Recurrent Neural Networks

DLAI – MARTA R. COSTA-JUSSÀ

Outline

1. The importance of context. Sequence modeling
2. Feed-forward networks review
3. Vanilla RNN
4. Vanishing gradient
5. Gating methodology
6. Other RNN extensions
7. Use case

1.The importance of context

The importance of context. Sequence modeling

- Recall the 5th digit of your phone number
- Sing your favourite song beginning at third sentence
- Recall 10th character of the alphabet

Probably you went straight from the beginning of the stream in each case…

because in sequences order matters!

Idea: retain the information preserving the importance of order

Language Applications

- Language Modeling (probability)
- Machine Translation
- Speech Recognition

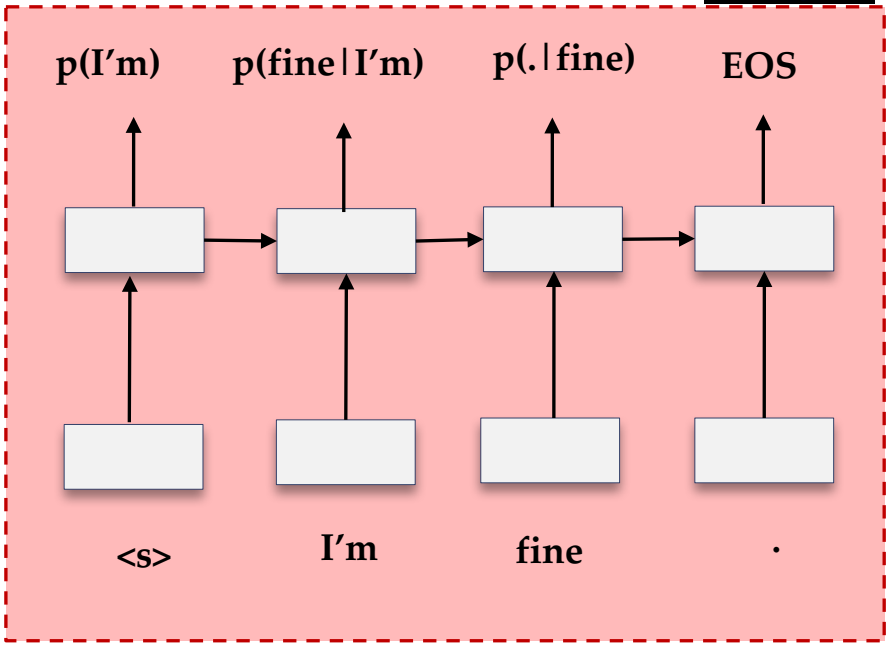


Image Applications

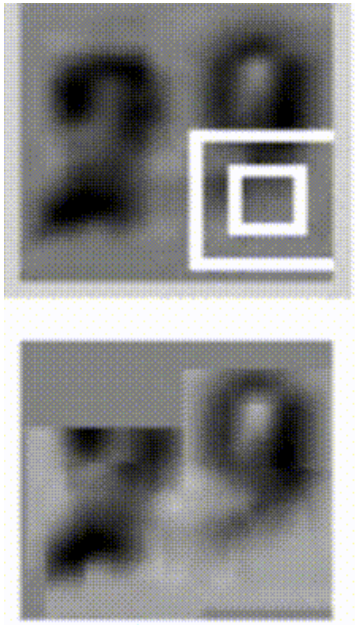


Image credit: The Unreasonable Effectiveness of RNNs, André Karpathy

2. Feedforward Network Review

Feedforward Networks

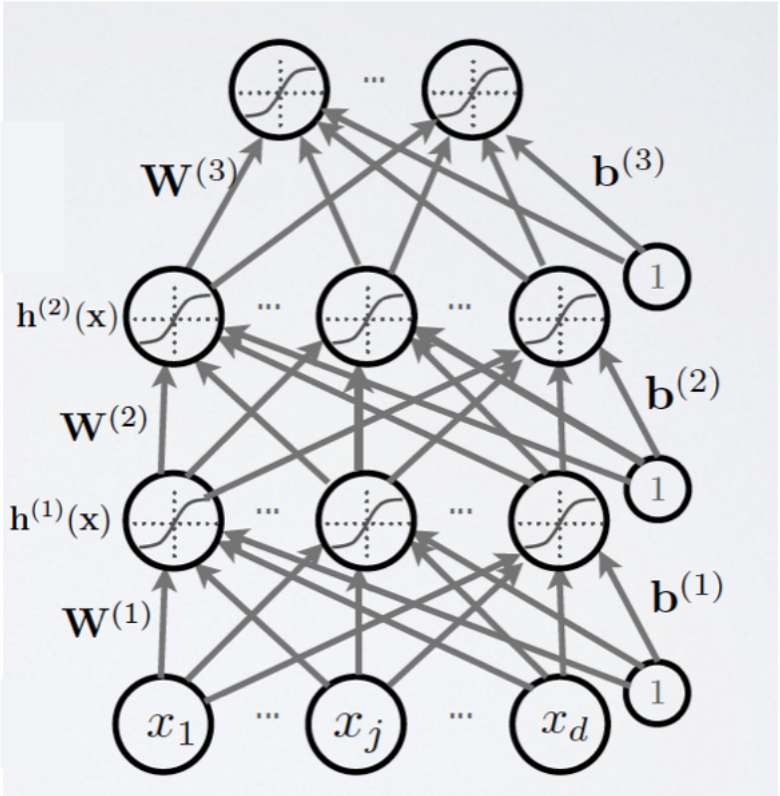
Every **y/hi** is computed from the sequence of forward activations out of input **x**.

$$y = f(W_3 \cdot h_2 + b_3)$$

$$h_2 = f(W_2 \cdot h_1 + b_2)$$

$$h_1 = f(W_1 \cdot x + b_1)$$

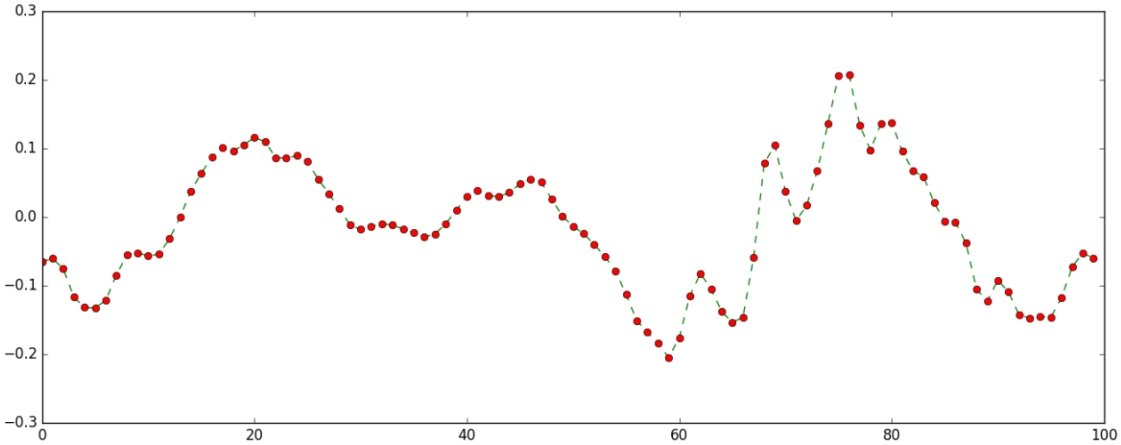
Feed FORWARD



Slide Credit: Hugo Laroche NN course

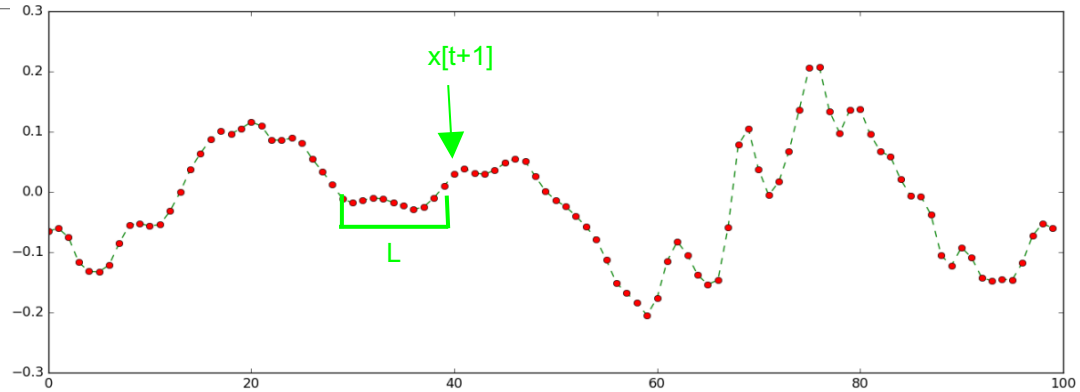
Where is the Memory I?

If we have a sequence of samples...



predict sample x[t+1] knowing previous values {x[t], x[t-1], x[t-2], …, x[t-τ]}

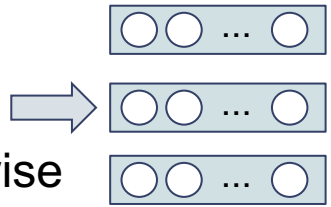
Where is the Memory II?



x[t+1]

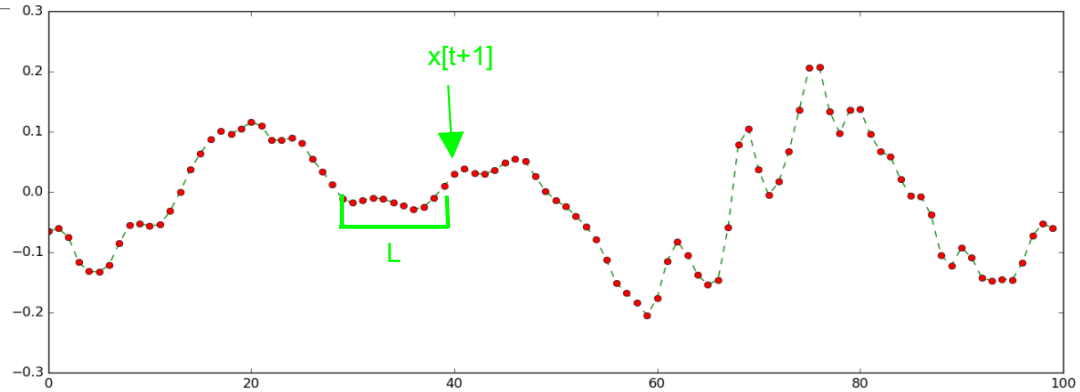
Feed Forward approach:

- static window of size L
- slide the window time-step wise



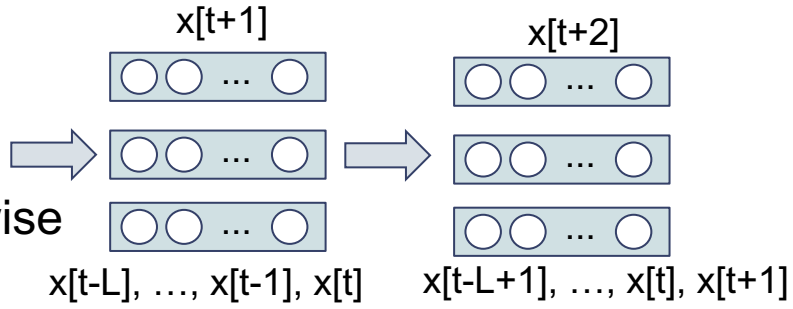
x[t-L], …, x[t-1], x[t]

Where is the Memory III?

