Recurrent Neural Networks

Jan Chorowski
University of Wroclaw

Slides originaly presented at TMLSS 2018

Outline

- Why RNNs?
- Gradient troubles.
- LSTMs
- Practical training tips
- Non-sequential data
- Discussion & controversies (why RNNs pt. 2)

WHY RNNS?

Supervised learning

Neural nets excel at implementing functions

$$f: \mathbb{R}^n \to \mathbb{R}^m$$

Crucial modeling assumption: n, m are known and fixed.

Variable sized data

- Classify if text is grammatical.
- Generate a sentence.
- Forecast demand for a product based on sales history
- Translation: map this sequence of English words to French ones
- Speech recognition: map this sequence of speech to characters.

Text generation

Generate a sentence

(or equivalently)

Learn a function that assigns probabilities to sentences.

Formally:

Let $x_1, x_2, \dots, x_T$ be the words of the sentence.

We want: $p(x_1, x_2, x_3, \dots, x_T)$.

Text generation solutions

$$p(x_1, x_2, x_3, \ldots, x_T) =$$
$$p(x_1)p(x_2|x_1) \ldots p(x_T|x_1, \ldots, x_{T-1}) =$$
$$\prod_{t=1}^{T} p(x_t|x_1, \ldots, x_{t-1})$$

Now we need to learn $p(x_t|x_1, \ldots, x_{t-1})$!

Learning $p(x_t|x_1, \dots, x_{t-1})$…

1. Assume distant past doesn't matter,
   $p(x_t|x_1, \dots, x_{t-1}) \approx p(x_t|x_{t-n+1}, \dots, x_{t-1})$

   These are called "n-gram text models"

Learning $p(x_t|x_1, \ldots, x_{t-1})$…

2. Simply reduce $x_1, \ldots, x_{t-1}$ to a fixed-size representation

   Can take weighted average of word embeddings

   This is nicely developed in Attention is All You Need paper.

Learning $p(x_t|x_1, \ldots, x_{t-1})$…

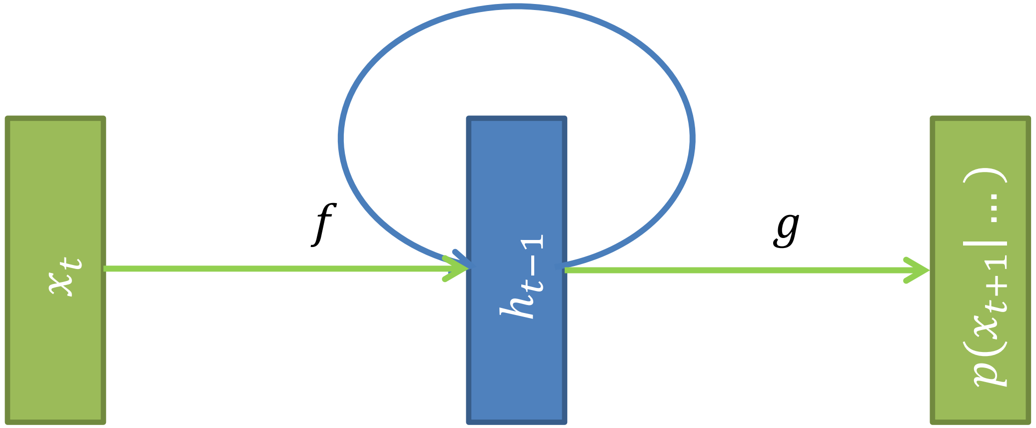
3. Introduce a hidden (i.e. not directly observed) state h_t.

   h_t summarizes $x_1, \ldots, x_{t-1}$.

   Compute h_t using this recurrence:

$$h_t = f(h_{t-1}, x_t)$$
$$p(x_{t+1}|x_1, \ldots, x_t) = g(h_t)$$

RNNs are networks with state



$$h_t = f(h_{t-1}, x_t)$$
$$p(x_{t+1}|x_1, \ldots, x_t) = g(h_t)$$

f, g are implemented as feedforward neural nets.

RNN Example

Input: a sequence of bits 1,0,1,0,0,1

Output: the parity

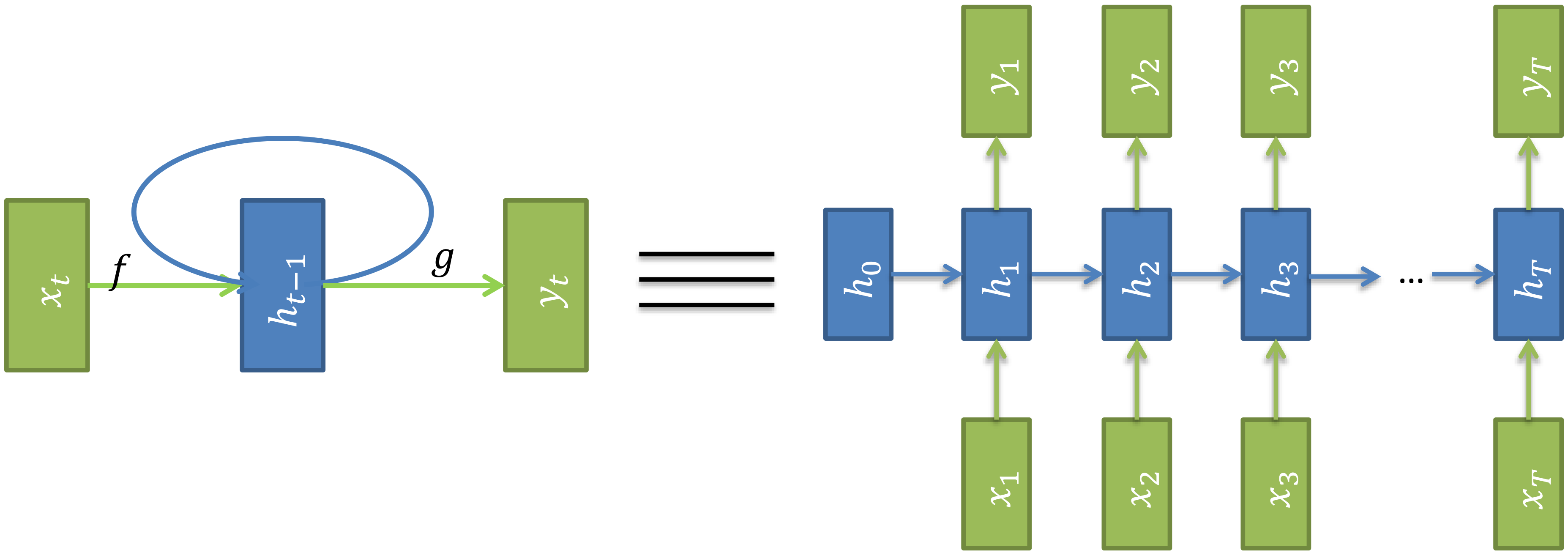
Solution:

The hidden state will be just 1 bit – parity so far:

$$h_0 = 0$$
$$h_t = XOR(h_{t-1}, x_t)$$
$$y_T = h_T$$

RNNs are dynamical systems

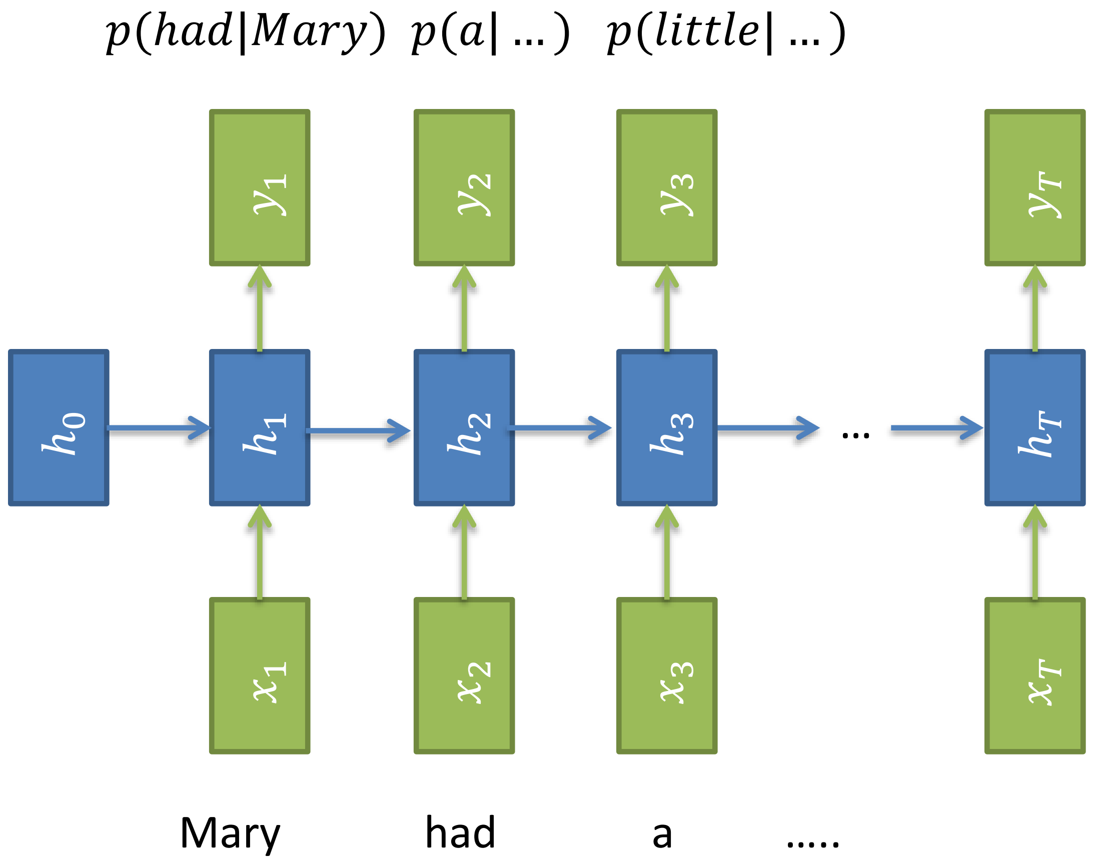
Time is discrete, we can unroll:



Thus the RNN is a very deep network, with same "arrows" computing the same function!

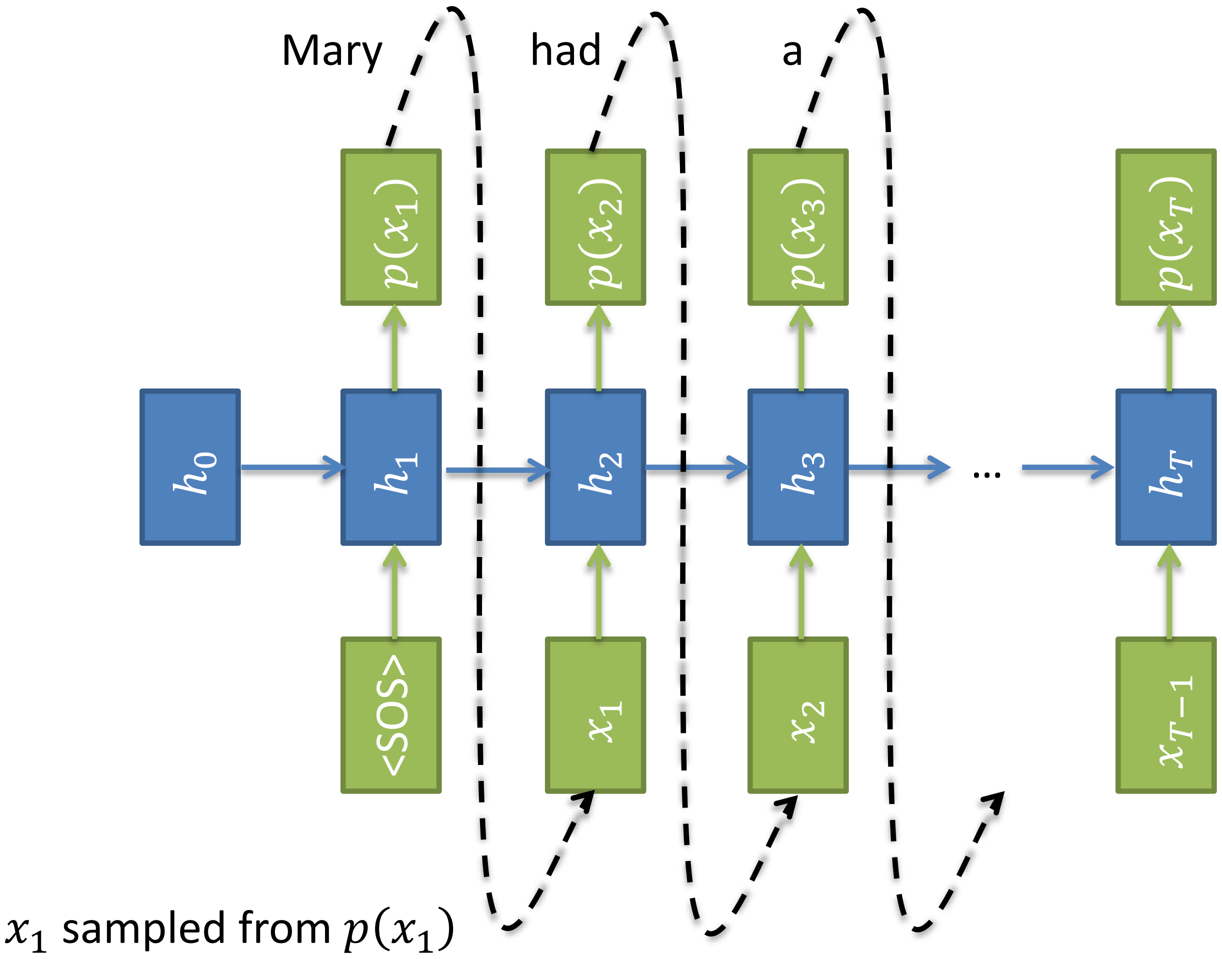
RNNs transform sequences

Transform a sequence $x_1, x_2, \dots, x_T$ into $y_1, y_2, \dots, y_T$:



