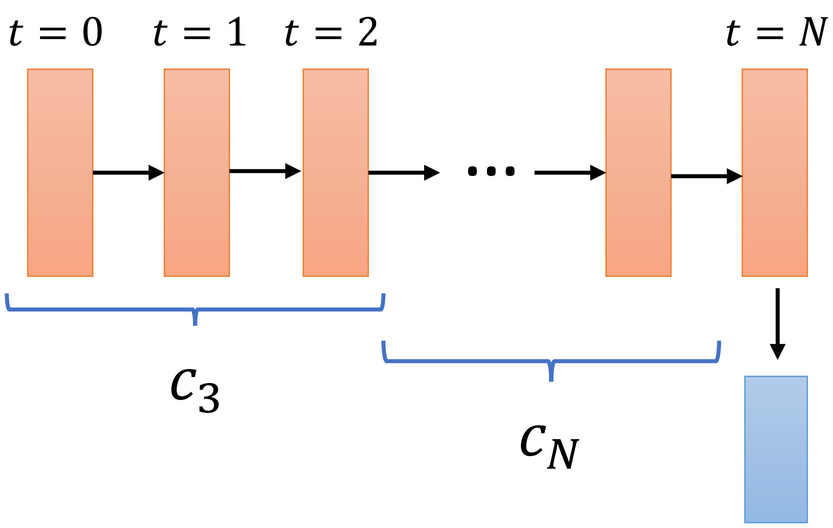
Gated Recurrent Networks

Davide Bacciu

Dipartimento di Informatica
Università di Pisa

Intelligent Systems for Pattern Recognition (ISPR)

Dealing with Sequences in NN



Variable size data describing sequentially dependent information

Neural models need to capture dynamic context c_t to perform predictions

- Recurrent Neural Network
  - Fully adaptive (Elman, SRN, …)
  - Randomized approaches (Reservoir Computing)
- Introduce (deep) gated recurrent networks

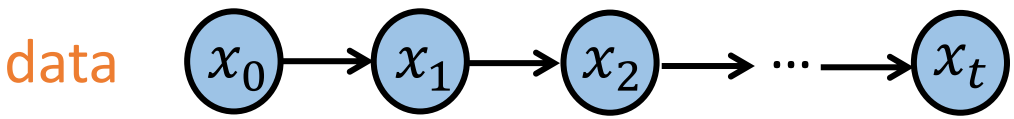
Lecture Outline

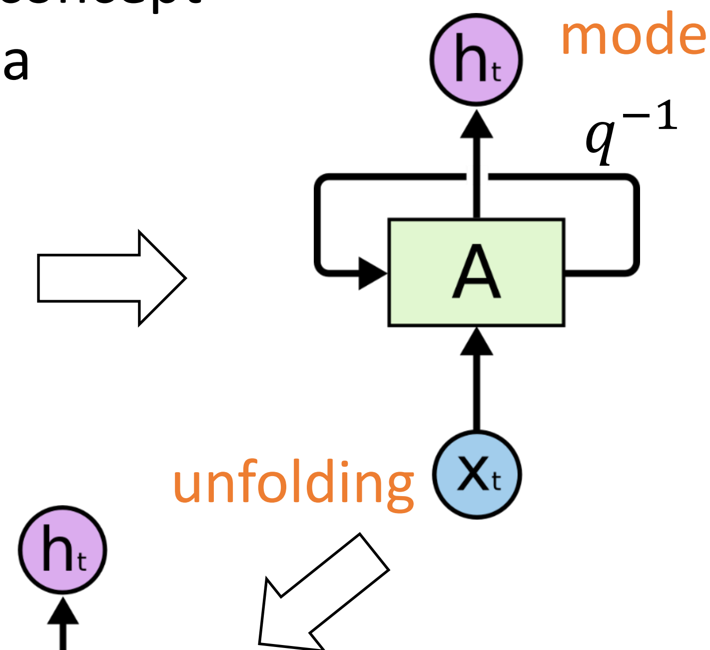
- RNN Repetita
- Motivations
  - Learning long-term dependencies is difficult
  - Gradient issues
- Gated RNN
  - Long-Short Term Memories (LSTM)
  - Gated Recurrent Units (GRU)
- Advanced topics
  - Understanding and exploiting memory encoding
  - Applications

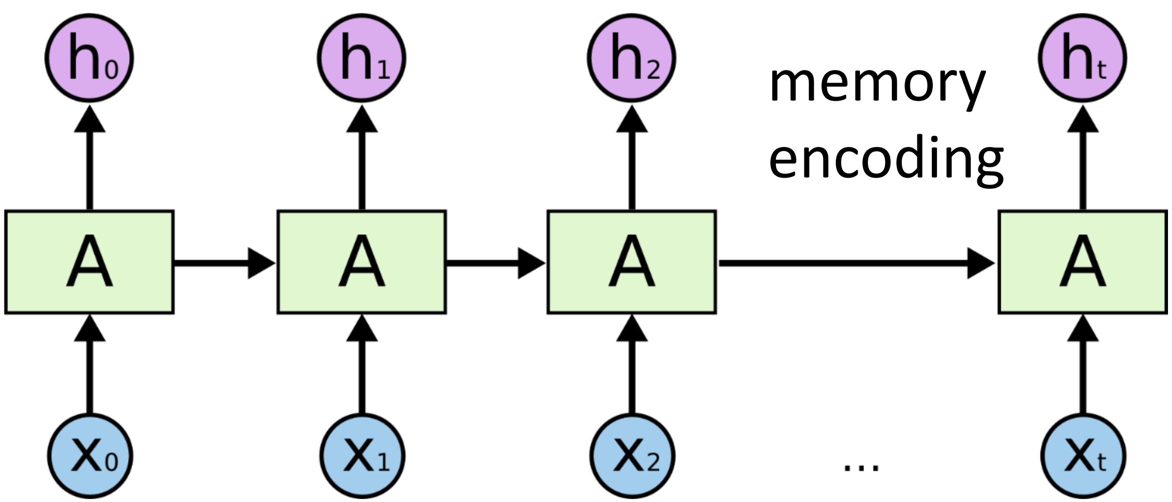
Unfolding RNN (Forward Pass)

Graphics credit @ colah.github.io

By now you should be familiar with the concept of model unfolding/unrolling on the data

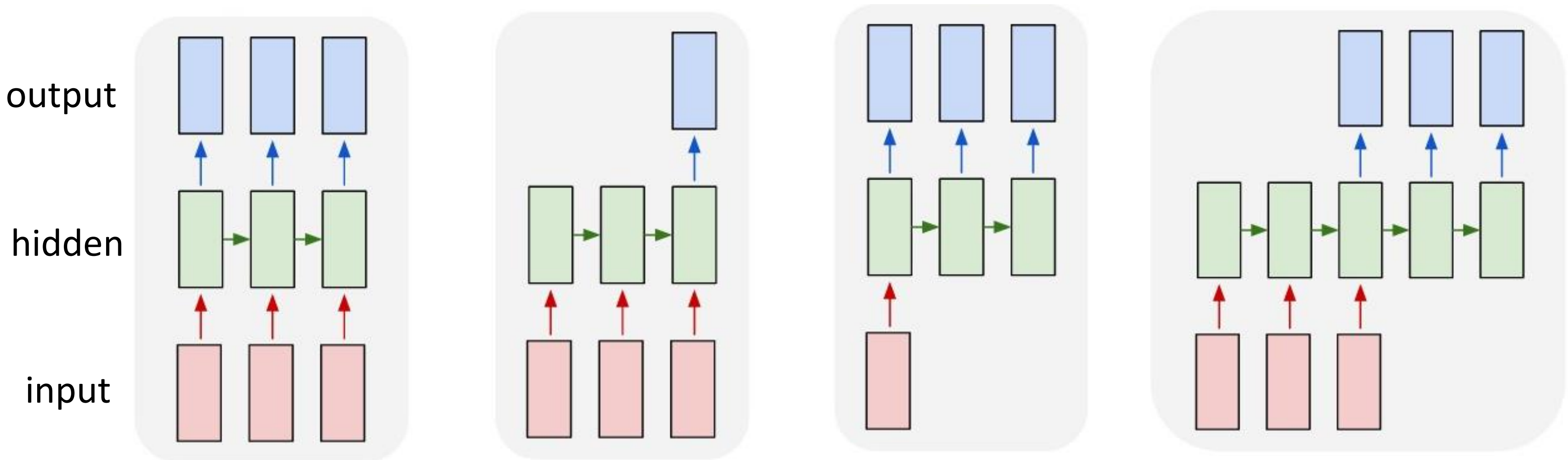






Map an arbitrary length sequence $x_0..x_t$ to fixed-length encoding $\boldsymbol{h}_t$

Supervised Recurrent Tasks

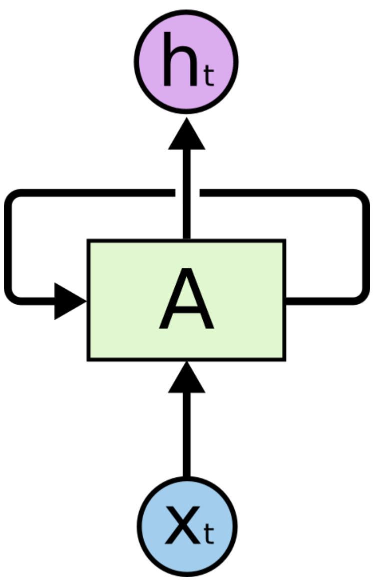


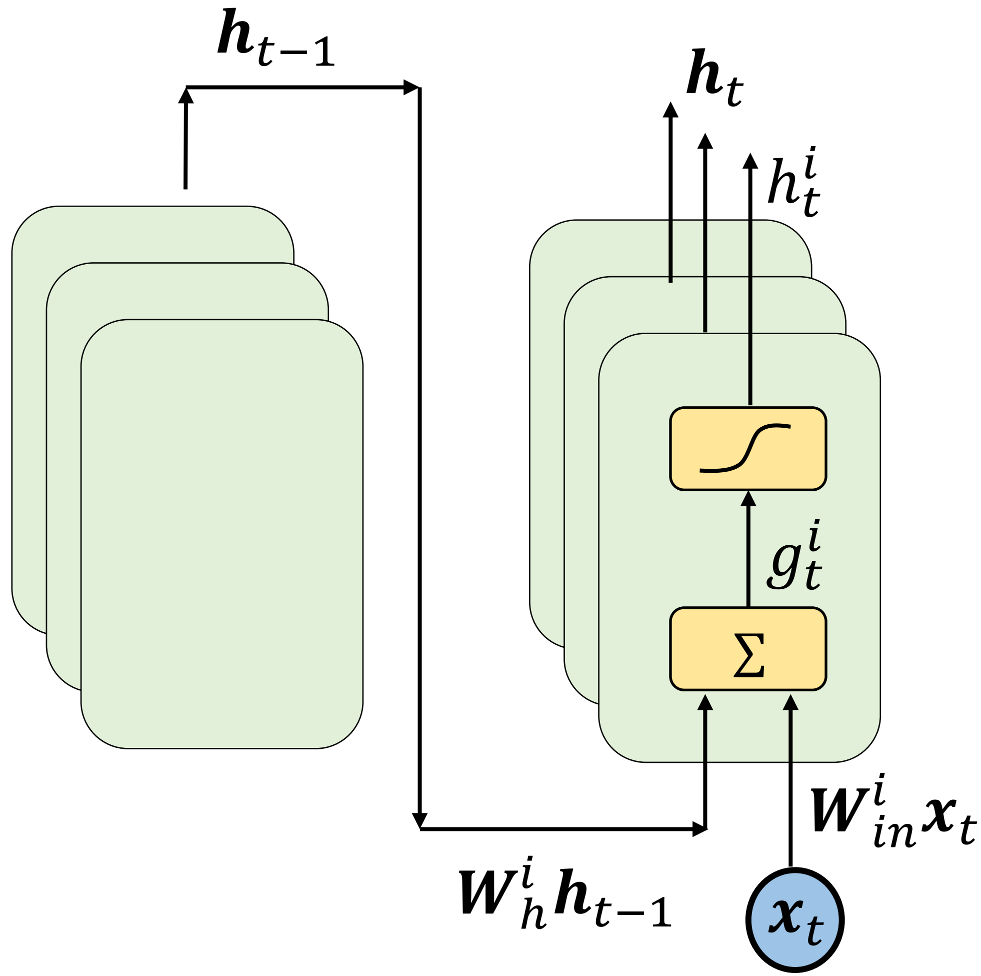
element to element   sequence to item   item to sequence   sequence to sequence

Graphics credit @ karpathy.github.io

A Non-Gated RNN (a.k.a. Vanilla)

$$\boldsymbol{y}_t = f(\boldsymbol{W}_{out}\boldsymbol{h}_t + \boldsymbol{b}_{out})$$