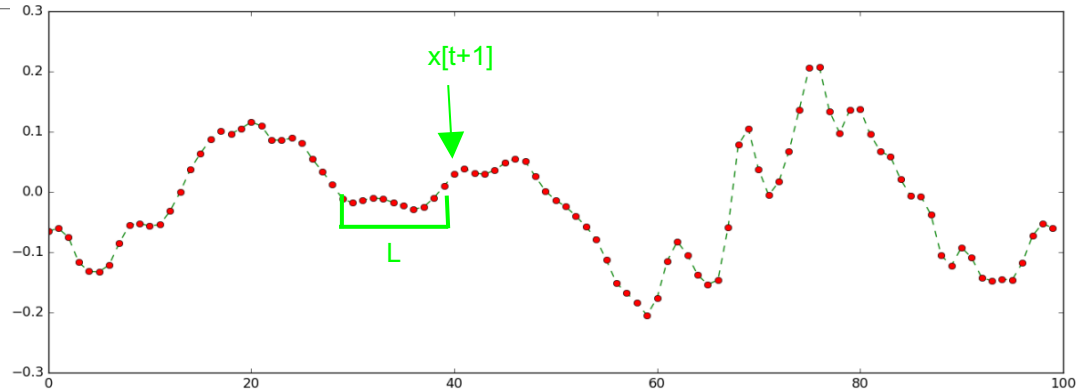


Feed Forward approach:

- static window of size L
- slide the window time-step wise

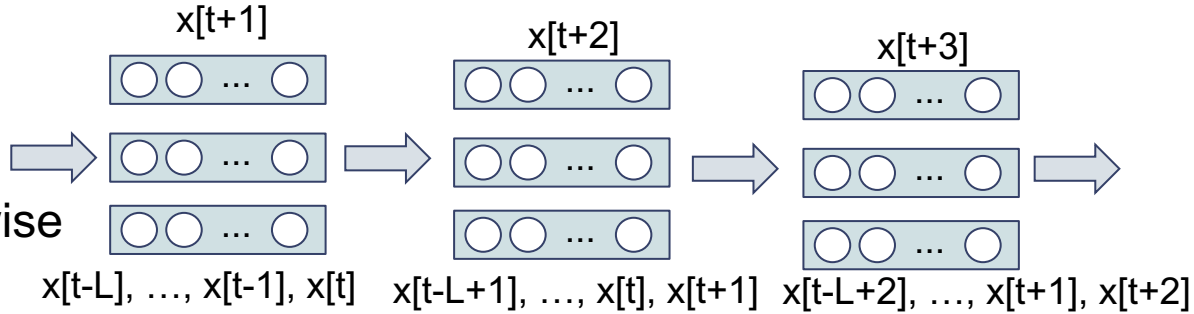


Where is the Memory IV?



Feed Forward approach:

- static window of size L
- slide the window time-step wise



Any problems with this approach?

Problems for the FNN + static window approach I

- If increasing L, fast growth of num of parameters!

Problems for the FNN + static window approach II

- If increasing L, fast growth of num of parameters!
- Decisions are independent between **time-steps**!
  - The network doesn't care about what happened at previous time-step, only present window matters → doesn't look good

TIME-STEPS: the memory do you want to include in your network.
If you want your network to have memory of 60 characters, this number should be 60.

Problems for the FNN + static window approach III

- If increasing L, fast growth of num of parameters!
- Decisions are independent between time-steps!
  - The network doesn't care about what happened at previous time-step, only present window matters → doesn't look good
- Cumbersome padding when there are not enough samples to fill L size
  - Can't work with variable sequence lengths

3. Vanilla RNN

Recurrent Neural Network (RNN) adding the “temporal” evolution

Allow to build specific connections capturing ”history”

$$h_t = f(W \cdot x_t + U \cdot h_{t-1} + b)$$

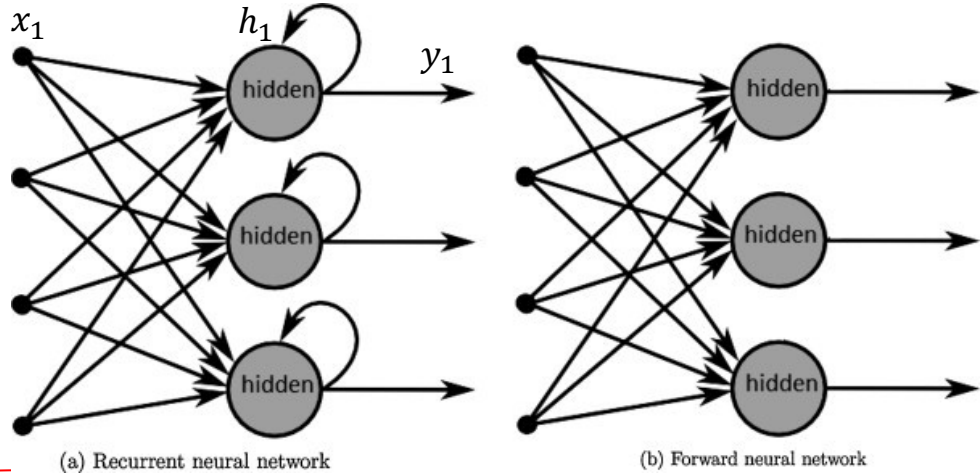
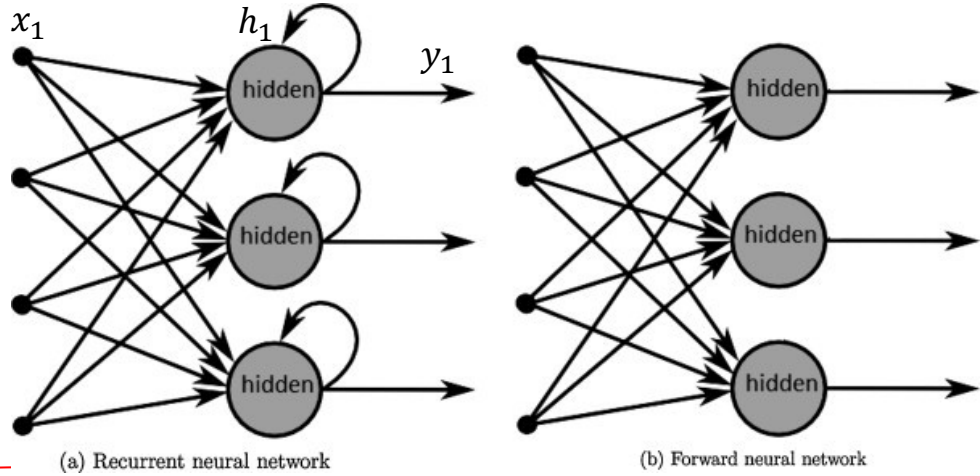


Image src: https://medium.com/towards-data-science/introduction-to-recurrent-neural-network-27202c3945f3

Recurrent Neural Network (RNN) adding the “temporal” evolution

Allow to build specific connections capturing ”history”

$$h_t = f(W \cdot x_t + U \cdot h_{t-1} + b)$$

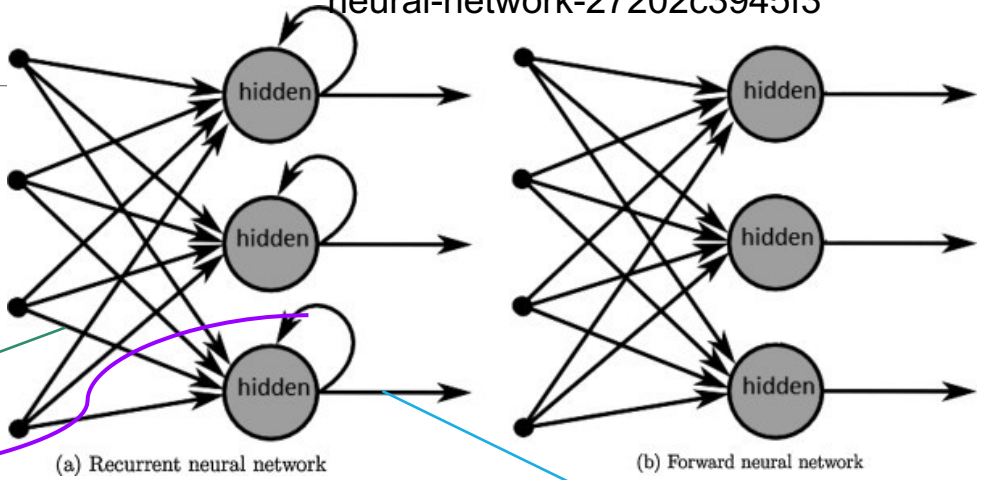


$$y_t = softmax(Vh_t)$$

Image src: https://medium.com/towards-data-science/introduction-to-recurrent-neural-network-27202c3945f3

RNN: parameters

Image src: https://medium.com/towards-data-science/introduction-to-recurrent-neural-network-27202c3945f3