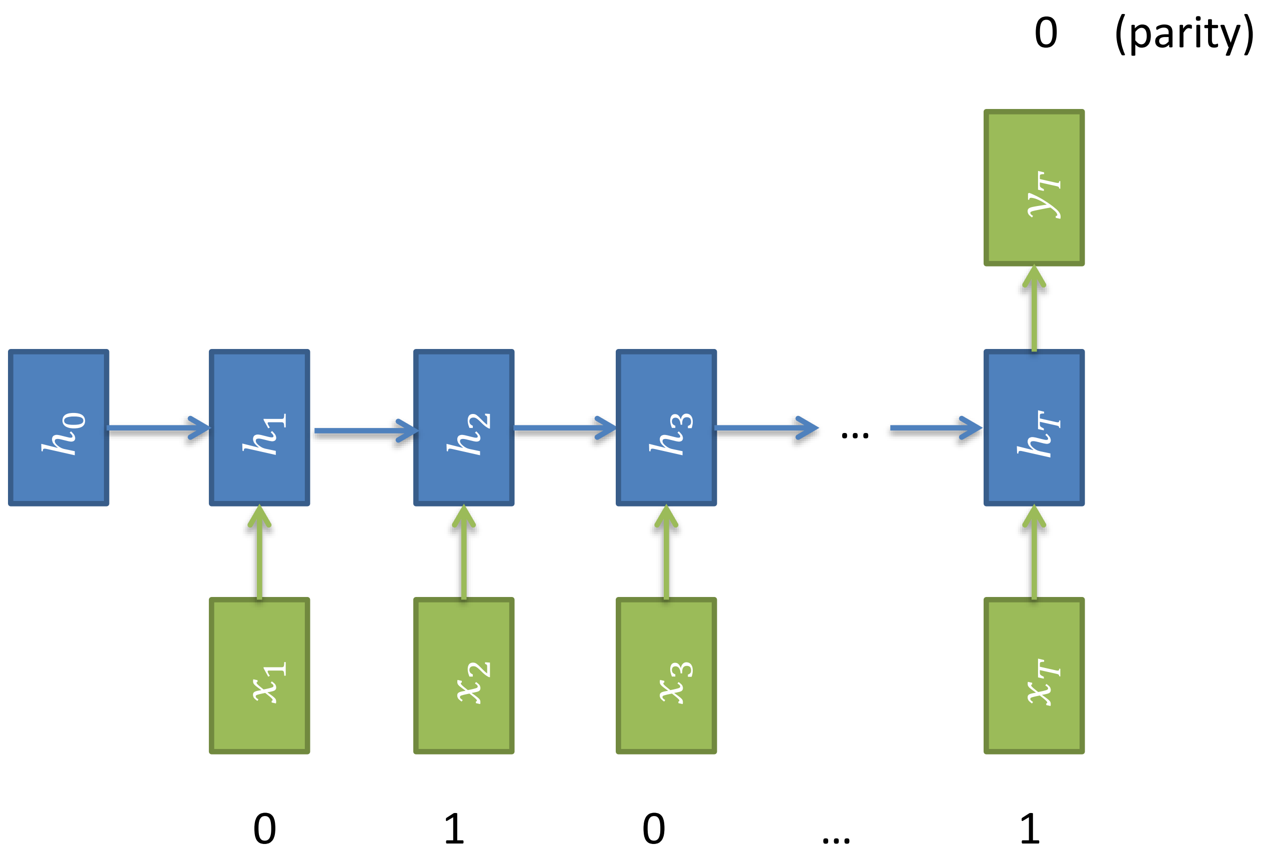
RNNs generate sequences

Generate a sequence (scan)



RNNs summarize sequences

Summarize a sequence (fold):



RNN TRAINING, GRADIENT TROUBLE!

Training RNNs

In principle very easy:

1. Unroll in time
2. Apply the backpropagation algorithm to get gradients with respect to parameters
3. Train as usual

In practice:

Gradient computation can be numerically unstable.

Getting good gradients is difficult.

A linear dynamical system

$$h_t = w \cdot h_{t-1} = w^t h_0$$

When:

1. $w > 1$ it diverges: $\lim_{t \to \infty} h_t = \mathrm{sign}(\mathrm{h}_0) \cdot \infty$
2. $w = 1$ it is constant $\lim_{t \to \infty} h_t = \mathrm{h}_0$
3. $-1 < w < 1$ it decays to 0: $\lim_{t \to \infty} h_t = 0$
4. $w = -1$ flips between $\pm h_0$
5. $w < -1$ diverges, changes sign at each step

A multidimensional linear dyn. system

$$h_t \in \mathbb{R}^n, W \in \mathbb{R}^{n\times n}$$
$$h_t = Wh_{t-1}$$

Compute the eigen-decomposition of W:

$$W = Q\Lambda Q^{-1} = Q \begin{bmatrix} \lambda_{11} & & \\ & \ddots & \\ & & \lambda_{nn} \end{bmatrix} Q^{-1}$$

Then

$$h_t = Wh_{t-1} = W^t h_0 = Q\Lambda^{\mathrm{t}} Q^{-1} h_0 =$$
$$= Q \begin{bmatrix} \lambda_{11}^t & & \\ & \ddots & \\ & & \lambda_{nn}^t \end{bmatrix} Q^{-1} h_0$$

A multidimensional dyn. system cont.

$$h_t = W h_{t-1} = W^t h_0 = Q \Lambda^{\mathrm{t}} Q^{-1} h_0$$

If largest eigenvalue has norm:

1. $|\lambda_1|$>1, system diverges
2. $|\lambda_1|$ =1, system is stable or oscillates
3. $|\lambda_1|$ <1, system decays to 0

This is similar to the scalar case.

A nonlinear dynamical system

$$h_t = \tanh(w \cdot h_{t-1})$$

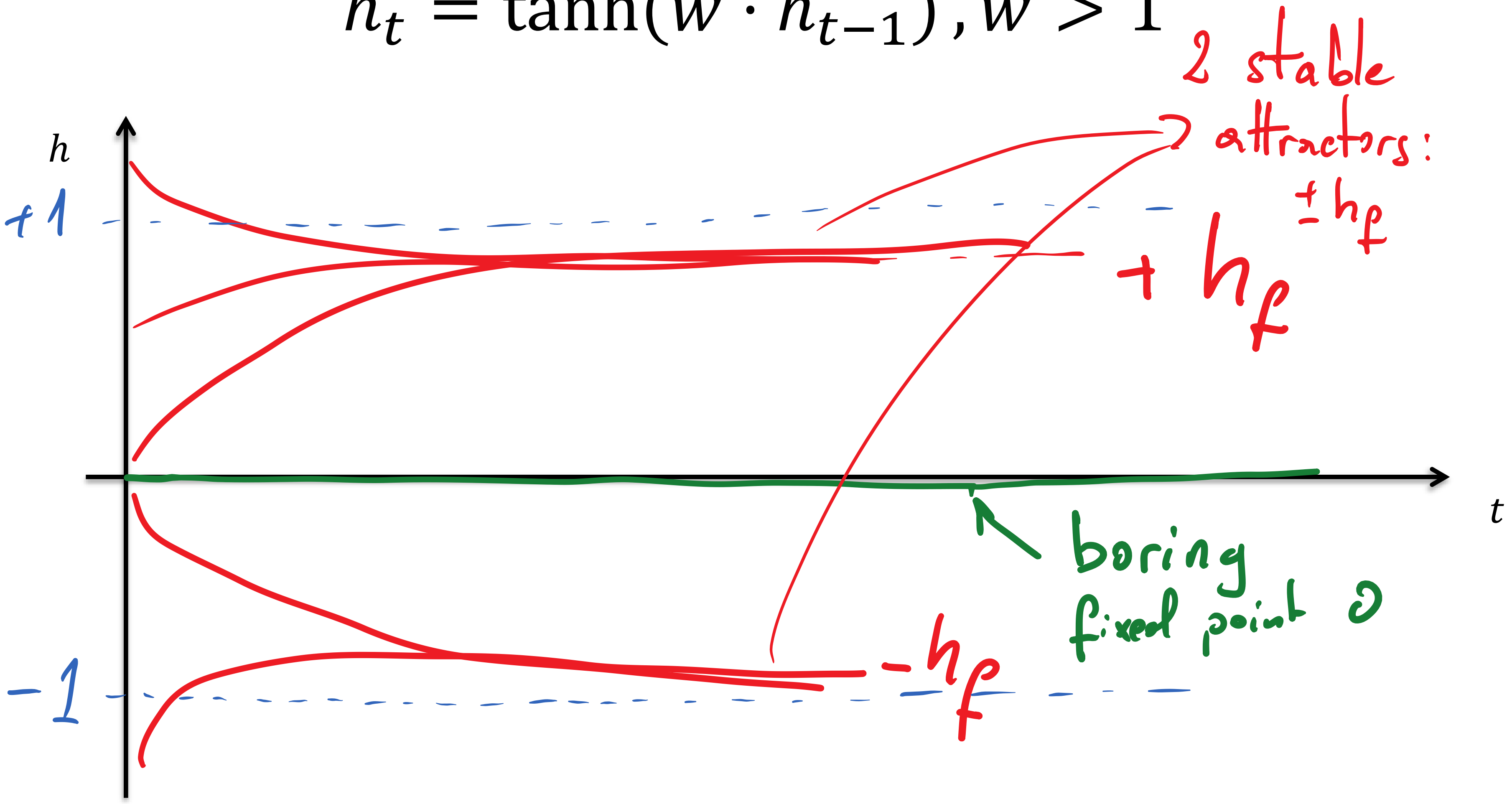
Output is bounded (can't diverge!)

If $w > 1$ has 3 fixed points: $0, \pm h_f$. Starting the iteration form $h_0 \neq 0$ it ends at $\pm h_f$ (effectively remembers the sign).

If $w \leq 1$ has one fixed point: 0

A nonlinear dynamical system

$$h_t = \tanh(w \cdot h_{t-1})\,, w > 1$$



Gradients in RNNs

Recall RNN equations:

$$h_t = f(h_{t-1}, x_t; \theta)$$
$$y_t = g(h_t)$$

Assume supervision only at the last step:

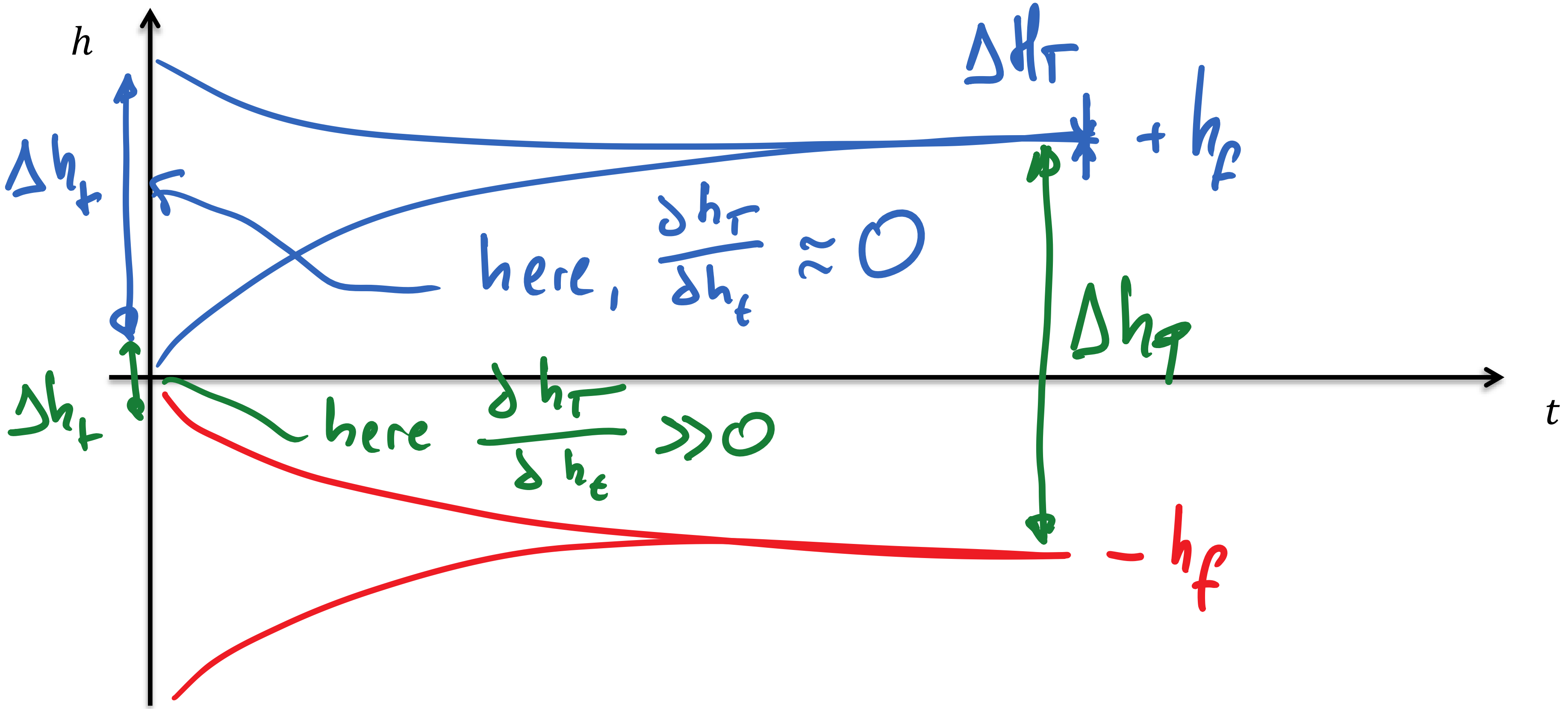
$$\mathrm{L} = e(y_T)$$

The gradient is

$$\frac{\partial L}{\partial \theta} = \sum_t \frac{\partial L}{\partial h_T} \boldsymbol{\frac{\partial h_T}{\partial h_t}} \frac{\partial h_t}{\partial \theta}$$

Trouble with $\frac{\partial h_T}{\partial h_t} \approx \frac{\Delta h_T}{\Delta h_t}$

$\frac{\partial h_T}{\partial h_t}$ measures how much h_T changes when h_t changes.



Another look at $\frac{\partial h_T}{\partial h_t}$

Let:

$$h_t = \tanh(w \cdot h_{t-1})$$

Then:

$$\frac{\partial h_t}{\partial h_0} = \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial h_{t-2}} \frac{\partial h_{t-2}}{\partial h_{t-3}} \dots \frac{\partial h_1}{\partial h_0}$$

$$\frac{\partial h_{i+1}}{\partial h_i} = \tanh'(wh_t)\, w$$

$$\frac{\partial h_t}{\partial h_0} = \prod_{i=0}^{t-1} \tanh'(wh_i)\, w = w^t \prod_{i=0}^{t-1} \tanh'(wh_i)$$

Backward phase is linear!

Vanishing gradient

If $\frac{\partial h_T}{\partial h_t} = 0$ the network forgets all information from time t when it reaches time T.

This makes it impossible to discover correlations between inputs at distant time steps.

This is a modeling problem.

