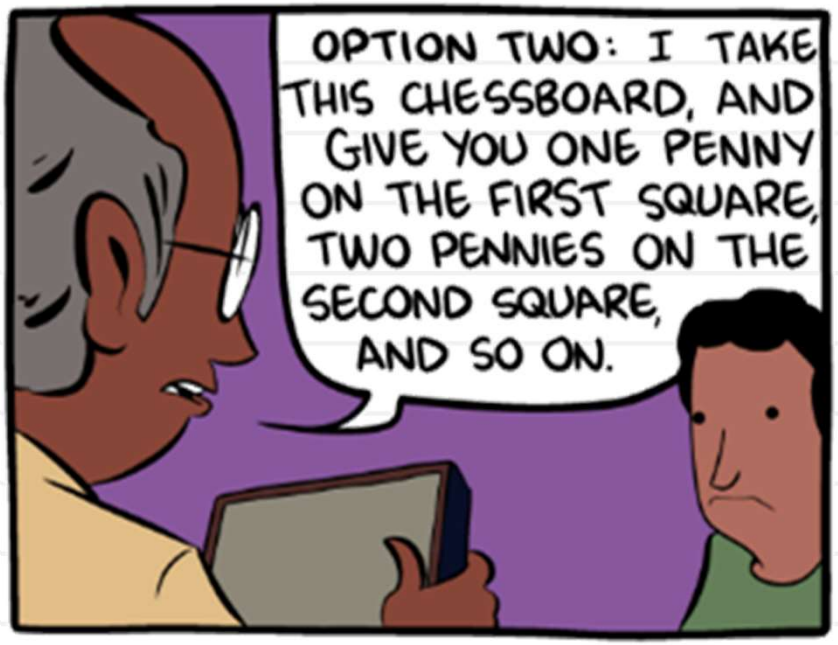
Lecture 10: Deep Sequence Modeling. Recurrent neural networks.

Predictive learning

- Given an element predict *nearby* elements (e.g. next, previous, adjacent, etc.)
- Does not require annotated data ("self-supervised")
- Usually considered as unsupervised, but often works much better than "plain" unsupervised
- Particularly prominent in NLP, but now gaining popularity in many fields

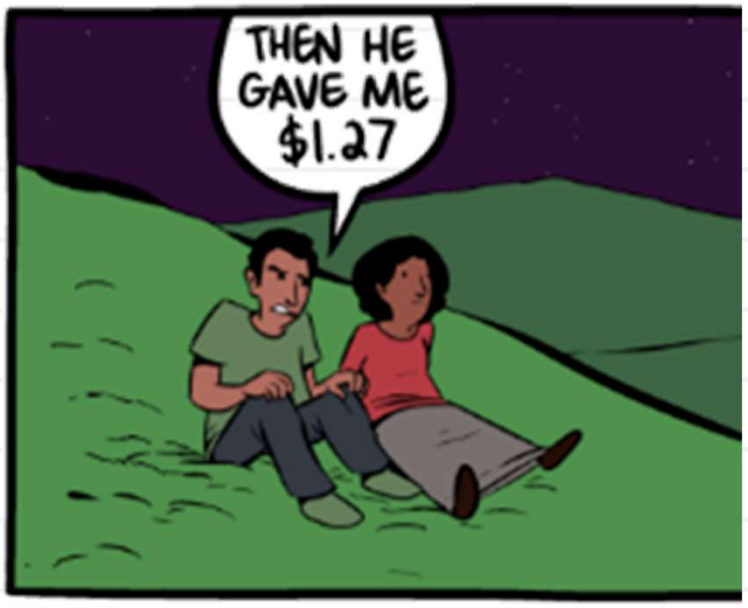
Today's focus: sequence modeling, sequence prediction

Predicting sequences matters



Applications:

- Synthesis (text, speech, etc.)
- Probabilistic modelling
- Compression



smbc-comics.com

Training sequence prediction

*A cat sat on a ma**?***

Inherently probabilistic: need to predict probabilities over lexicon

Training sequence prediction

A cat sat on a ma?

Predominantly maximum likelihood learning:

$$\max_{\theta} \sum_{i} \log P_{\theta}(x_t^i \mid x_{t-1}^i \; x_{t-2}^i \cdots x_1^i)$$

Many models goes back fixed number of steps:

$$\max_{\theta} \sum_{i} \log P_{\theta}(x_t^i \mid x_{t-1}^i \; x_{t-2}^i \cdots x_{t-N}^i)$$

Temporal window

Fixed window/order architectures

$$P_\theta(x_t \mid x_{t-1}\ x_{t-2} \cdots x_{t-N})$$

- N-grams (with smoothing)
- CNNs (aka TDNNs)
- Any probabilistic classifier (e.g. decision forest, etc.)

NB: using padding for the special symbol (UNK) we can train model for shorter sequences

Probabilistic modeling of long sequences

Assume given $p_\theta(x_t \mid x_{t-1}\, x_{t-2} \cdots x_{t-N})$

$$p(x_M\, x_{M-1} \cdots x_1) =$$

$$p(x_M \mid x_{M-1} \cdots x_1) \cdot p(x_{M-1} \cdots x_1) =$$

$$p(x_M \mid x_{M-1} \cdots x_1)\, p(x_{M-1} \mid x_{M-2} \cdots x_1) \cdots \approx$$

$$p_\theta(x_M \mid x_{M-1} \cdots x_{M-N})\, p_\theta(x_{M-1} \mid x_{M-2} \cdots x_{M-N-1}) \cdots$$

$$= \prod_{j=1}^{M} p_\theta(x_j \mid x_{j-1} \cdots x_{j-N})$$

Assessing a probabilistic model

1. Train $p_\theta(x_t \mid x_{t-1}\ x_{t-2} \cdots x_{t-N})$
2. Evaluate $\prod_{j=1}^{M} p_\theta(x_j \mid x_{j-1} \cdots x_{j-n})$ on a hold-out set

Common measure (*perplexity):*

$$PP(x_1 \ldots x_M) = \sqrt[M]{\prod_{j=1}^{M} p_\theta(x_j \mid x_{j-1} \cdots x_{j-n})}^{-1}$$

ML sequence generation

Task: draw a sample sequence with high-probability

$$\prod_{j=1}^{M} p_\theta(x_j \mid x_{j-1} \dots x_{j-N})$$

Option 1: synthesize one-by-one greedily, picking the symbol with highest probability

$$\hat{x}_j = \operatorname{argmax} p_\theta(x_j \mid \hat{x}_{j-1} \dots \hat{x}_{j-N})$$

Option 2: *beam search*

Beam search

$$\prod_{j=1}^{M} p_\theta(x_j \mid x_{j-1} \cdots x_{j-w})$$

The c??????

The ca?????

The cat????

The cap????

The cam????

The co?????

The cor????

The col????

The cow????

The ch?????

The cha????

The cho????

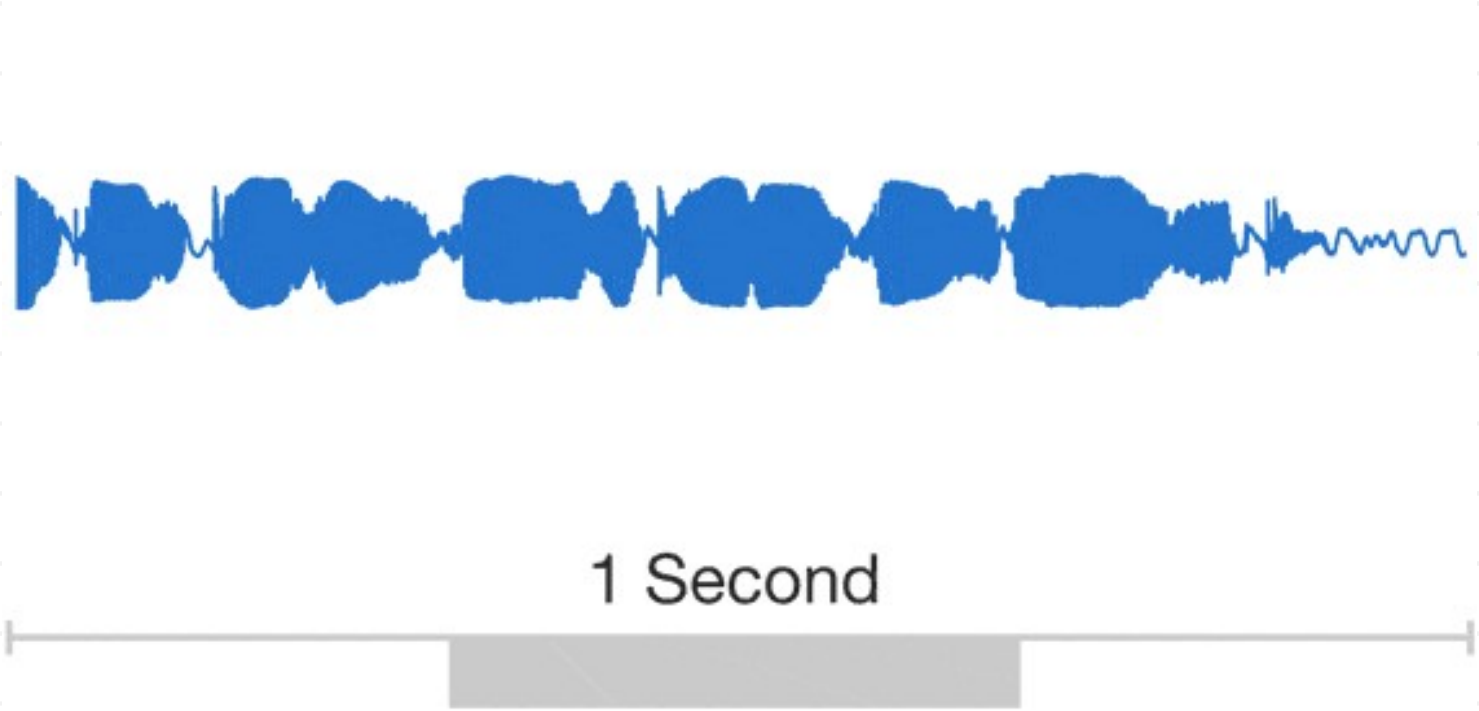
The chi????

The cat????

The cap????