(a) Recurrent neural network

(b) Forward neural network

$$h_t = f(W \cdot x_t + U \cdot h_{t-1} + b)$$

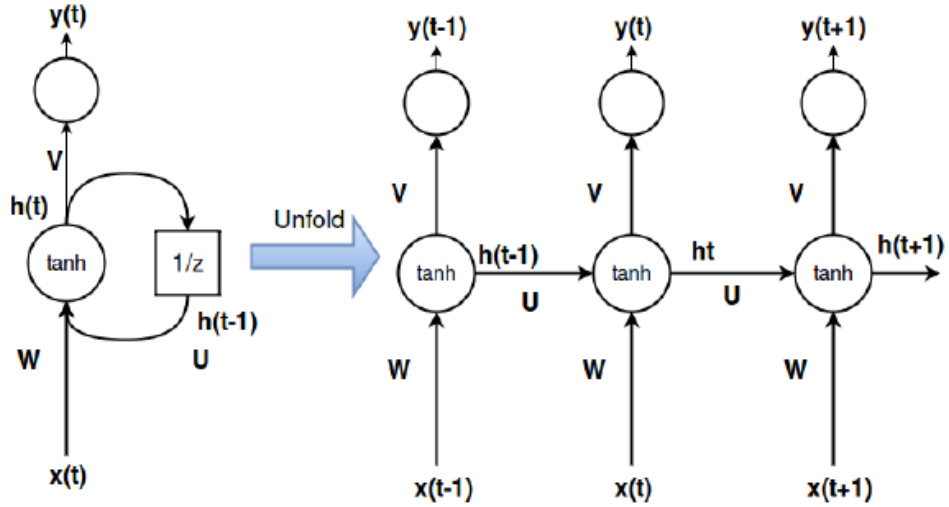
$$y_t = softmax(Vh_t)$$

$$N^i_{params} = N^i_{inputs} \times N^i_{units} + N^i_{units} \times N^i_{units} + N^i_{units}$$

$$N^i_{units} \times N^i_{outputs}$$

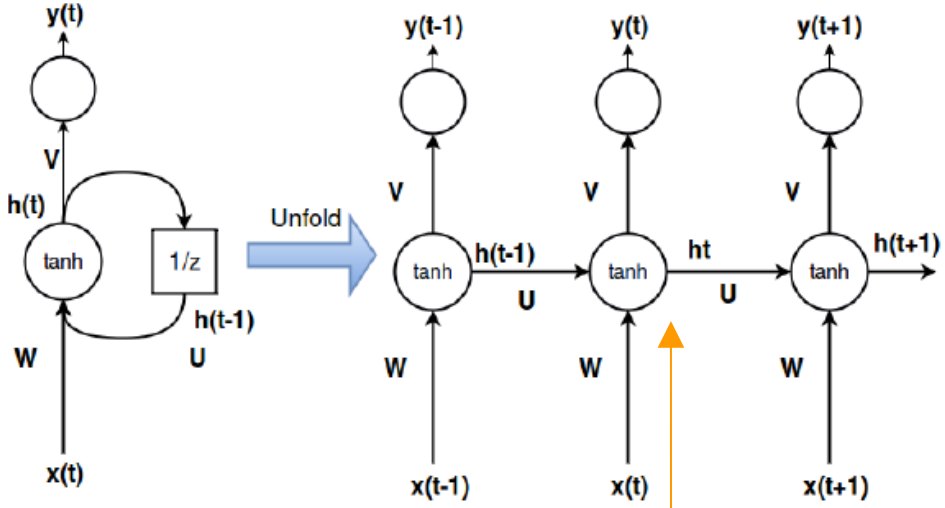
RNN: unfolding

BEWARE: We have extra depth now! Every time-step is an extra level of depth (as a deeper stack of layers in a feed-forward fashion!)



RNN: depth 1

Forward in space propagation

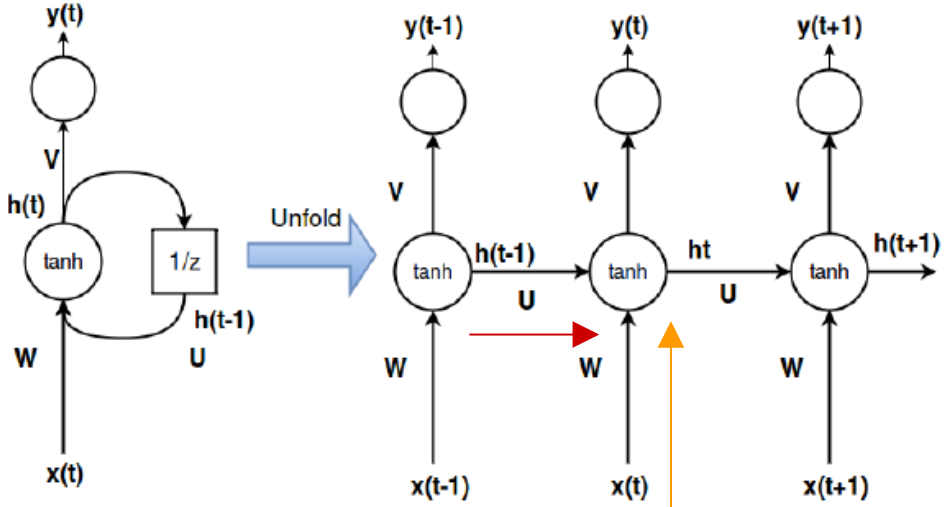


$$\mathbf{h}_t = g(\mathbf{W} \cdot \mathbf{x}_t + \mathbf{U} \cdot \mathbf{h}_{t-1} + \mathbf{b}_h)$$

$\mathbf{x}_t \in \mathbb{R}^n$

RNN: depth 2

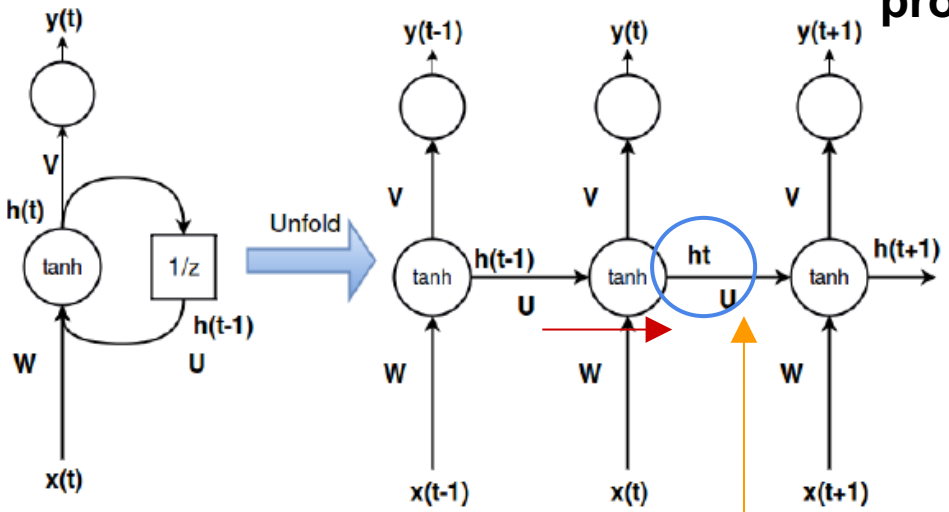
Forward in time propagation



$$\mathbf{h}_t = g(\mathbf{W} \cdot \mathbf{x}_t + \mathbf{U} \cdot \mathbf{h}_{t-1} + \mathbf{b}_h)$$

$\mathbf{x}_t \in \mathbb{R}^n$

RNN: depth in space + time



Hence we have two data flows: **Forward in space + time** propagation: **2 projections per layer activation**

Last time-step includes the context of our decisions recursively

W, U, V shared across all steps

$$\mathbf{h}_t = g(\mathbf{W} \cdot \mathbf{x}_t + \mathbf{U} \cdot \mathbf{h}_{t-1} + \mathbf{b}_h)$$

$$\boldsymbol{y_t = softmax(V h_t)}$$

$\mathbf{x}_t \in \mathbb{R}^n$

Alternatives of recurrence

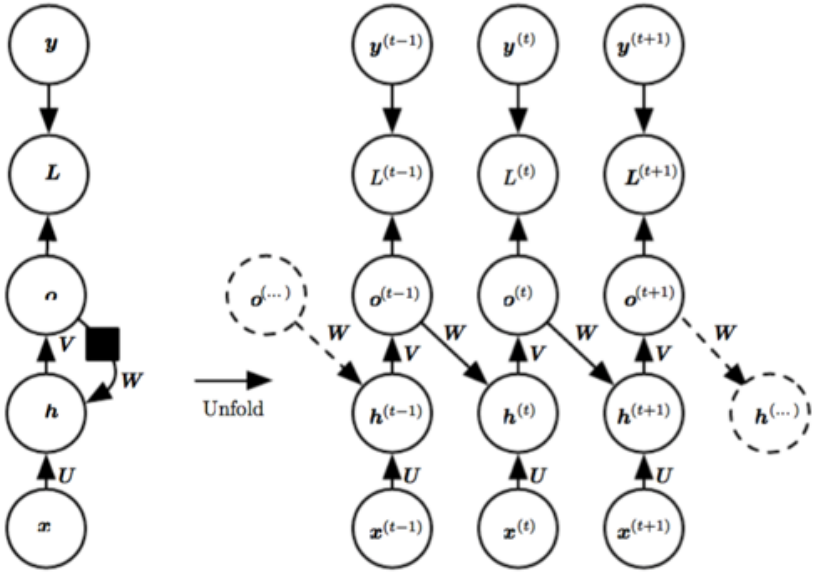


Image src: Goodfellow et al. Deep Learning. The MIT Press

Alternatives of outputs

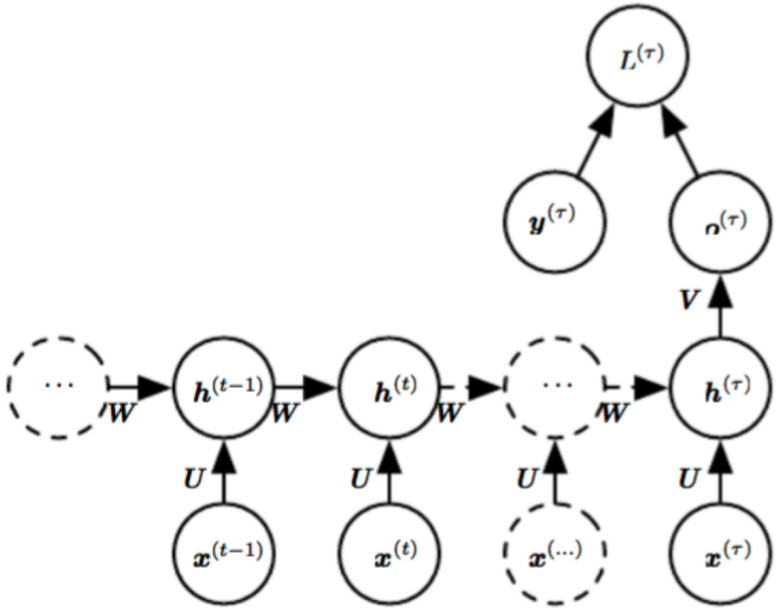


Image src: Goodfellow et al. Deep Learning. The MIT Press

Training a RNN I: BPTT

Backpropagation through time (BPTT): The training algorithm for updating network weights to minimize error including time

Remember BackPropagation

1. Present a training input pattern and propagate it through the network to get an output.
2. Compare the predicted outputs to the expected outputs and calculate the error.
3. Calculate the derivatives of the error with respect to the network weights.
4. Adjust the weights to minimize the error.
5. Repeat.