Exploding gradient

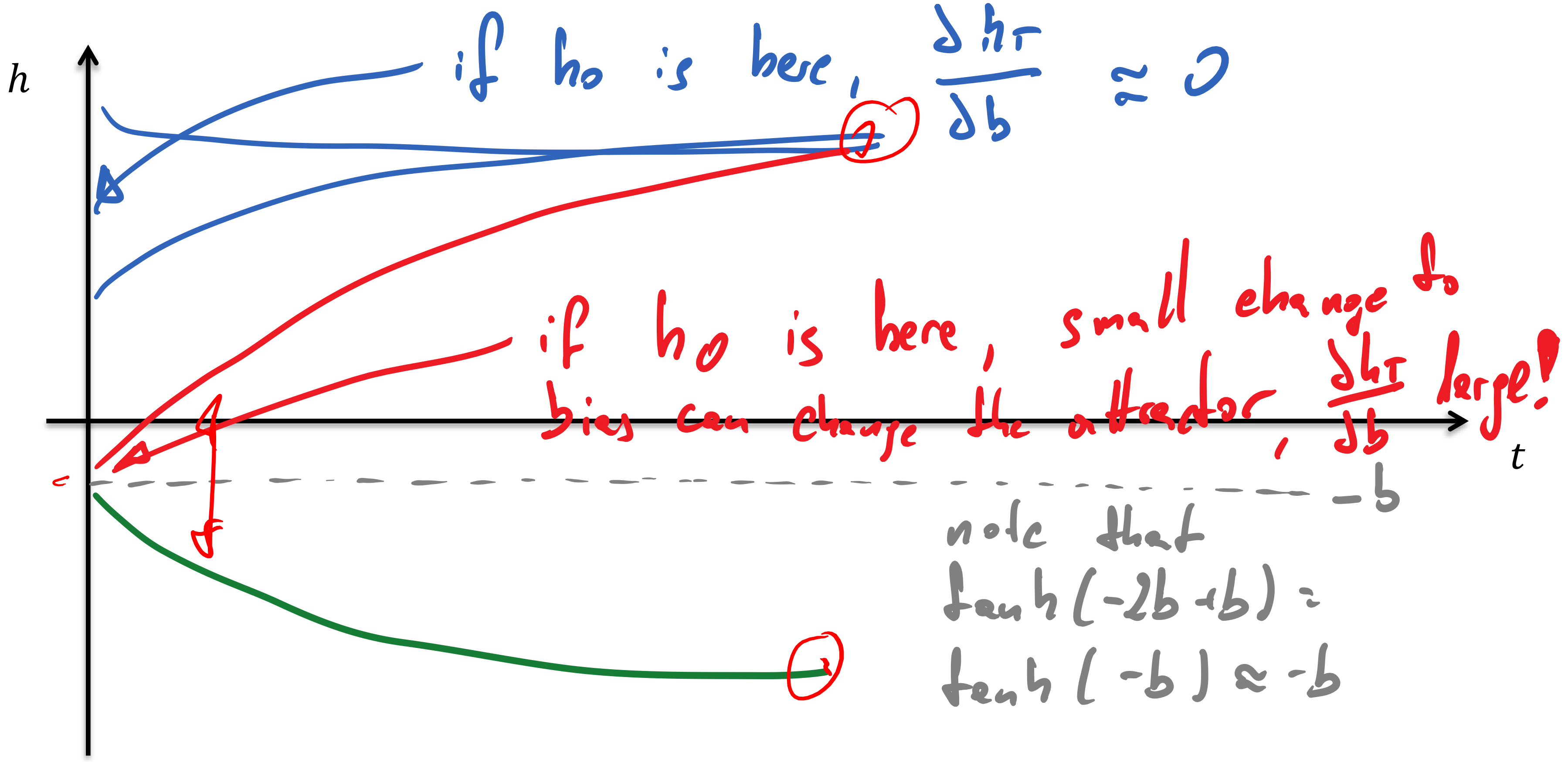
When $\frac{\partial h_T}{\partial h_t}$ is large $\frac{\partial \mathrm{Loss}}{\partial \theta}$ will be large too.

Making a gradient step $\theta \leftarrow \alpha \frac{\partial \mathrm{Loss}}{\partial \theta}$ can drastically change the network, or even destroy it.

This is not a problem of information flow, but of training stability!

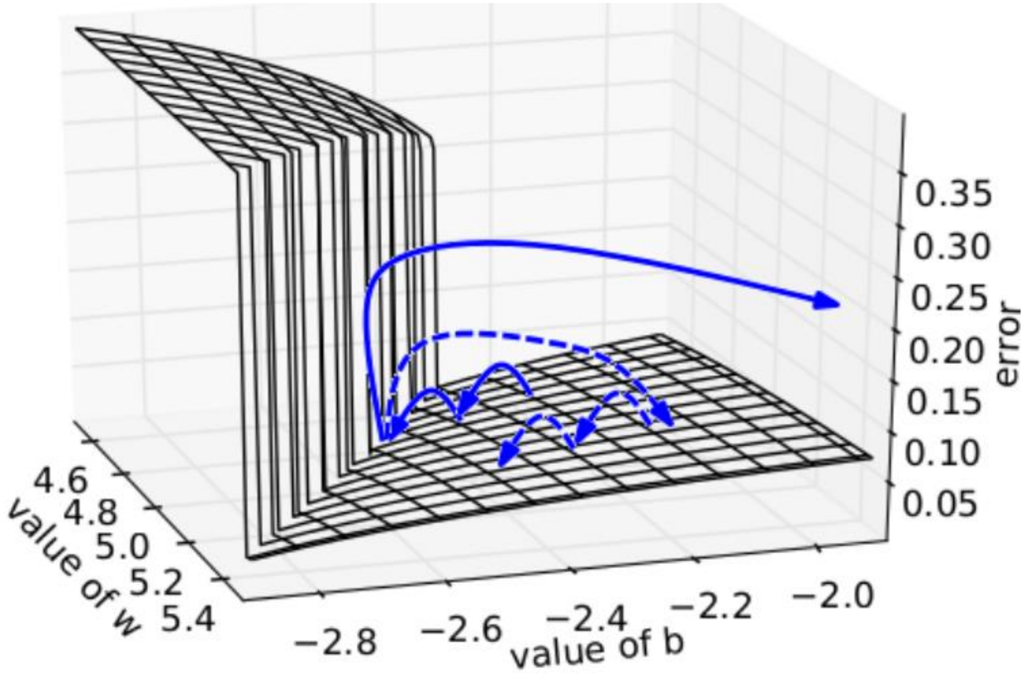
Exploding gradient intuition #1

$$h_t = \tanh(2h_{t-1} + b)$$



Exploding gradient intuition #2

$$h_0 = \sigma(0.5)$$
$$h_t = \sigma(wh_{t-1} + b)$$
$$L = (h_{50} - 0.7)^2$$



R. Pascanu et al. https://arxiv.org/abs/1211.5063

Summary: trouble with $\frac{\partial h_T}{\partial h_t}$

RNNs are difficult to train because:

$\frac{\partial h_T}{\partial h_t}$ can be 0 – vanishing gradient

$\frac{\partial h_T}{\partial h_t}$ can be ∞ – exploding gradient

With both phenomena governed by the spectral norm of the weight matrix (norm of the largest eigenvalue, magnitude of w in the scalar case).

Exploding gradient solution

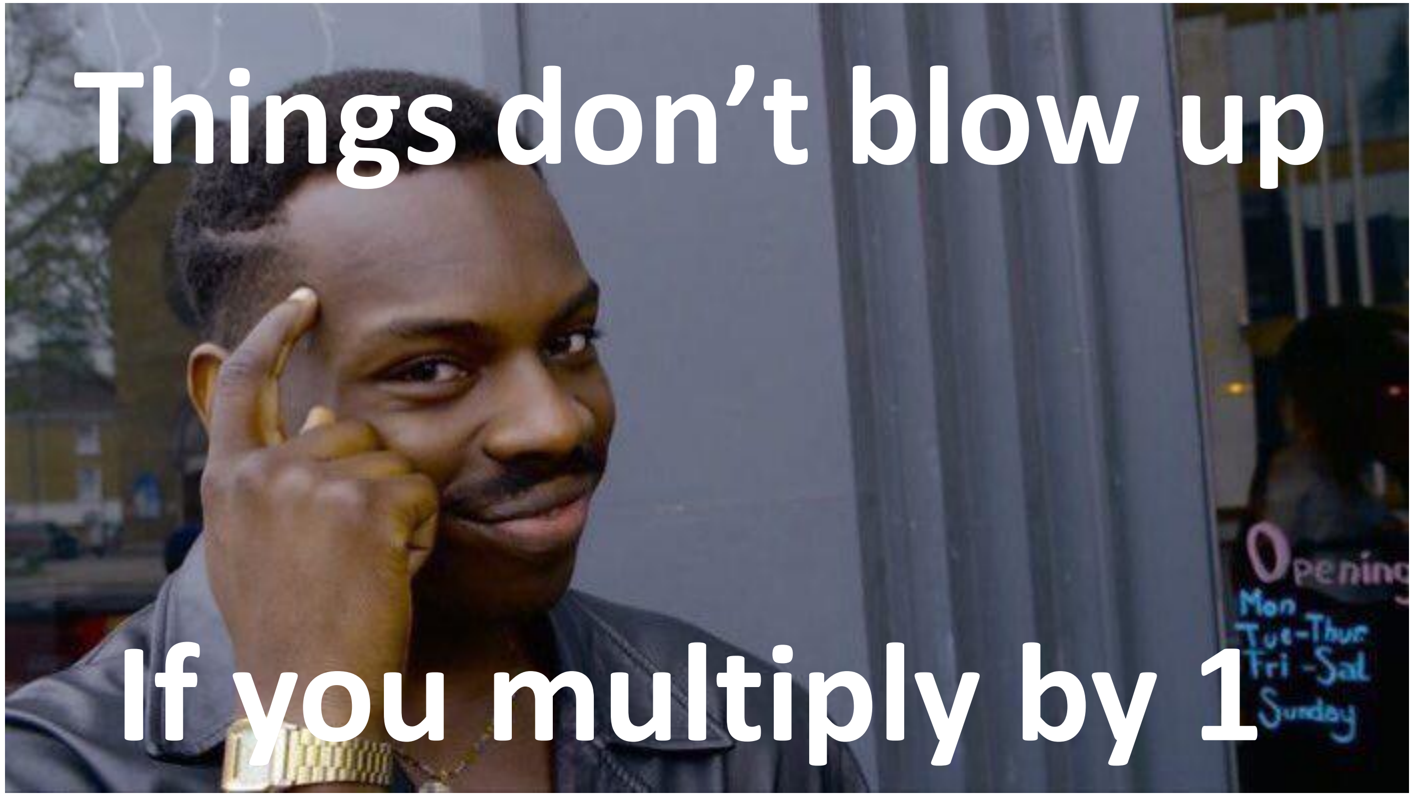
Don't do large steps.

Pick a gradient norm threshold and scale down all larger gradients.

This prevents the model from doing a large learning update and destroying itself.

VANISHING GRADIENT SOLUTION: LSTM

LSTM


Things don't blow up
If you multiply by 1
Opening
Mon
Tue-Thur
Fri -Sat
Sunday

LSTM intuitions

Recall the scalar case

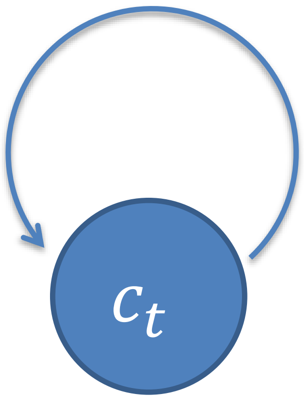
$$h_t = wh_{t-1}$$

It maximally preserves information when $w = 1$

The LSTM introduces a memory cell c_t that will keep information forever:

$$c_t = 1 \cdot c_{t-1}$$

Memory cell

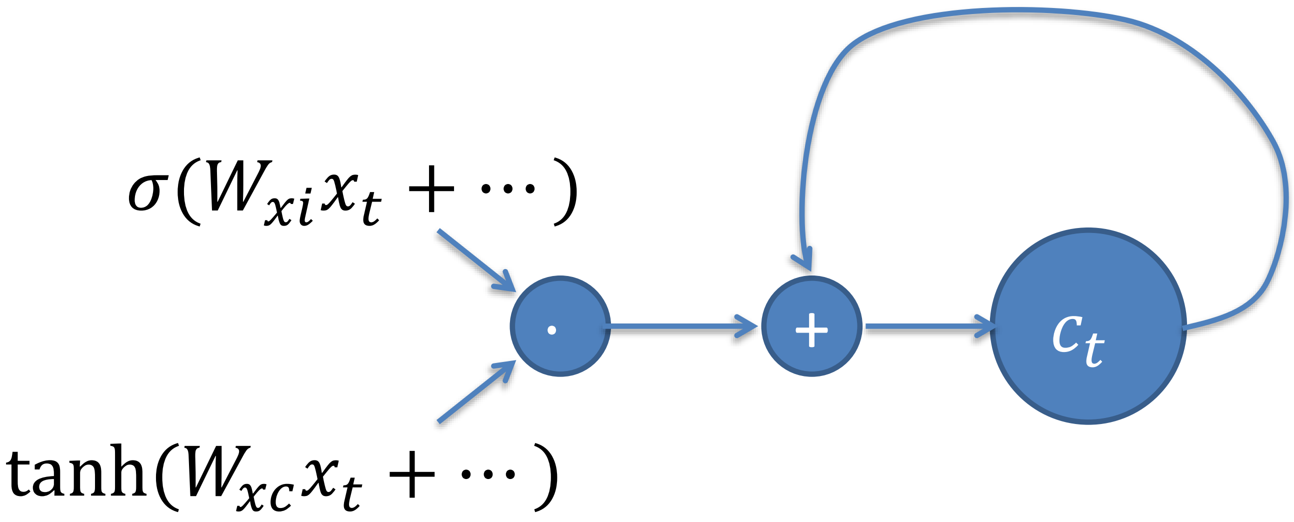


Memory cell preserves information

$$c_t = c_{t-1}$$

$$\frac{\partial c_T}{\partial c_t} = 1$$

Gates



Gates selectively load information into the memory cell:

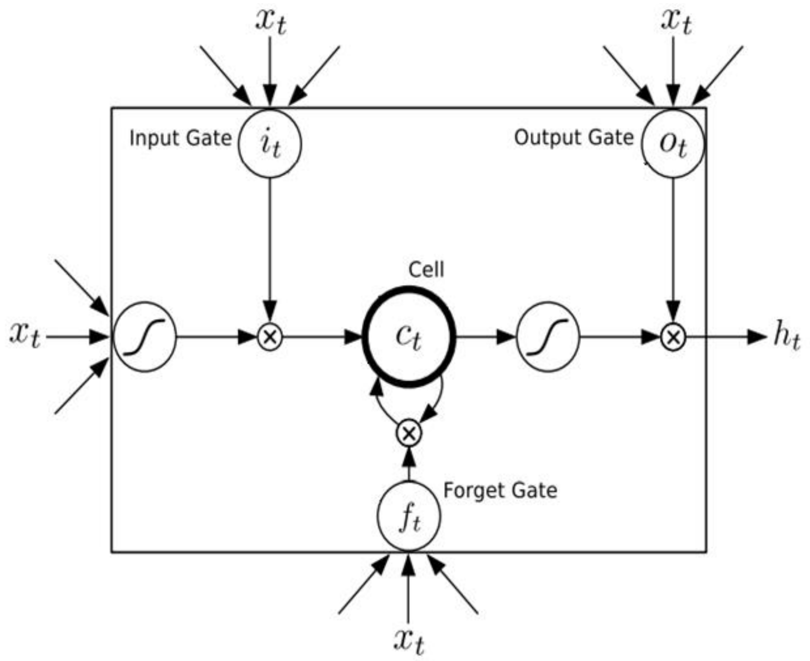
$$i_t = \sigma(W_{xi}x_t + \cdots)$$
$$c_t = c_{t-1} + i_t \cdot \tanh(W_{xc}x_t + \cdots)$$

