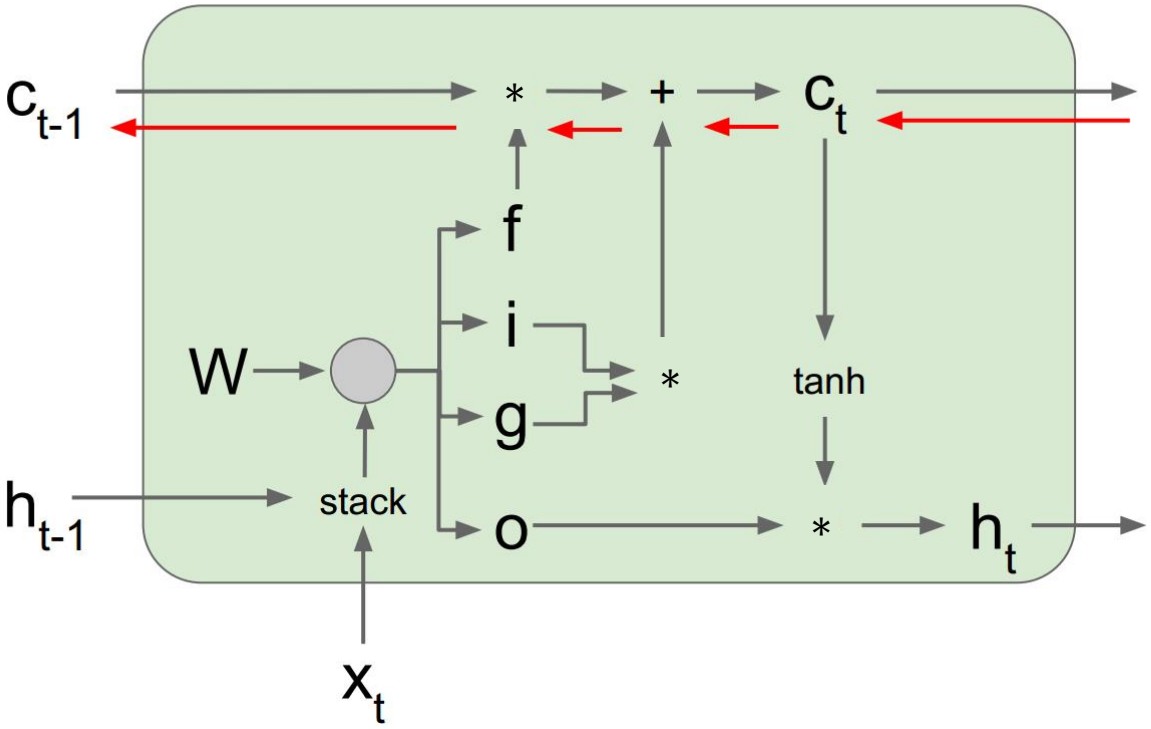
Input
Forget
Output
Cell Net

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f * c_{t-1} + i * g$$

$$h_t = o * \tanh(c_t)$$

Gradient calculation in LSTM



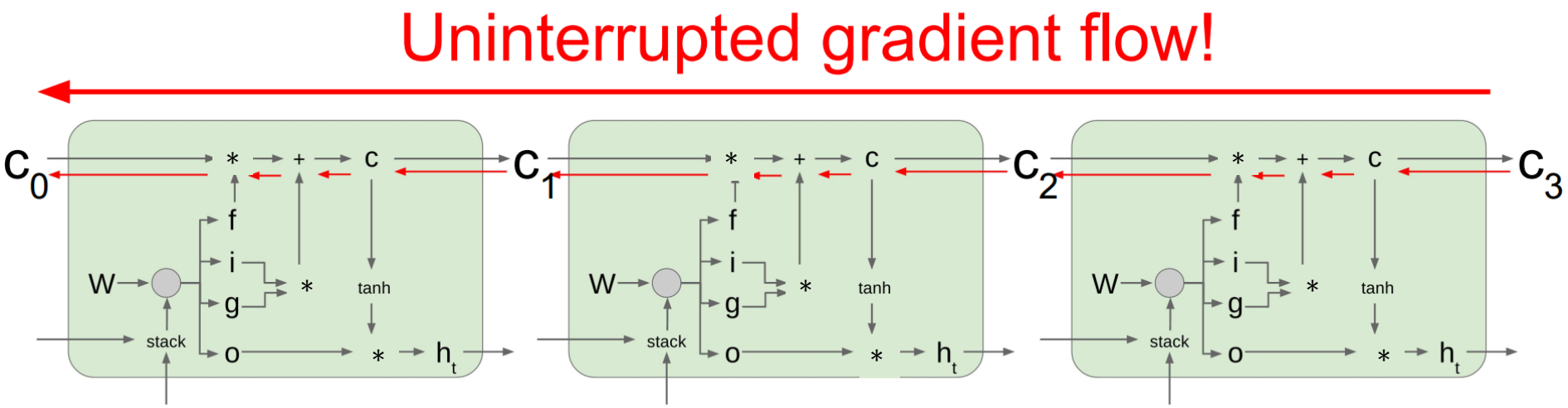
Backpropagation from c_t to c_{t-1} only elementwise multiplication by f, no matrix multiply by W