BPTT I

Cross Entropy Loss

$$\boldsymbol{h_t} = f(\boldsymbol{W} \cdot \boldsymbol{x_t} + \boldsymbol{U} \cdot \boldsymbol{h_{t-1}} + \boldsymbol{b})$$

$$y_t = softmax(Vh_t)$$

$$\begin{aligned} E_t(y_t, \hat{y}_t) &= -y_t \log \hat{y}_t \\ E(y, \hat{y}) &= \sum_t E_t(y_t, \hat{y}_t) \\ &= -\sum_t y_t \log \hat{y}_t \end{aligned}$$

BPTT II

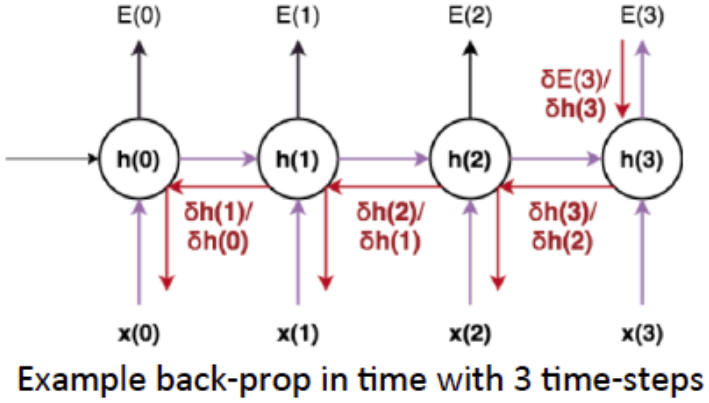
$$\frac{\partial E}{\partial W} = \sum_t \frac{\partial E_t}{\partial W}.$$

NOTE: our goal is to calculate the gradients of the error with respect to our parameters U, W and V and and then learn good parameters using Stochastic Gradient Descent.
Just like we sum up the errors, we also sum up the gradients at each time step for one training example:

BPTT III (E3 computation)

$$\frac{\partial E_3}{\partial W} = \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial h_3}\frac{\partial h_3}{\partial W}$$

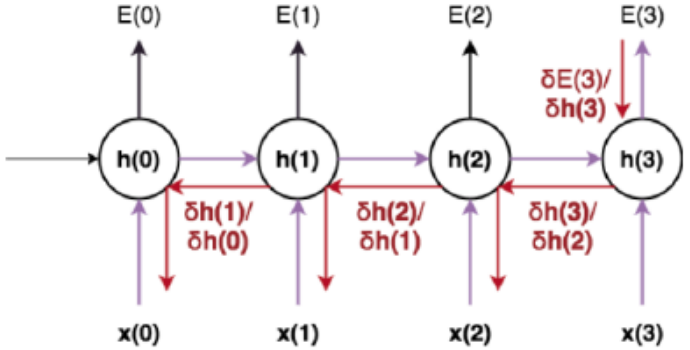
$$h_3 = f(Ux_t + Wh_2)$$



Example back-prop in time with 3 time-steps

BPTT IV

$$\frac{\partial E_3}{\partial W} = \sum_{h=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial h_3} \frac{\partial h_3}{\partial h_k} \frac{\partial h_k}{\partial W}$$



Example back-prop in time with 3 time-steps

4. Vanishing gradient

Example

Maria has studied law and she is a lawyer.

Vanishing gradient I

- **During training gradients explode/vanish easily because of depth-in-time →** Exploding/Vanishing gradients!

Vanishing gradient II

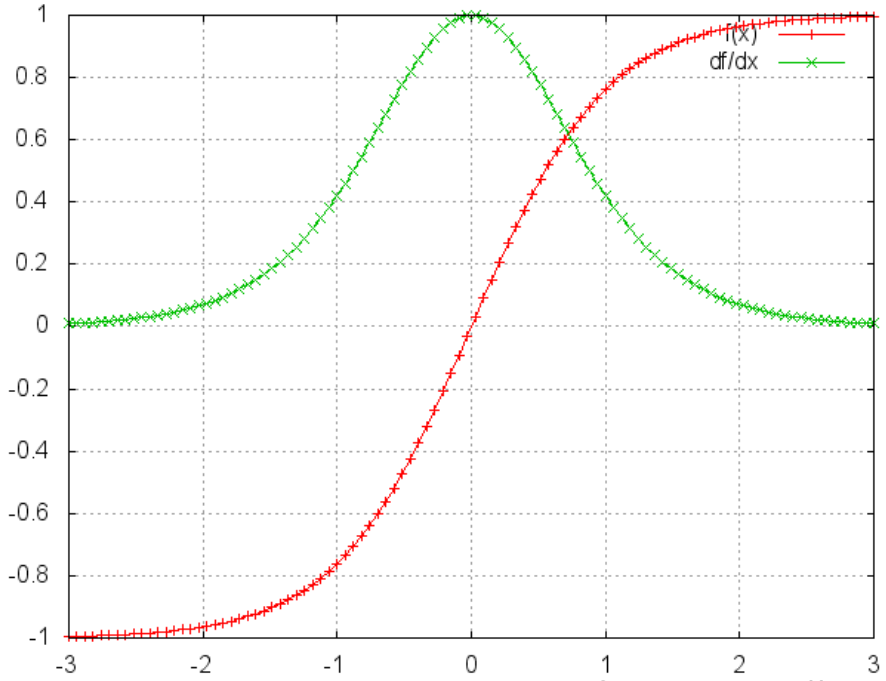
For example,

$$\frac{\partial E_3}{\partial W} = \sum_{h=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial h_3} \frac{\partial h_3}{\partial h_k} \frac{\partial h_k}{\partial W}$$

$$\frac{\partial h_3}{\partial h_1} = \frac{\partial h_3}{\partial h_2} \frac{\partial h_2}{\partial h_1}$$

$$\frac{\partial E_3}{\partial W} = \sum_{h=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial h_3} \left(\prod_{j=k+1}^{3} \frac{\partial h_j}{\partial h_{j-1}} \right) \frac{\partial h_k}{\partial W}$$

Vanishing gradient III



tanh and derivative. Source: http://nn.readthedocs.org/en/rtd/transfer/

$$\frac{\partial E_3}{\partial W} = \sum_{h=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial h_3} \left(\prod_{j=k+1}^{3} \frac{\partial h_j}{\partial h_{j-1}} \right) \frac{\partial h_k}{\partial W}$$

Standard Solutions

Proper initialization of Weight Matrix

Regularization of outputs or Dropout

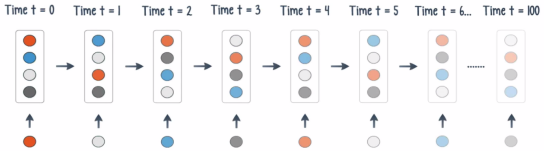
Use of ReLU Activations as it’s derivative is either 0 or 1