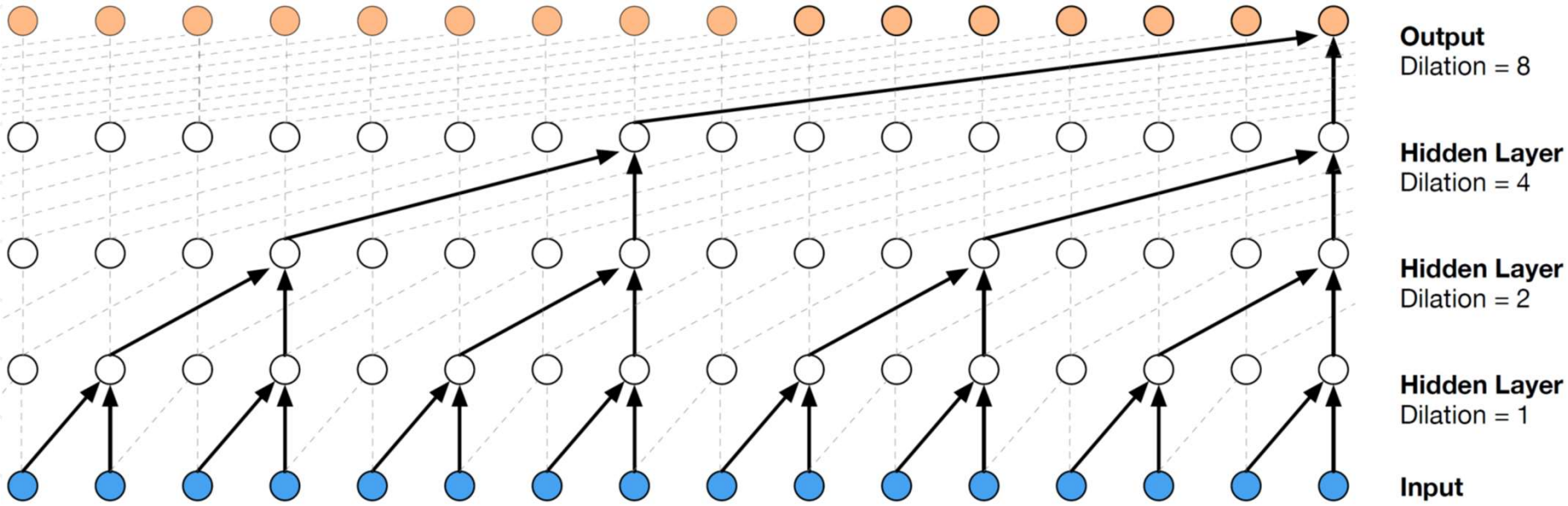
The cor????

WaveNet



- Generating raw waveforms at 16 kHz (very uncommon)

WaveNet: dilated ConvNet



- Repeated pattern of dilations:

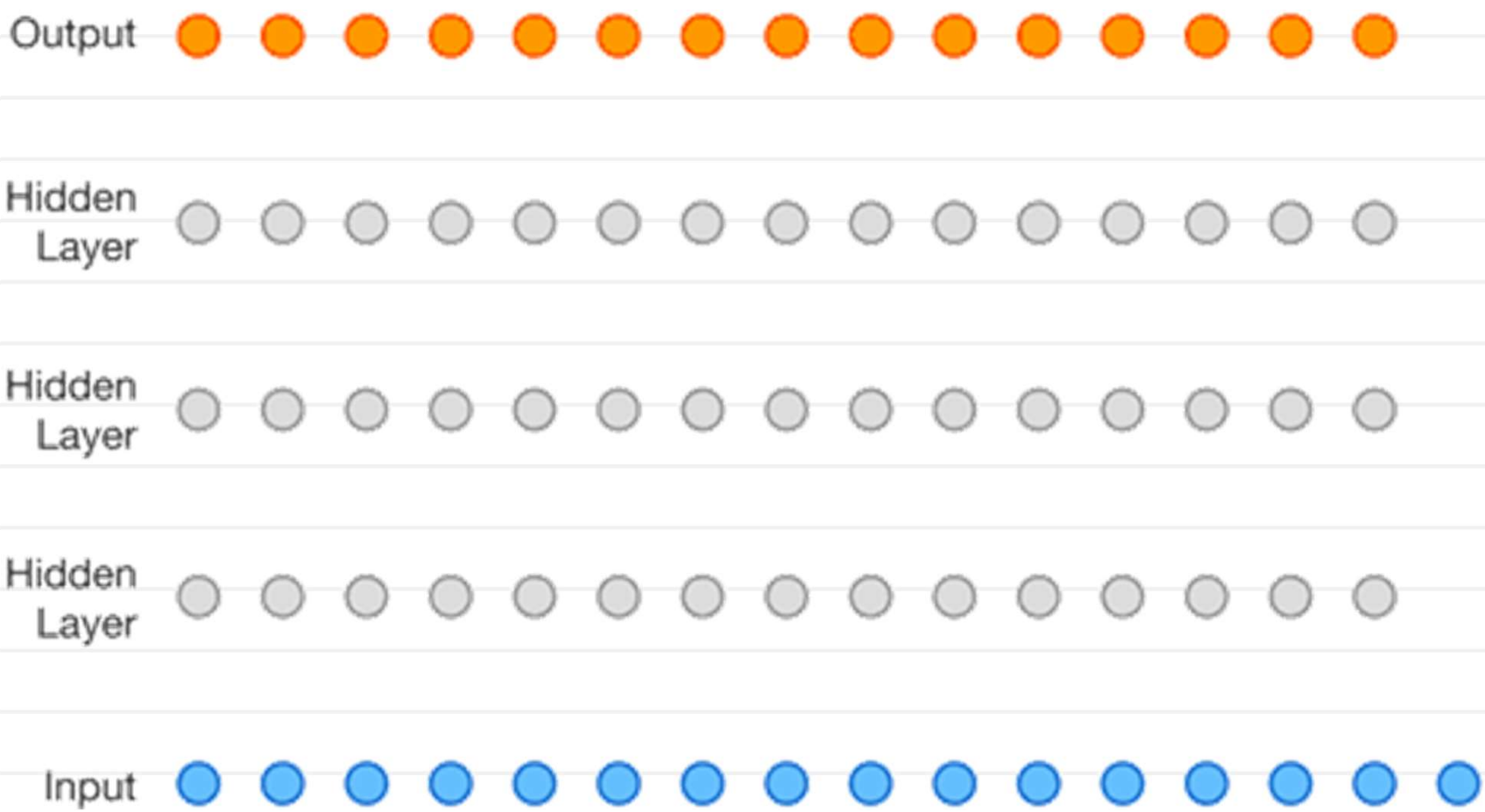
  1,2,…512,1,2,…512,…
- Gated (bilinear) non-linearity:

$$\mathbf{z} = \tanh\left(W_{f,k} * \mathbf{x}\right) \odot \sigma\left(W_{g,k} * \mathbf{x}\right)$$

- There are also skip connections

[van der Oord et al, 2016]

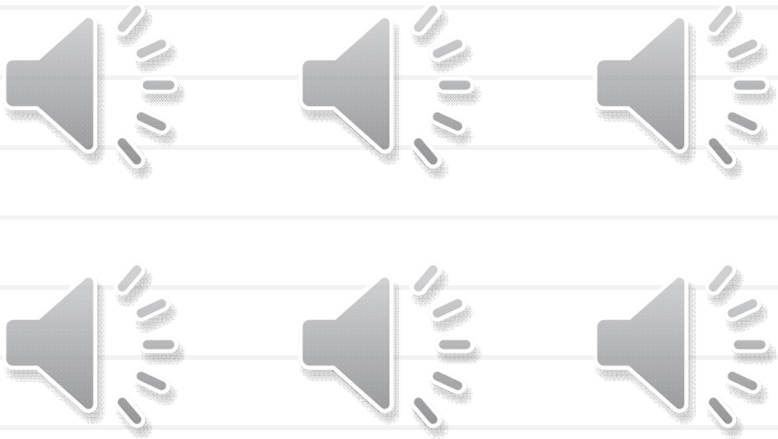
WaveNet: dilated ConvNet



[van der Oord et al, 2016]

WaveNet: piano results

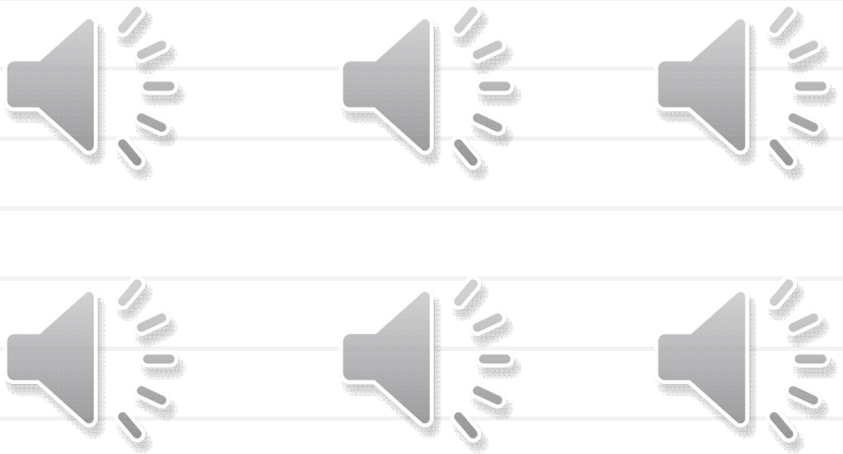
- Trained on 60 hours of piano (from YouTube)



[van der Oord et al, 2016]

WaveNet: speech results

- Trained on 24.6 hours of speech
- Receptive field is 0.24 seconds
- Conditioned on the speaker ID



[van der Oord et al, 2016]