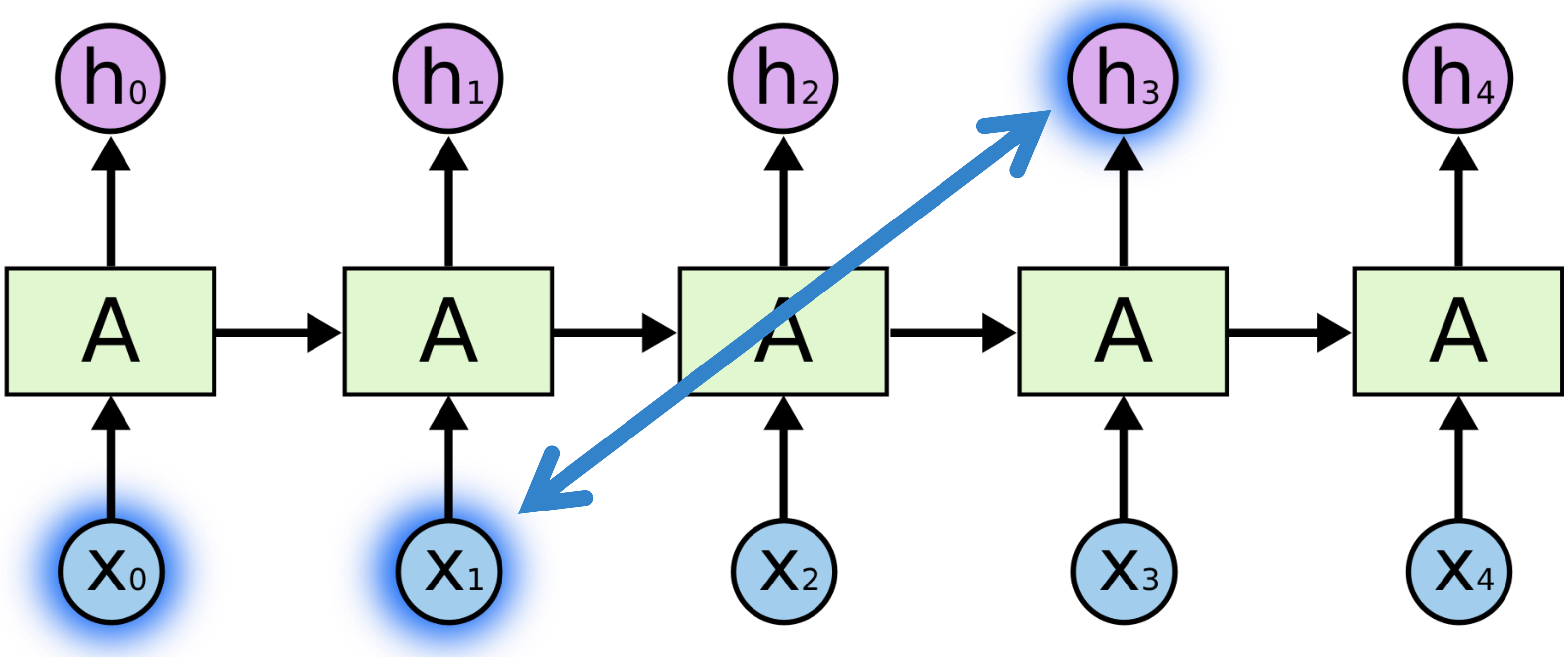




$$\boldsymbol{h}_t = tanh(\boldsymbol{g}_t)$$

$$\boldsymbol{g}_t(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t) = \boldsymbol{W}_h\boldsymbol{h}_{t-1} + \boldsymbol{W}_{in}\boldsymbol{x}_t + \mathbf{b}_\mathrm{h}$$

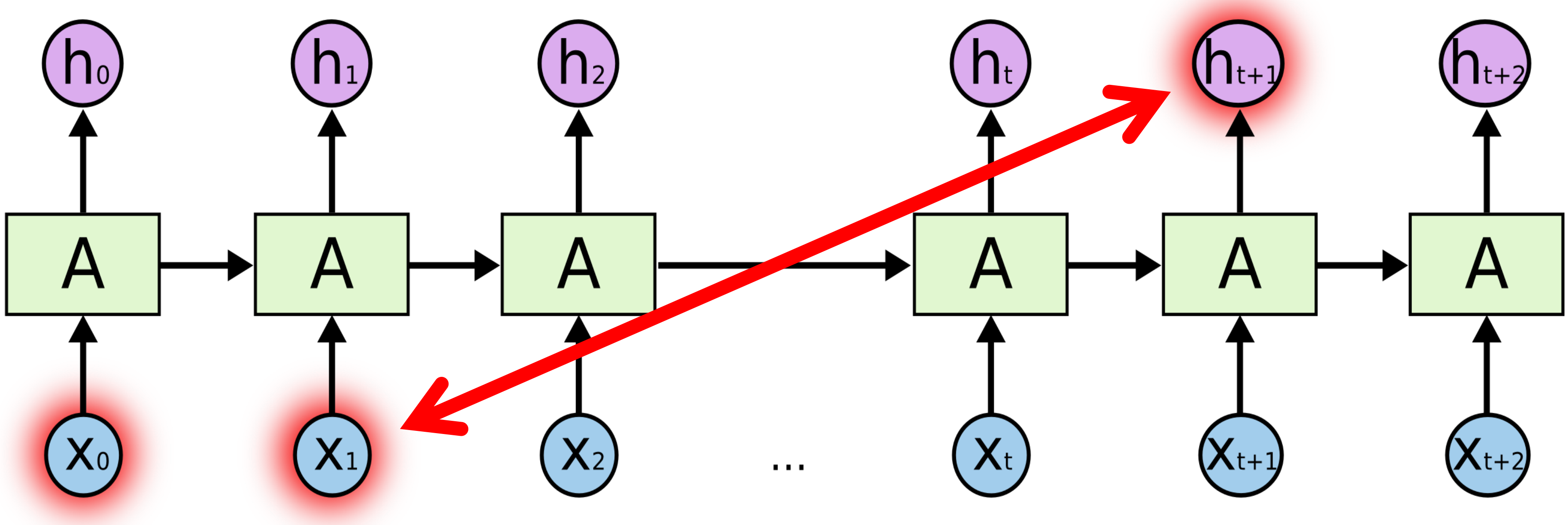
Learning to Encode Input History



Hidden state $\boldsymbol{h}_t$ summarizes information on the history of the input signal up to time t

Learning Long-Term Dependencies is Difficult

When the time gap between the observation and the state grows there is little residual information of the input inside of the memory



What is the cause?

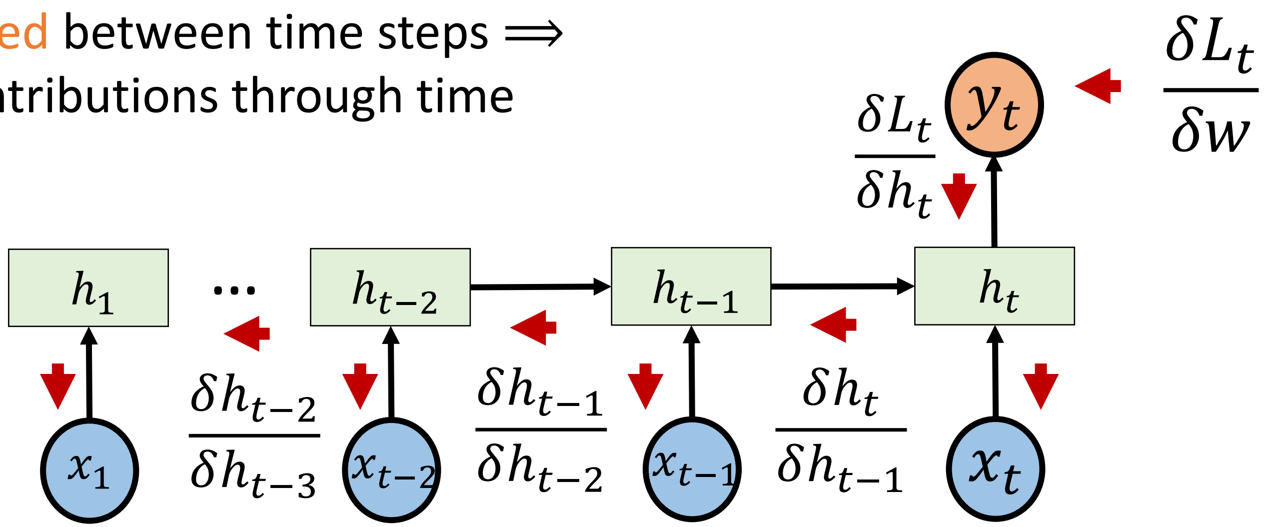
J. Hochreiter. Untersuchungen zu dynamischen neuronalen Netzen, TUM, 1991

Exploding/Vanishing Gradient

Short story: Gradients propagated over many stages tend to

- Vanish (often) $\Longrightarrow$ No learning
- Explode (rarely) $\Longrightarrow$ Instability and oscillations

Weights are shared between time steps $\Longrightarrow$ sum gradient contributions through time



Bengio, Simard and Frasconi, Learning long-term dependencies with gradient descent is difficult. TNN, 1994

A Closer Look at the Gradient

$$\frac{\delta L_t}{\delta W} = \sum_{k=1}^{t} \frac{\delta L_t}{\delta h_t} \boxed{\frac{\delta h_t}{\delta h_k}} \frac{\delta h_k}{\delta W}$$

This is a parameter matrix $\Longrightarrow$ we have a Jacobian

Inside here you have chain rule

$$\frac{\delta h_t}{\delta h_k} = \frac{\delta h_t}{\delta h_{t-1}} \times \frac{\delta h_{t-1}}{\delta h_{t-2}} \times \cdots \times \frac{\delta h_{k+1}}{\delta h_k}$$

$$\frac{\delta L_t}{\delta W} = \sum_{k=1}^{t} \boxed{\frac{\delta L_t}{\delta h_t} \left(\prod_{l=k}^{t-1} \frac{\delta h_{l+1}}{\delta h_l} \right)} \frac{\delta h_k}{\delta W}$$

The gradient is a recursive product of hidden activation gradients (Jacobian)

Bounding the Gradient (I)

Given $\boldsymbol{h}_l = tanh(\boldsymbol{W}_{hl}\boldsymbol{h}_{l-1} + \boldsymbol{W}_{in}\boldsymbol{x}_l)$ then $\dfrac{\delta \boldsymbol{h}_{l+1}}{\delta \boldsymbol{h}_l} = \boldsymbol{D}_{l+1}\boldsymbol{W}_{hl}^T$

where the activation Jacobian is

$$\boldsymbol{D}_{l+1} = diag(1 - \tanh^2(\boldsymbol{W}_{hl}\boldsymbol{h}_l + \boldsymbol{W}_{in}\boldsymbol{x}_{l+1}))$$

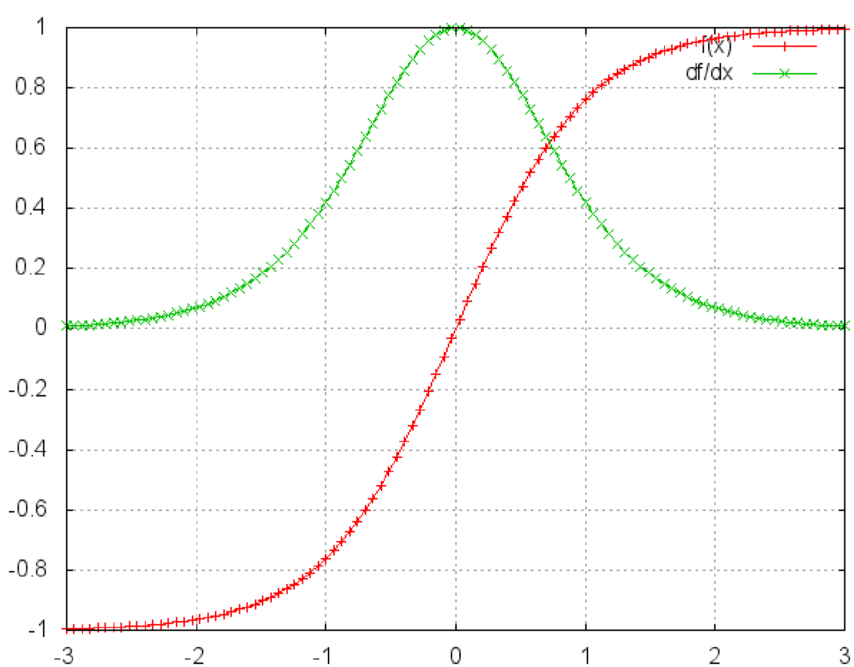
$$\frac{\delta L_t}{\delta \boldsymbol{h}_k} = \frac{\delta L_t}{\delta \boldsymbol{h}_t}\left(\prod_{l=k}^{t-1}\frac{\delta \boldsymbol{h}_{l+1}}{\delta \boldsymbol{h}_l}\right) = \frac{\delta L_t}{\delta \boldsymbol{h}_t}\prod_{l=k}^{t-1}\boldsymbol{D}_{l+1}\boldsymbol{W}_{hl}^T$$

We are interested in the gradient magnitude $\left\|\dfrac{\delta L_t}{\delta \boldsymbol{h}_k}\right\|$