Quizz

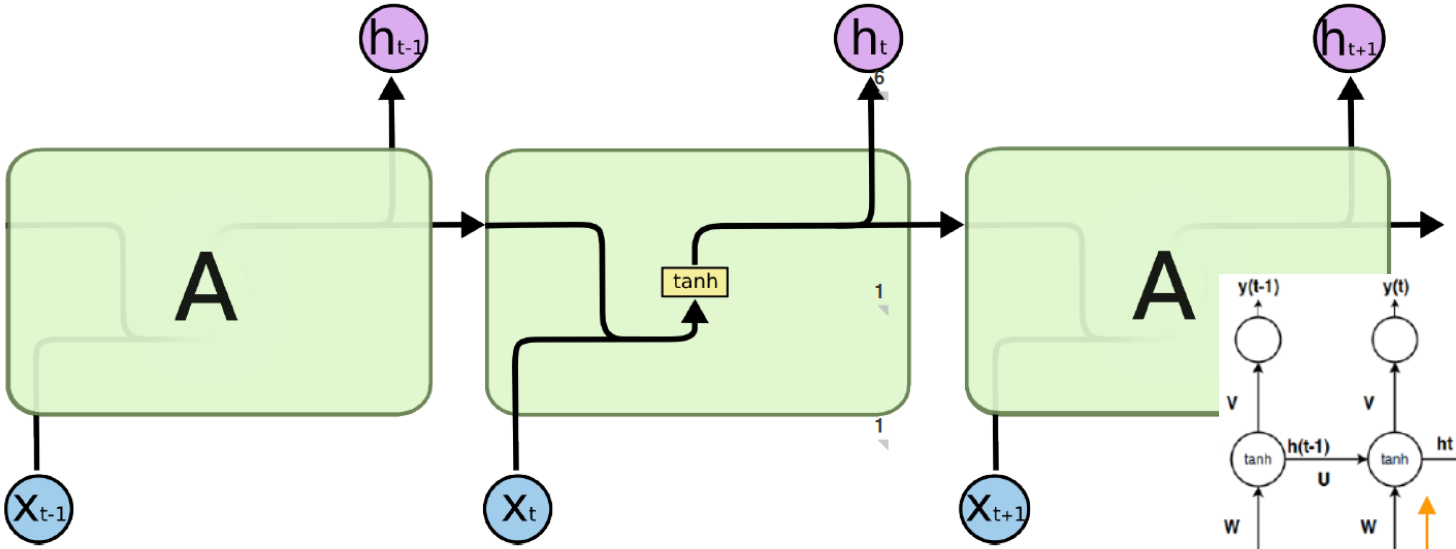
5. Gating method

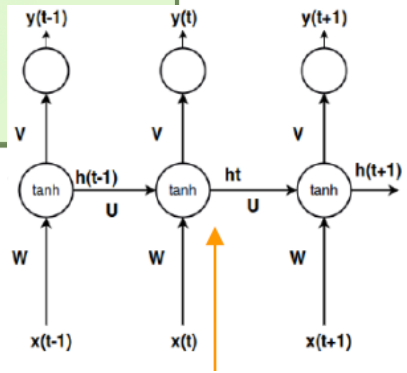
Standard RNN

Decay of information through time



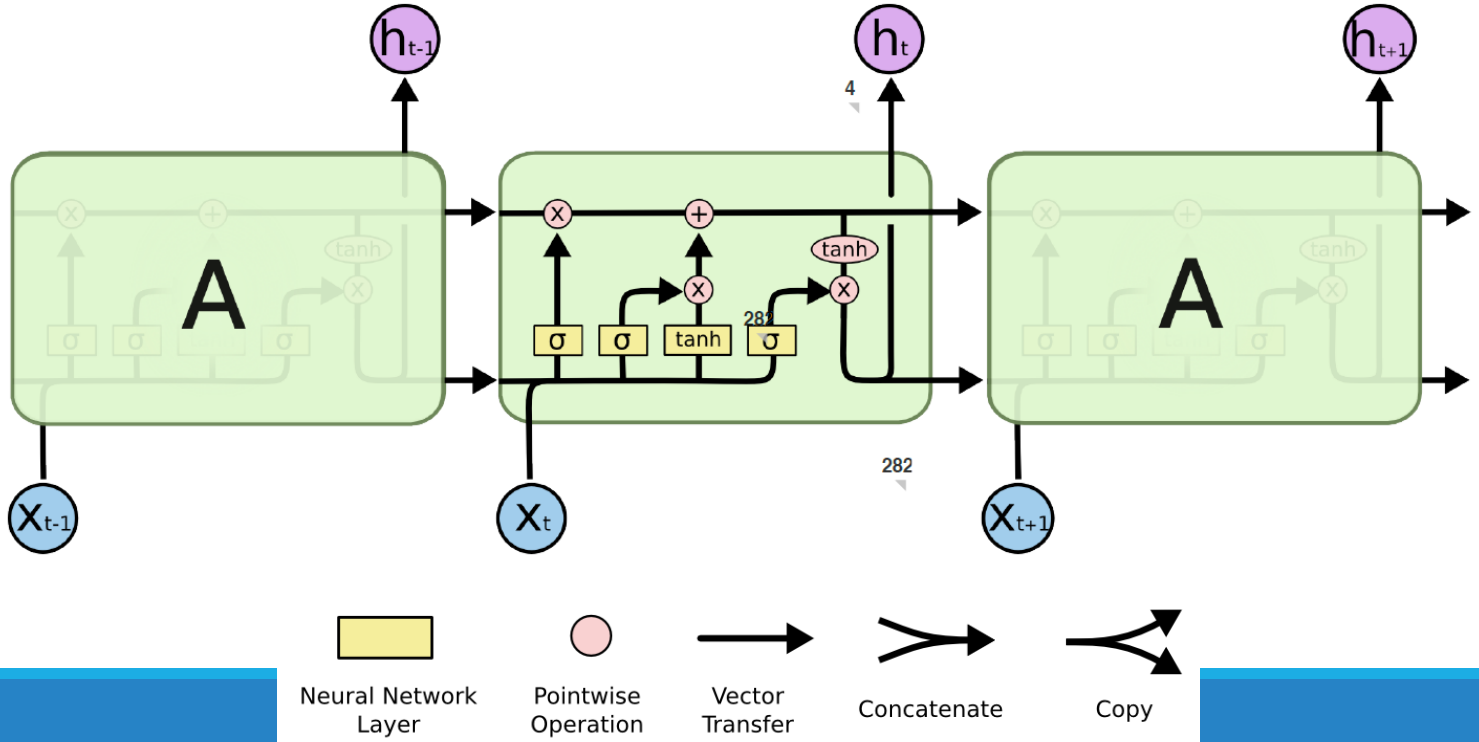
https://www.nextbigfuture.com/2016/03/recurrent-neural-nets.html





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

Long-Short Term Memory (LSTM)



Gating method

1. Change the way in which past information is kept → create the notion of **cell state, a memory unit that keeps long-term information in a safer way by protecting it from recursive operations**

2. **Make every RNN unit able to decide whether the current time-step information matters or not**, to accept or discard (optimized reading mechanism)

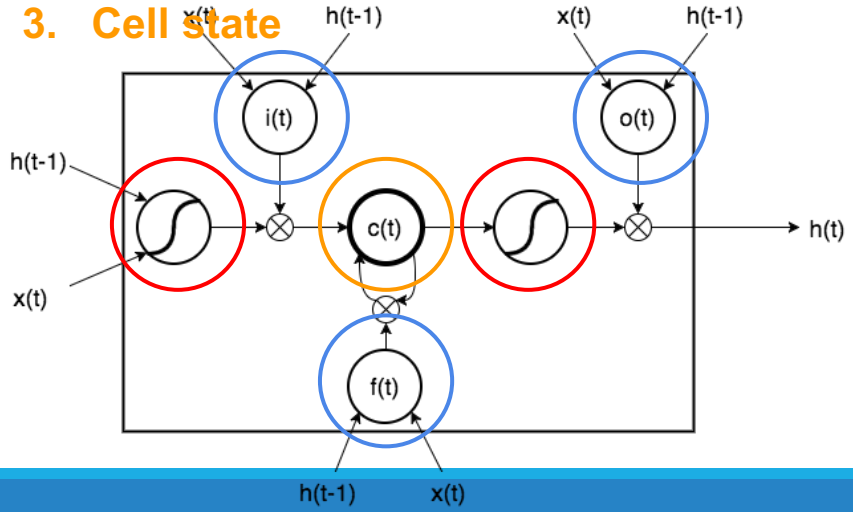
3. **Make every RNN unit able to forget whatever may not be useful anymore** by clearing that info from the cell state (optimized clearing mechanism)

4. **Make every RNN unit able to output the decisions whenever it is ready to do so** (optimized output mechanism)

Long Short Term Memory (LSTM) cell

An LSTM cell is defined by two groups of neurons plus the cell state (memory unit):

1. **Gates**
2. **Activation units**
3. **Cell state**



$$\mathbf{i}_t = \sigma(\mathbf{W}_i\mathbf{x}_t + \mathbf{U}_i\mathbf{h}_{t-1} + \mathbf{b}_i)$$

$$\hat{\mathbf{C}}_t = \tanh(\mathbf{W}_c\mathbf{x}_t + \mathbf{U}_c\mathbf{h}_{t-1} + \mathbf{b}_c)$$

$$\mathbf{f}_t = \sigma(\mathbf{W}_f\mathbf{x}_t + \mathbf{U}_f\mathbf{h}_{t-1} + \mathbf{b}_f)$$

$$\mathbf{C}_t = \mathbf{i}_t \odot \hat{\mathbf{C}}_t + \mathbf{f}_t \odot \mathbf{C}_{t-1}$$

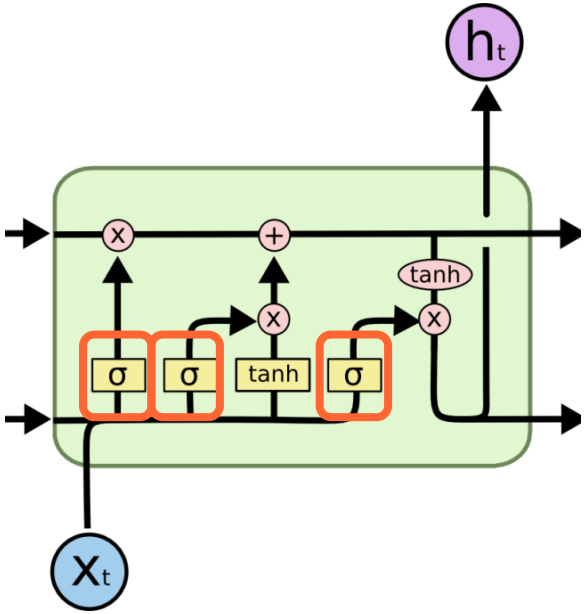
$$\mathbf{o}_t = \sigma(\mathbf{W}_o\mathbf{x}_t + \mathbf{U}_o\mathbf{h}_{t-1} + \mathbf{b}_o)$$

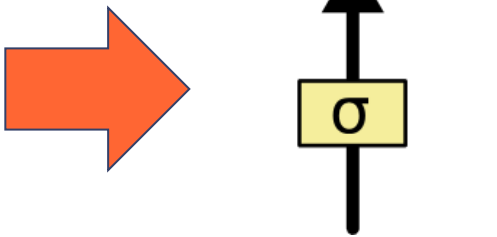
$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{C}_t)$$

Computation Flow

Long Short-Term Memory (LSTM)

Three **gates** are governed by *sigmoid* units (btw [0,1]) define the control of in & out information..