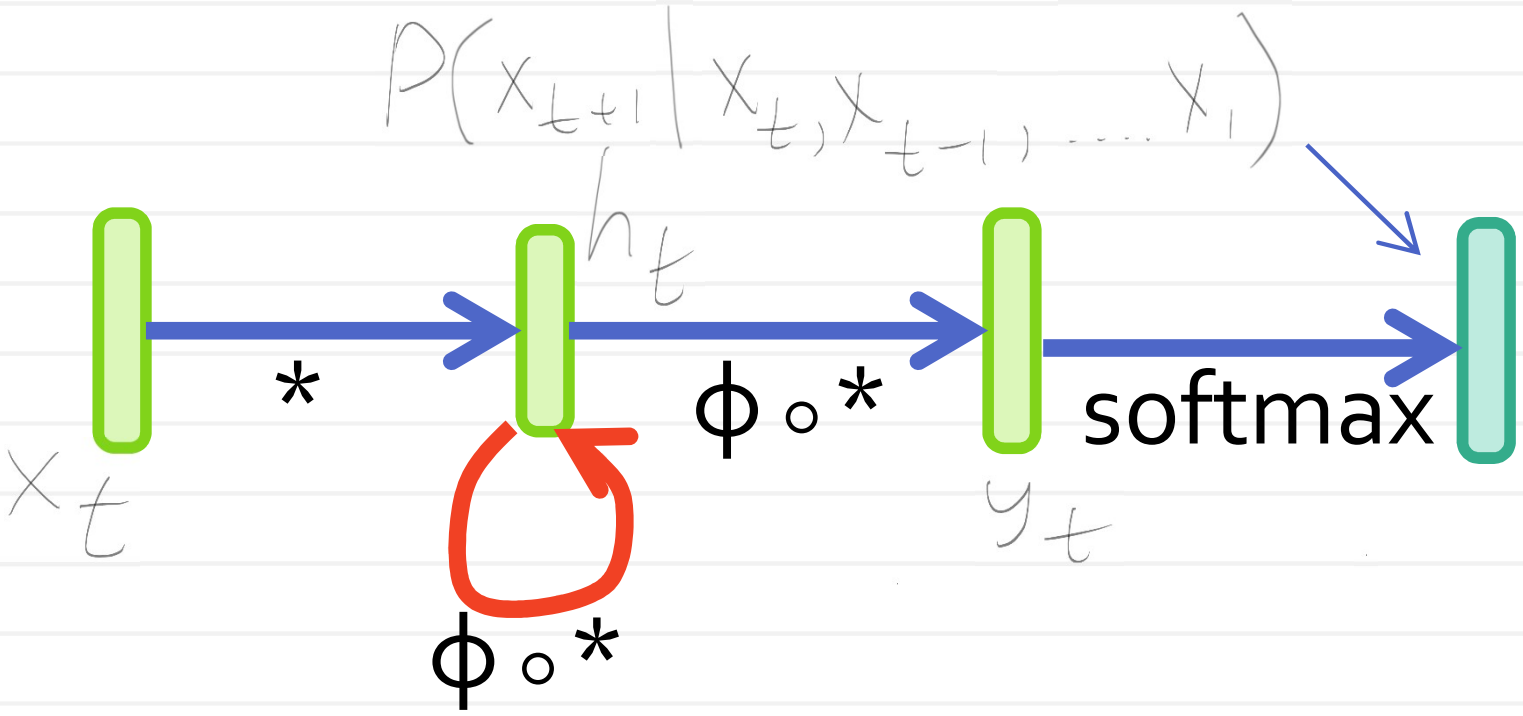
Picking a probabilistic model

- N-grams
- CNNs (aka TDNNs)
- Any probabilistic classifier

Common problem: picking size of the window

- Avoiding overfitting
- To work on instances of different length
- To track long-range behavior

Recurrent neural network (RNN)

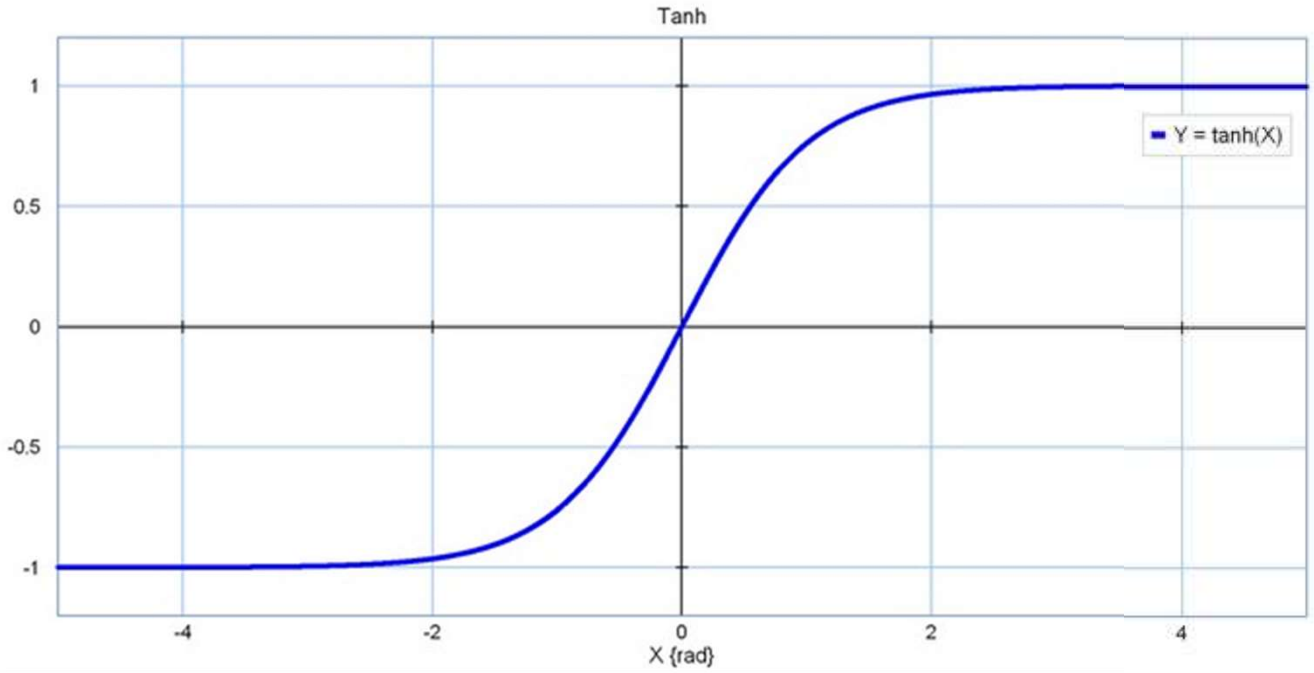


$$h_t = W \phi(h_{t-1}) + W_x x_t$$
$$y_t = W_y \phi(h_t)$$

NB: I omit bias terms but they can be useful!