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f * c_{t-1} + i * g$$

$$h_t = o * \tanh(c_t)$$

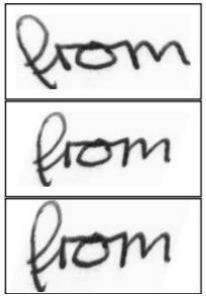
Gradient calculation in LSTM



- Though we multiply the memory content with a smaller than 1 number
- And the W matrix is part of the memory update
- But it still preserves the content for longer time
- As it comes from the name: It is a elongated time **short term** memory

Achevements with LSTM networks

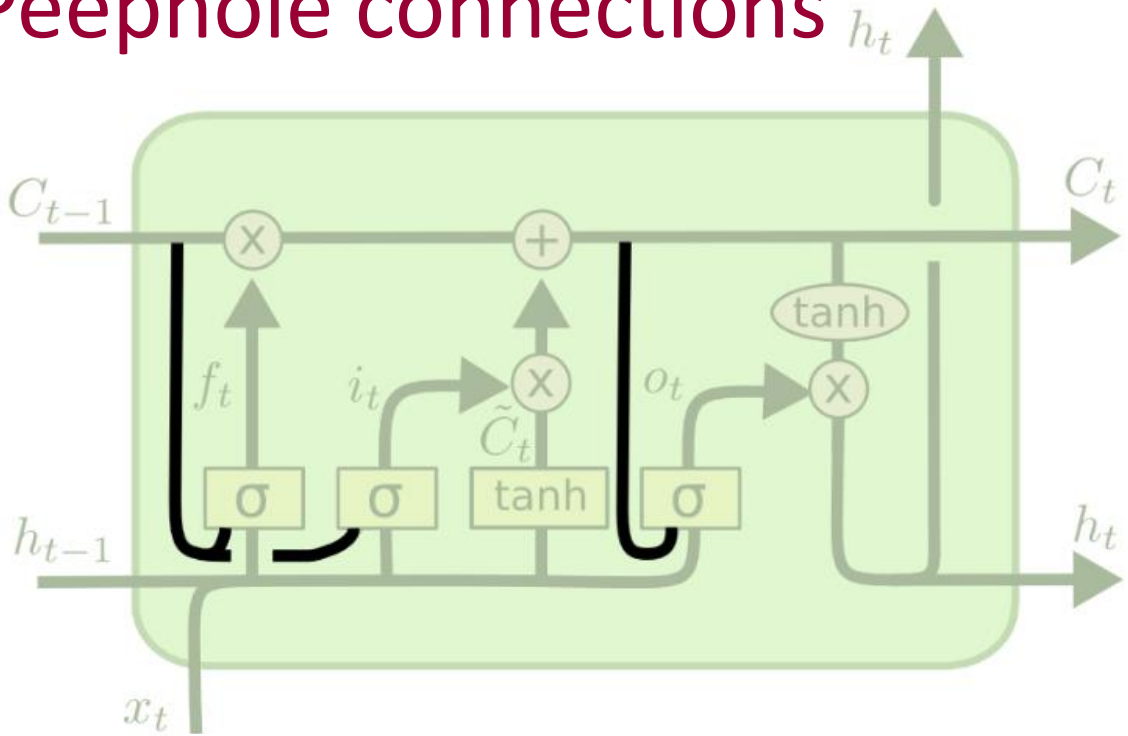
- Record results in natural language text compression
- Unsegmented connected handwriting recognition
- Natural speech recognition
- Smart voice assistants
  - Google Translate
  - Amazon Alexa
  - Microsoft Cortana
  - Apple Quicktype



- 95.1% recognition accuracy on the Switchboard corpus, incorporating a vocabulary of 165,000 words
  - Continuous spontaneous English native speech

Variants of LSTM I : Peephole connections

- Introduced by Gers & Schmidhuber (2000)
- All the three gates receives input from the previous state and the input
- Since output can be sparse this version has more information for gating
  - addressing and weighting