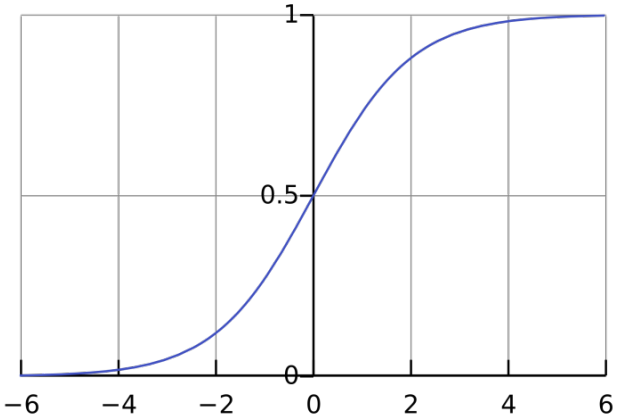
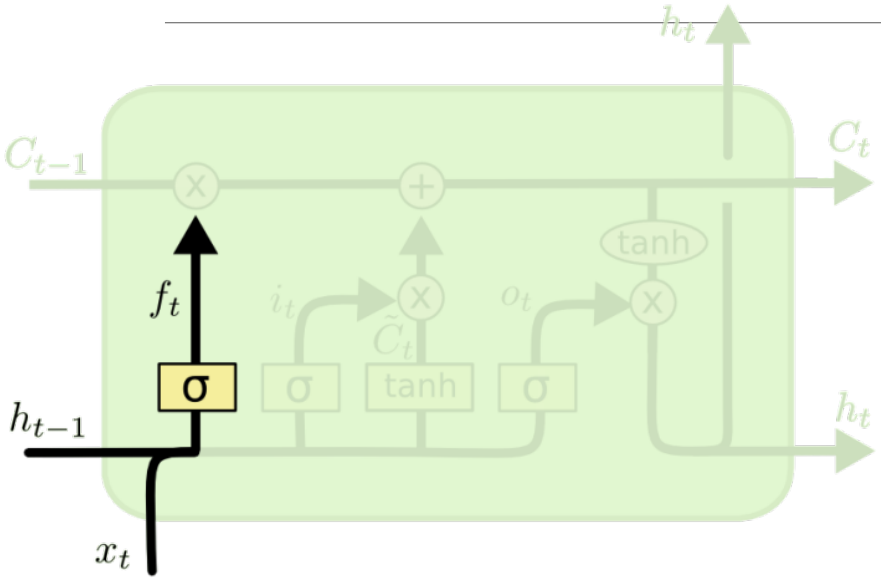


Figure: Cristopher Olah, "Understanding LSTM Networks" (2015)

slide credit: Xavi Giro

Forget Gate



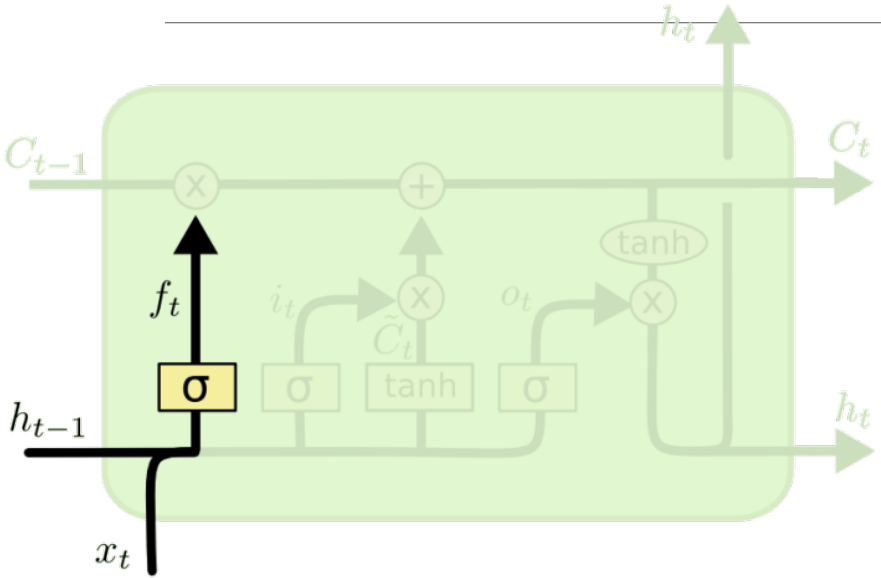
Forget Gate:

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

Concatenate

Figure: Cristopher Olah, "Understanding LSTM Networks" (2015) / Slide: Alberto Montes

Forget Gate: Example



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

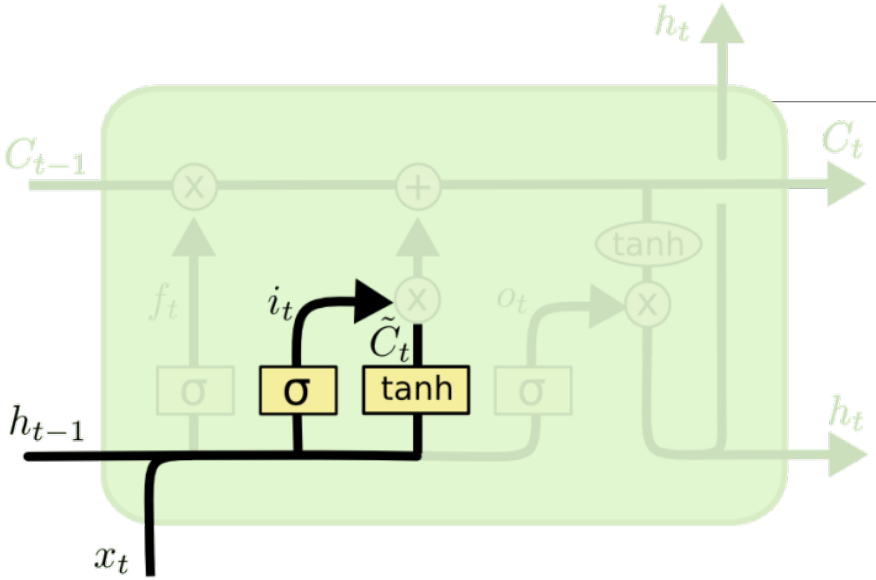
LANGUAGE MODELING

Joan es un chico activo y Anna es una chica calmada

Forget about ”male” gender

Figure: Cristopher Olah, “Understanding LSTM Networks” (2015) / Slide: Alberto Montes

Input Gate



Input Gate Layer

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

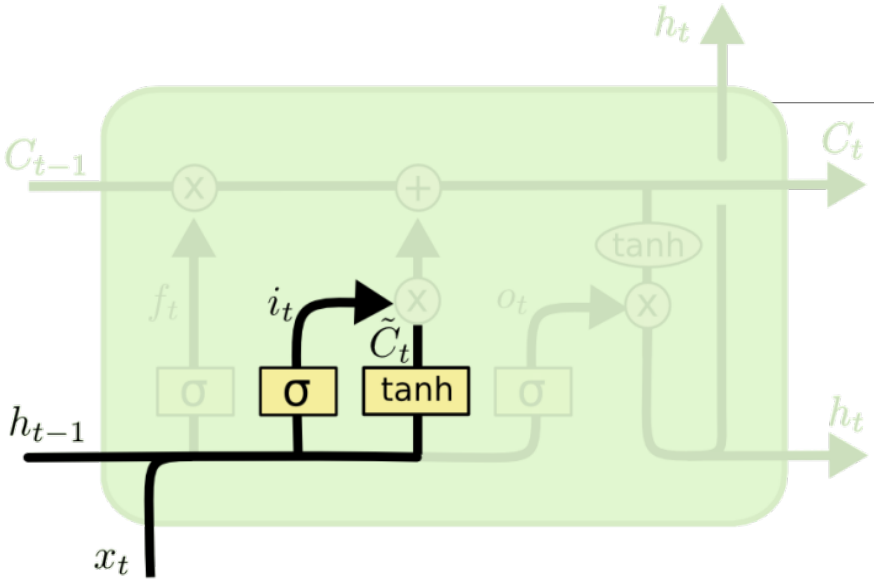
New contribution to cell state

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

Classic neuron

Figure: Cristopher Olah, "Understanding LSTM Networks" (2015) / Slide: Alberto Montes

Input Gate: Example



Input Gate Layer

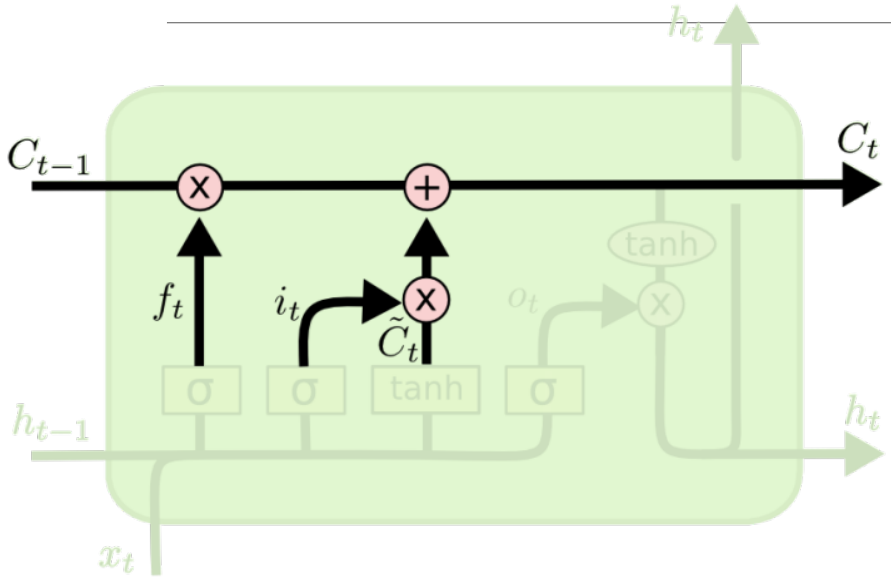
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

LANGUAGE MODELING

Joan es un chico activo y Anna es una chica calmada

Input about ”female” gender

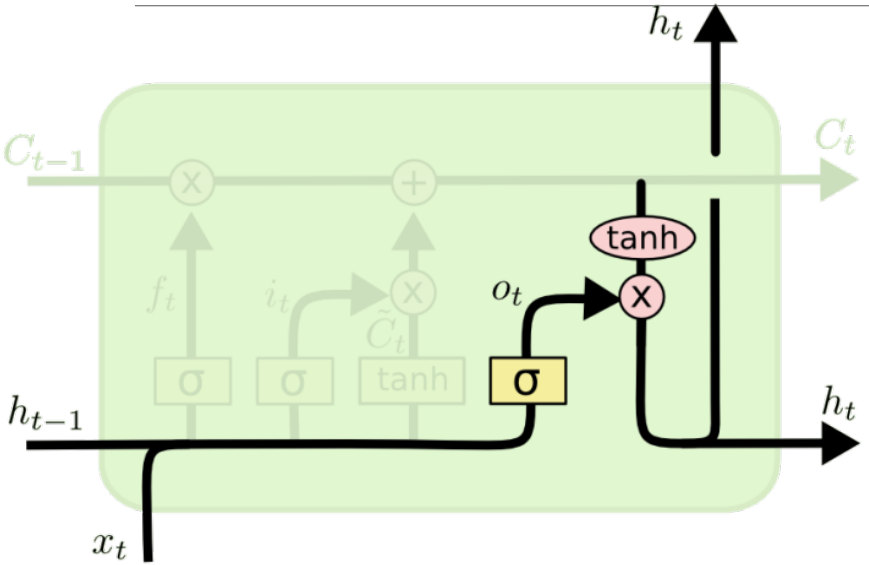
Update Cell State



Update Cell State (memory):