Bounding the Gradient (II)

$$\left\|\frac{\delta L_t}{\delta \boldsymbol{h}_k}\right\| = \left\|\frac{\delta L_t}{\delta \boldsymbol{h}_t}\prod_{l=k}^{t-1}\boldsymbol{D}_{l+1}\boldsymbol{W}_{hl}^T\right\| \leq \left\|\frac{\delta L_t}{\delta \boldsymbol{h}_t}\right\|\prod_{l=k}^{t-1}\|\boldsymbol{D}_{l+1}\boldsymbol{W}_{hl}^T\| = \left\|\frac{\delta L_t}{\delta \boldsymbol{h}_t}\right\|\prod_{l=k}^{t-1}\sigma(\boldsymbol{D}_{l+1})\sigma(\boldsymbol{W}_{hl}^T)$$

Bounded by the spectral radius σ



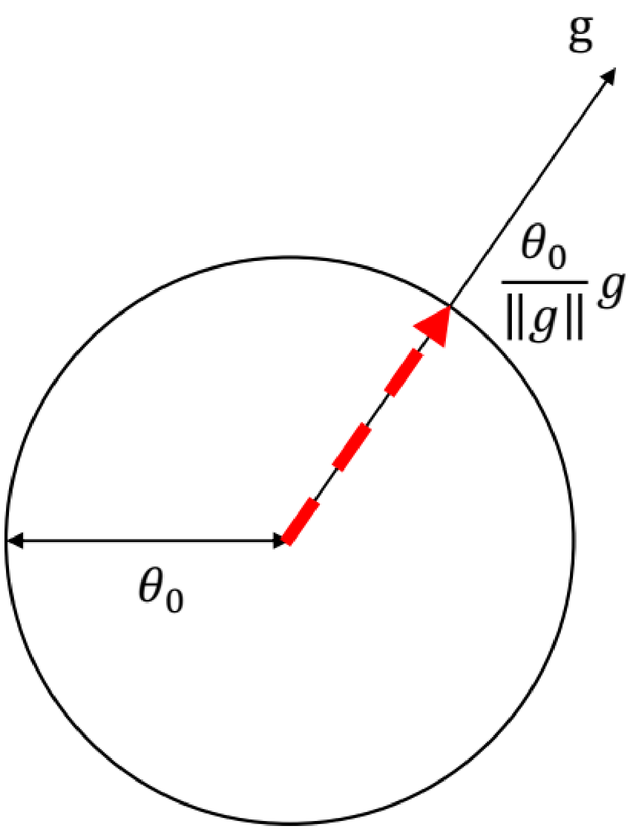
Can shrink to zero or increase exponentially depending on the spectral properties

- $\sigma < 1 \Longrightarrow$ vanishish
- $\sigma > 1 \Longrightarrow$ exploding

Gradient Clipping for Exploding Gradients

- Take $g = \frac{\delta L_t}{\delta \boldsymbol{W}}$
- If $||g|| > \theta_0$ then

$$g = \frac{\theta_0}{||g||} g$$



Rescaling does not work for gradient vanish as total gradient is a sum of time dependent gradients (preserving relative contribution from each time makes it exponentially decay)

$$\frac{\delta L_t}{\delta \boldsymbol{W}} = \sum_{k=1}^{t} \frac{\delta L_t}{\delta h_t} \frac{\delta h_t}{\delta h_k} \cdots$$

Tackling Gradient Issues

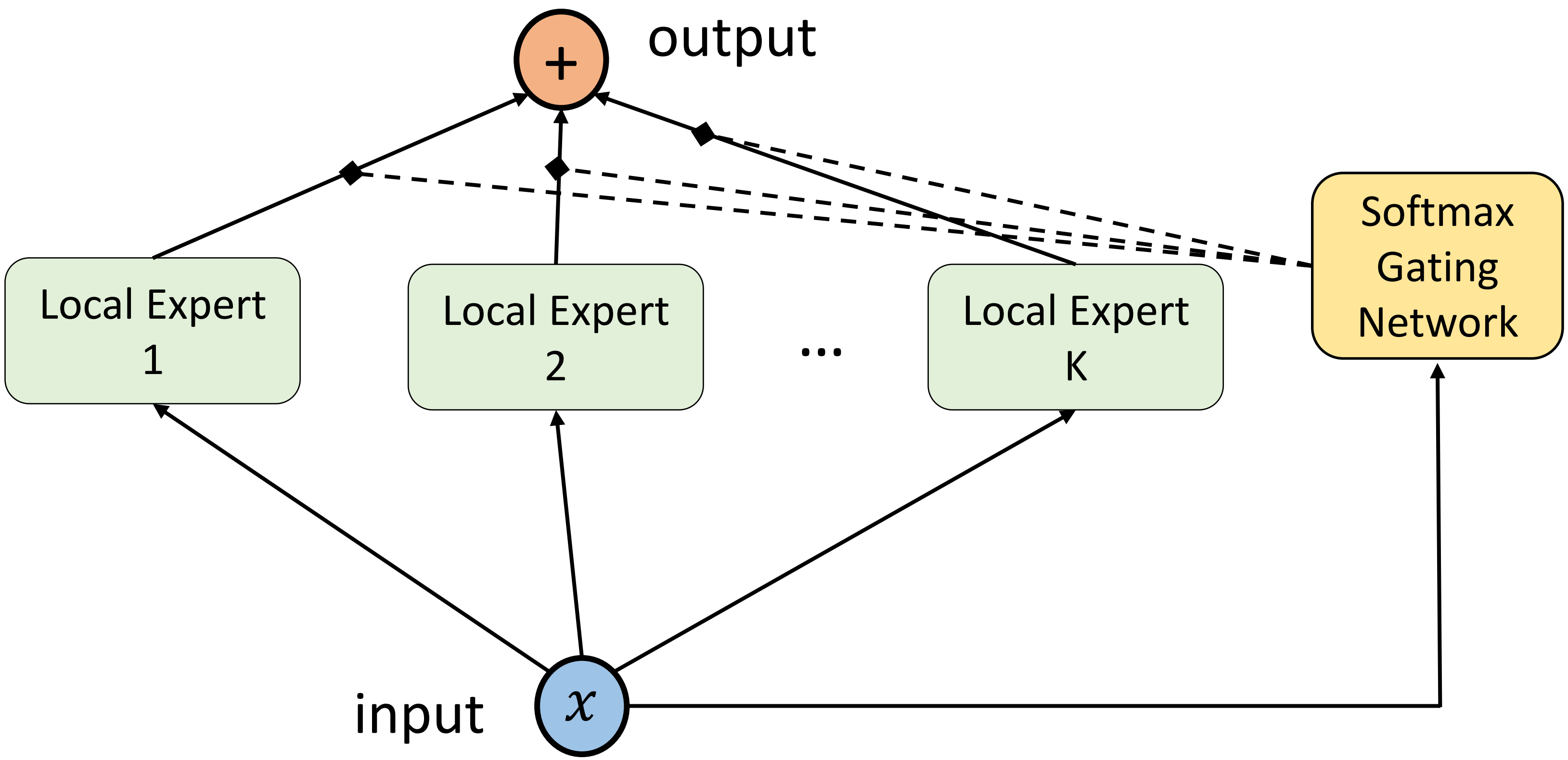
- Solution seems to be having the Jacobian with $\sigma = 1$ (activation function?)
- Linear $\Rightarrow$ dominated by the eigenvalues of $\boldsymbol{W}_h$ to the power of t
- Linear with weight 1 (state identity)

$$\boldsymbol{h}_t = \boldsymbol{h}_{t-1} + \hat{c}(\boldsymbol{x}_t)$$

Has the desired spectral properties but does not work in practice as it quickly saturates memory (e.g. with replicated/non-useful inputs and states)

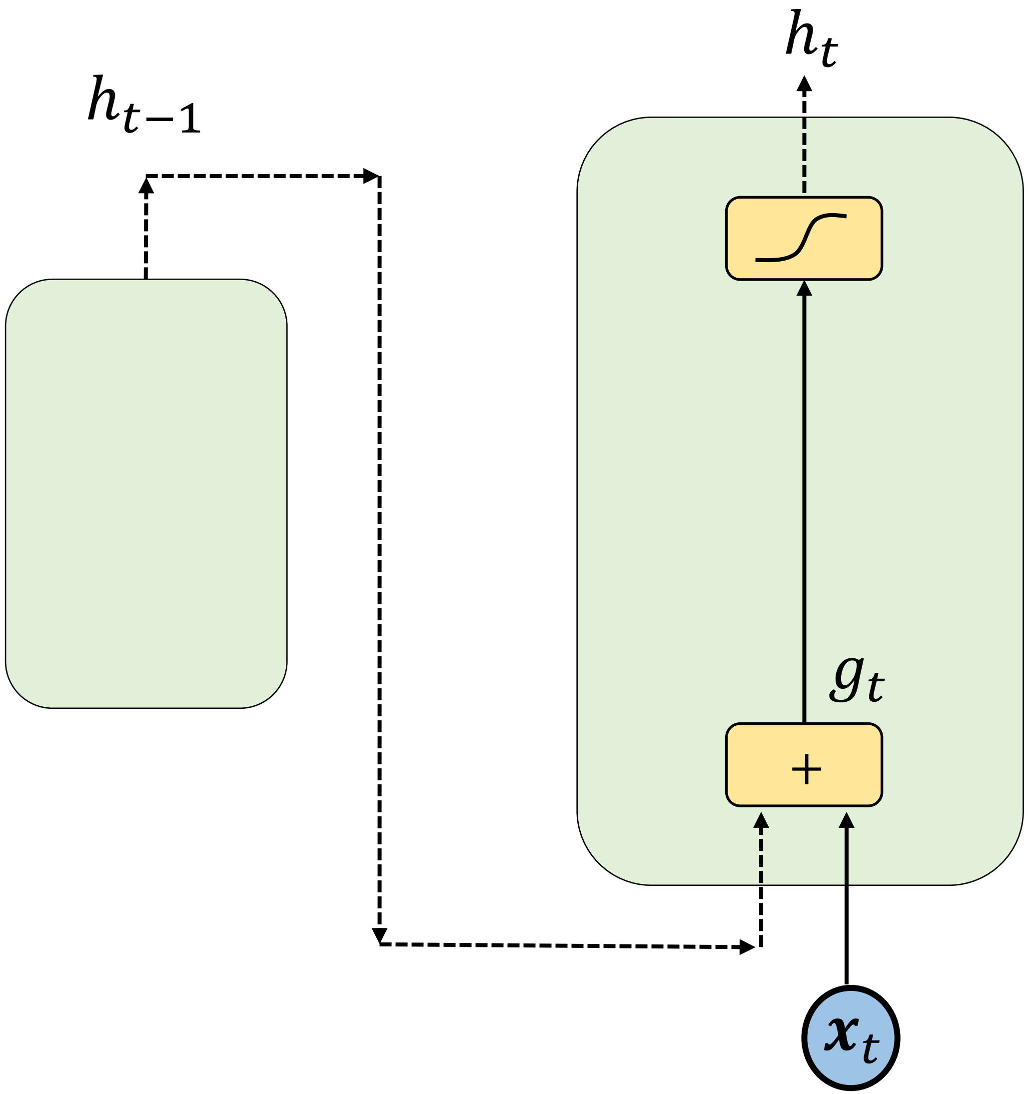
Gating Units

Mixture of experts ⇒ the origin of gating



Jacobs et al (1991), Adaptive Mixtures of Local Experts, ...