Most popular non-linearity for RNNs

$$\tanh x : \mathbb{R} \longrightarrow (-1; 1)$$



$$\tanh x' =$$

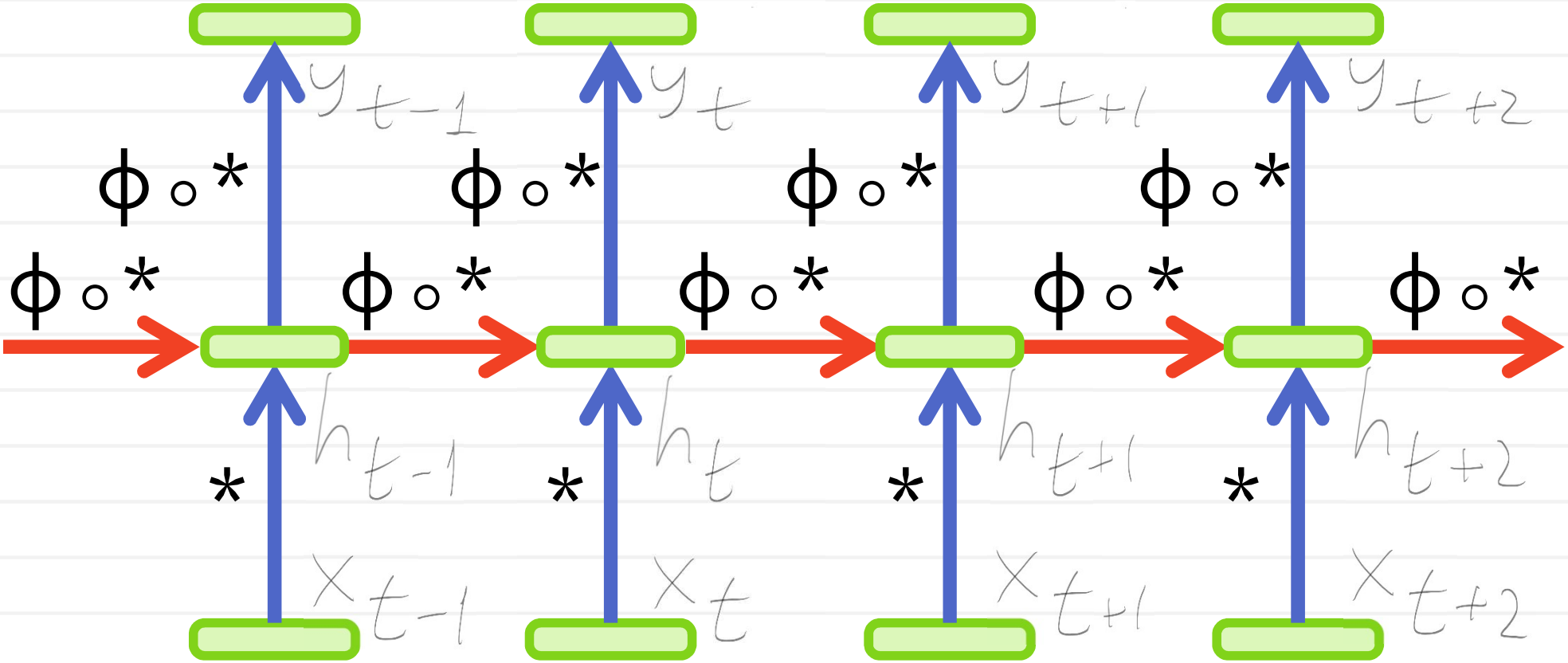
$$1 - \tanh^2 x$$

$$|\tanh x'| \leq 1$$

$$h_t = W \phi(h_{t-1}) + W_x x_t$$

$$y_t = W_y \phi(h_t)$$

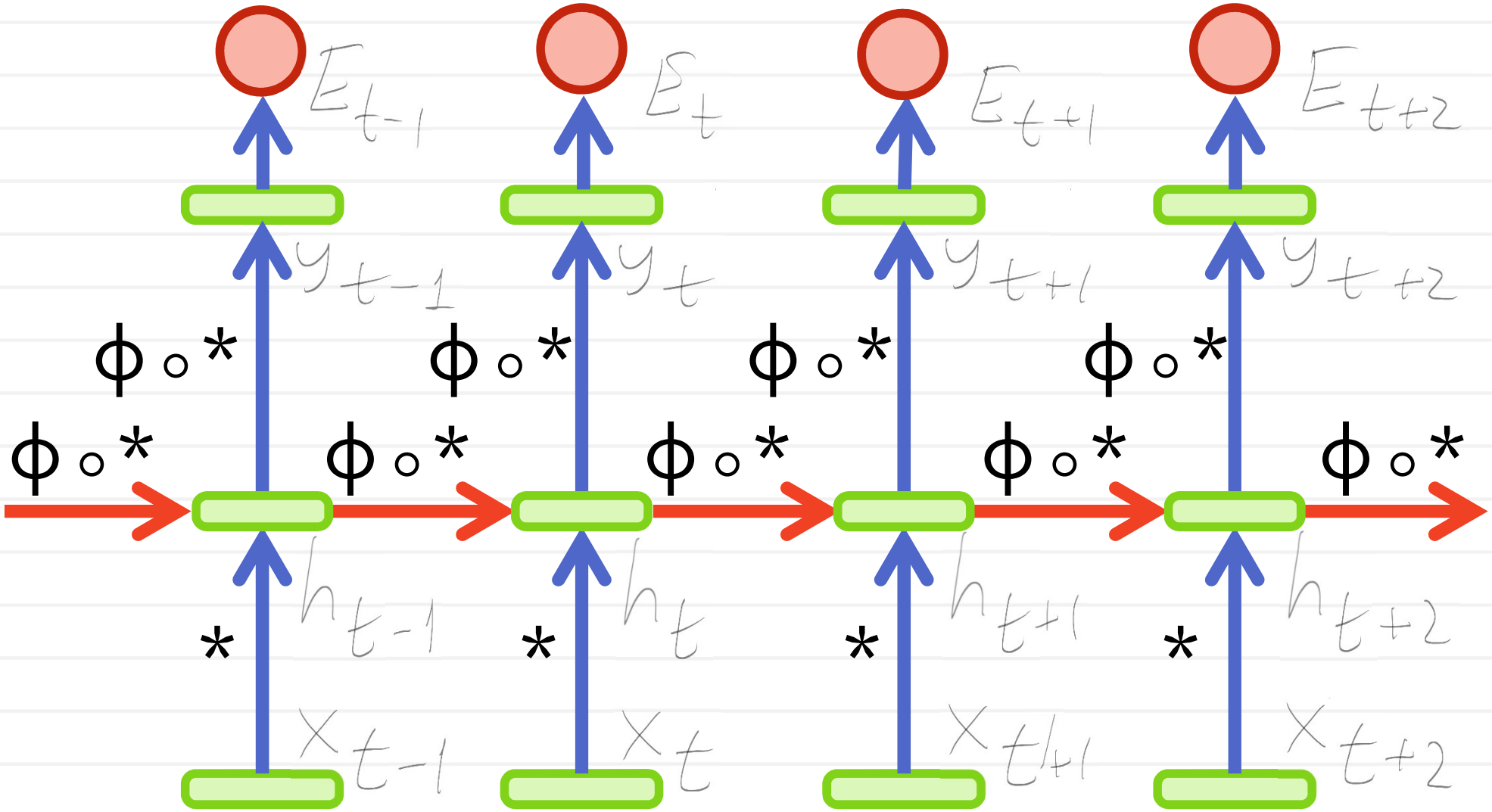
Unwrapping RNN



$$h_t = W\,\phi(h_{t-1}) + W_x x_t$$

$$y_t = W_y\,\phi(h_t)$$

Training RNN



$$h_t = W\,\phi(h_{t-1}) + W_x x_t \qquad y_t = W_y\,\phi(h_t)$$

Training RNN

$$h_t = W\,\phi(h_{t-1}) + W_x x_t$$

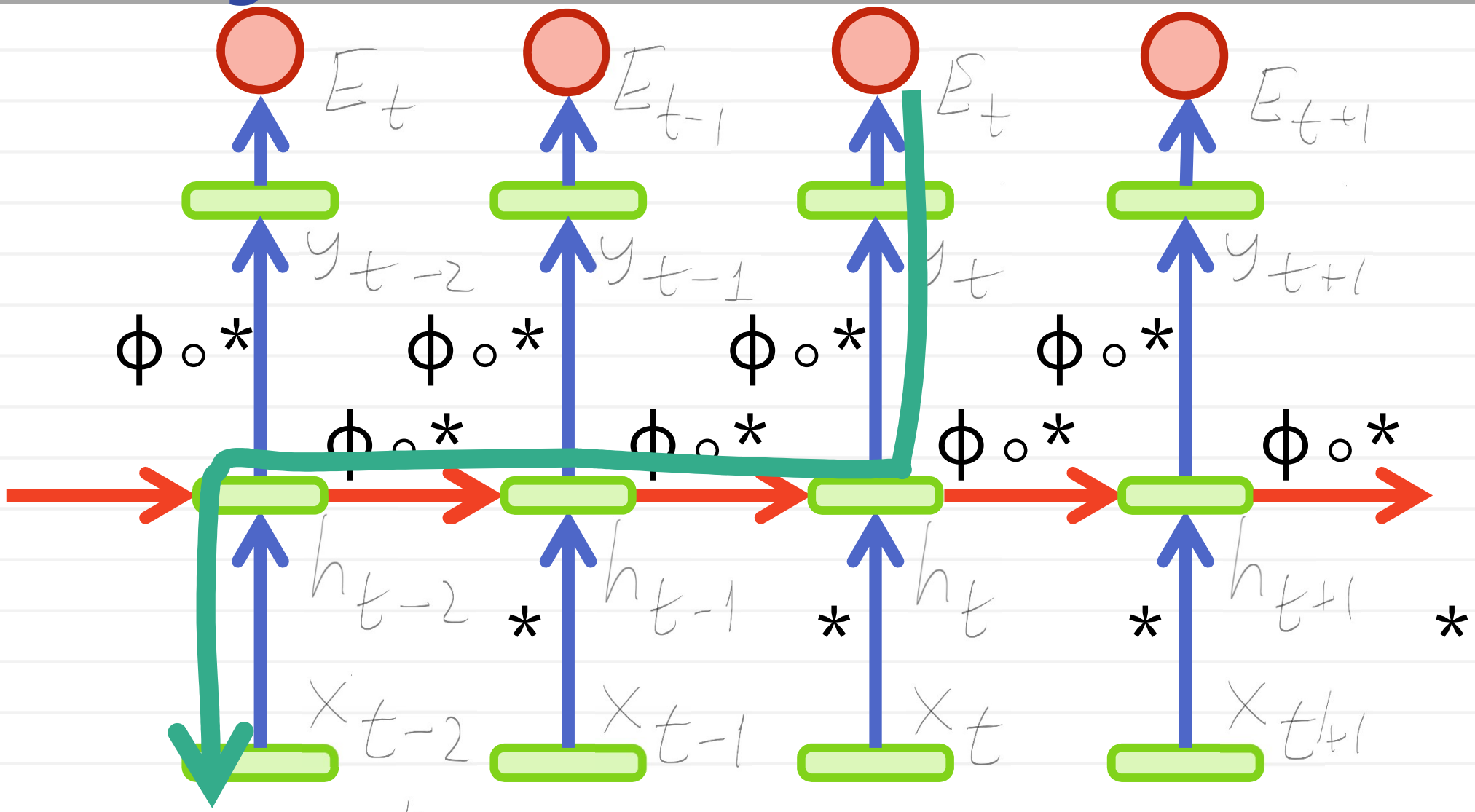
$$y_t = W_y\,\phi(h_t)$$

$$E = \sum_{t=1}^{s} E_t$$

$$\frac{\partial E}{\partial W} = \sum_{t=1}^{s} \frac{\partial E_t}{\partial W}$$

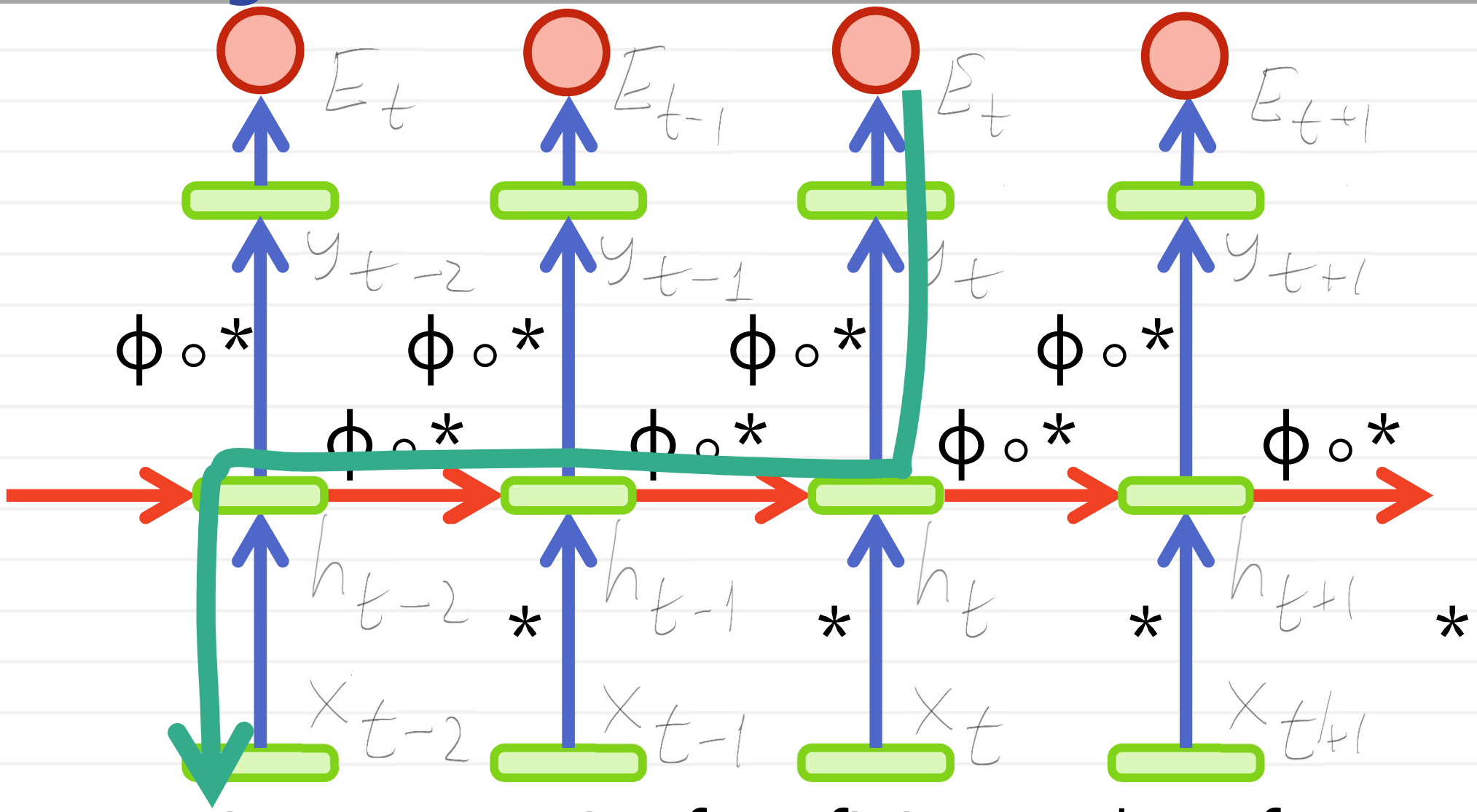
$$\frac{\partial E_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t}\frac{\partial y_t}{\partial h_t}\frac{\partial h_t}{\partial h_k}\frac{\partial h_k}{\partial W}$$