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

Output Gate



Output Gate Layer

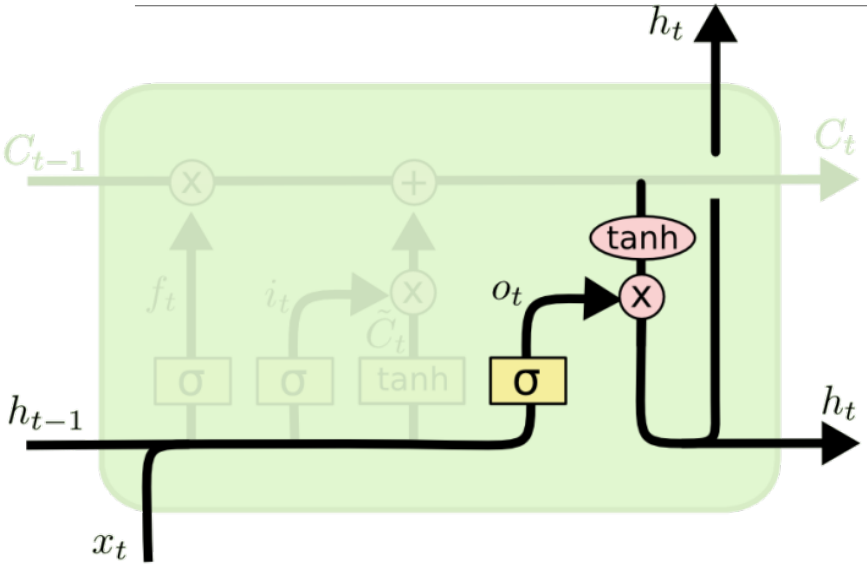
$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

Output to next layer

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

Figure: Cristopher Olah, "Understanding LSTM Networks" (2015) / Slide: Alberto Montes

Output Gate: Example



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

LANGUAGE MODELING

Joan es un chico activo y Anna es una chica calmada

Info relevant for a verb? : ”female”, “singular”, “3rd person”

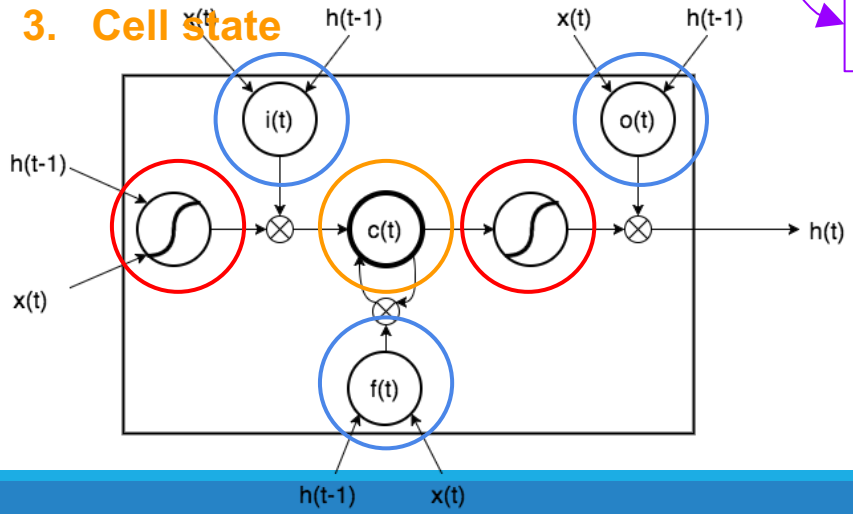
LSTM: parameters

An LSTM cell is defined by two groups of neurons plus the cell state (memory unit):

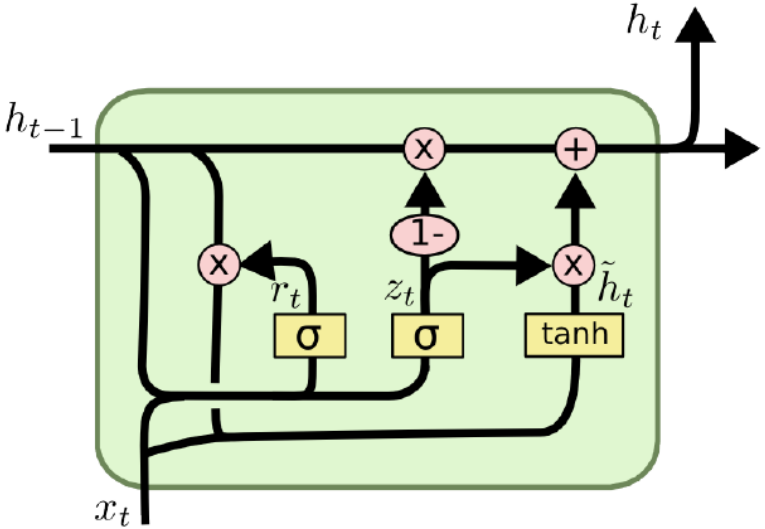
1. **Gates**
2. **Activation units**
3. **Cell state**

$$N^i_{params} = 4 \times (N^i_{inputs} \times N^i_{units} + N^i_{units} \times N^i_{units} + N^i_{units})$$

3 gates + input activation



Gated Recurrent Unit (GRU)



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

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

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

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

$$N^i_{params} = 3 \times (N^i_{inputs} \times N^i_{units} + N^i_{units} \times N^i_{units} + N^i_{units})$$

Cho, Kyunghyun, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. "Learning phrase representations using RNN encoder-decoder for statistical machine translation." AMNLP 2014.