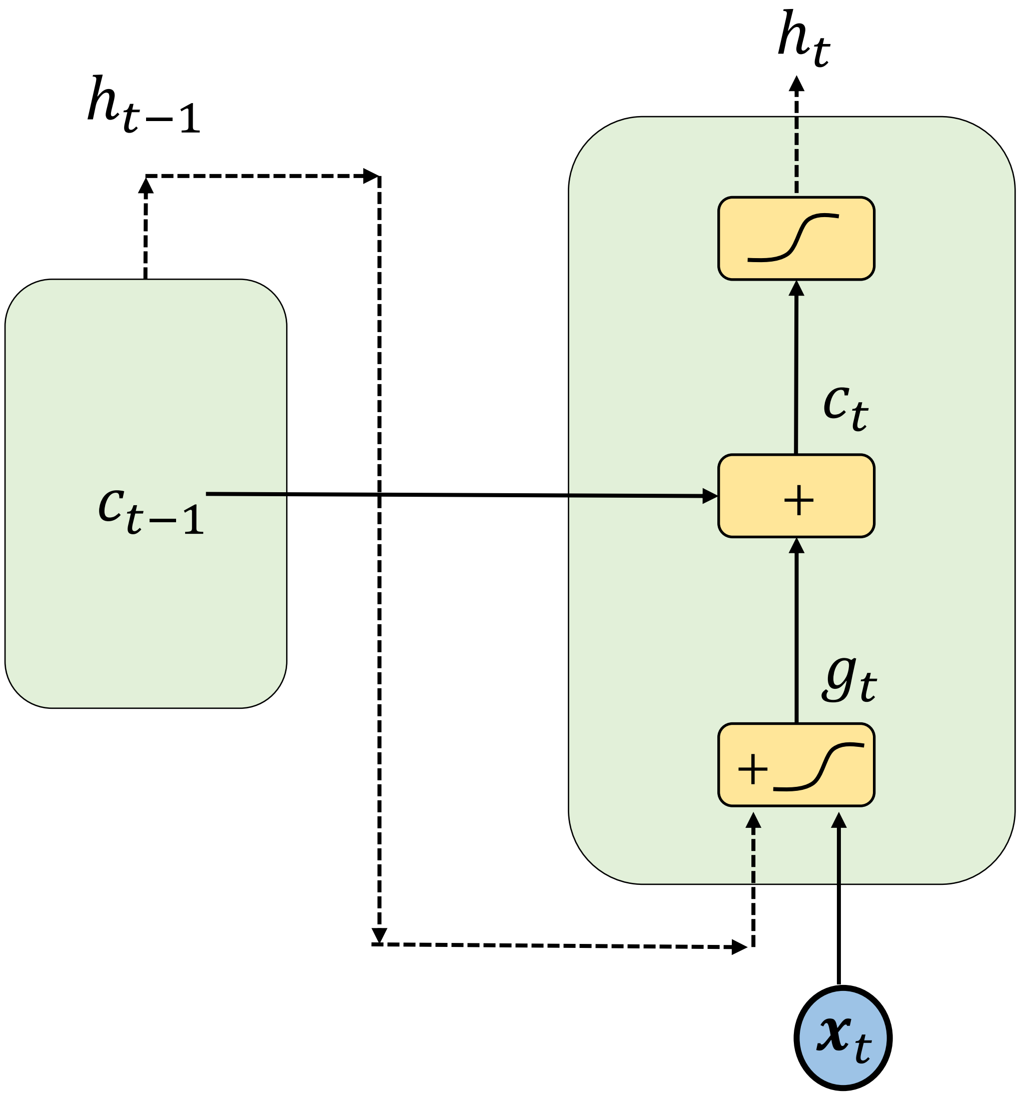
Long-Short Term Memory (LSTM) Cell



Lets start from the vanilla RNN unit

S. Hochreiter, J. Schmidhuber, Long short-term memory". Neural Computation, Neural Comp. 1997

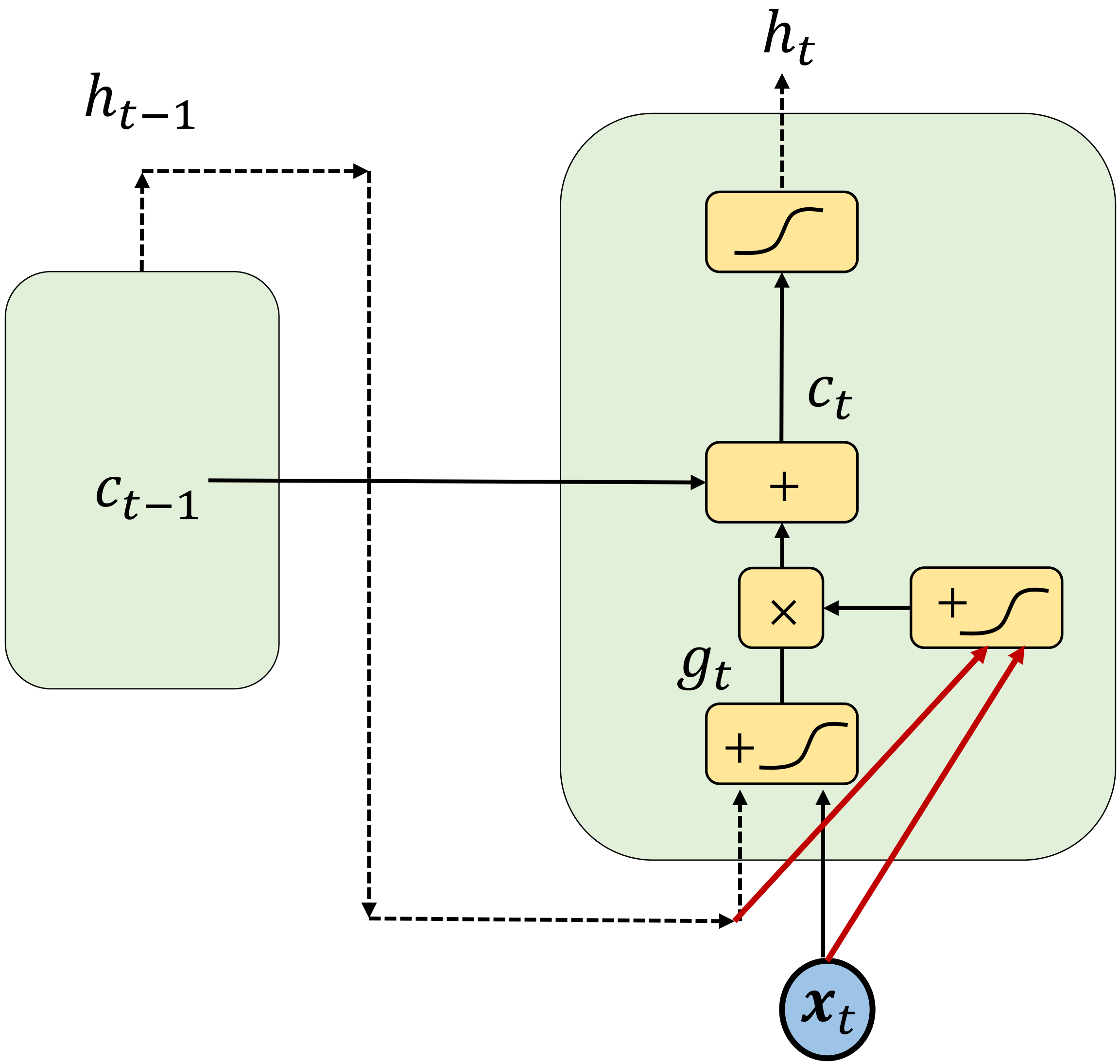
LSTM Design – Step 1



Introduce a linear/identity memory c_t

Combines past internal state c_{t-1} with current input $\boldsymbol{x}_t$

LSTM Design – Step 2 (Gates)



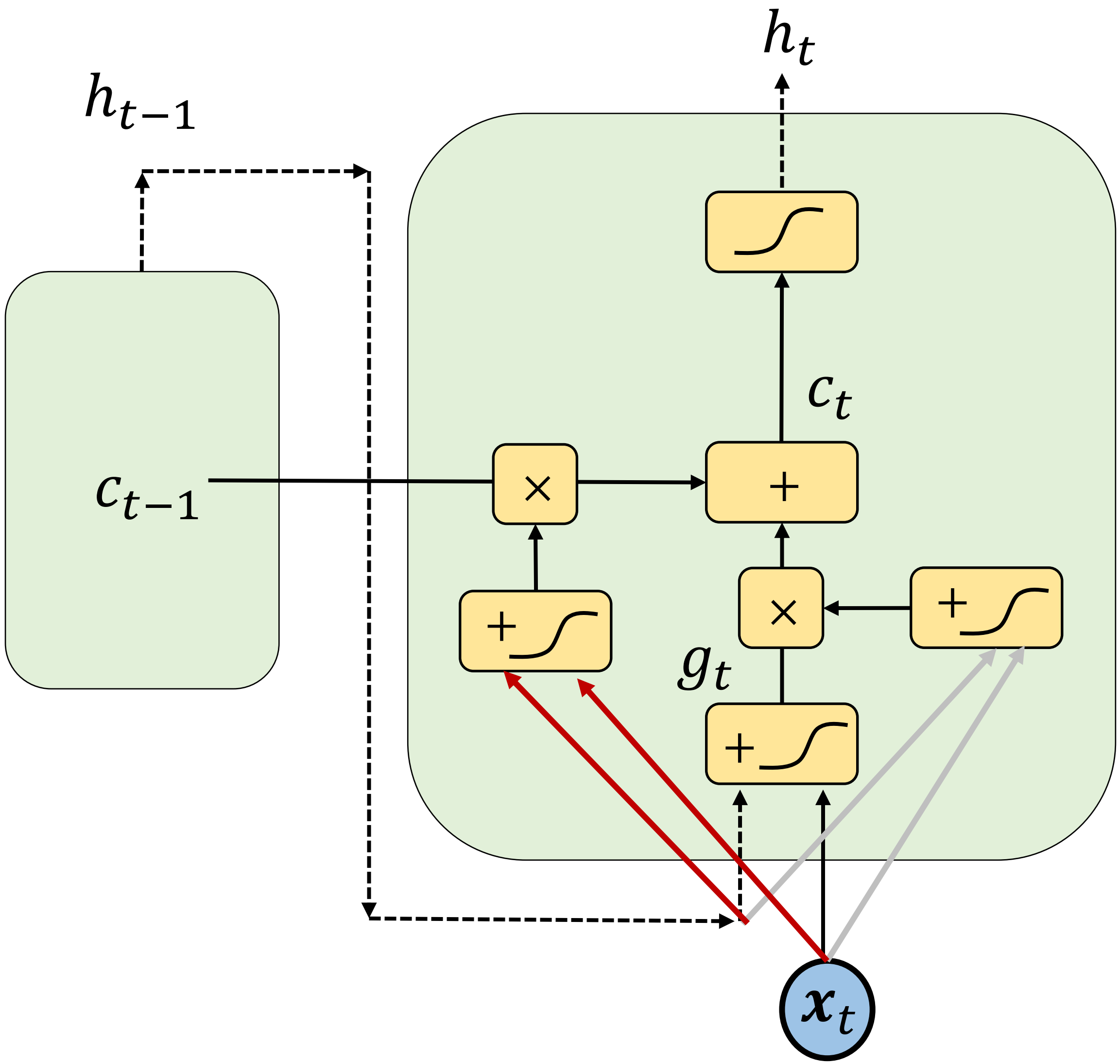
Input gate

Controls how inputs contribute to the internal state

$$I_t(x_t, h_{t-1})$$

Logistic sigmoid

LSTM Design – Step 2 (Gates)



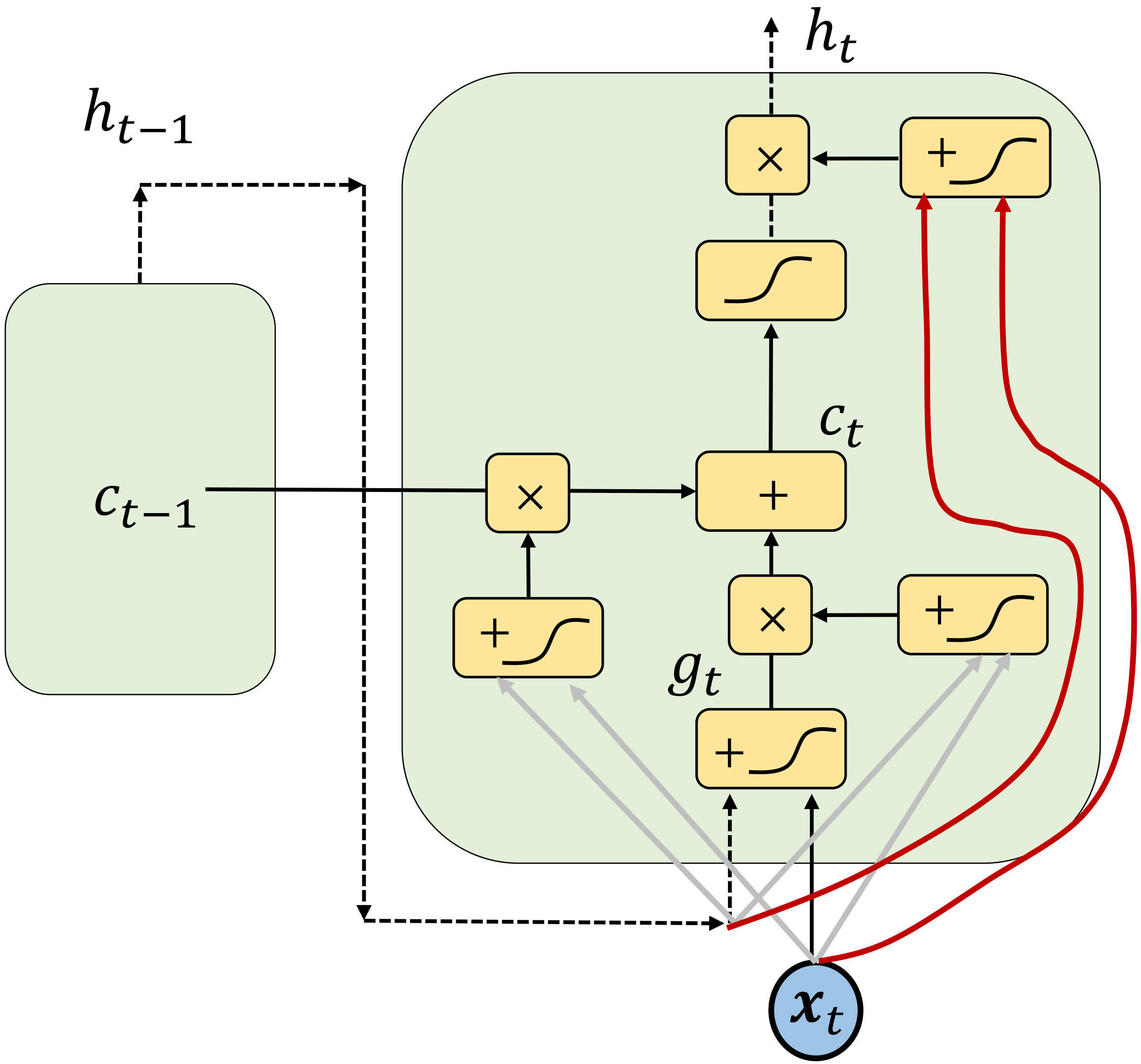
Forget gate

Controls how past internal state c_{t-1} contributes to c_t

$$F_t(x_t, h_{t-1})$$

Logistic sigmoid

LSTM Design – Step 2 (Gates)