Training RNN



$$\frac{\partial E_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W}$$

Training RNN



In practice: unwrapping for a finite number of time-steps (or training on bounded length sequences)

Training RNN

$$\frac{\partial E_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W}$$

$$\frac{\partial h_t}{\partial h_k} = \prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}} = \prod_{i=k+1}^{t} W^T \operatorname{diag} \phi'(h_{i-1})$$

$$\left\| \frac{\partial h_i}{\partial h_{i-1}} \right\|_2 \leq \|W^T\|_2 \cdot L_\phi = \sigma_{max} L_\phi$$

$$\left\| \frac{\partial h_t}{\partial h_k} \right\|_2 \leq (\sigma_{max} L_\phi)^{t-k}$$

Challenges with training RNN

$$\frac{\partial E_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W}$$

$$\left\| \frac{\partial h_t}{\partial h_k} \right\|_2 \leq \left(\sigma_{max} L_\phi \right)^{t-k}$$

$$\sigma_{max} L_\phi \leq 1 \qquad \sigma_{max} L_\phi >> 1$$

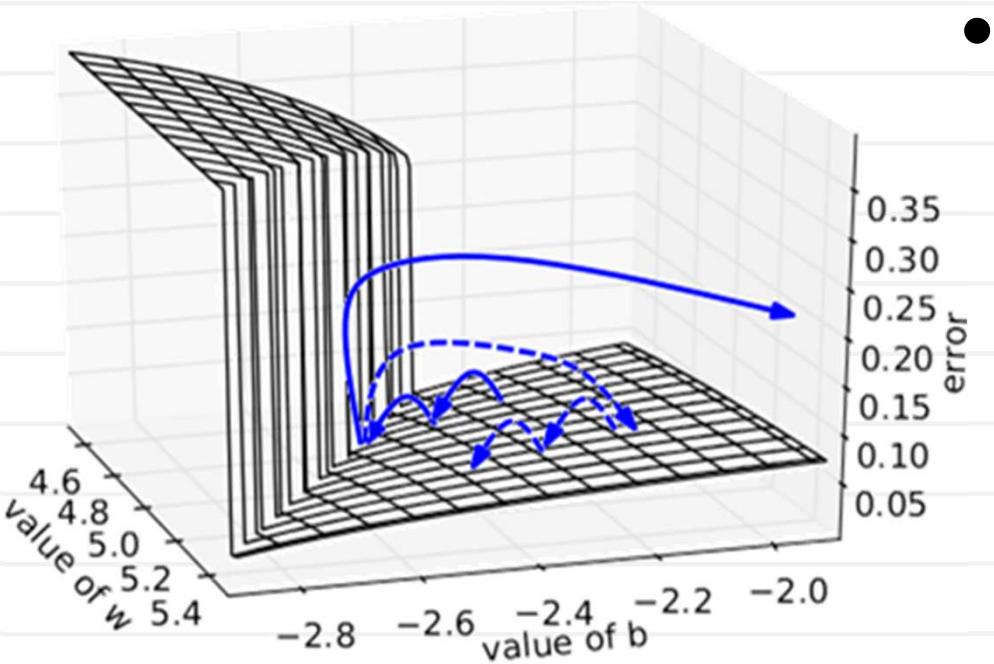
vanishing gradient: network forgets information

exploding gradient: learning quickly "explodes"

Gradient clipping

Algorithm 1 Pseudo-code for norm clipping

$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$
if $\|\hat{\mathbf{g}}\| \geq threshold$ **then**
  $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|}\hat{\mathbf{g}}$
end if



- Simple trick handles gradient explosion (provided that the "valley" is wide)

[Pascanu et al. 2013]