LSTM: the details

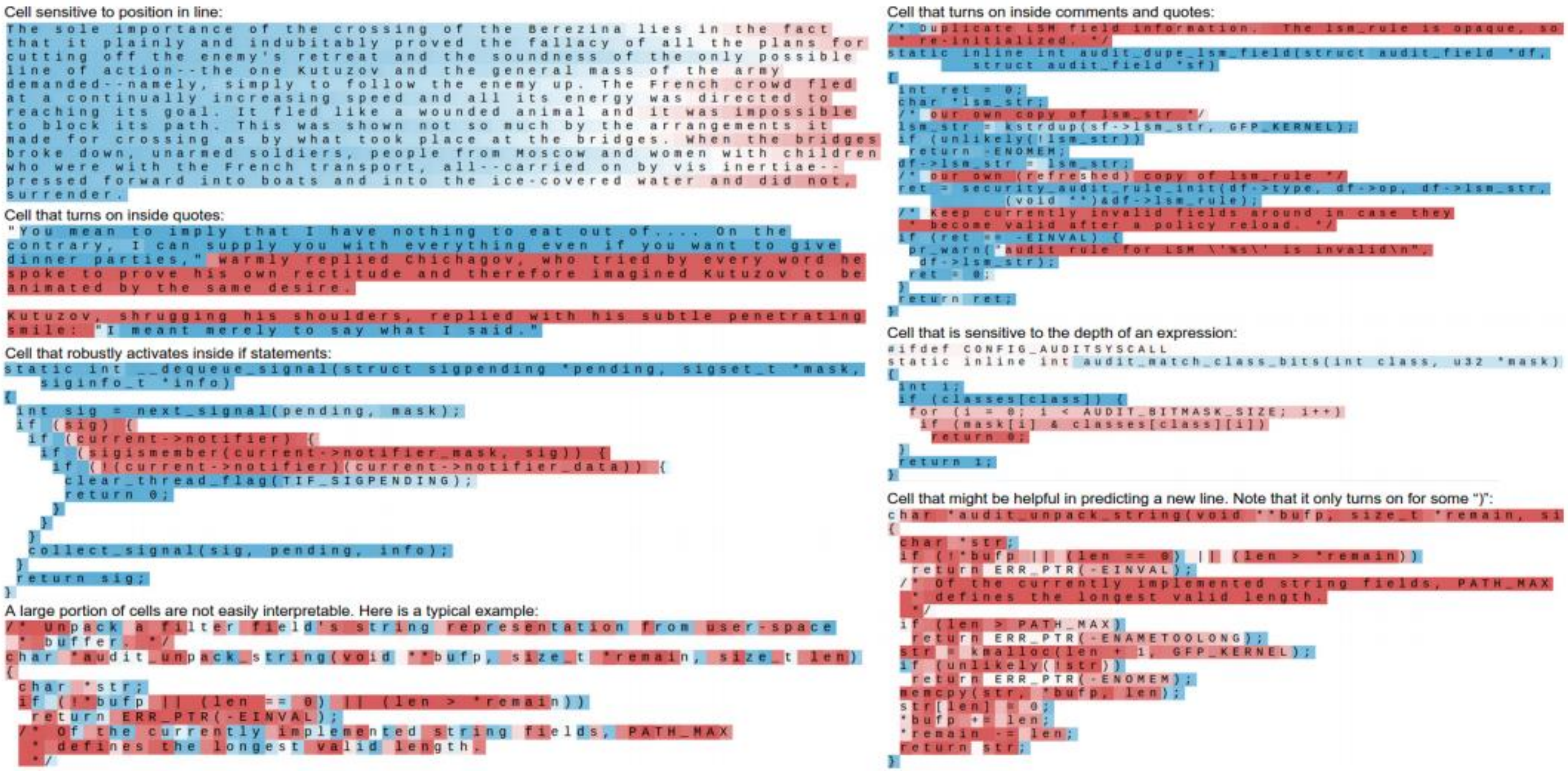
Hidden state is a pair of:
-c_t information in the cell, hidden from the rest of the network
-h_t information extracted form the cell into the network

Update equations:

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i)$$
$$f_t = \sigma\left(W_{xf}x_t + W_{hf}h_{t-1} + b_f\right)$$
$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o)$$
$$c_t = i_t \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) + f_t c_{t-1}$$
$$h_t = o_t \tanh c_t$$



LSTM in action



A. Karpathy et al., arxiv.org/pdf/1506.02078.pdf

LSTM fixes vanishing gradient

- With input gate closed, and forget gate opened, LSTM preserves information in the memory cell - gradients flow into distant time steps.

- The gates decouple the decision **if** information should be stored from **what** information should be stored.

LSTM variant: peephole connections

Peephole connections allow the gates to "peep" into the cell value:

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + \boldsymbol{W_{ci}c_{t-1}} + b_i)$$
$$f_t = \sigma\left(W_{xf}x_t + W_{hf}h_{t-1} + \boldsymbol{W_{cf}c_{t-1}} + b_f\right)$$
$$c_t = f_t c_{t-1} + i_t \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c)$$
$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + \boldsymbol{W_{co}c_t} + b_o)$$
$$h_t = o_t \tanh c_t$$

LSTM Variant: projection

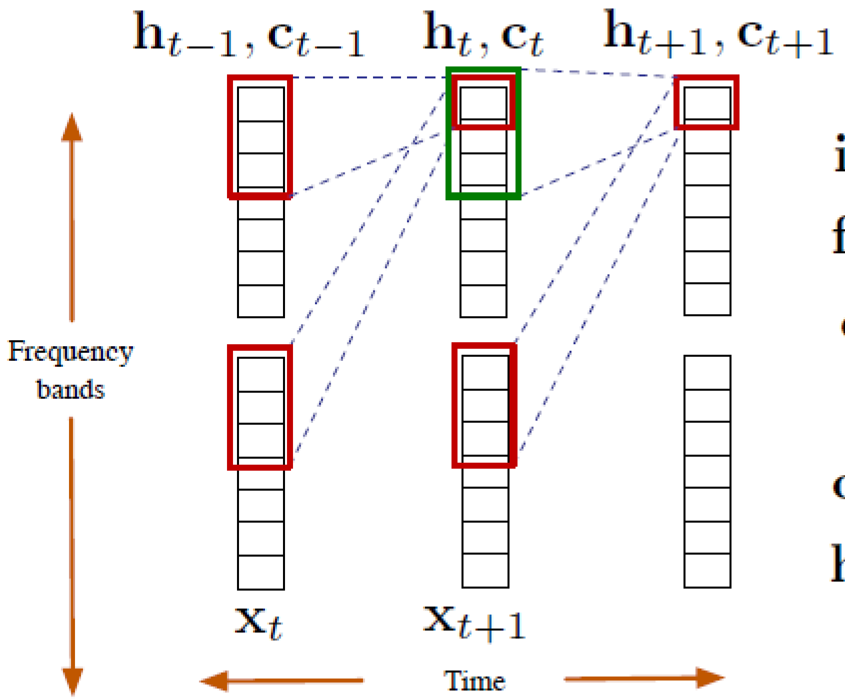
Projections reduce the number of parameters and computations: for N LSTM units we need $O(N^2)$ parameters and flops.

With projections we only need $O(2NP)$, which is beneficial when $P \ll N$:

$$
\begin{aligned}
i_t &= \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i) \\
f_t &= \sigma\left(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f\right) \\
c_t &= f_t c_{t-1} + i_t \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \\
o_t &= \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o) \\
h'_t &= o_t \tanh c_t \\
\boldsymbol{h_t} &= \boldsymbol{W_p h'_t}
\end{aligned}
$$

LSTM variant: ConvLSTM

Replace matrix multiplications with convolutions!
Both are linear operations.



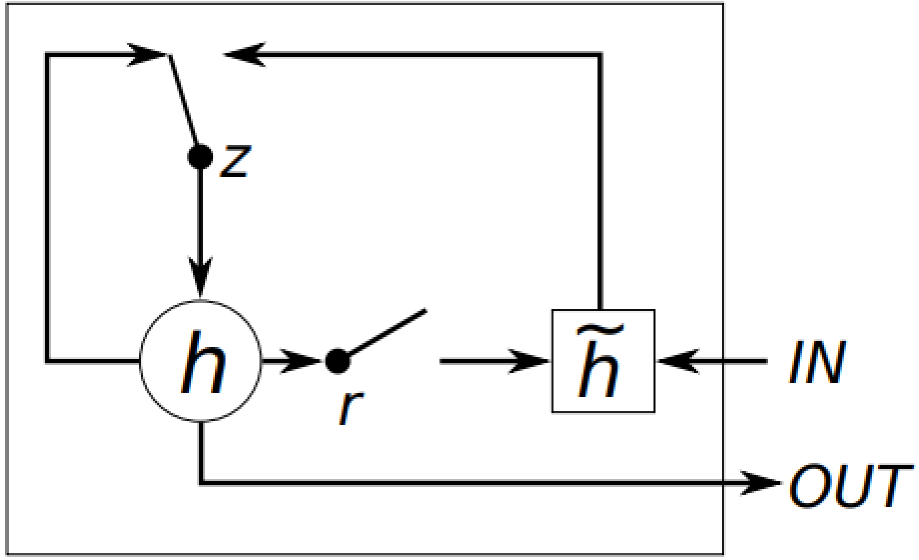
$$\mathbf{i}_t = \sigma(\mathbf{W}_{xi} * \mathbf{x}_t + \mathbf{W}_{hi} * \mathbf{h}_{t-1} + \mathbf{b}_i)$$
$$\mathbf{f}_t = \sigma(\mathbf{W}_{xf} * \mathbf{x}_t + \mathbf{W}_{hf} * \mathbf{h}_{t-1} + \mathbf{b}_f)$$
$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + $$
$$+ \mathbf{i}_t \odot \tanh(\mathbf{W}_{xc} * \mathbf{x}_t + \mathbf{W}_{hc} * \mathbf{h}_{t-1} + \mathbf{b}_c)$$
$$\mathbf{o}_t = \sigma(\mathbf{W}_{xo} * \mathbf{x}_t + \mathbf{W}_{ho} * \mathbf{h}_{t-1} + \mathbf{b}_o)$$
$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$$

Shi X., et al. arxiv.org/pdf/1506.04214.pdf   Y. Zhang et al, https://arxiv.org/abs/1610.03022

The GRU cell – an LSTM alternative

The GRU is similar to the LSTM, but:

- uses only two gates: reset (r) and update (z)
- Doesn't have a separate c_t from h_t.



$$r_t^j = \sigma\left(W_r \mathbf{x}_t + U_r \mathbf{h}_{t-1}\right)^j$$

$$\tilde{h}_t^j = \tanh\left(W \mathbf{x}_t + U\left(\mathbf{r}_t \odot \mathbf{h}_{t-1}\right)\right)^j$$

$$z_t^j = \sigma\left(W_z \mathbf{x}_t + U_z \mathbf{h}_{t-1}\right)^j$$

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

arxiv.org/abs/1412.3555

Echo state networks

Echo state networks

Dynamics:

$$h_t = (1-a)h_{t-1} + a\tanh(W_h h_{t-1} + W_x x_t)$$
$$y_t = W_y h_t$$

Algorithm:

Repeat until it works:

- Initialize W_x randomly
- Initialize W_h randomly, scale to have largest eigenvalue ≈ 1
- Fit W_y, can use the matrix pseudo-inverse formula

Each "training" run is very fast, can sample thousands of configurations.

Jaeger, H., 2002. Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the echo state network approach

Unitary Evolution RNN

What if

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

But W is unitary (all complex eigenvalues have norm 1)?

The we have no exponential explosion/decay!

How to ensure W is unitary?

Let: $$\mathbf{W} = \mathbf{D}_3 \mathbf{R}_2 \mathcal{F}^{-1} \mathbf{D}_2 \mathbf{\Pi} \mathbf{R}_1 \mathcal{F} \mathbf{D}_1$$

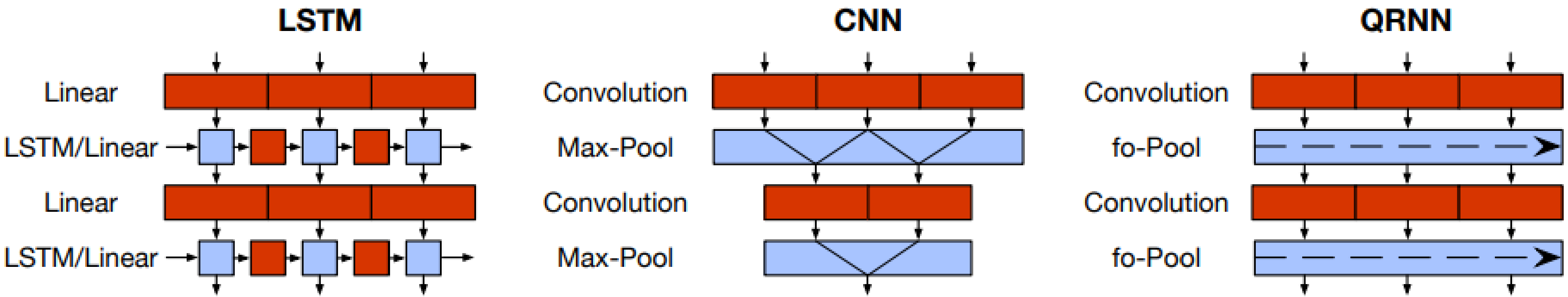
- $\mathbf{D}$, a diagonal matrix with $\mathbf{D}_{j,j} = e^{iw_j}$, with parameters $w_j \in \mathbb{R}$,
- $\mathbf{R} = \mathbf{I} - 2\frac{vv^*}{\|v\|^2}$, a reflection matrix in the complex vector $v \in \mathbb{C}^n$,
- $\mathbf{\Pi}$, a fixed random index permutation matrix, and
- $\mathcal{F}$ and $\mathcal{F}^{-1}$, the Fourier and inverse Fourier transforms.

arxiv.org/pdf/1511.06464.pdf

QuasiRNNs

RNNs are inherently sequential.