Output gate

Controls what part of the internal state is propagated out of the cell

$$O_t(x_t, h_{t-1})$$

Logistic sigmoid

LSTM in Equations

1) Compute activation of input and forget gates

$$\boldsymbol{I}_t = \sigma(\boldsymbol{W}_{Ih}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{Iin}\boldsymbol{x}_t + \mathbf{b}_{\mathrm{I}})$$

$$\boldsymbol{F}_t = \sigma(\boldsymbol{W}_{Fh}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{Fin}\boldsymbol{x}_t + \mathbf{b}_{\mathrm{F}})$$

2) Compute input potential and internal state

$$\boldsymbol{g}_t = tanh(\boldsymbol{W}_{h}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{in}\boldsymbol{x}_t + \mathbf{b}_{\mathrm{h}})$$

$$\boldsymbol{c}_t = \boldsymbol{F}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{I}_t \odot \boldsymbol{g}_t$$

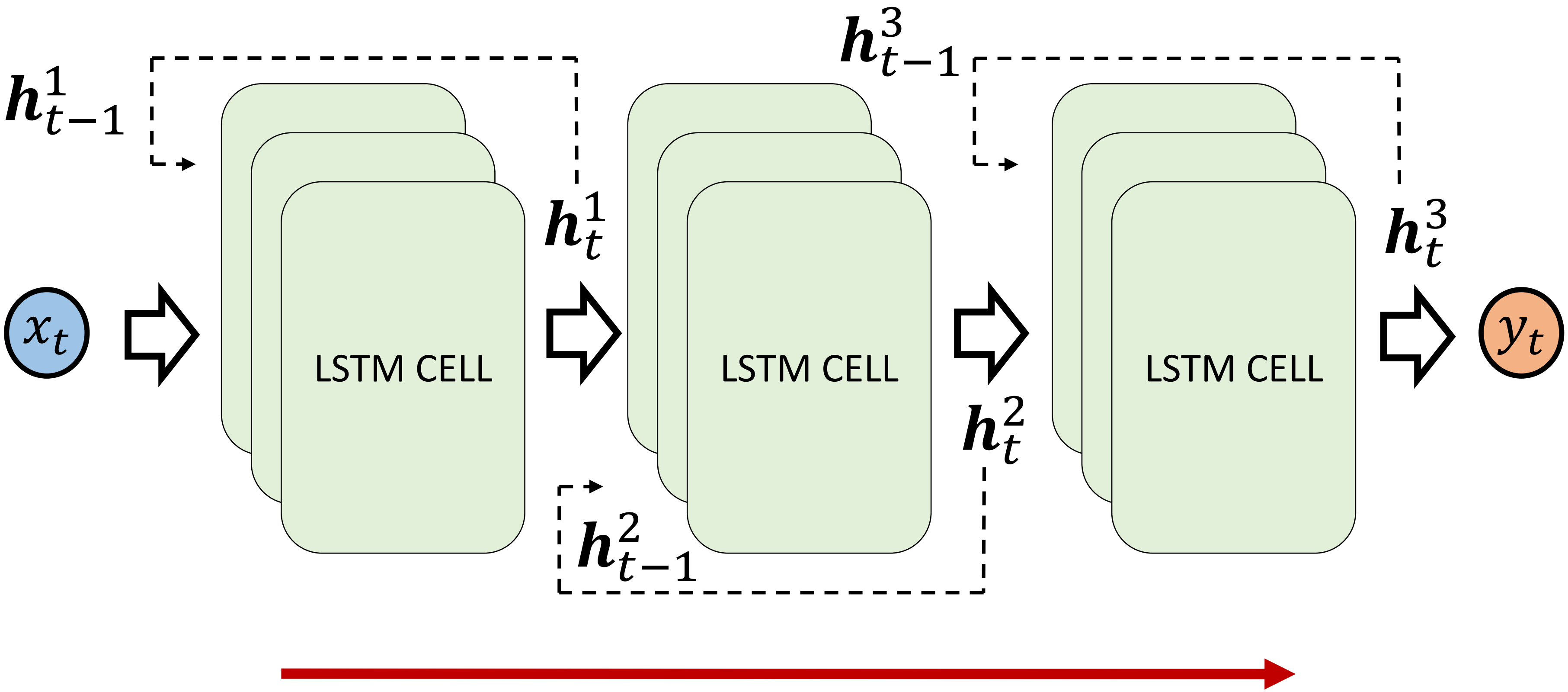
$\odot$ element-wise multiplication

3) Compute output gate and output state

$$\boldsymbol{O}_t = \sigma(\boldsymbol{W}_{Oh}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{Oin}\boldsymbol{x}_t + \mathbf{b}_{\mathrm{O}})$$

$$\boldsymbol{h}_t = \boldsymbol{O}_t \odot tanh(\boldsymbol{c}_t)$$

Deep LSTM



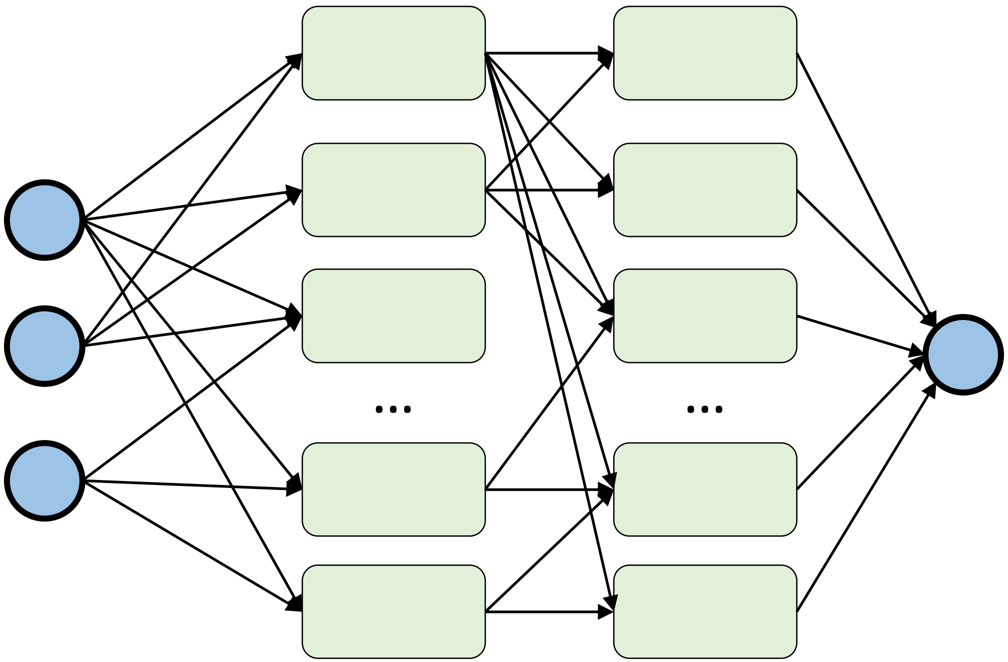
LSTM layers extract information at increasing levels of abstraction (enlarging context)

Training LSTM

- Original LSTM training algorithm was a mixture of RTRL and BPTT
  - BPTT on internal state gradient
  - RTRL-like truncation on other recurrent connections
  - No exact gradient calculation!
- All current LSTM implementation use full BPTT training
  - Introduced by Graves and Schmidhuber in 2005
  - Typically use Adam or RMSProp optimizer

Regularizing LSTM - Dropout

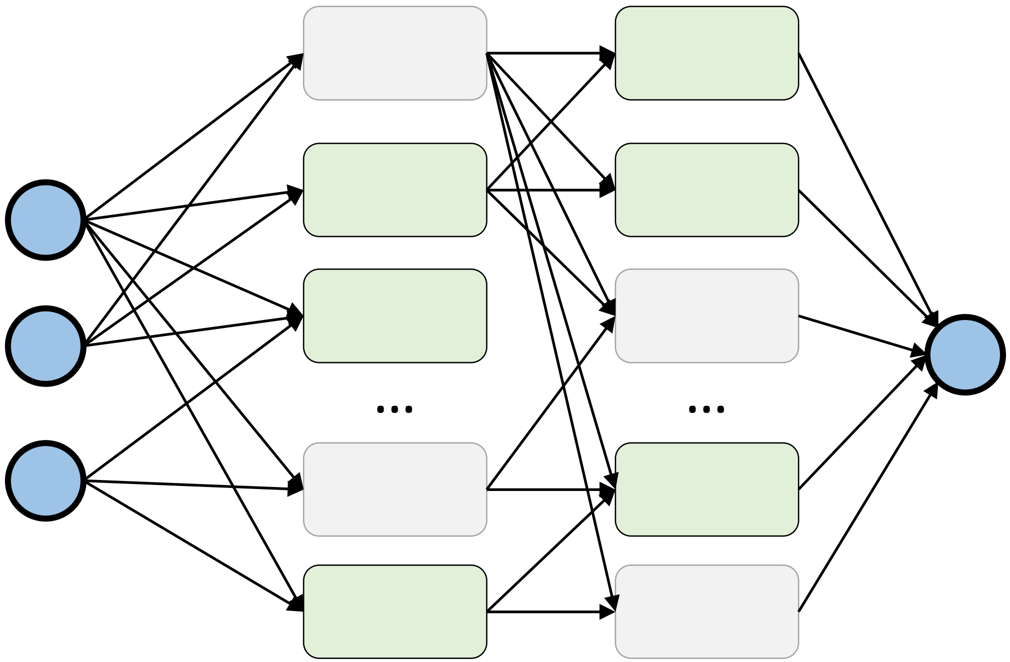
Randomly disconnect units from the network during training



N. Srivastava et al, Dropout: A Simple Way to Prevent Neural Networks from Overfitting, JLMR 2014

Regularizing LSTM - Dropout

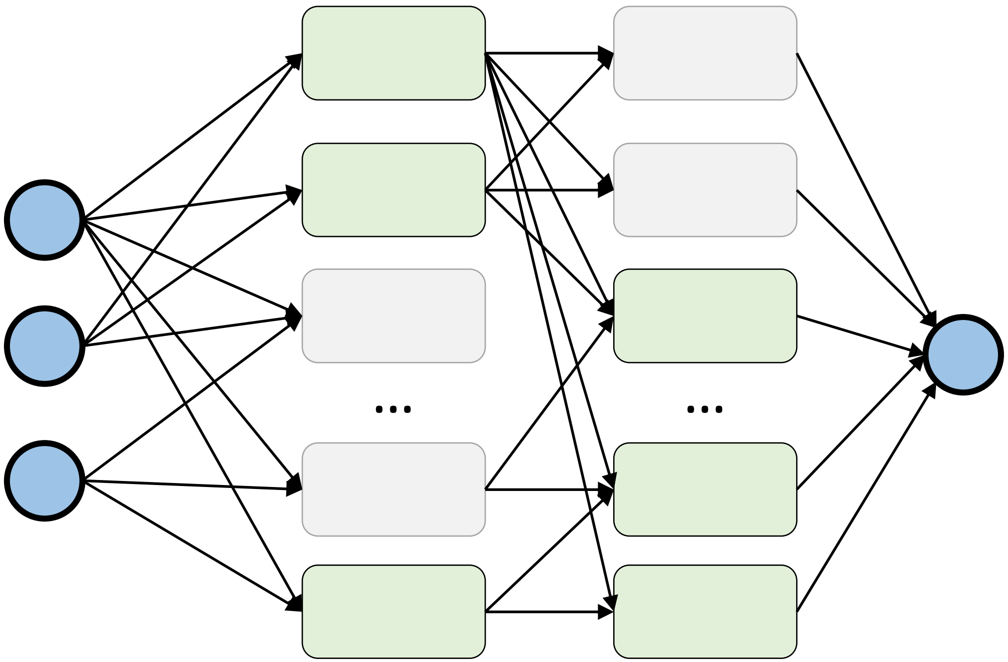
Randomly disconnect units from the network during training