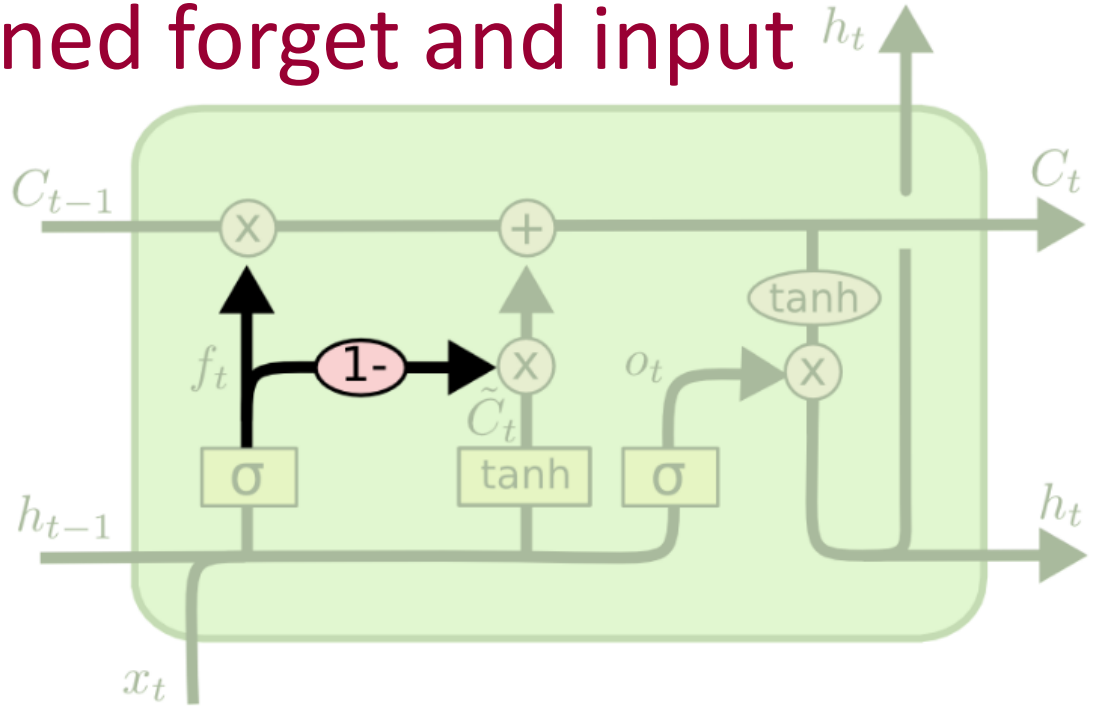
$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_f\right)$$

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

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

Variants of LSTM II : Joined forget and input

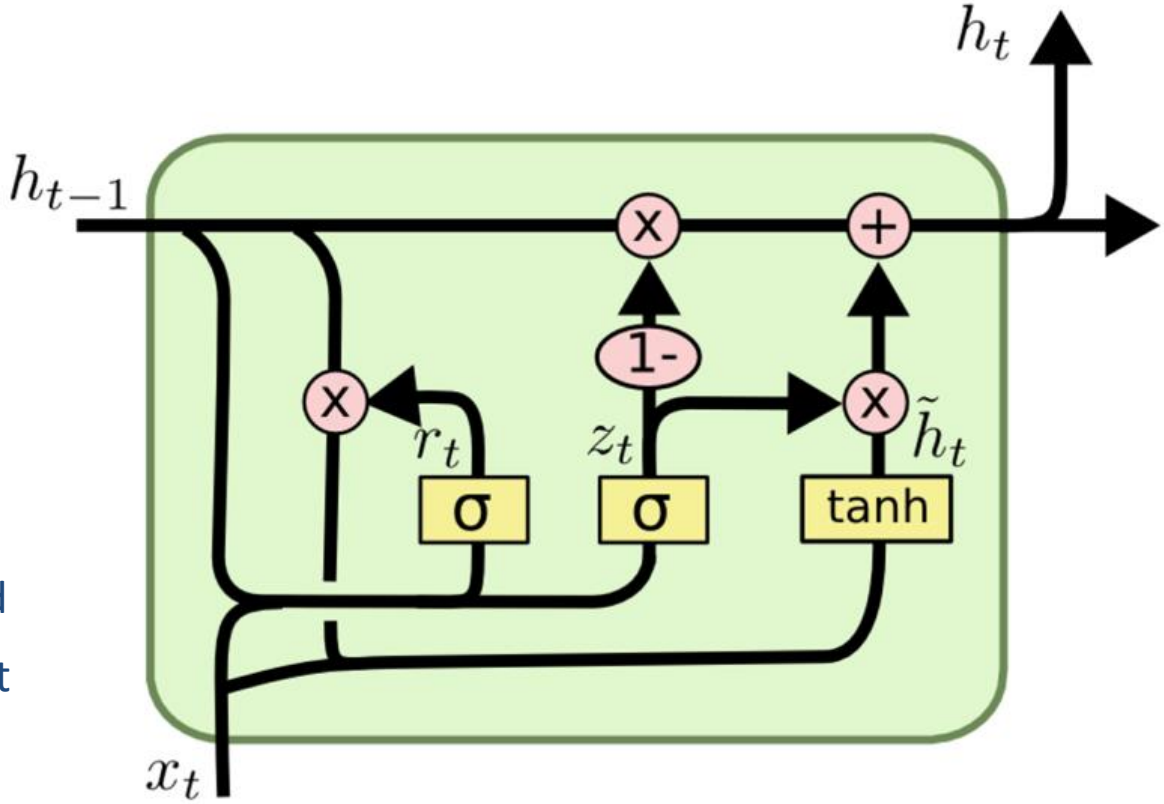
- Input and forget gates has practically the same role
- Why not to join them?