How much can we parallelize?



$$\mathbf{Z} = \tanh(\mathbf{W}_z * \mathbf{X})$$
$$\mathbf{F} = \sigma(\mathbf{W}_f * \mathbf{X})$$
$$\mathbf{O} = \sigma(\mathbf{W}_o * \mathbf{X}),$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + (1 - \mathbf{f}_t) \odot \mathbf{z}_t$$
$$\mathbf{h}_t = \mathbf{o}_t \odot \mathbf{c}_t.$$

arxiv.org/pdf/1611.01576.pdf

No-nonlinearity RNN

Model uses a different weight matrix for each input symbol:

$$\mathbf{h}_t = \mathbf{W}_{\mathbf{x_t}}\ \mathbf{h}_{t-1} + \mathbf{b}_{\mathbf{x_t}}$$
$$p\left(\mathbf{x}_{t+1}\right) = \text{softmax}\left(\mathbf{l}_t\right)$$
$$\mathbf{l}_t = \mathbf{W}_{ro}\ \mathbf{h}_t + \mathbf{b}_{ro}$$

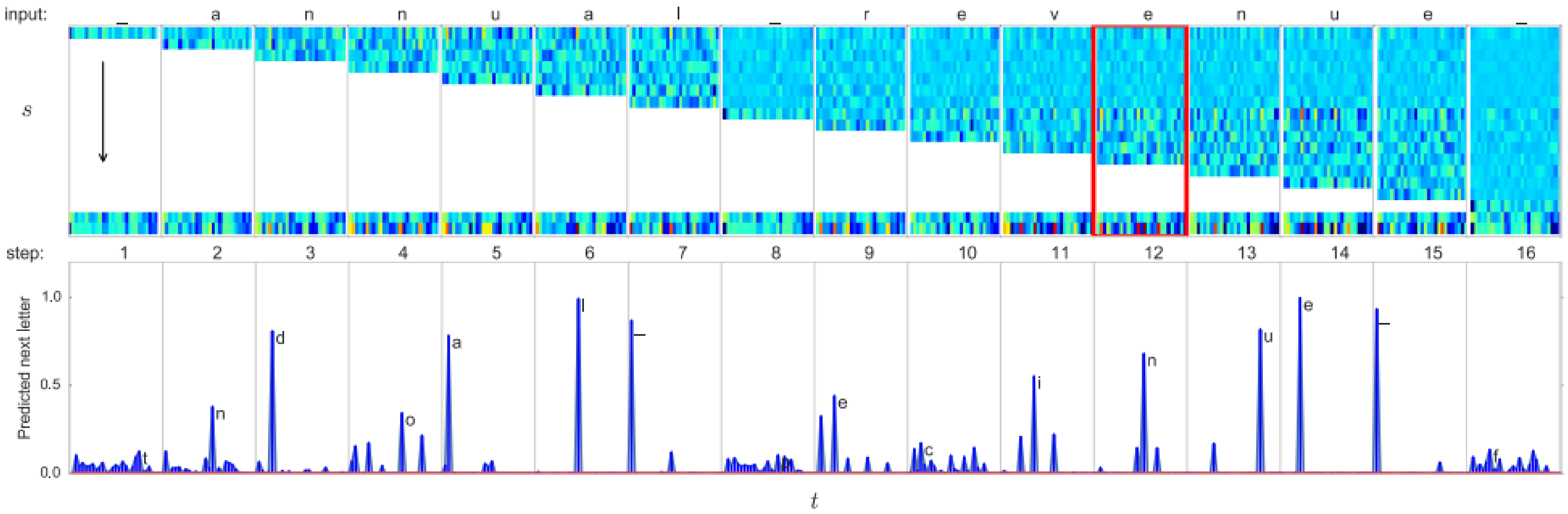
We can decompose an output at time t into contributions from previous time states:

$$\mathbf{l}_t = \mathbf{b}_{ro} + \sum_{s=0}^{t} \boldsymbol{\kappa}_s^t$$
$$\boldsymbol{\kappa}_s^t = \mathbf{W}_{ro}\left(\prod_{s'=s+1}^{t} \mathbf{W}_{\mathbf{x}_{s'}}\right)\mathbf{b}_{\mathbf{x}_s}$$

arxiv.org/pdf/1611.09434.pdf

No-nonlinearity RNN

Visualization of contributions from past timesteps:

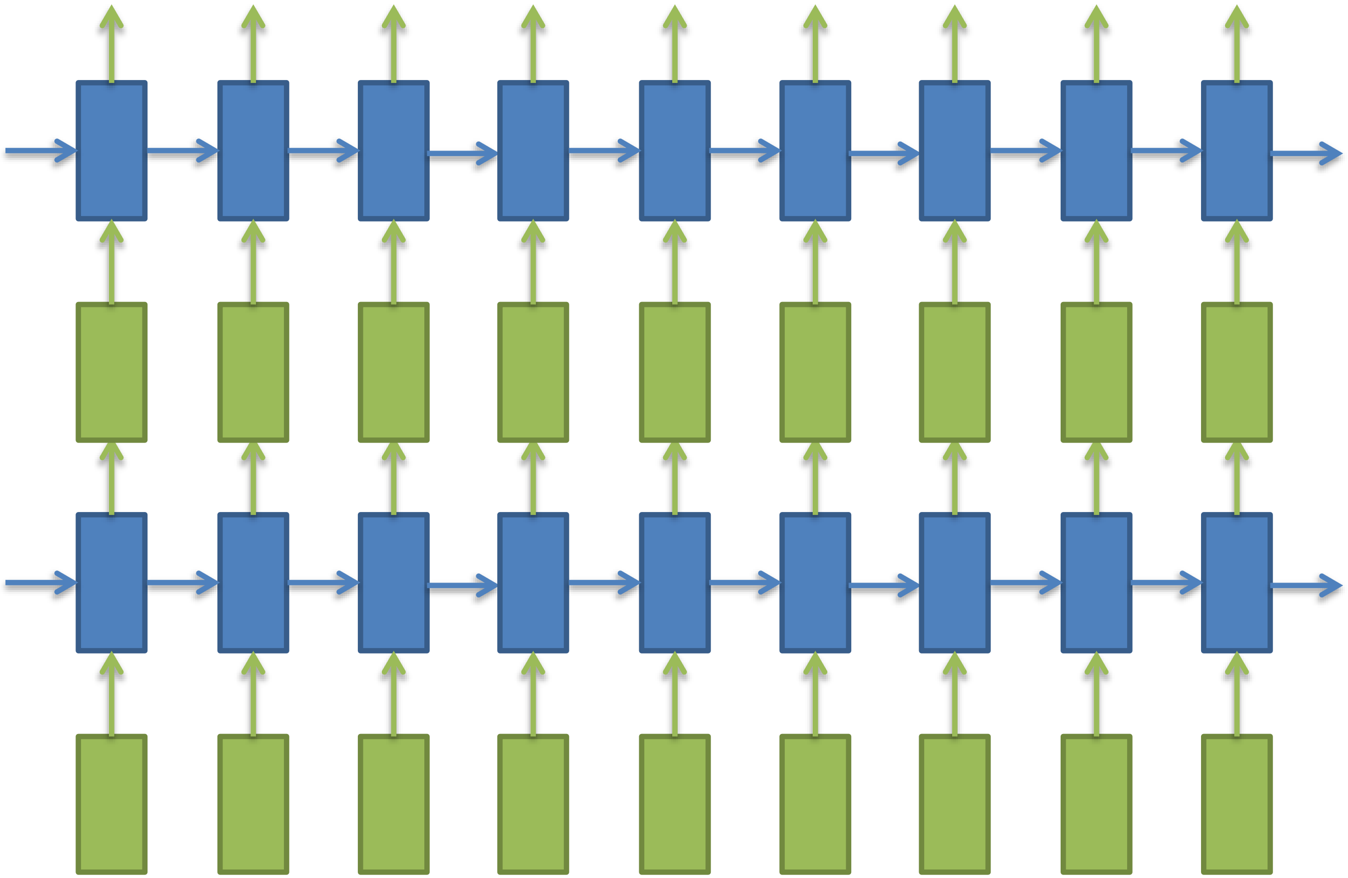


arxiv.org/pdf/1611.09434.pdf

RNNS – PRACTICAL ASPECTS

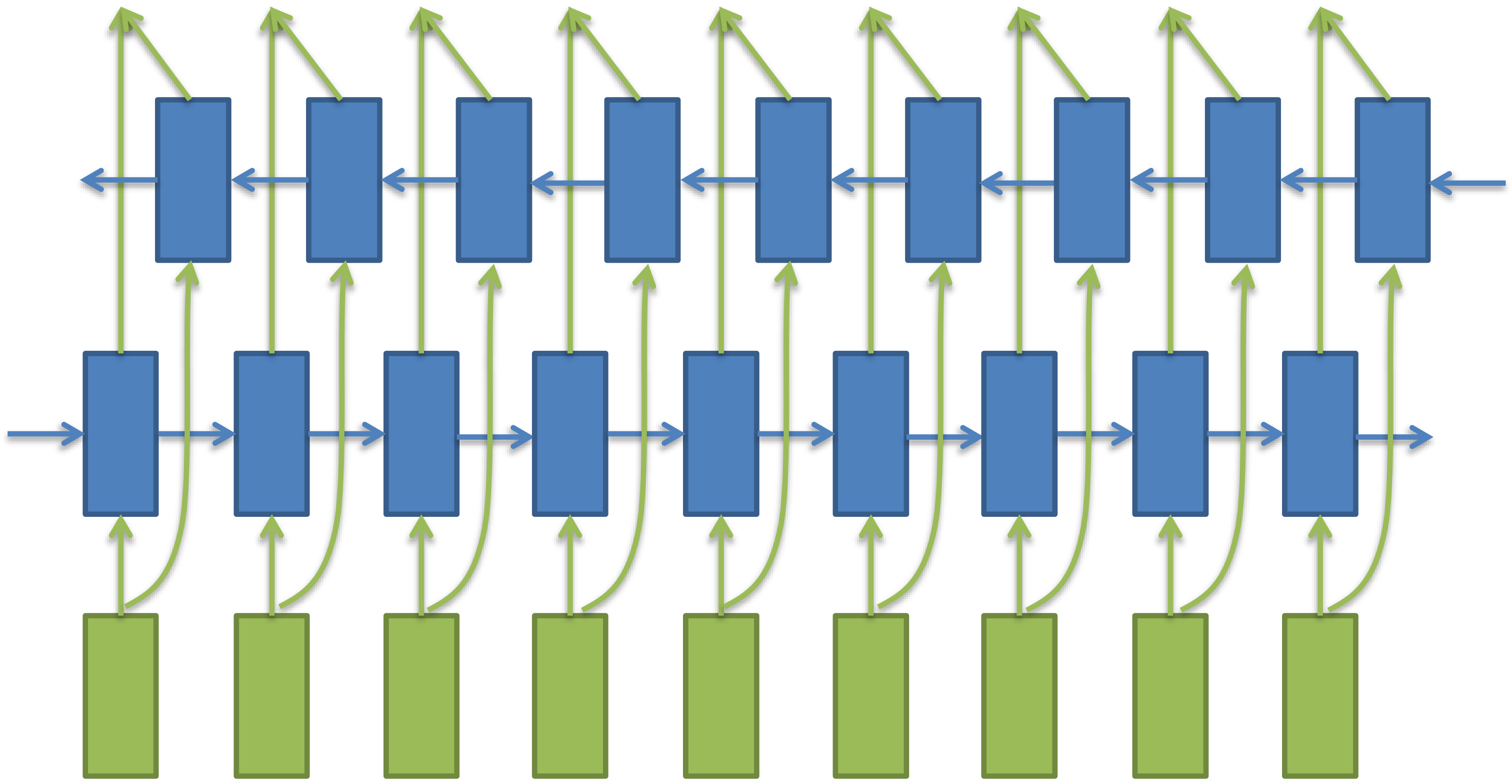
Stacking RNNs

You can create a deep RNN by stacking



Bidirectional RNNs

Concatenate two RNNs: one going forward in time, one back in time.

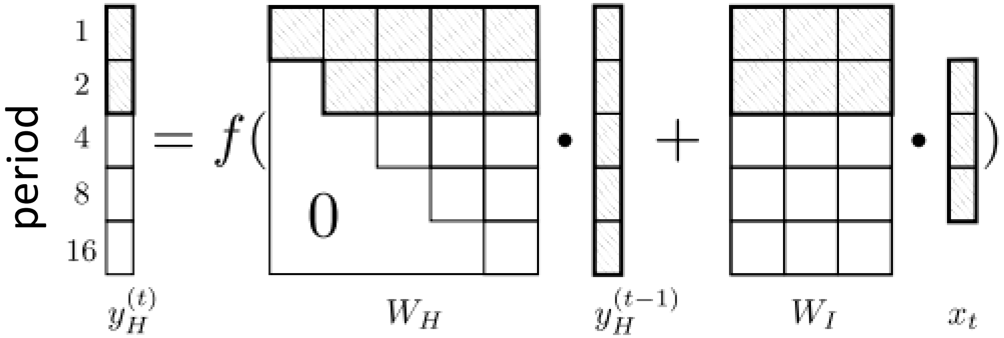


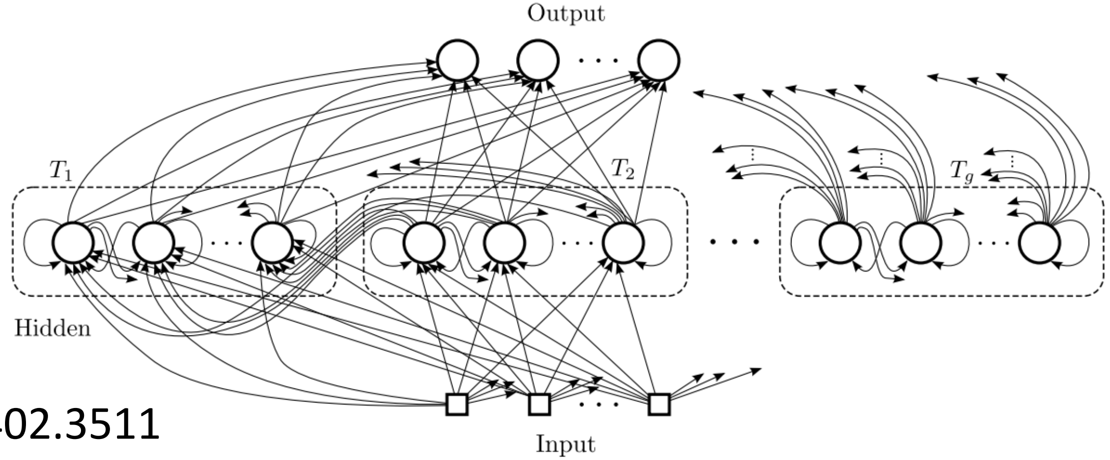
Schuster, M. and Paliwal, K.K., 1997. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, *45*(11), pp.2673-2681.