N. Srivastava et al, Dropout: A Simple Way to Prevent Neural Networks from Overfitting, JLMR 2014

Regularizing LSTM - Dropout

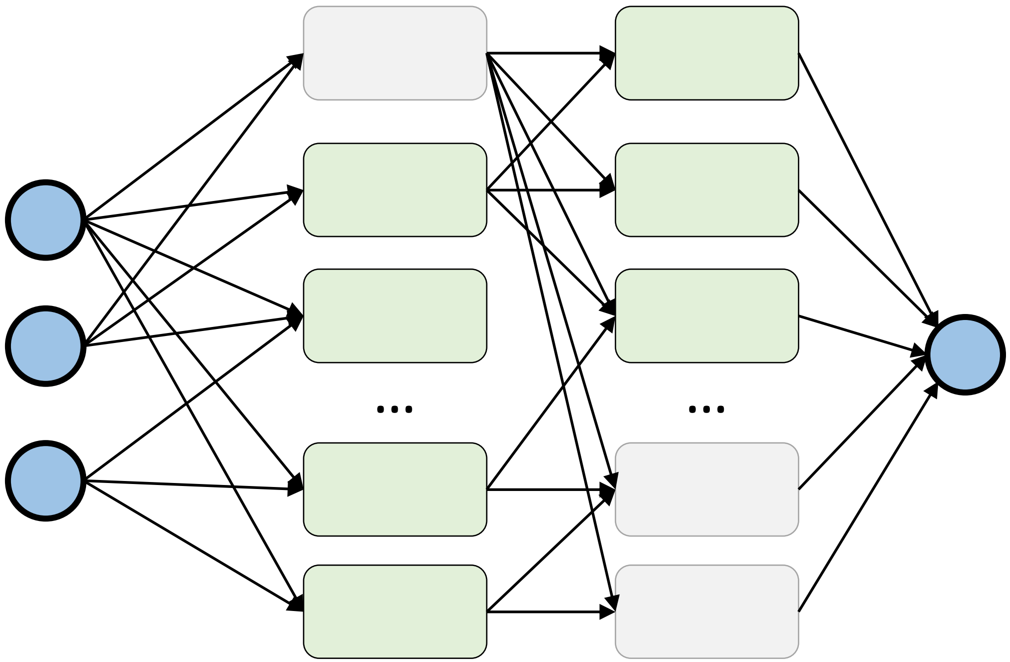
Randomly disconnect units from the network during training



N. Srivastava et al, Dropout: A Simple Way to Prevent Neural Networks from Overfitting, JLMR 2014

Regularizing LSTM - Dropout

Randomly disconnect units from the network during training

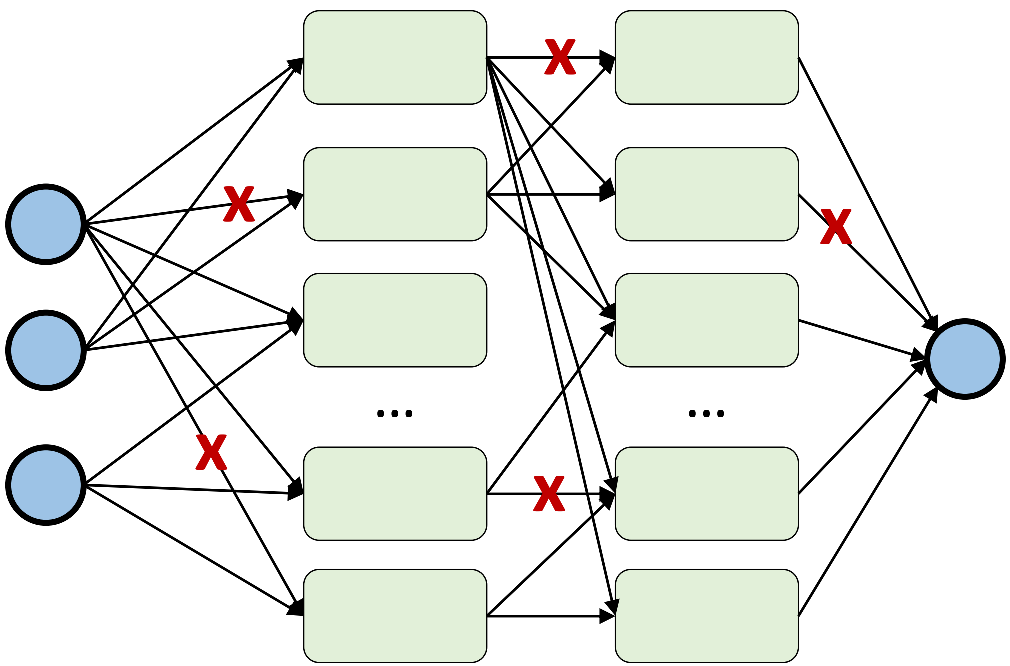


- Regulated by unit dropping hyperparameter
- Prevents unit coadaptation
- Committee machine effect
- Need to adapt prediction phase
- Drop units for the whole sequence!

N. Srivastava et al, Dropout: A Simple Way to Prevent Neural Networks from Overfitting, JLMR 2014

Regularizing LSTM - Dropout

Randomly disconnect units from the network during training

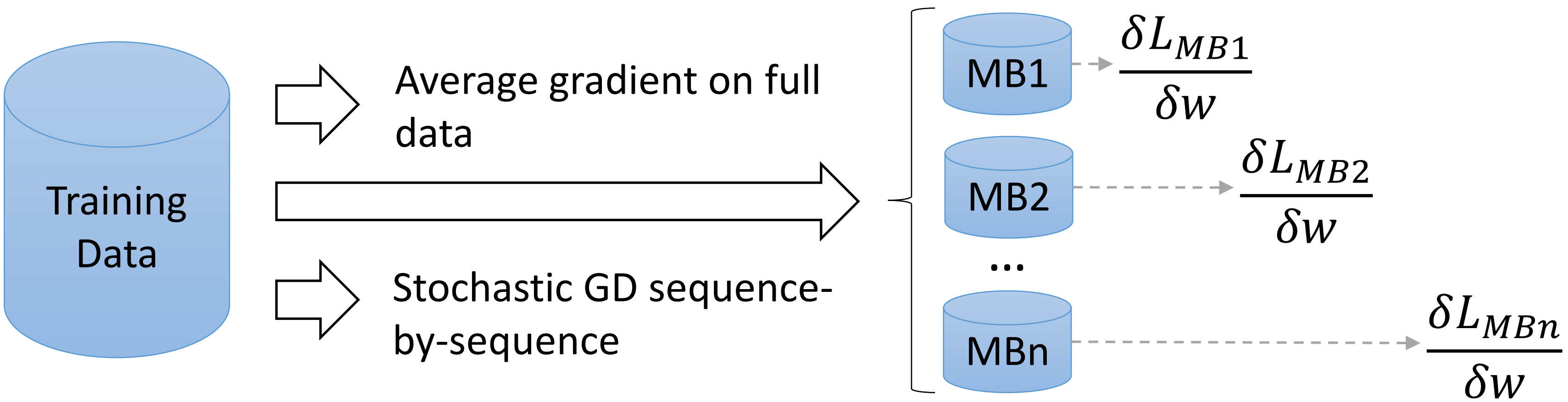


- Regulated by unit dropping hyperparameter
- Prevents unit coadaptation
- Committee machine effect
- Need to adapt prediction phase
- Drop units for the whole sequence!

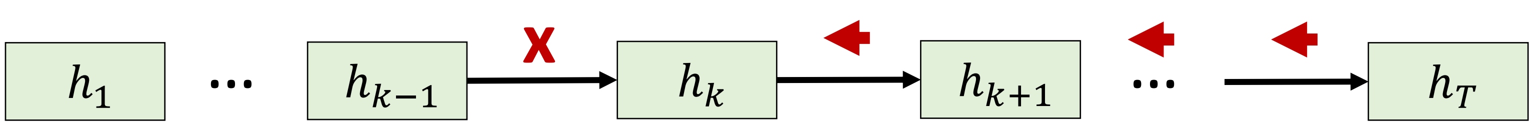
You can also drop single connections (dropconnect)

Practicalities – Minibatch and Truncated BP

Minibatch (MB)



Truncated gradient propagation



Gated Recurrent Unit (GRU)

Reset acts directly on output state (no internal state and no output gate)

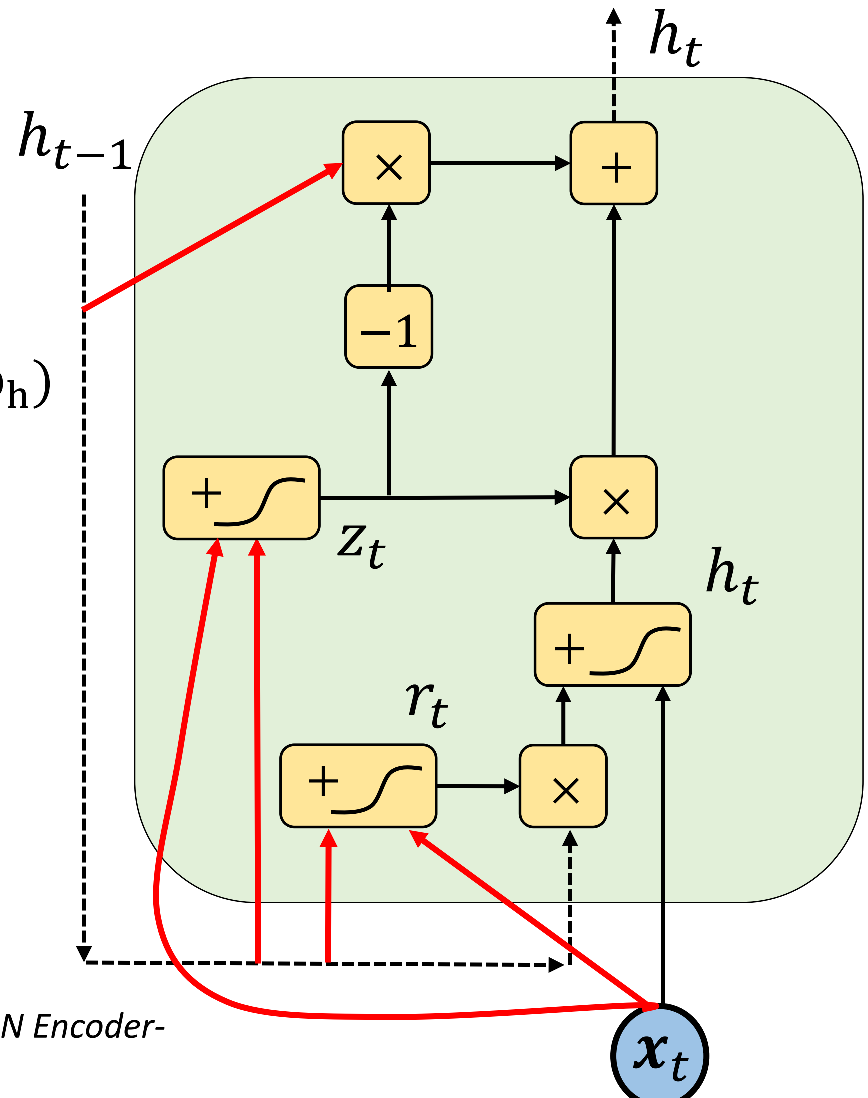
$$\boldsymbol{h}_t = (1 - \boldsymbol{z}_t) \odot \boldsymbol{h}_{t-1} + \boldsymbol{z}_t \odot \boldsymbol{h}_t$$

$$\boldsymbol{h}_t = tanh(\boldsymbol{W}_{hh}(\boldsymbol{r}_t \odot \boldsymbol{h}_{t-1}) + \boldsymbol{W}_{hin}\boldsymbol{x}_t + \mathbf{b}_{\mathrm{h}})$$

Reset and update gates when coupled act as input and forget gates

$$\boldsymbol{z}_t = \sigma(\boldsymbol{W}_{zh}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{zin}\boldsymbol{x}_t + \mathbf{b}_{\mathrm{z}})$$

$$\boldsymbol{r}_t = \sigma(\boldsymbol{W}_{rh}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{rin}\boldsymbol{x}_t + \mathbf{b}_{\mathrm{r}})$$