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

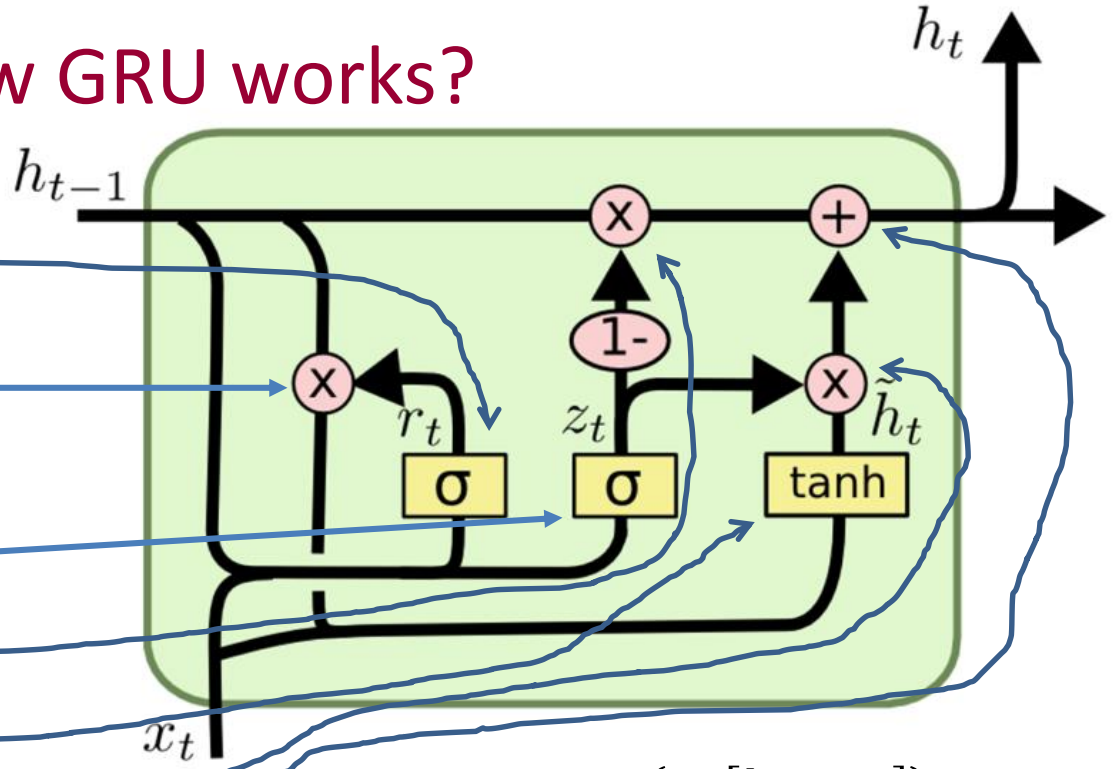
Gated Recurrent Unit (GRU)

- Another variant of LSTM
- Introduced by Kyunghyun Cho (2014)
- There is no separate State and Output
- Only three neural nets
- At GRU the output will not be sparse (not gated)
- Similar performance in music and speech signal modelling and
- Learns faster for smaller data set



How GRU works?

- Concatenate h_{t-1} and x_t
- Calculate the Input Gate
- Suppress the values to be forgotten in h_{t-1} (get sparse memory vector)
- Calculate the joint Forgot and output Gates
- Gate h_{t-1}
- Calculate function of the Cell Network
- Gate $\tilde{h}_{t-1}$
- Calculate the new output (h_t)



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

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

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

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