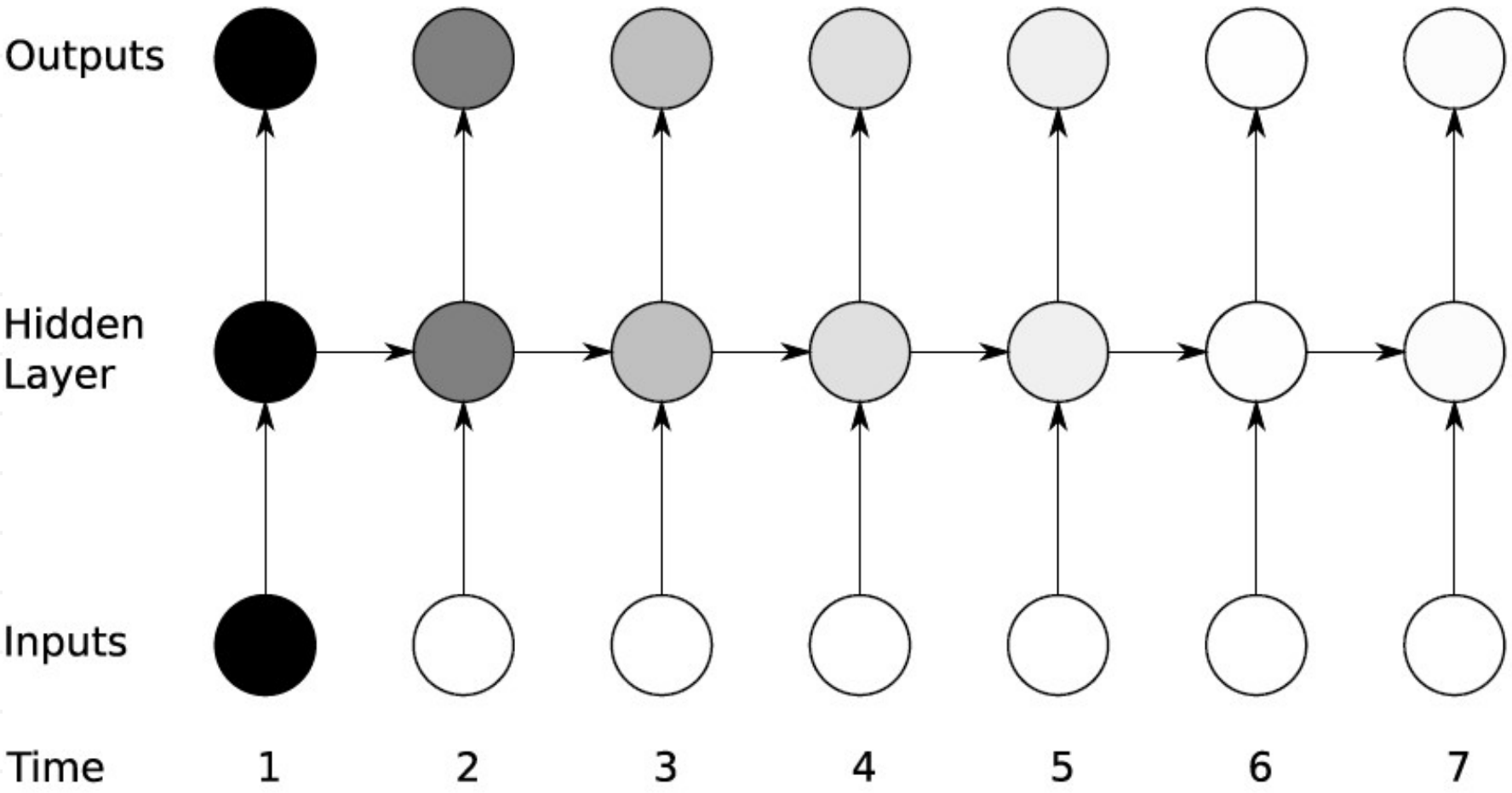
Handling vanishing gradient

$$\frac{\partial E_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W}$$

$$\sigma_{max} L_\phi \leq 1$$

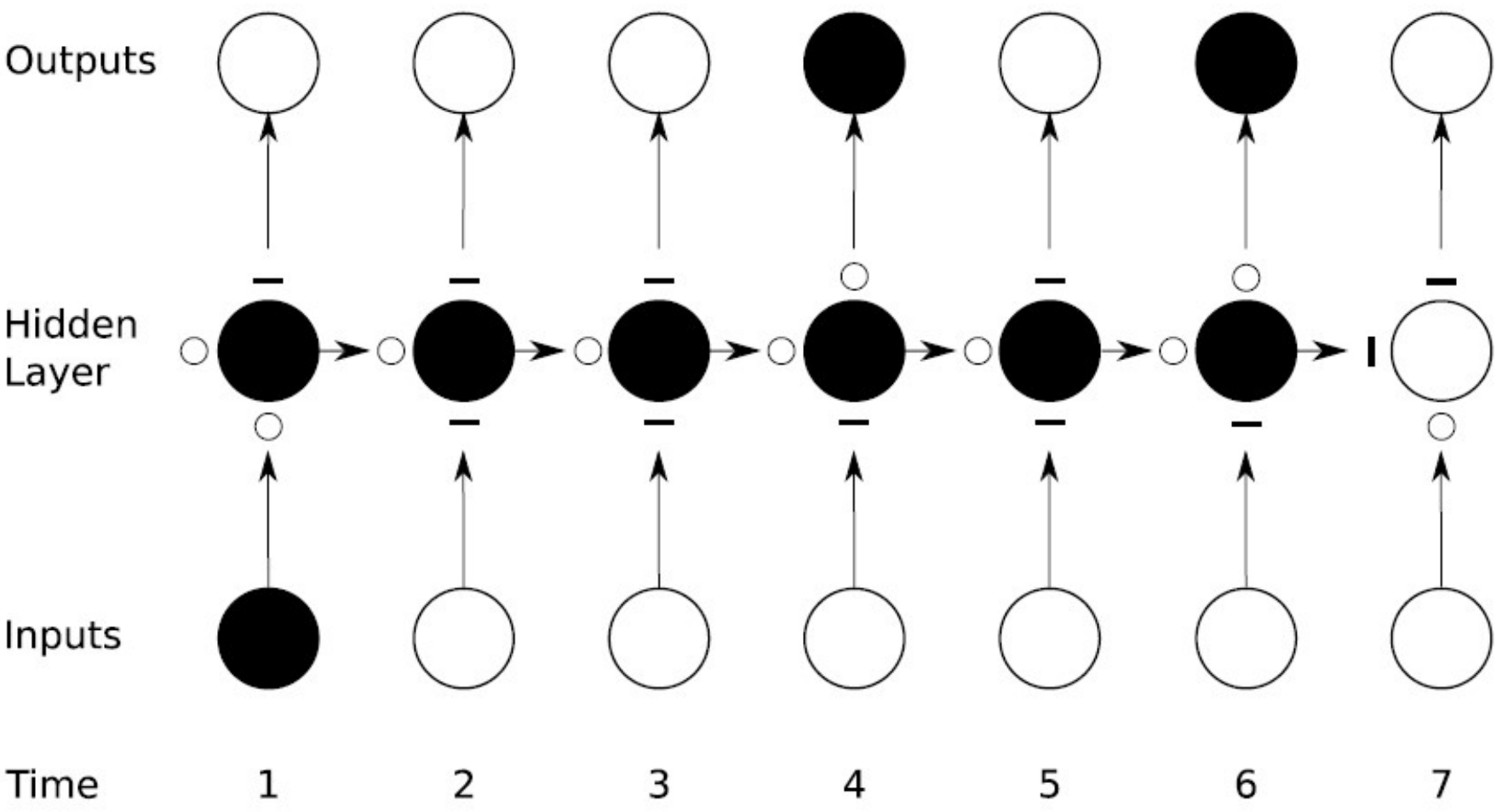
- Even if the gradient does not vanish totally, the information stored in low-energy subspaces will not be propagated
- Idea: we need mechanism to ensure long-term propagation.

Vanishing gradient visualization



The influence of an input unit quickly vanishes with time [Graves 12]

Long Short-term Memory



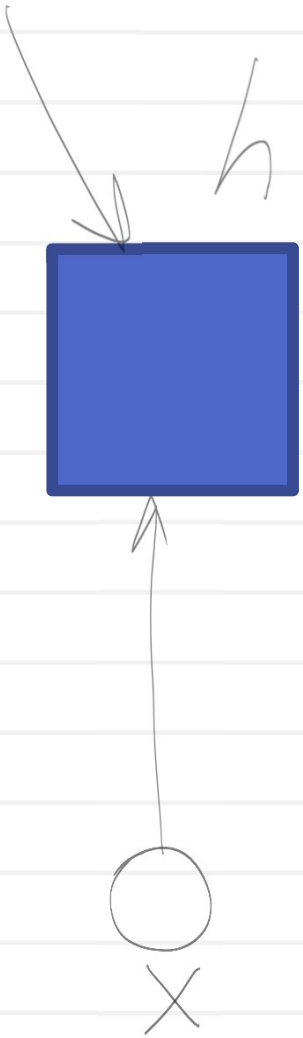
O = gate open

– = gate closed

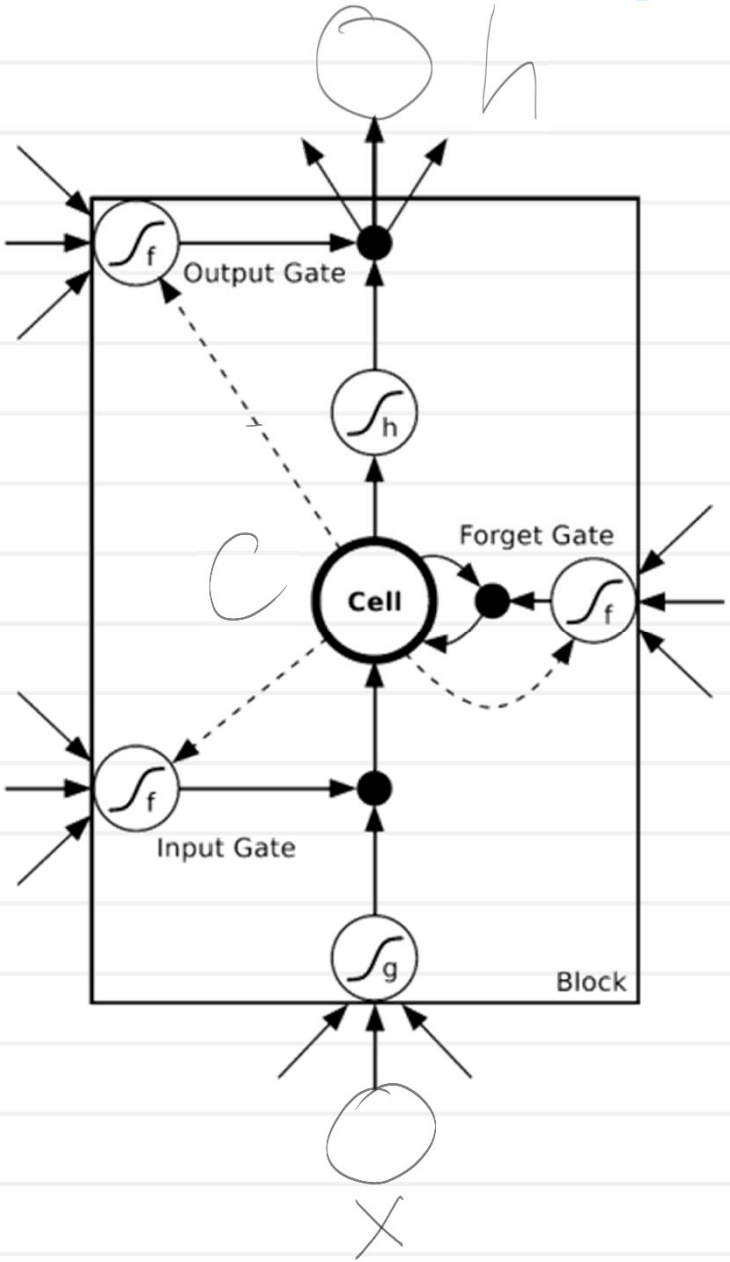
[Graves 12]

[Hochreiter & Schmidhuber 97]

Long Short-term Memory



Plain RNN

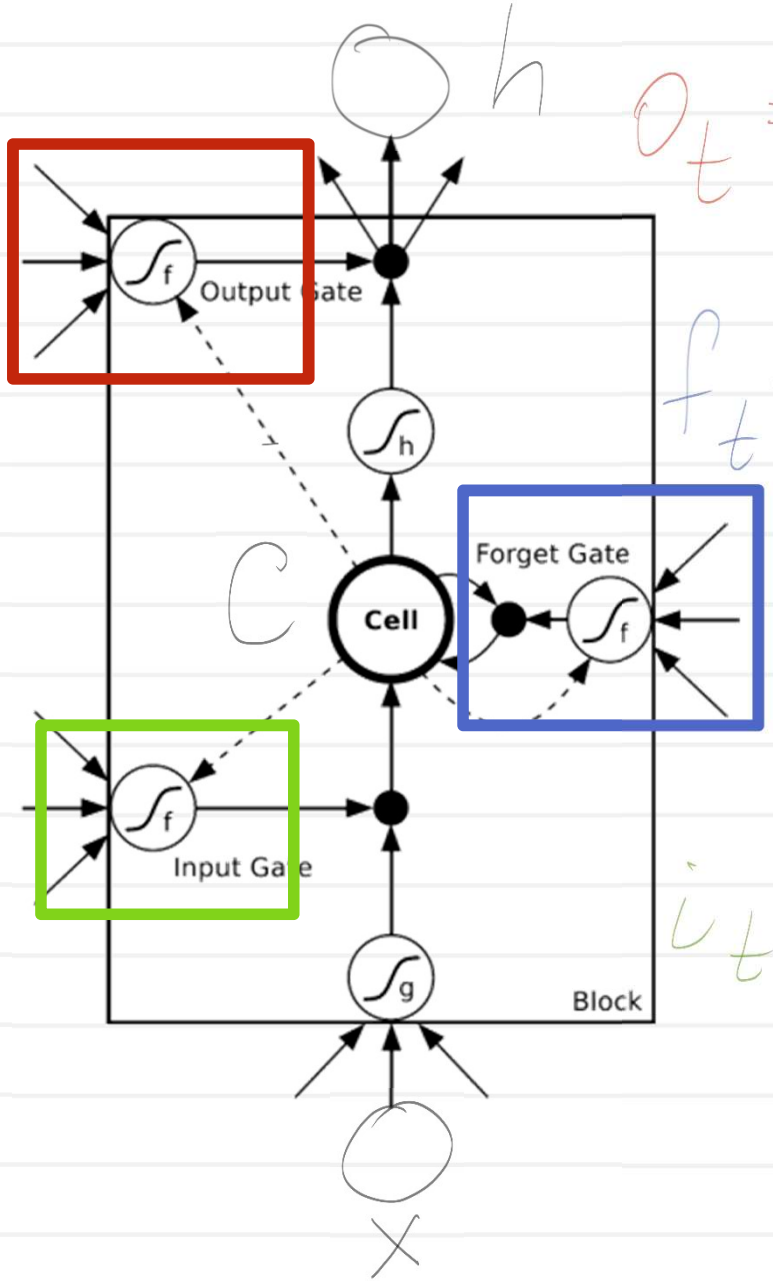


LSTM RNN

Gate activations are trainable functions of x and h

[Graves 12]
[Hochreiter & Schmidhuber 97]