C. Kyunghyun et al, Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation, EMNLP 2014

Bidirectional LSTM – Character Recognition

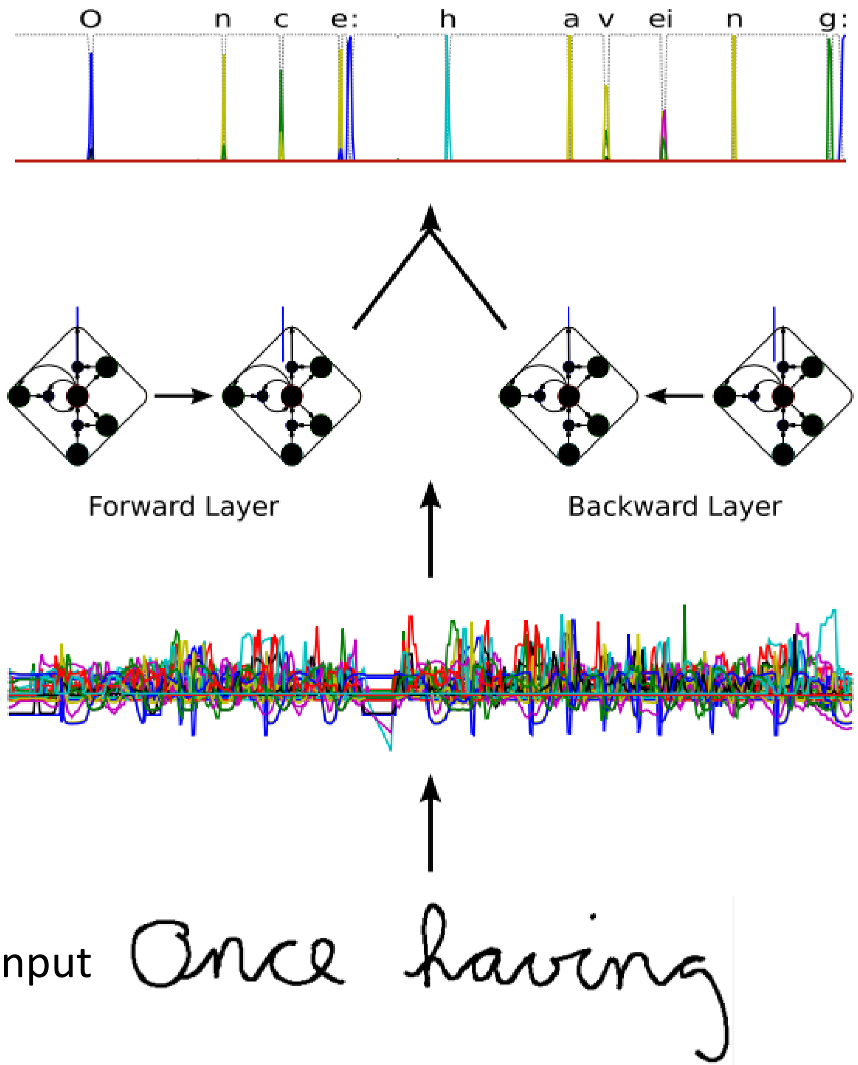
Character distribution

1 output for each character plus no output symbol

LSTM layers

Preprocessed input

Original input



A. Graves, A novel connectionist system for unconstrained handwriting recognition, TPAMI 2009

Bidirectional LSTM – Character Recognition

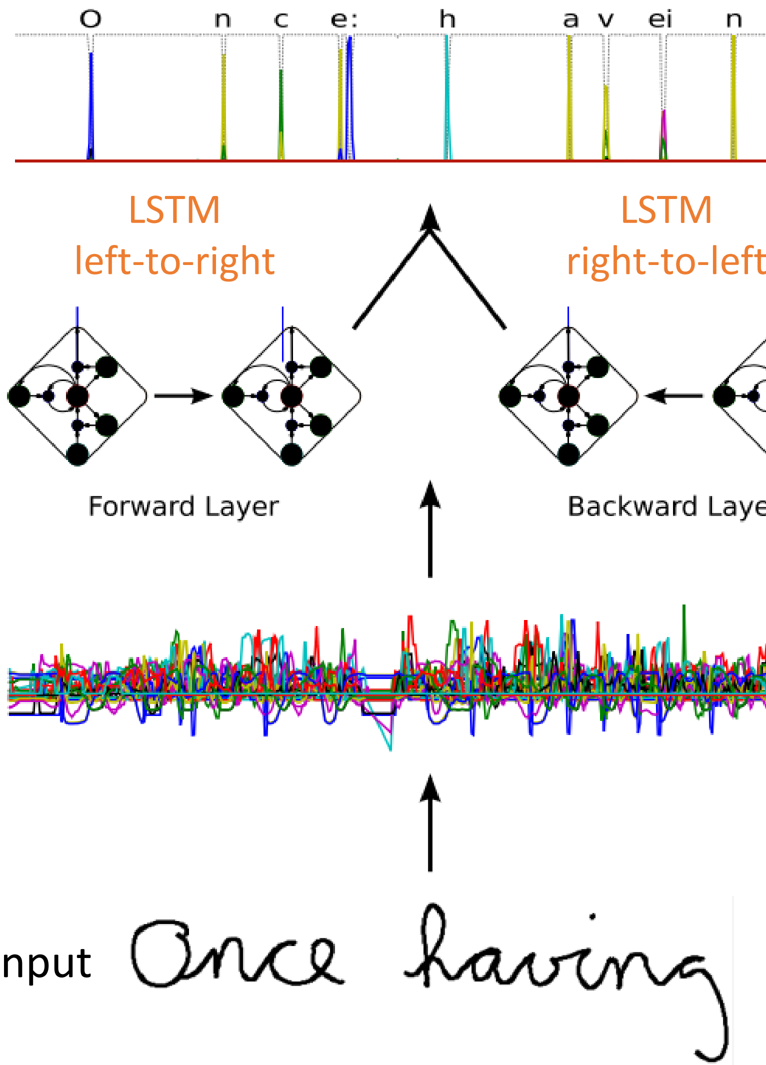
Character distribution

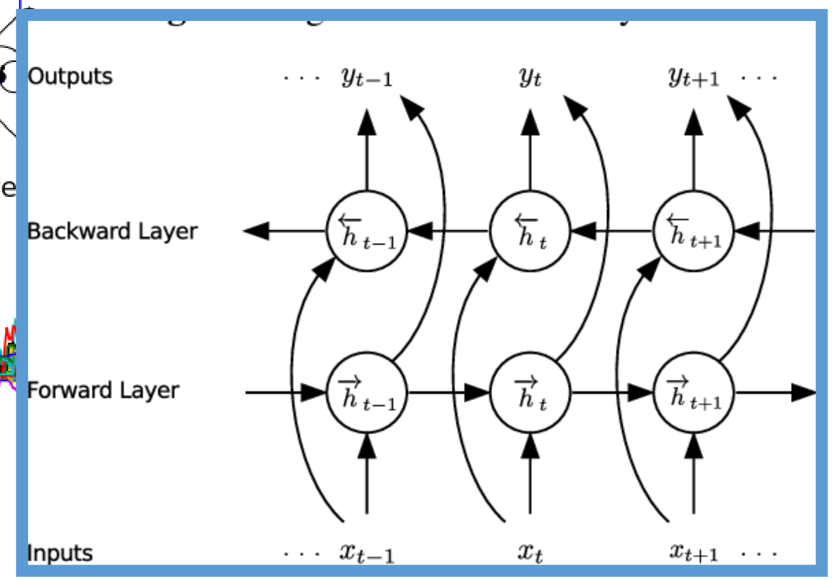
1 output for each character plus no output symbol

LSTM layers

Preprocessed input

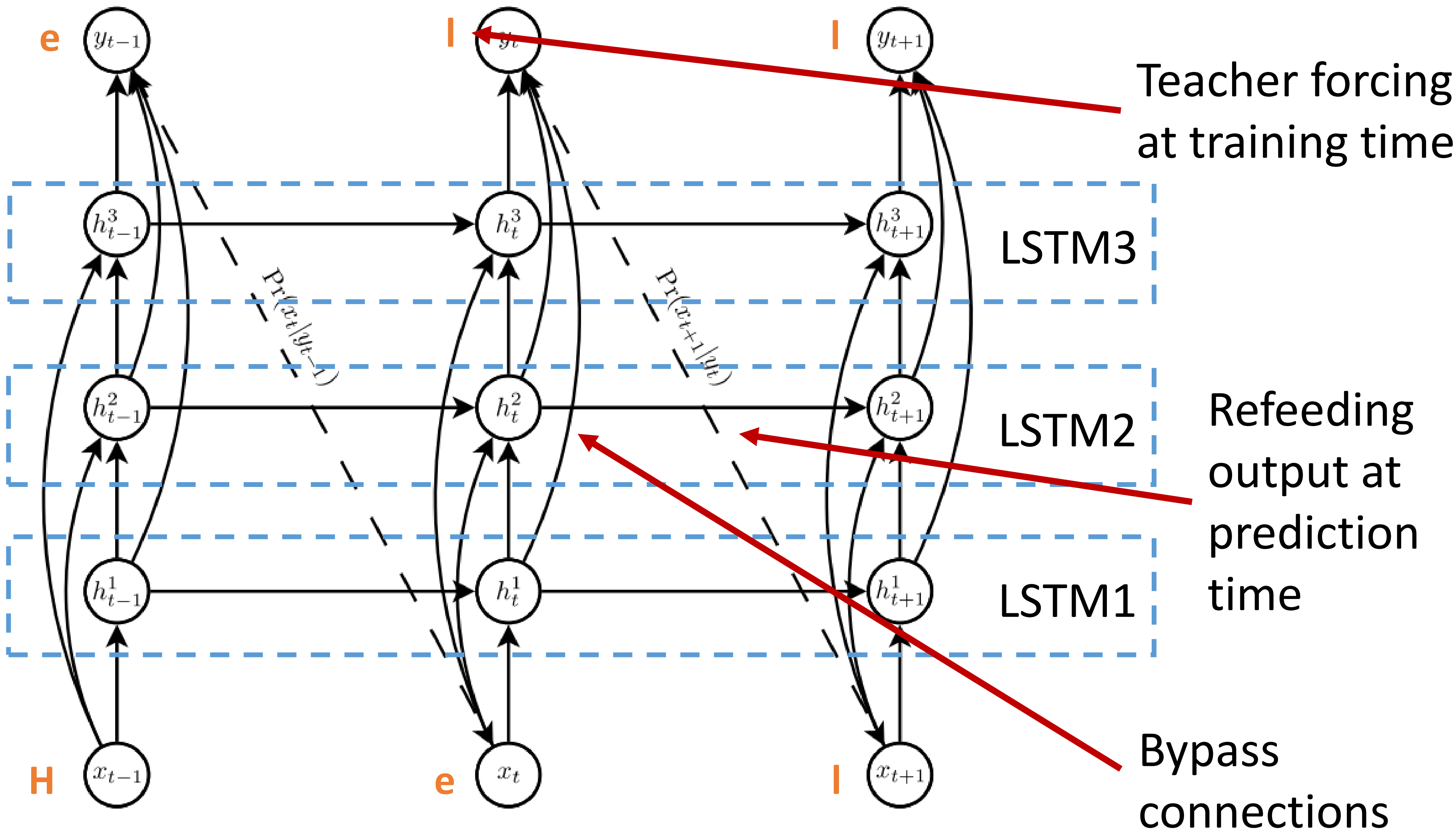
Original input





A. Graves, A novel connectionist system for unconstrained handwriting recognition, TPAMI 2009

Generative Use of LSTM/GRU



A. Graves, Generating Sequences With Recurrent Neural Networks, 2013

Character Generation Fun

Shakespeare

PANDARUS:
Alas, I think he shall be come approached and the day
When little srain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:
They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

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

Character Generation Fun

Linux Kernel Code

```
/*
 * If this error is set, we will need anything right after that BSD.
 */
static void action_new_function(struct s_stat_info *wb)
{
  unsigned long flags;
  int lel_idx_bit = e->edd, *sys & ~((unsigned long) *FIRST_COMPAT);
  buf[0] = 0xFFFFFFFF & (bit << 4);
  min(inc, slist->bytes);
  printk(KERN_WARNING "Memory allocated %02x/%02x, "
    "original MLL instead\n"),
    min(min(multi_run - s->len, max) * num_data_in),
    frame_pos, sz + first_seg);
  div_u64_w(val, inb_p);
  spin_unlock(&disk->queue_lock);
  mutex_unlock(&s->sock->mutex);
  mutex_unlock(&func->mutex);
  return disassemble(info->pending_bh);
}
```

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

Generate Sad Jokes

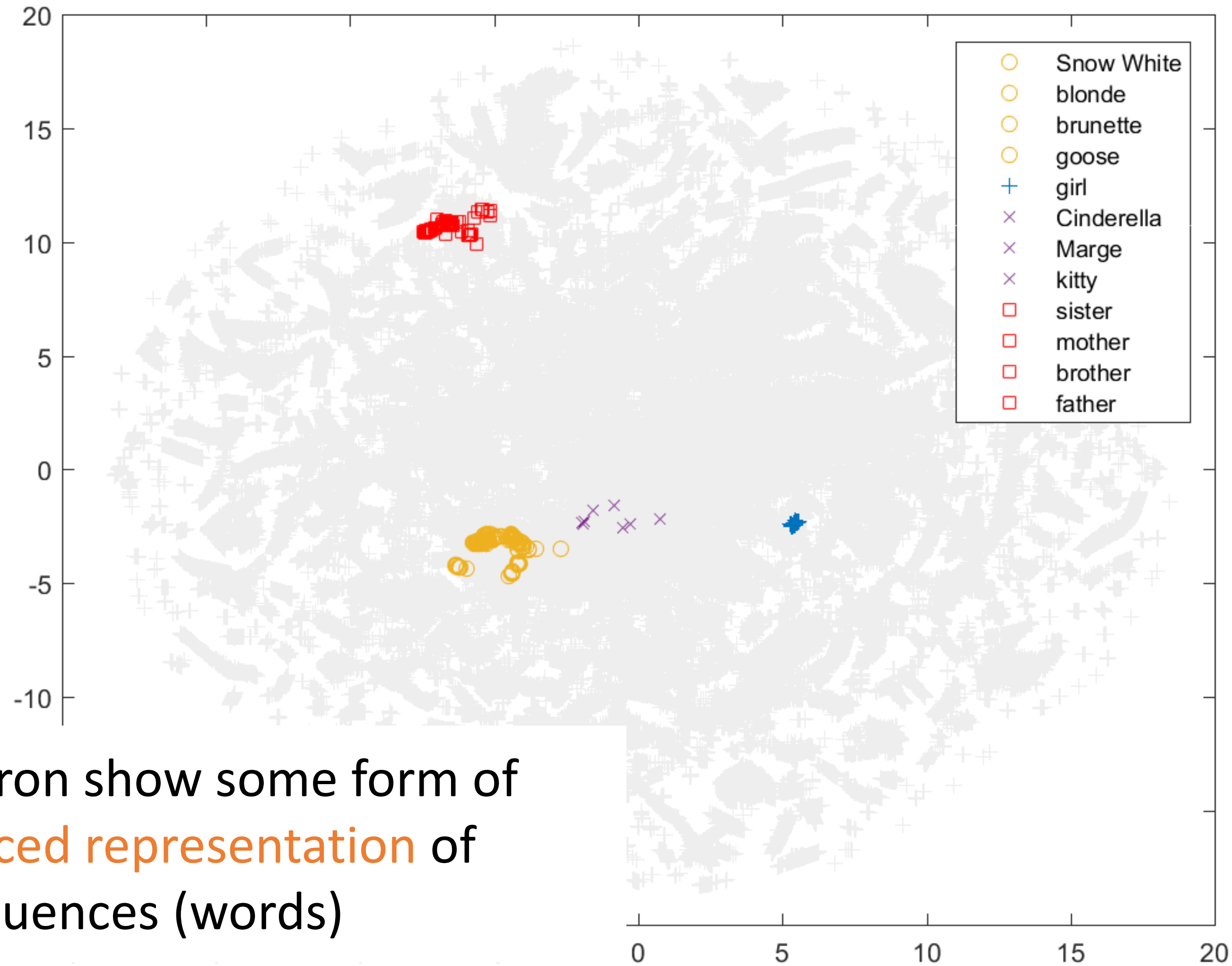
A 3-LSTM layers neural network to generate English jokes character by character

Why did the boy stop his homework?
Because they're bunny boo!