Clockwork RNN

Units operate at different frequencies.

Slower neurons feed data to faster ones, but not vice versa.

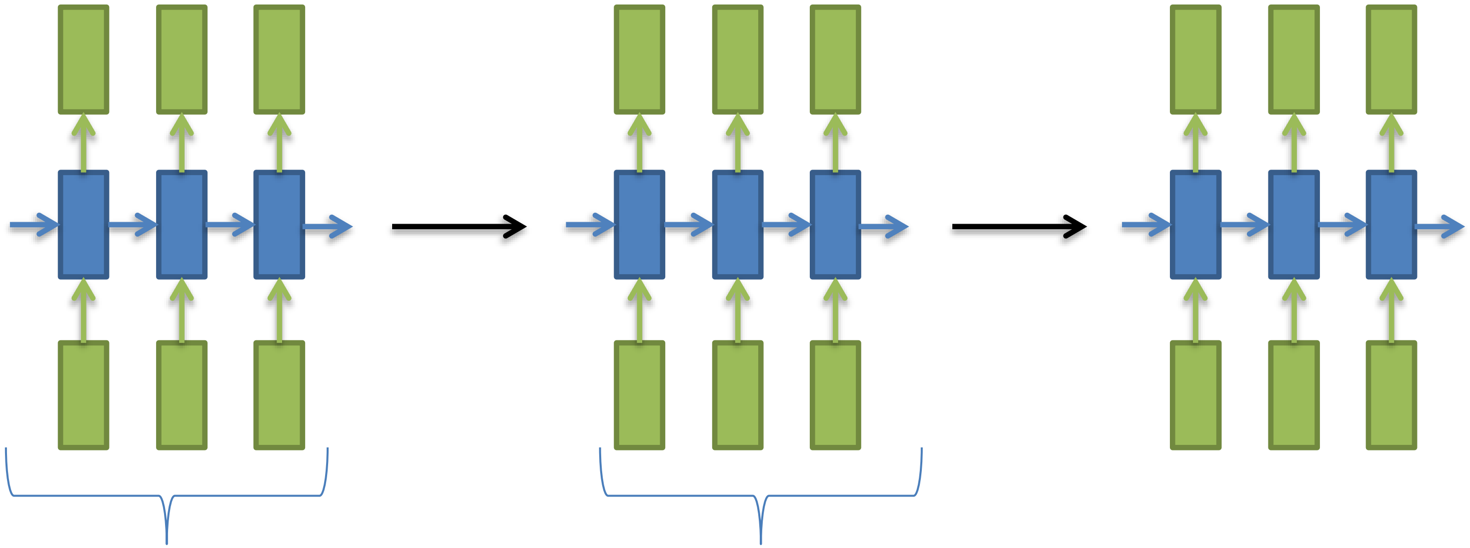




arxiv.org/abs/1402.3511

Truncated BPTT

Sometimes, it is not necessary to unroll the RNN until the very beginning.



1. Unroll for K steps, do fprob, bprop, update weights

2. Copy the hidden state

3. Start from saved state, unroll do fprob & bprop & update

4. Copy the hidden state, repeat

Dynamic evaluation

Train on test set during eval:

- For language modeling, we get implicit feedback during evaluation – at each step we make a prediction and get the correct answer
- We can use this for training (nb. we become dependent on test set ordering)
- Used by A. Graves for Hutter Prize Wikipedia experiments.

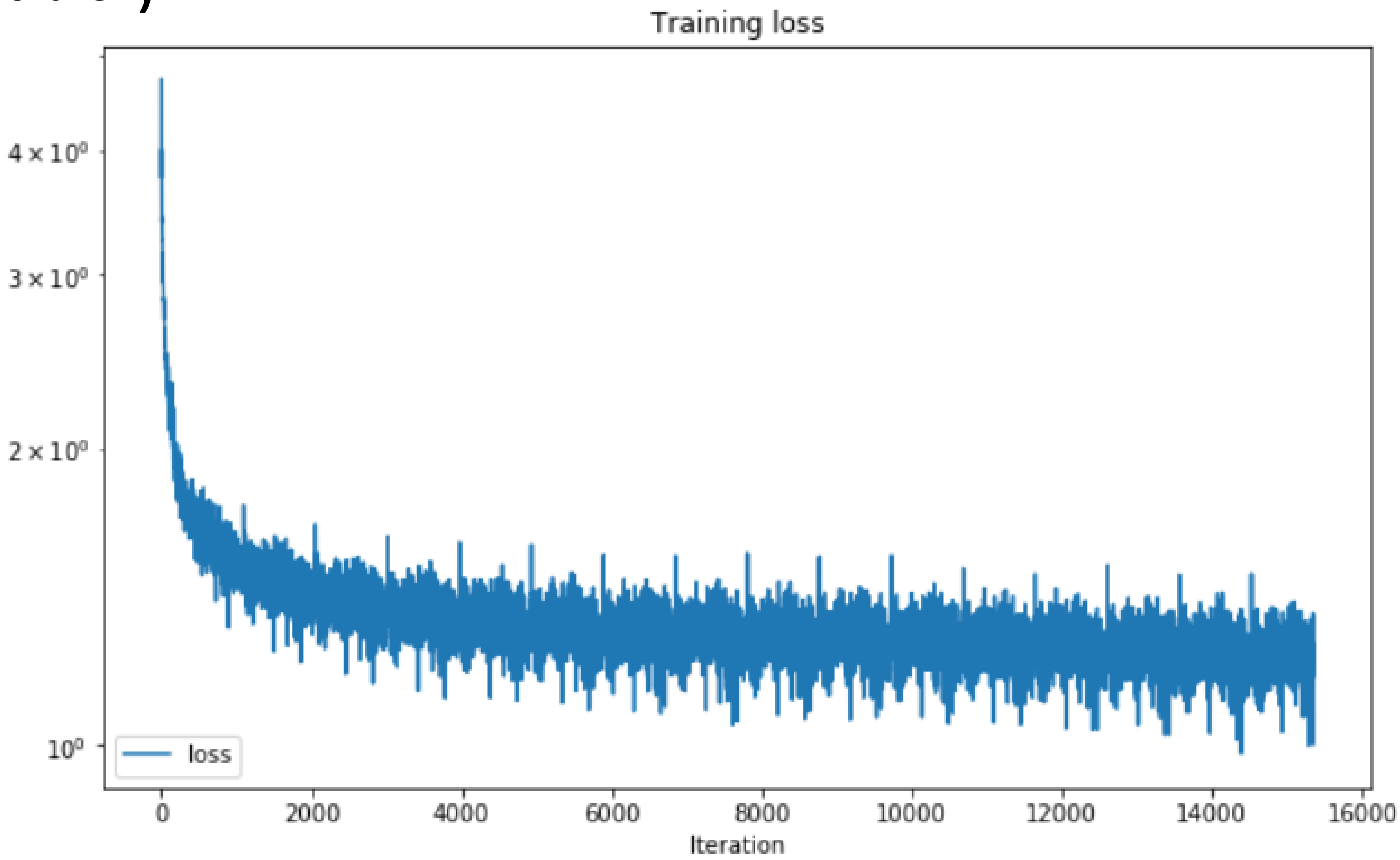
arxiv.org/pdf/1308.0850.pdf

Practical aspects of RNN training

1. Use as much supervision as possible
   when intermediate targets are available, use them!
2. Use curriculum learning
   start with shortest examples, then do longer ones.
3. Monitor your gradients.
4. Use gradient clipping
   aim to clip a few % of all steps.
5. Play with echo-state-like or orthogonal initialization
   It works well for LSTMs & GRUs too

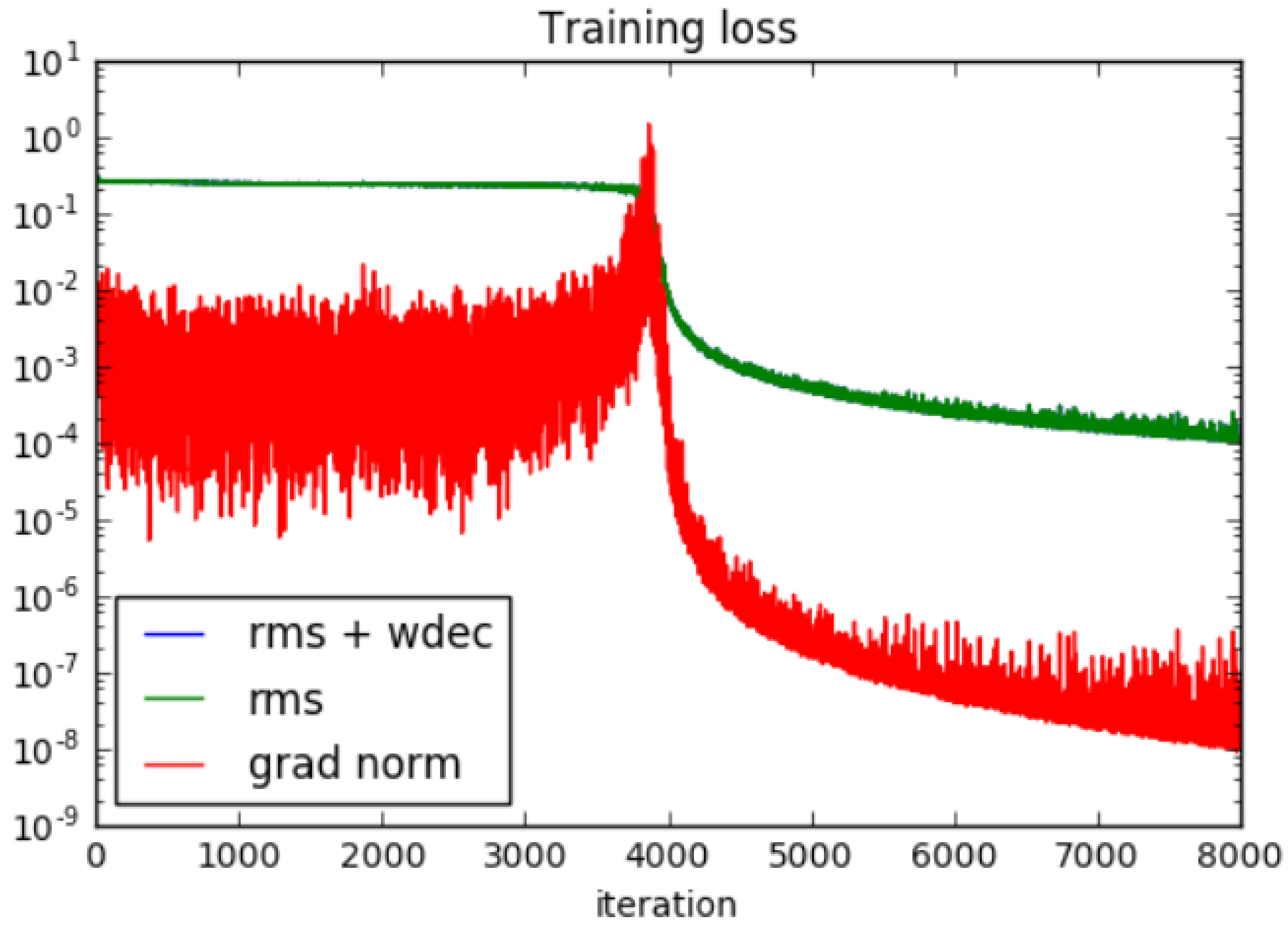
What to expect during RNN training?

Task with frequent supervision (e.g. language model)



What to expect during RNN training?

Task with distant supervision (parity)



Batch norm for RNNs?

Recall batch normalization:

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

S. Ioffe, C. Szegedy, arxiv.org/pdf/1502.03167.pdf

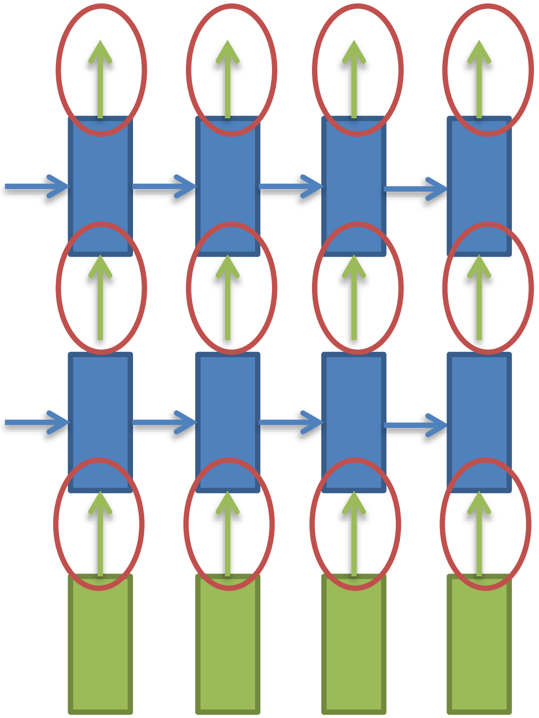
Easy BN for RNNs

Apply to the "vertical" arrows only:

$$\begin{pmatrix} f_t \\ i_t \\ o_t \\ g_t \end{pmatrix} = W_h h_{t-1} + \mathbf{BN}(W_x x_t; \gamma) + b$$

$$c_t = \sigma(f_t) c_{t-1} + \sigma(i_t) \odot \tanh(g_t)$$

$$h_t = \sigma(o_t) \odot \tanh(c_t)$$



Full BN for RNNs

Can also apply BN to recurrent connections:

$$\begin{pmatrix} f_t \\ i_t \\ o_t \\ g_t \end{pmatrix} = \mathbf{BN}(W_h h_{t-1}, \gamma_h) + \mathbf{BN}(W_x x_t; \gamma_x) + b$$

$$c_t = \sigma(f_t) c_{t-1} + \sigma(i_t) \odot \tanh(g_t)$$

$$h_t = \sigma(o_t) \odot \tanh(\mathbf{BN}(c_t; \gamma_c, \beta_c))$$

But:

1. Need to use separate batch norms for hiddens, inputs and cell values.
2. Need to use per-timestep statistics (γ are the same, but mean and variance are per time step)

LayerNorm (arxiv.org/abs/1607.06450) works too and is somewhat easier to use, as it stores no stats.

T. Cooijmans et al., arxiv.org/pdf/1603.09025.pdf

How to regularize an RNN

L2 penalty/Weight decay is bad:

- Scaling a weight matrix scales its eigenvalues.
- Small weight matrices with small eigenvalues will lead to fast information decay!

Weight noise

We want low-precision weights, not small weights.

Add noise to weights!