Long Short-term Memory: gate activations



$$o_t = \sigma(W_{xo} x_t + W_{ho} h_{t-1} + b_o)$$

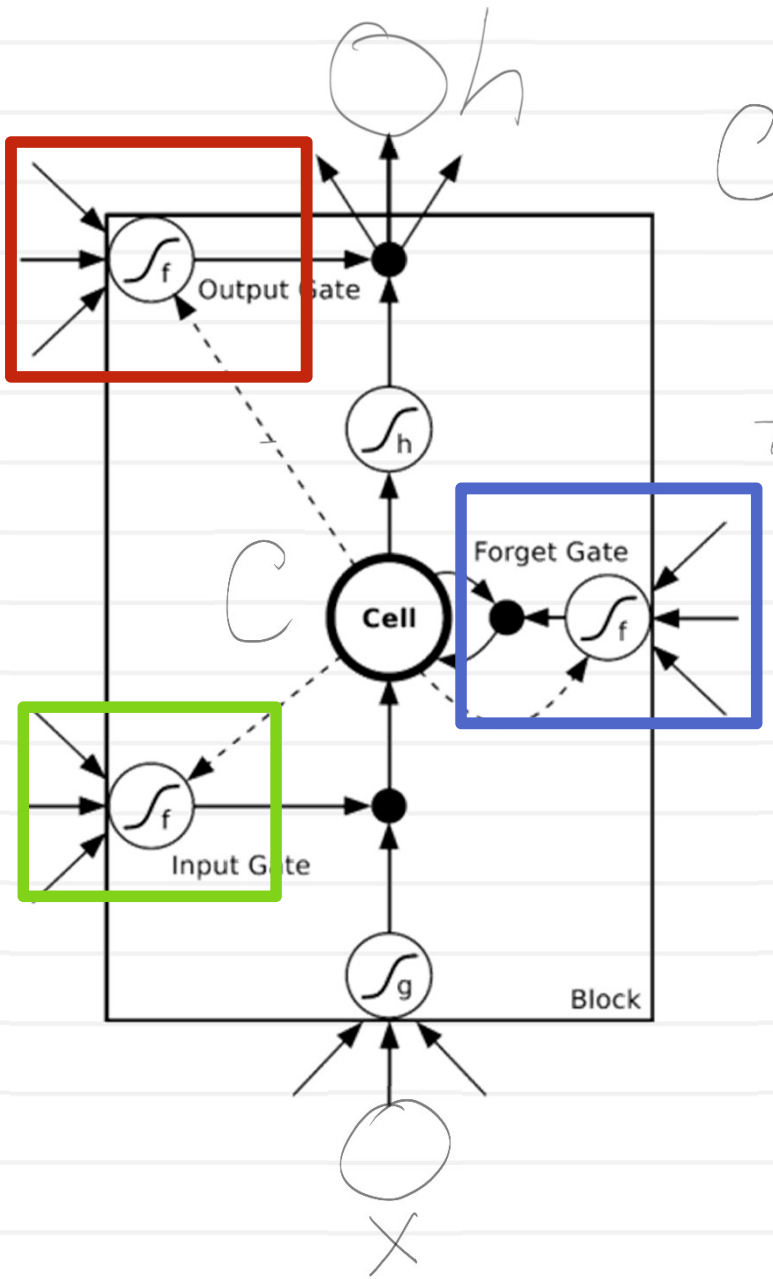
$$f_t = \sigma(W_{xf} x_t + W_{hf} h_{t-1} + b_f)$$

$$i_t = \sigma(W_{xi} x_t + W_{hi} h_{t-1} + b_i)$$

LSTM RNN

[Graves 12]
[Hochreiter & Schmidhuber 97]

Long Short-term Memory: cell update



$$c_t = f_t \odot c_{t-1} +$$

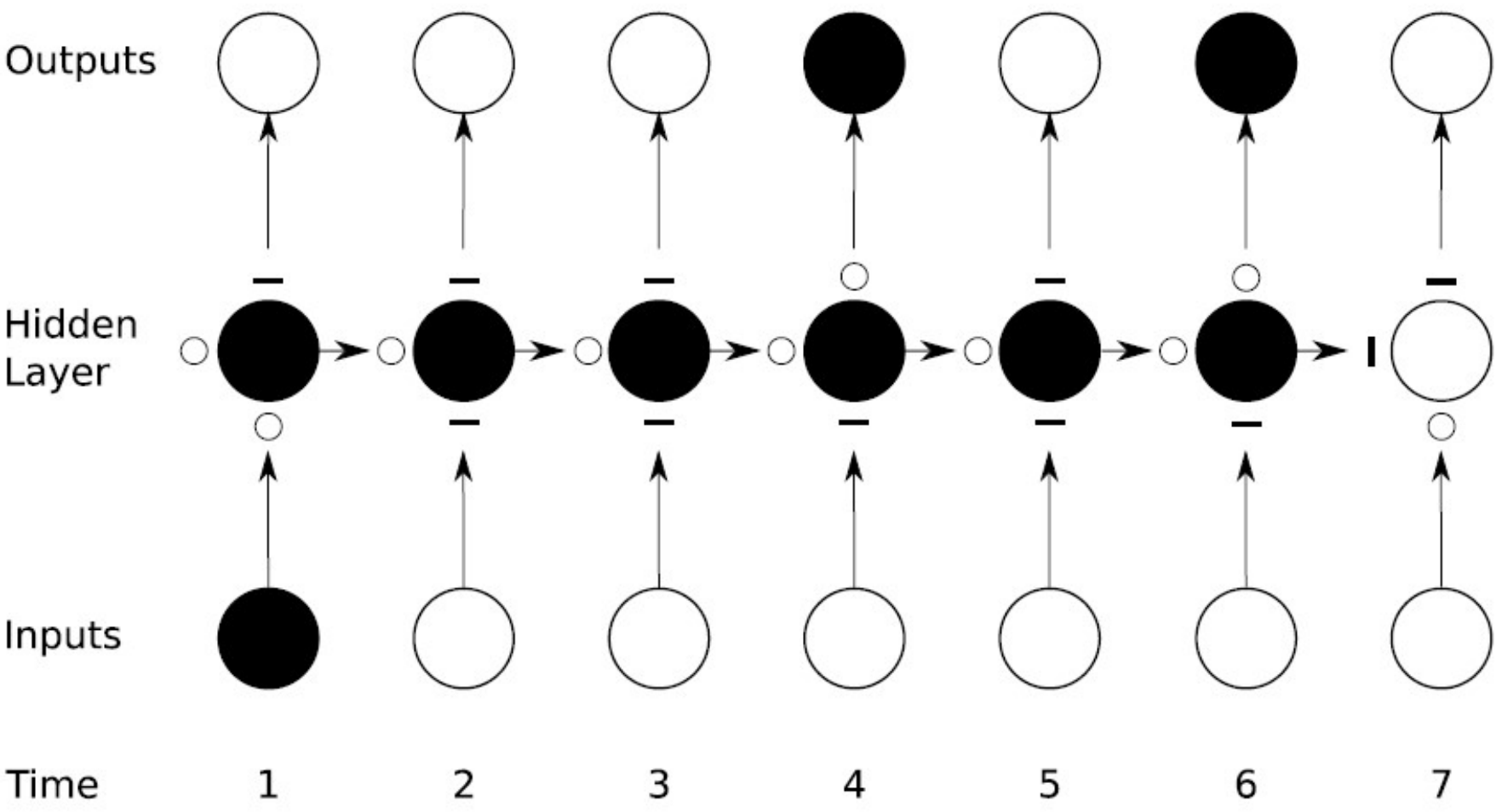
$$+ i_t \odot \phi(W_x x_t + W h_{t-1})$$

$$h_t = o_t \odot \phi(c_t)$$

LSTM RNN

[Graves 12]
[Hochreiter & Schmidhuber 97]