slide credit: Xavi Giro

Visual Comparison FNN, Vanilla RNNs and LSTMs

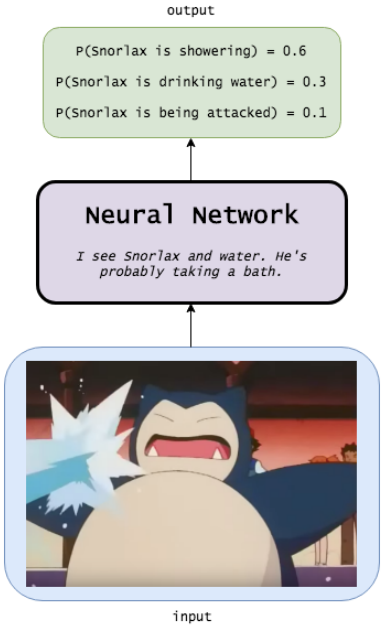


Image src http://blog.echen.me/2017/05/30/exploring-lstms/

Visual Comparison FNN, Vanilla RNNs and LSTMs

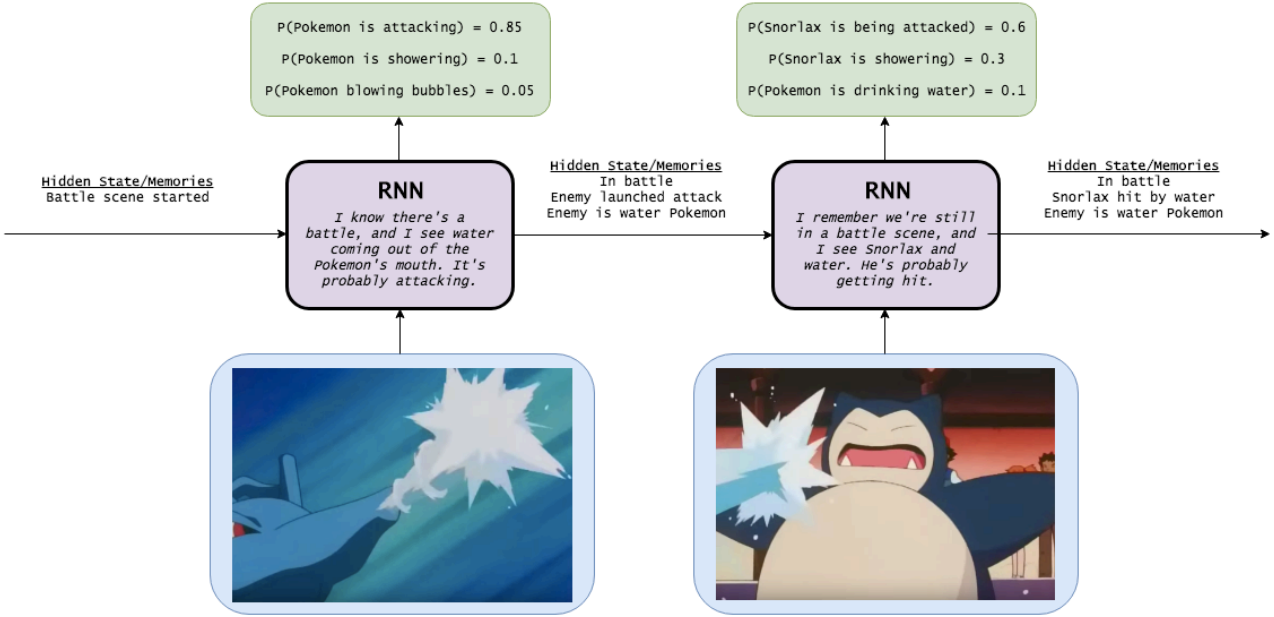


Image src http://blog.echen.me/2017/05/30/exploring-lstms/

Visual Comparison FNN, Vanilla RNNs and LSTMs

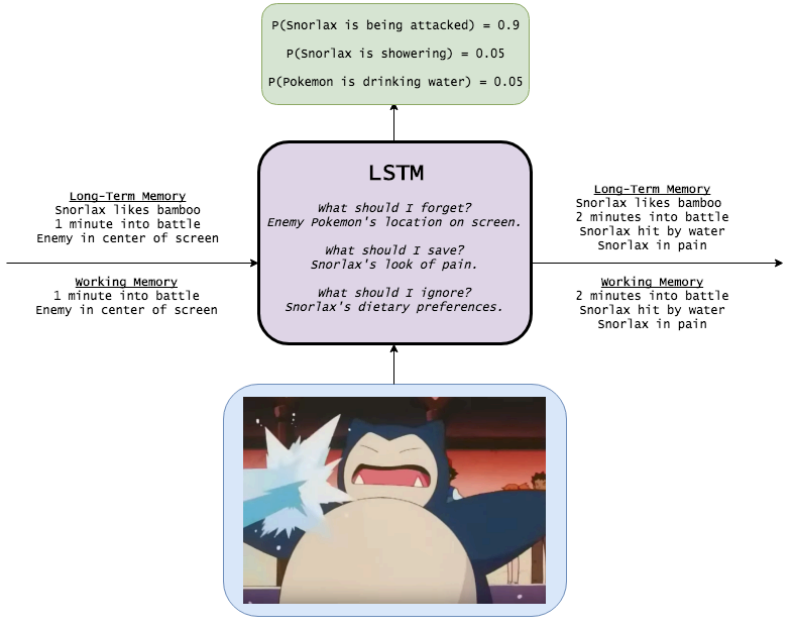


Image src http://blog.echen.me/2017/05/30/exploring-lstms/

Visualization

6. Other RNN extensions

Bidirectional RNNs

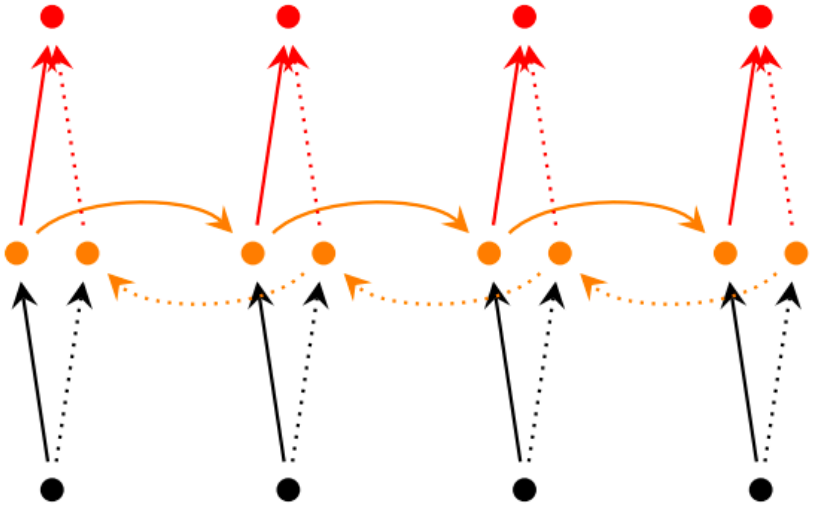


Image src: http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/

Deep RNNs

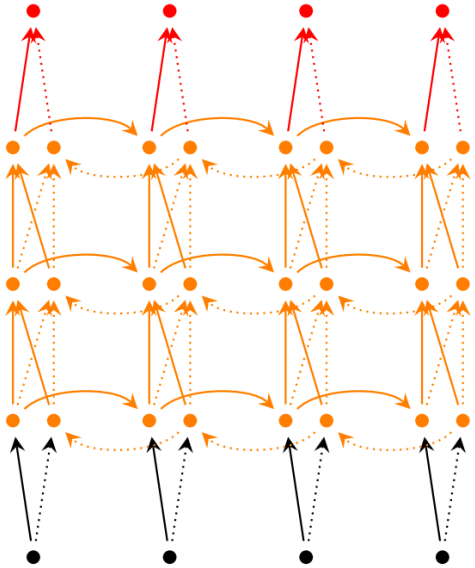


Image src: http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/

7. Use Case

Character-level language models

P(‘he’)=P(e|h)

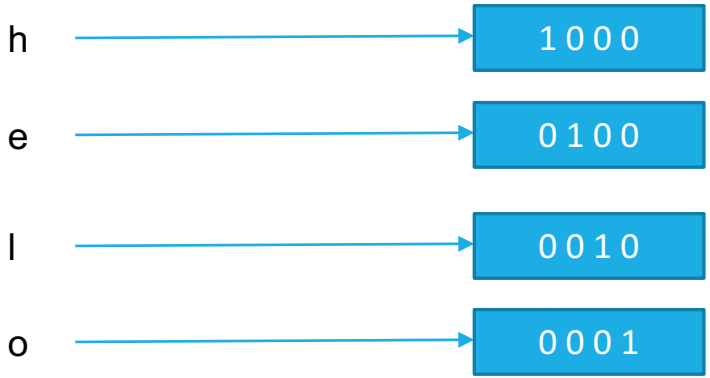
P(‘hel’)=P(l|’he’)

P(‘hell’)=P(l|’hel’)

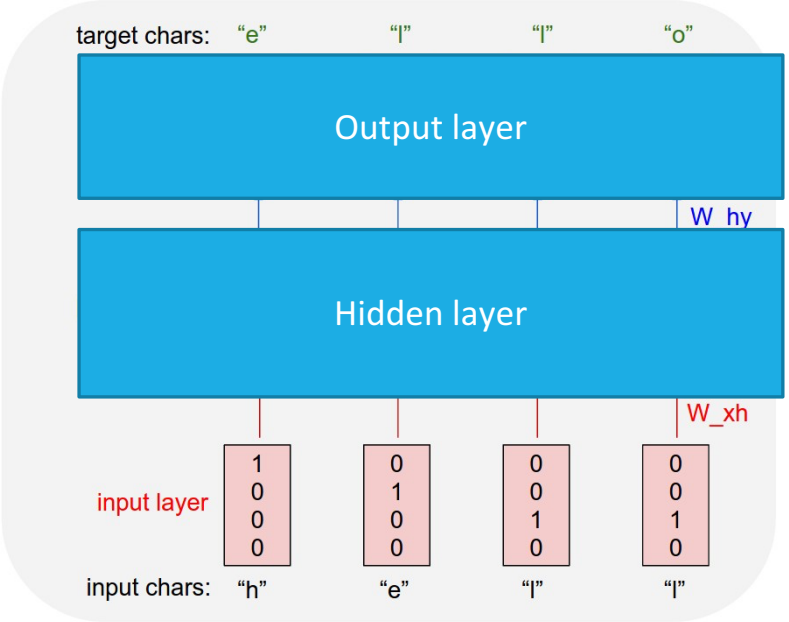
P(‘hello’)=P(o|’hell’)

All these probabilities should be likely

Character-level language models with RNN

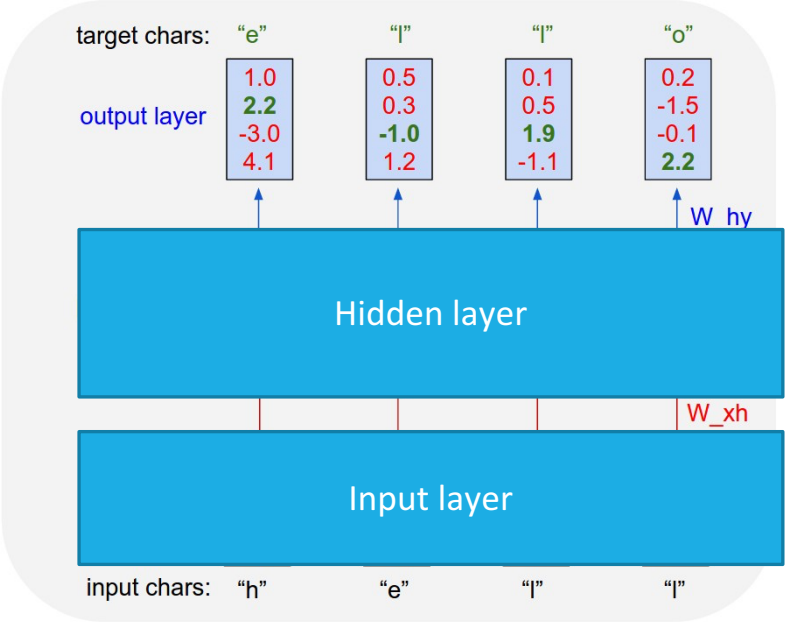


Character-level language models diagram I



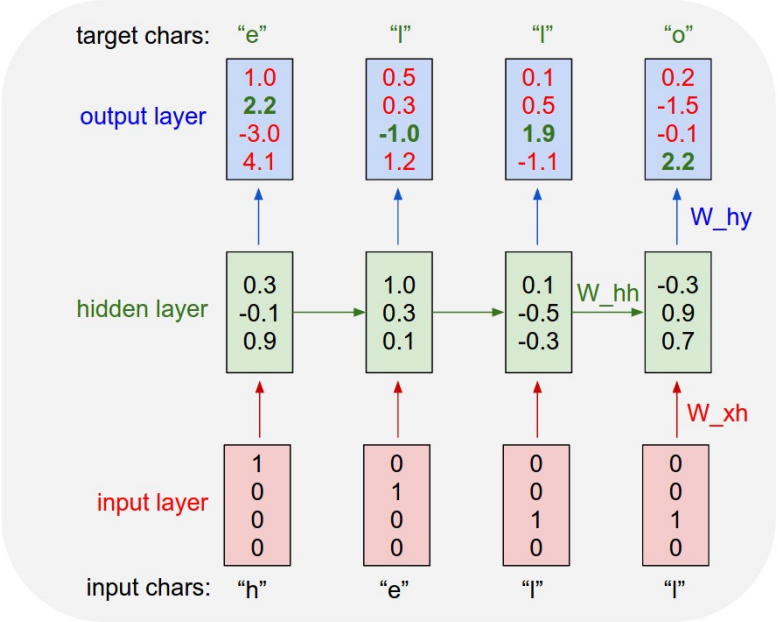
Imagesrc: http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Character-level language diagram II



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

Character-level language diagram III



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

Code Available: https://gist.github.com/karpathy/

min-char-rnn.py

```
Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
BSD License
"""
import numpy as np

# data I/O

# hyperparameters


# model parameters
Wxh = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
Whh = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
Why = np.random.randn(vocab_size, hidden_size)*0.01 # hidden to output
bh = np.zeros((hidden_size, 1)) # hidden bias
by = np.zeros((vocab_size, 1)) # output bias

def lossFun(inputs, targets, hprev):
  """
  inputs,targets are both list of integers.
  hprev is Hx1 array of initial hidden state
  returns the loss, gradients on model parameters, and last hidden state
  """
  return loss, dWxh, dWhh, dWhy, dbh, dby, hs[len(inputs)-1]

def sample(h, seed_ix, n):
  """
  sample a sequence of integers from the model
  h is memory state, seed_ix is seed letter for first time step
  """
 return ixes

n, p = 0, 0
mWxh, mWhh, mWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
while True:
  # prepare inputs (we're sweeping from left to right in steps seq_length long)

  # sample from the model now and then

  # forward seq_length characters through the net and fetch gradient

  # perform parameter update with Adagrad
```

Still ISSUES with RNNs??

Thanks ! Q&A ?