For all weights w:

```
        noise_w= random.normal()*0.02      (or 0.05)
        w += noise_w
```

Do fprop, do backprop

For all weights w:

```
        w -= noise_w
```

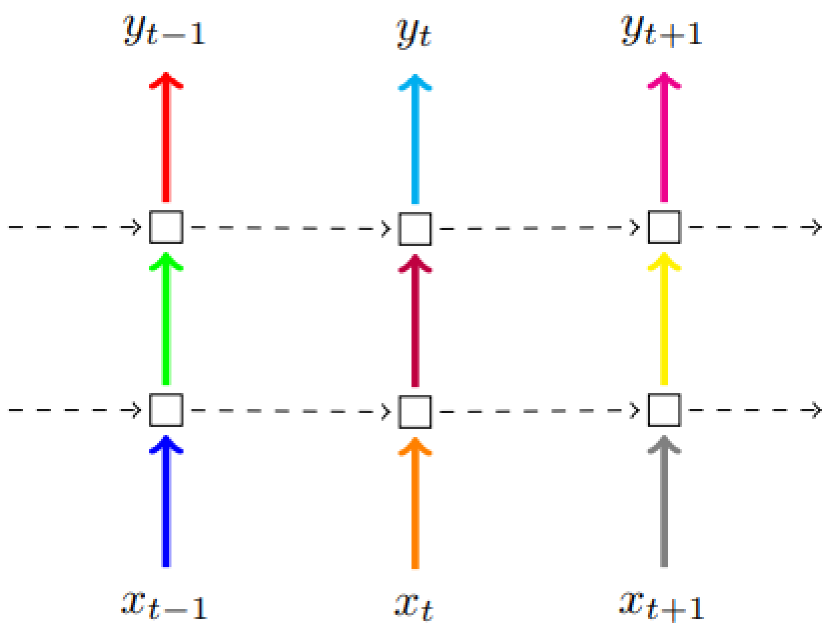
Apply gradients.

More principled: A. Graves "Practical variational inference for neural networks"

Recurrent dropout

Simplest approach (Zaremba)

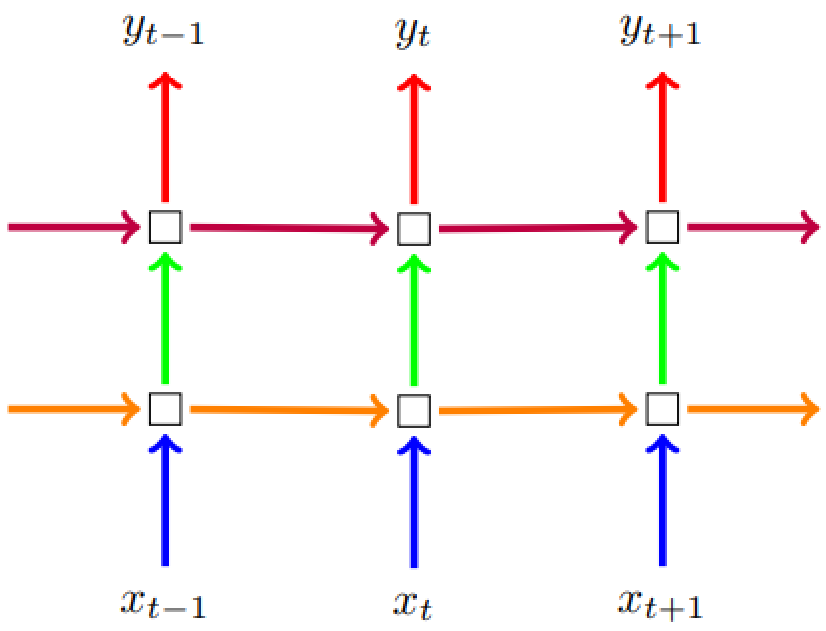
Dropout only to the “vertical” arrows



arxiv.org/pdf/1409.2329.pdf arxiv.org/pdf/1512.05287.pdf

Better dropout

Apply the same dropout mask across all time steps (Gal's dropout)



arxiv.org/pdf/1512.05287.pdf

Zoneout

Randomly keep past values for h_t or c_t:

$$c_t = d_t^c \odot c_{t-1} + (1 - d_t^c) \odot \left(f_t \odot c_{t-1} + i_t \odot g_t\right)$$

$$h_t = d_t^h \odot h_{t-1} + (1 - d_t^h) \odot \left(o_t \odot \tanh\left(f_t \odot c_{t-1} + i_t \odot g_t\right)\right)$$

Seems to work best with probability of keeping cs larger than that of keeping hs.

arxiv.org/pdf/1606.01305.pdf

Teacher forcing

At training time, the net sees only correct data.

Input: Mary had a little lamb

p(Mary)

p(had|Mary)

p(a|Mary had)

p(little|Mary had a)…

At generation time:

p(?) -> sampled Mary

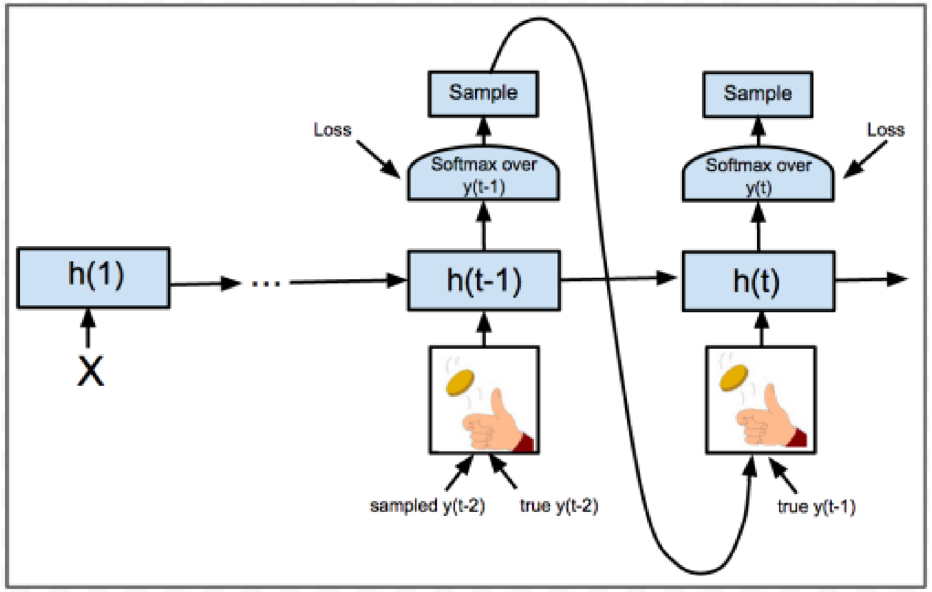
p(?|Mary) -> sampled cucumber

p(?|Mary cucumber) -> panic!!!!

Scheduled sampling

Pragmatic solution to teacher forcing:

Allow mistakes during training, too.

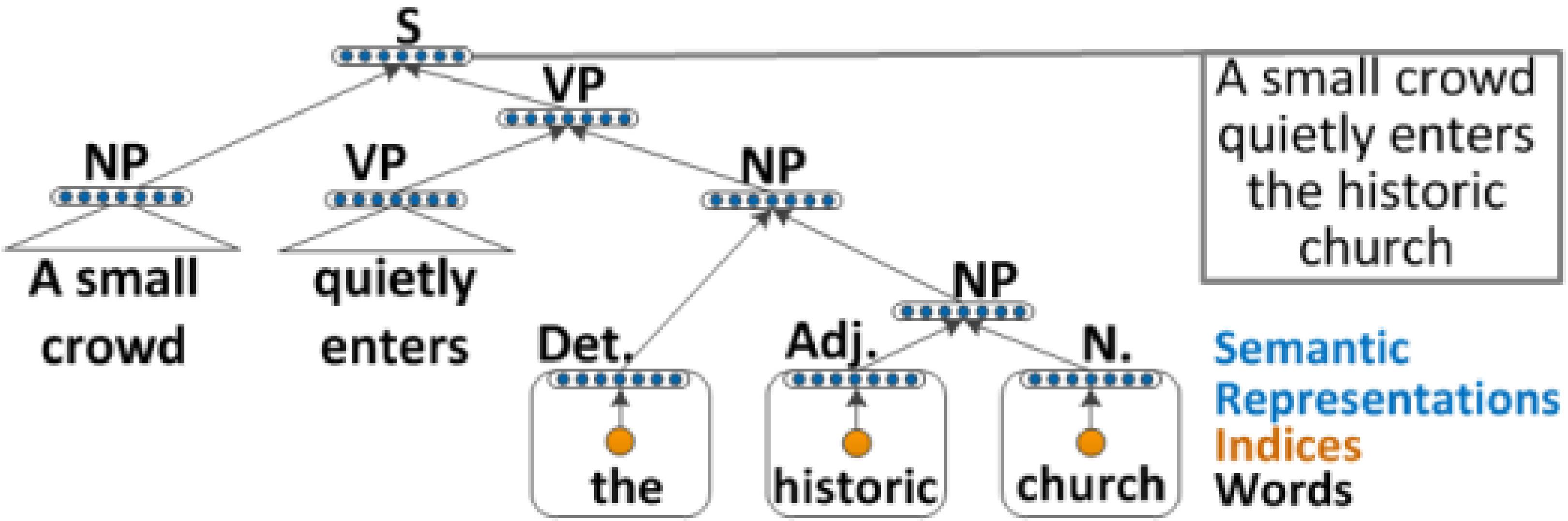


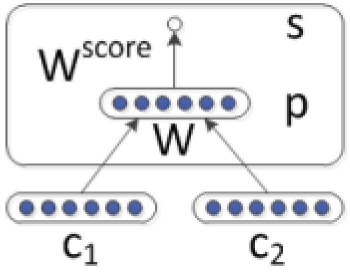
S. Bengio et al. “Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks“ NIPS 2015

BEYOND SEQUENCES

Recursive Neural Networks

Neural Network operating on binary trees



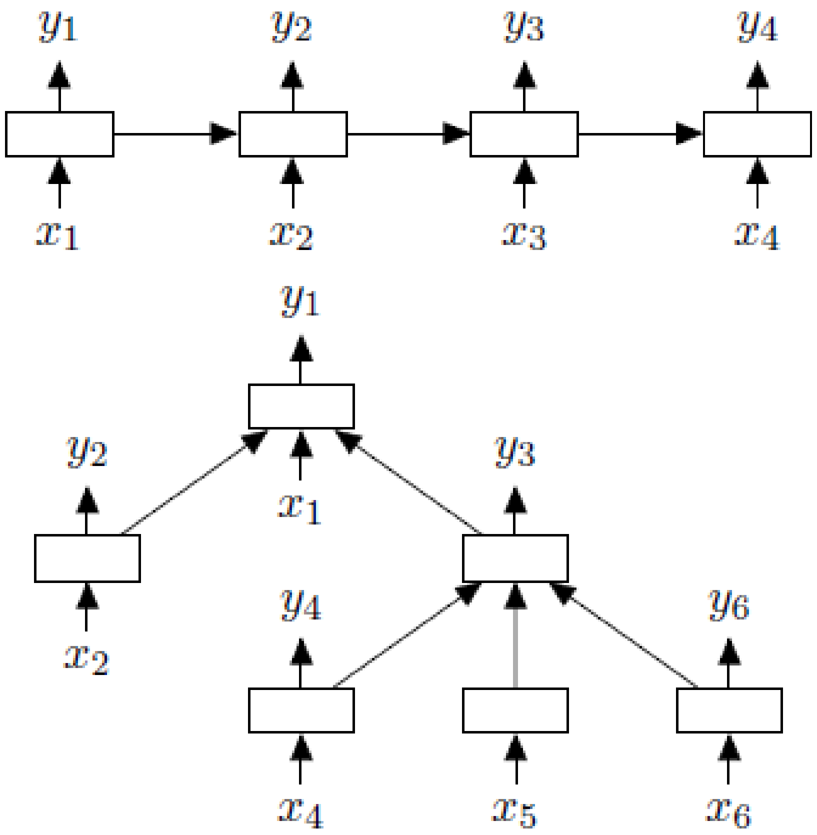


$$s = W^{score} p \quad (9)$$
$$p = f(W[c_1; c_2] + b)$$

R. Socher et al. “Parsing natural scenes and natural language with recursive neural networks”, ICML 11

Tree LSTM

Extend LSTM form sequences to trees: need to aggregate all incoming nodes $C(j)$.



$$\tilde{h}_j = \sum_{k \in C(j)} h_k,$$

$$i_j = \sigma\left(W^{(i)} x_j + U^{(i)} \tilde{h}_j + b^{(i)}\right),$$

$$f_{jk} = \sigma\left(W^{(f)} x_j + U^{(f)} h_k + b^{(f)}\right),$$

$$o_j = \sigma\left(W^{(o)} x_j + U^{(o)} \tilde{h}_j + b^{(o)}\right),$$

$$u_j = \tanh\left(W^{(u)} x_j + U^{(u)} \tilde{h}_j + b^{(u)}\right),$$

$$c_j = i_j \odot u_j + \sum_{k \in C(j)} f_{jk} \odot c_k,$$

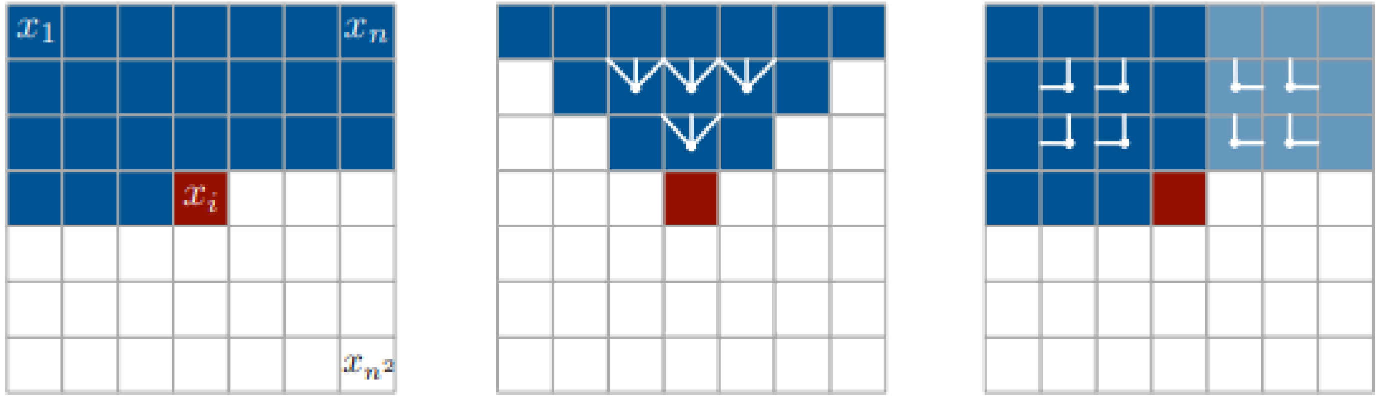
$$h_j = o_j \odot \tanh(c_j),$$

KS Tai et al. https://www.aclweb.org/anthology/P15-1150

Pixel RNN

Generate images pixel by pixel.