What do you get if you cross a famous California little boy with an elephant for players?
Market holes.

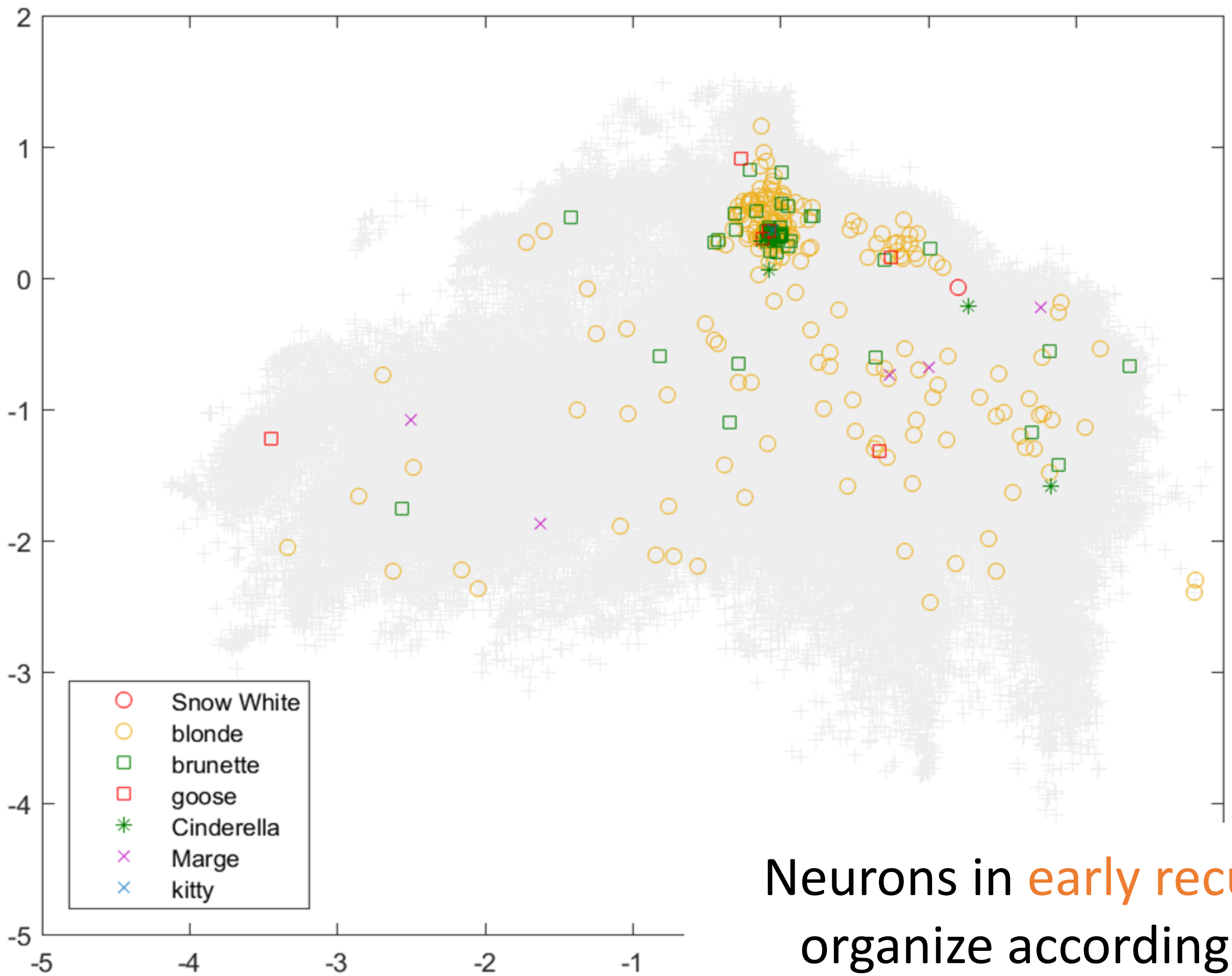
Q: Why did the death penis learn string?
A: Because he wanted to have some roasts case!

Understanding Memory Representation



At Layer-3 neuron show some form of context induced representation of subsequences (words)

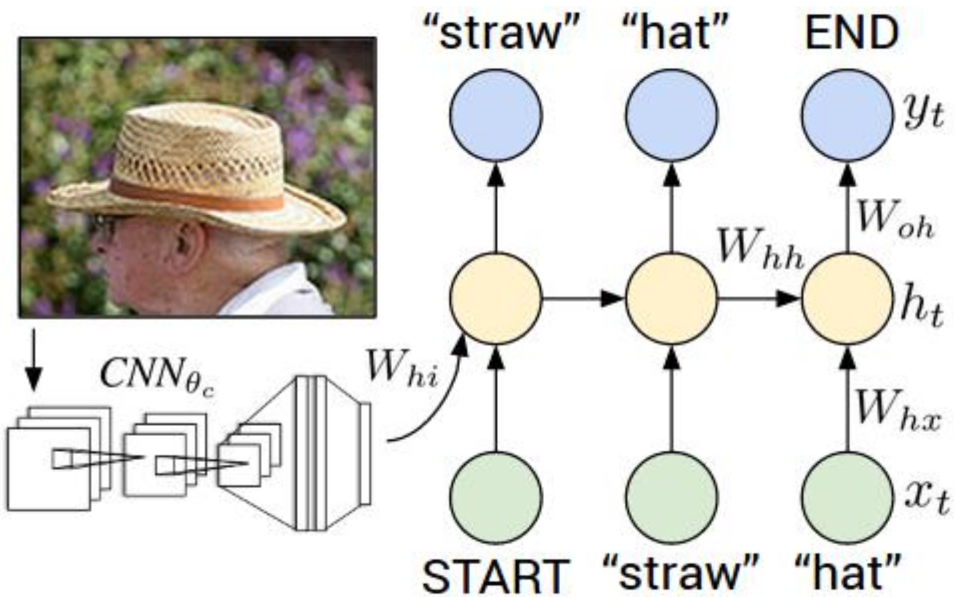
Understanding Memory Representation

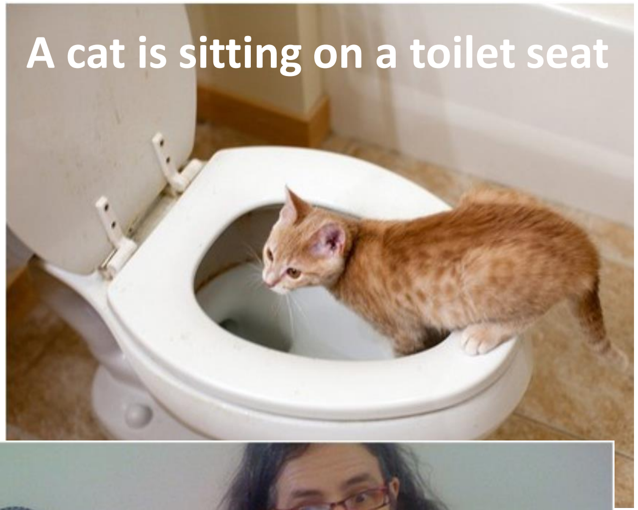


Neurons in early recurrent layers tend to organize according to sequence suffix

More Differentiable Compositions

CNN-LSTM Composition for image-to-sequence (NeuralTalk)

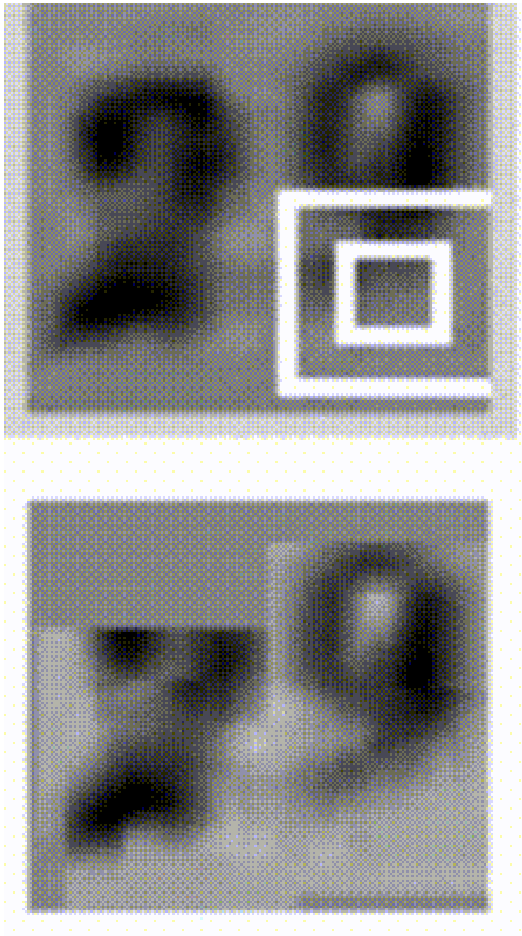


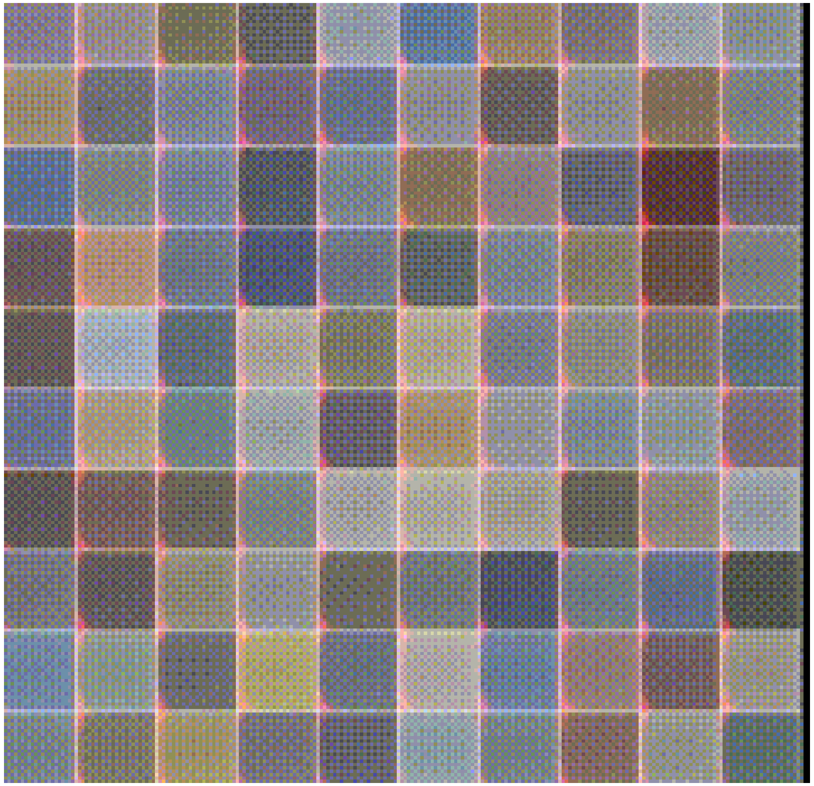


A. Karpathy and L. Fei-Fei, Deep Visual-Semantic Alignments for Generating Image Descriptions, CVPR 2015
https://github.com/karpathy/neuraltalk2

RNN – A Modern View

RNN are only for sequential/structured data?





RNN as stateful systems

Software

- Standard LSTM and GRU are available in all deep learning frameworks (Python et al) as well as in Matlab's Neural Network Toolbox
- If you want to play with one-element ahead sequence generation try out char-RNN implementations
    - https://github.com/karpathy/char-rnn (ORIGINAL)
    - https://github.com/sherjilozair/char-rnn-tensorflow
    - https://github.com/crazydonkey200/tensorflow-char-rnn
    - http://pytorch.org/tutorials/intermediate/char_rnn_generation_tutorial.html

Take Home Messages

- Learning long-term dependencies can be difficult due to gradient vanish/explosion
- Gated RNN solution
  - Gates are neurons whose output is used to scale another neuron's output
  - Use gates to determine what information can enter (or exit) the internal state
  - Training gated RNN non always straightforward
- Deep RNN can be used in generative mode
  - Can seed the network with neural embeddings
- Deep RNN as stateful and differentiable machines