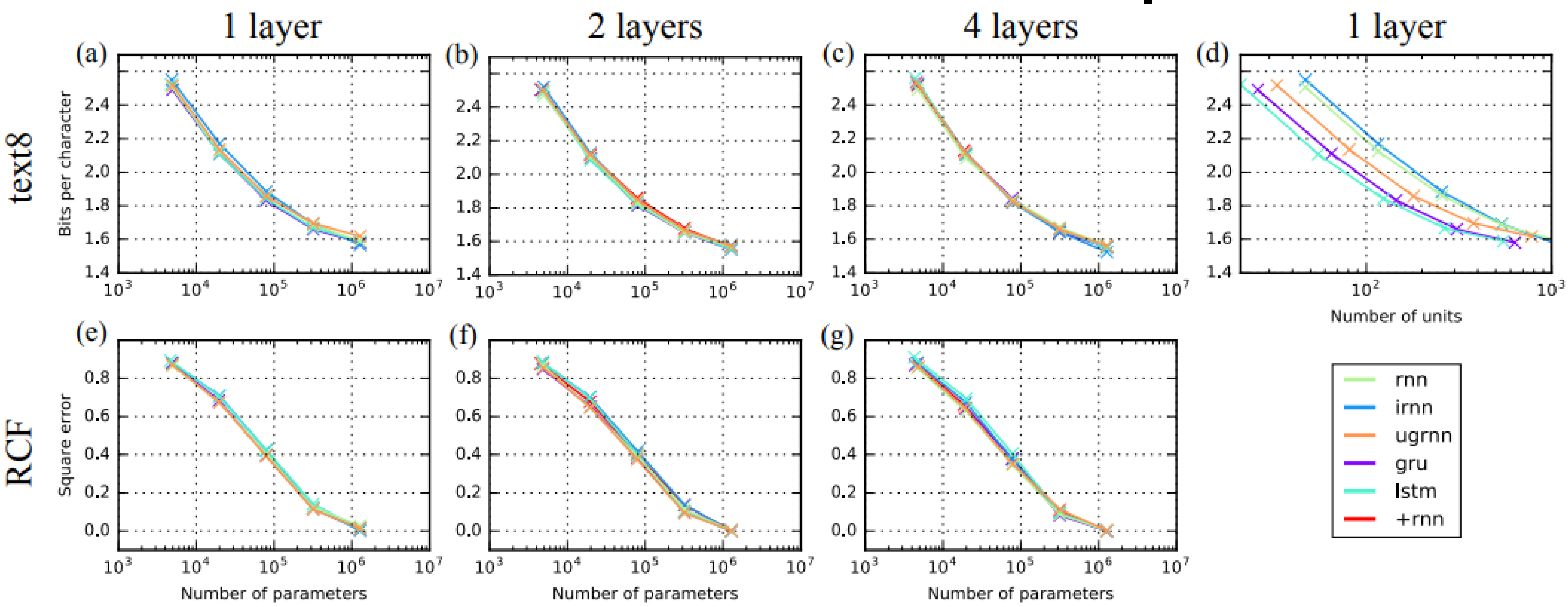
Generate pixels row-wise – condition only on pixels higher/to the left



A. Van den Oord et al., arxiv.org/pdf/1601.06759v2.pdf

DISCUSSIONS AND CONTROVERSIES

Are all RNNs created equal?



All RNN architectures reach same performance*

All models store about 5 bits/parameter.

* When you can afford the search for good hyper-parameters, and have enough data to not need regularization.

arxiv.org/pdf/1611.09913.pdf

Are all RNNs created equal?

On practical tasks the gated RNNs are typically better.

Cell	newstest2013	Params
LSTM	**22.22** $\pm$ 0.08 (22.33)	68.95M
GRU	21.78 $\pm$ 0.05 (21.83)	66.32M
Vanilla-Dec	15.38 $\pm$ 0.28 (15.73)	63.18M

Table 2: BLEU scores on newstest2013, varying the type of encoder and decoder cell.

Maybe this is because tuning vanilla RNNs is expensive.

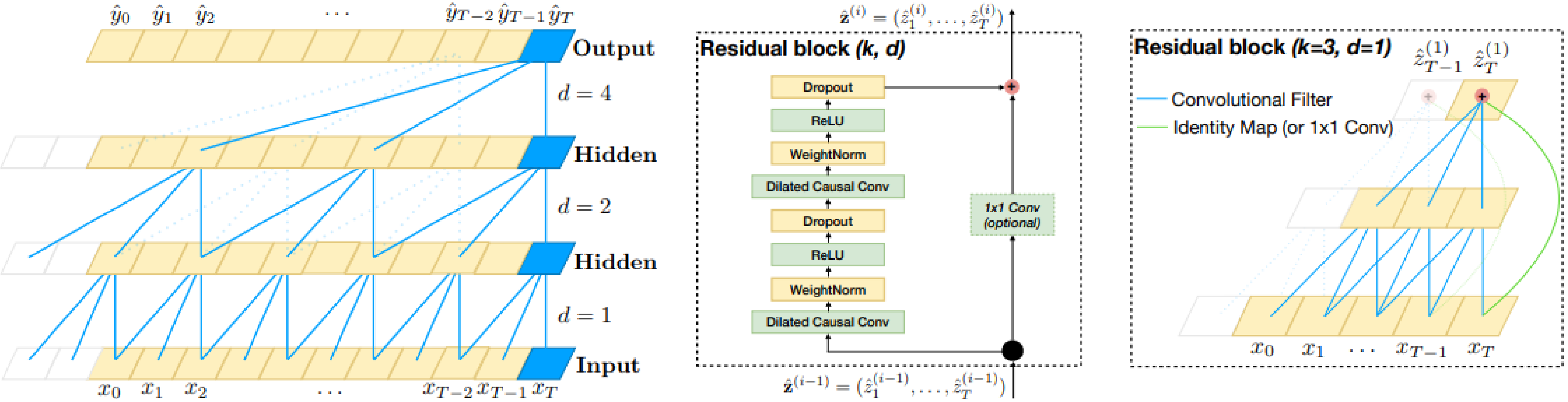
Sidenote: Baidu used vanilla RNNs in deep speech to make inference cheap.

arxiv.org/abs/1703.03906

TCN: do we need RNNs at all?

Autoregressive models with truncated history give good results, e.g. the Wavenet.

The TCN architecture is similar:



arxiv.org/pdf/1803.01271.pdf

TCN Results

Sequence Modeling Task	Model Size (≈)	Models			
		LSTM	GRU	RNN	**TCN**
Seq. MNIST (accuracy^h)	70K	87.2	96.2	21.5	**99.0**
Permuted MNIST (accuracy)	70K	85.7	87.3	25.3	**97.2**
Adding problem T=600 (loss$^\ell$)	70K	0.164	**5.3e-5**	0.177	**5.8e-5**
Copy memory T=1000 (loss)	16K	0.0204	0.0197	0.0202	**3.5e-5**
Music JSB Chorales (loss)	300K	8.45	8.43	8.91	**8.10**
Music Nottingham (loss)	1M	3.29	3.46	4.05	**3.07**
Word-level PTB (perplexity$^\ell$)	13M	**78.93**	92.48	114.50	88.68
Word-level Wiki-103 (perplexity)	-	48.4	-	-	**45.19**
Word-level LAMBADA (perplexity)	-	4186	-	14725	**1279**
Char-level PTB (bpc$^\ell$)	3M	1.36	1.37	1.48	**1.31**
Char-level text8 (bpc)	5M	1.50	1.53	1.69	**1.45**

arxiv.org/pdf/1803.01271.pdf

TCN results, SOTA?

TCN vs. SoTA Results					
Task	**TCN Result**	**Size**	**SoTA**	**Size**	**Model**
Seq. MNIST (acc.)	99.0	21K	99.0	21K	Dilated GRU (Chang et al., 2017)
P-MNIST (acc.)	97.2	42K	95.9	42K	Zoneout (Krueger et al., 2017)
Adding Prob. 600 (loss)	5.8e-5	70K	5.3e-5	70K	Regularized GRU
Copy Memory 1000 (loss)	3.5e-5	70K	0.011	70K	EURNN (Jing et al., 2017)
JSB Chorales (loss)	8.10	300K	3.47	-	DBN+LSTM (Vohra et al., 2015)
Nottingham (loss)	3.07	1M	1.32	-	DBN+LSTM (Vohra et al., 2015)
Word PTB (ppl)	88.68	13M	47.7	22M	AWD-LSTM-MoS + Dynamic Eval. (Yang et al., 2018)
Word Wiki-103 (ppl)	45.19	148M	40.4	>300M	Neural Cache Model (Large) (Grave et al., 2017)
Word LAMBADA (ppl)	1279	56M	138	>100M	Neural Cache Model (Large) (Grave et al., 2017)
Char PTB (bpc)	1.31	3M	1.22	14M	2-LayerNorm HyperLSTM (Ha et al., 2017)
Char text8 (bpc)	1.45	4.6M	1.29	>12M	HM-LSTM (Chung et al., 2016)

arxiv.org/pdf/1803.01271.pdf

Stability: do we need RNNs at all?

J. Miller, M. Hardt: arxiv.org/abs/1805.10369

A system is $\lambda < 1$ contractive if

$$\left\|\phi_w(h, x) - \phi_w(h', x)\right\| \leq \lambda \left\|h - h'\right\|$$

The difference of one step outputs is smaller than the difference of the hidden states.

Proposition (Informal version of Proposition 1). *Assuming the system ϕ is λ-contractive and under additional Lipschitz assumptions, we show if $k \geq O(\log(1/(1-\lambda)\varepsilon))$, then the difference in predictions between the recurrent and truncated model is negligible, $\left\|y_t - y_t^k\right\| \leq \varepsilon$.*

Note: if the LSTM has a forget gate, it will never be fully open, the LSTM is contractive!

Conclusions
What RNNs have brought to DL?

1. Hidden state and recurrence ☺

2. Gating units
   Highway networks, Wavenet etc. all use gates despite being feed-forward.
   Gates decouple **if** from **what**.
3. Training hacks, Gradient clipping.
   It allows stable training with larger learning rates, and helps feedforward net training too.

References

Graves, A., 2013. Generating sequences with recurrent neural networks. arXiv preprint arXiv:1308.0850.

Bengio, Y., Simard, P. and Frasconi, P., 1994. Learning long-term dependencies with gradient descent is difficult. IEEE transactions on neural networks, 5(2), pp.157-166.

Hochreiter, S. and Schmidhuber, J., 1997. Long short-term memory. Neural computation, 9(8), pp.1735-1780.

Pascanu, R., Mikolov, T. and Bengio, Y., 2013, February. On the difficulty of training recurrent neural networks. In International Conference on Machine Learning (pp. 1310-1318).

Jaeger, H., 2002. Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the echo state network approach (Vol. 5). Bonn: GMD-Forschungszentrum Informationstechnik.

Xingjian, S.H.I., Chen, Z., Wang, H., Yeung, D.Y., Wong, W.K. and Woo, W.C., 2015. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. In Advances in neural information processing systems (pp. 802-810).

Zhang, Y., Chan, W. and Jaitly, N., 2017, March. Very deep convolutional networks for end-to-end speech recognition. In Acoustics, Speech and Signal Processing (ICASSP), 2017 IEEE International Conference on (pp. 4845-4849). IEEE.

Chung, J., Gulcehre, C., Cho, K. and Bengio, Y., 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555.

Arjovsky, M., Shah, A. and Bengio, Y., 2016, June. Unitary evolution recurrent neural networks. In International Conference on Machine Learning (pp. 1120-1128).

References

Schuster, M. and Paliwal, K.K., 1997. Bidirectional recurrent neural networks. IEEE Transactions on Signal Processing, 45(11), pp.2673-2681.

Koutnik, J., Greff, K., Gomez, F. and Schmidhuber, J., 2014. A clockwork rnn. arXiv preprint arXiv:1402.3511.

Ioffe, S. and Szegedy, C., 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167.

Cooijmans, T., Ballas, N., Laurent, C., Gülçehre, Ç. and Courville, A., 2016. Recurrent batch normalization. arXiv preprint arXiv:1603.09025.

Graves, A., 2011. Practical variational inference for neural networks. In Advances in neural information processing systems (pp. 2348-2356).

Zaremba, W., Sutskever, I. and Vinyals, O., 2014. Recurrent neural network regularization. arXiv preprint arXiv:1409.2329.

Gal, Y. and Ghahramani, Z., 2016. A theoretically grounded application of dropout in recurrent neural networks. In Advances in neural information processing systems (pp. 1019-1027).

Krueger, D., Maharaj, T., Kramár, J., Pezeshki, M., Ballas, N., Ke, N.R., Goyal, A., Bengio, Y., Courville, A. and Pal, C., 2016. Zoneout: Regularizing rnns by randomly preserving hidden activations. arXiv preprint arXiv:1606.01305.

Bengio, S., Vinyals, O., Jaitly, N. and Shazeer, N., 2015. Scheduled sampling for sequence prediction with recurrent neural networks. In Advances in Neural Information Processing Systems (pp. 1171-1179).

References

Socher, R., Lin, C.C., Manning, C. and Ng, A.Y., 2011. Parsing natural scenes and natural language with recursive neural networks. In Proceedings of the 28th international conference on machine learning (ICML-11) (pp. 129-136).

Tai, K.S., Socher, R. and Manning, C.D., 2015. Improved semantic representations from tree-structured long short-term memory networks. arXiv preprint arXiv:1503.00075.

Oord, A.V.D., Kalchbrenner, N. and Kavukcuoglu, K., 2016. Pixel recurrent neural networks. arXiv preprint arXiv:1601.06759.

Collins, J., Sohl-Dickstein, J. and Sussillo, D., 2016. Capacity and trainability in recurrent neural networks. arXiv preprint arXiv:1611.09913.

Britz, D., Goldie, A., Luong, M.T. and Le, Q., 2017. Massive exploration of neural machine translation architectures. arXiv preprint arXiv:1703.03906.

Bai, S., Kolter, J.Z. and Koltun, V., 2018. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. arXiv preprint arXiv:1803.01271.

Miller, J. and Hardt, M., 2018. When Recurrent Models Don't Need To Be Recurrent. arXiv preprint arXiv:1805.10369.

Bradbury, J., Merity, S., Xiong, C. and Socher, R., 2016. Quasi-recurrent neural networks. arXiv preprint arXiv:1611.01576.

Foerster, J.N., Gilmer, J., Chorowski, J., Sohl-Dickstein, J. and Sussillo, D., 2016. Input switched affine networks: An RNN architecture designed for interpretability. arXiv preprint arXiv:1611.09434.

Karpathy, A., Johnson, J. and Fei-Fei, L., 2015. Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*.