Long Short-term Memory



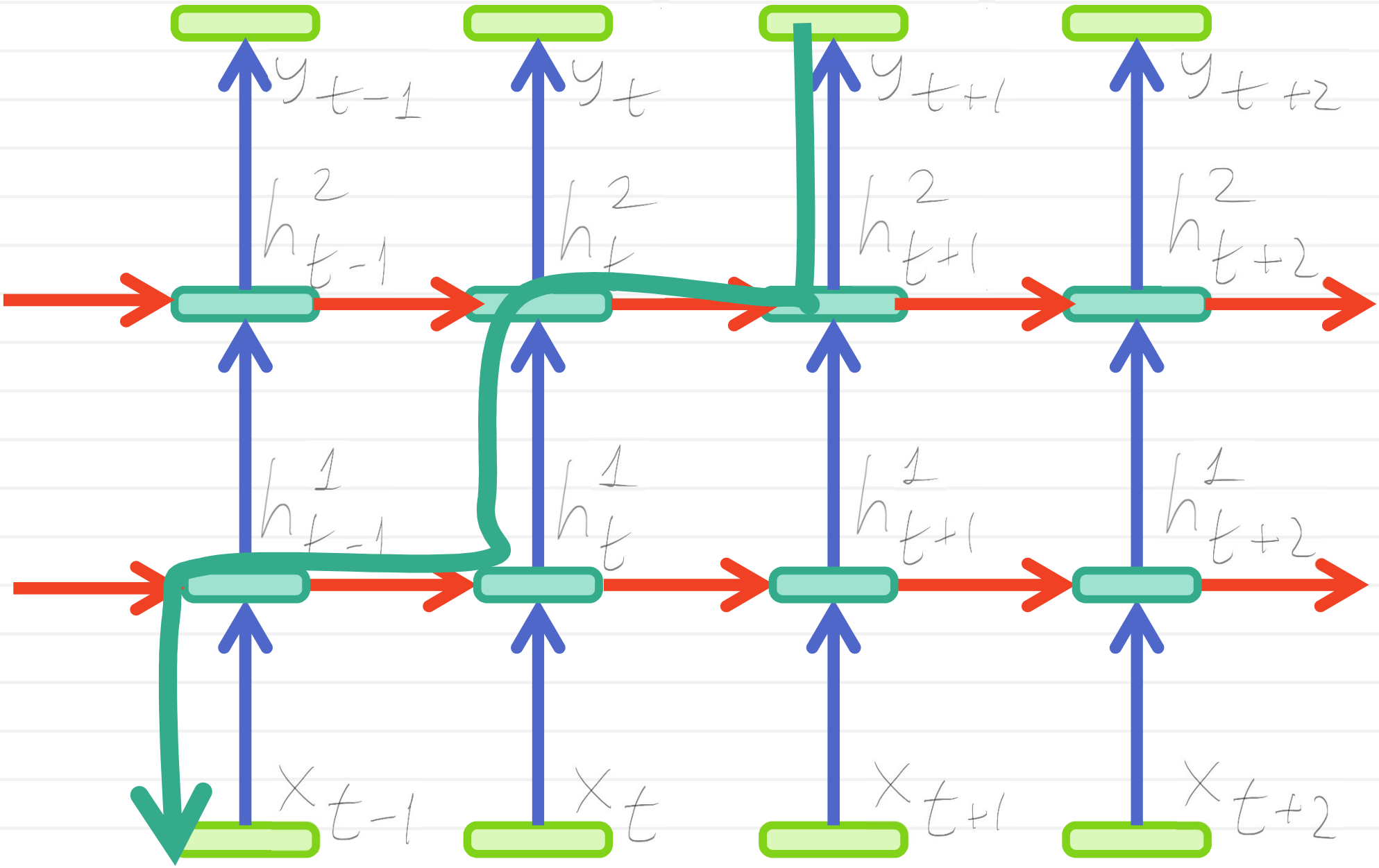
O = gate open

– = gate closed

[Graves 12]

[Hochreiter & Schmidhuber 97]

Multi-layer RNNs



Recap: RNN-LSTM as a probabilistic model

$$P(x_t / x_1 \dots x_{t-1}) = ?$$

$$h_t = LSTM(x_{t-1}, h_{t-1})$$

$$y_t = W h_t$$

$$p_t^i = \frac{\exp(y_t^i)}{\sum_k \exp(y_t^k)} = P(i / x_1 \dots x_{t-1})$$

$$x_t \sim \{p_t^i\}$$

Interpretable LSTM Cells

Cell sensitive to position in line:

```
The sole importance of the crossing of the Berezina lies in the fact
that it plainly and indubitably proved the fallacy of all the plans for
cutting off the enemy's retreat and the soundness of the only possible
line of action--the one Kutuzov and the general mass of the army
demanded--namely, simply to follow the enemy up. The French crowd fled
at a continually increasing speed and all its energy was directed to
reaching its goal. It fled like a wounded animal and it was impossible
to block its path. This was shown not so much by the arrangements it
made for crossing as by what took place at the bridges. When the bridges
broke down, unarmed soldiers, people from Moscow and women with children
who were with the French transport, all--carried on by vis inertiae--
pressed forward into boats and into the ice-covered water and did not,
surrender.
```

Cell that turns on inside quotes:

```
"You mean to imply that I have nothing to eat out of.... On the
contrary, I can supply you with everything even if you want to give
dinner parties," warmly replied Chichagov, who tried by every word he
spoke to prove his own rectitude and therefore imagined Kutuzov to be
animated by the same desire.

Kutuzov, shrugging his shoulders, replied with his subtle penetrating
smile: "I meant merely to say what I said."
```

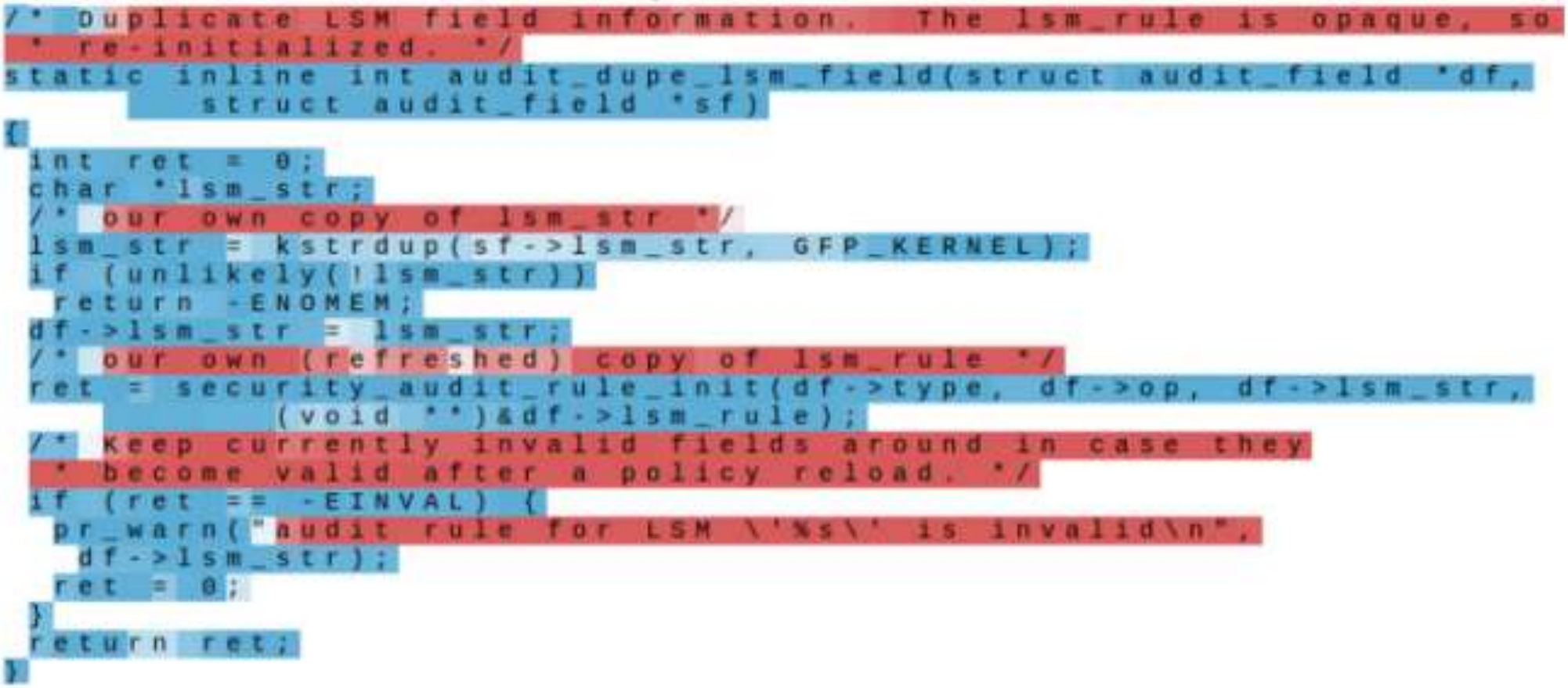
Cell that robustly activates inside if statements:

```
static int __dequeue_signal(struct sigpending *pending, sigset_t *mask,
        siginfo_t *info)
{
  int sig = next_signal(pending, mask);
  if (sig) {
    if (current->notifier) {
      if (sigismember(current->notifier_mask, sig)) {
        if (!(current->notifier)(current->notifier_data)) {
          clear_thread_flag(TIF_SIGPENDING);
          return 0;
        }
      }
    }
    collect_signal(sig, pending, info);
  }
  return sig;
}
```

[Karpathy et al. ICLR16]

Interpretable LSTM Cells

Cell that turns on inside comments and quotes:



```
/* Duplicate LSM field information.  The lsm_rule is opaque, so
 * re-initialized. */
static inline int audit_dupe_lsm_field(struct audit_field *df,
                                       struct audit_field *sf)
{
  int ret = 0;
  char *lsm_str;
  /* our own copy of lsm_str */
  lsm_str = kstrdup(sf->lsm_str, GFP_KERNEL);
  if (unlikely(!lsm_str))
    return -ENOMEM;
  df->lsm_str = lsm_str;
  /* our own (refreshed) copy of lsm_rule */
  ret = security_audit_rule_init(df->type, df->op, df->lsm_str,
                                 (void **)&df->lsm_rule);
  /* Keep currently invalid fields around in case they
   * become valid after a policy reload. */
  if (ret == -EINVAL) {
    pr_warn("audit rule for LSM \'%s\' is invalid\n",
            df->lsm_str);
    ret = 0;
  }
  return ret;
}
```

[Karpathy et al. ICLR16]

Interpretable LSTM Cells

Cell that is sensitive to the depth of an expression:

```
#ifdef CONFIG_AUDITSYSCALL
static inline int audit_match_class_bits(int class, u32 *mask)
{
  int i;
  if (classes[class]) {
    for (i = 0; i < AUDIT_BITMASK_SIZE; i++)
      if (mask[i] & classes[class][i])
        return 0;
  }
  return 1;
}
```

Cell that might be helpful in predicting a new line. Note that it only turns on for some ")":

```
char *audit_unpack_string(void **bufp, size_t *remain, si
{
  char *str;
  if (!*bufp || (len == 0) || (len > *remain))
    return ERR_PTR(-EINVAL);
  /* Of the currently implemented string fields, PATH_MAX
   * defines the longest valid length.
   */
  if (len > PATH_MAX)
    return ERR_PTR(-ENAMETOOLONG);
  str = kmalloc(len + 1, GFP_KERNEL);
  if (unlikely(!str))
    return ERR_PTR(-ENOMEM);
  memcpy(str, *bufp, len);
  str[len] = 0;
  *bufp += len;
  *remain -= len;
  return str;
}
```

[Karpathy et al. ICLR16]

Non-interpretable LSTM Cells

A large portion of cells are not easily interpretable. Here is a typical example:

```
/*  Unpack a filter field's string representation from user-space
 *  buffer.  */
char *audit_unpack_string(void **bufp, size_t *remain, size_t len)
{
  char *str;
  if (!*bufp || (len == 0) || (len > *remain))
    return ERR_PTR(-EINVAL);
  /*  Of the currently implemented string fields, PATH_MAX
   *  defines the longest valid length.
   */
```

[Karpathy et al. ICLR16]

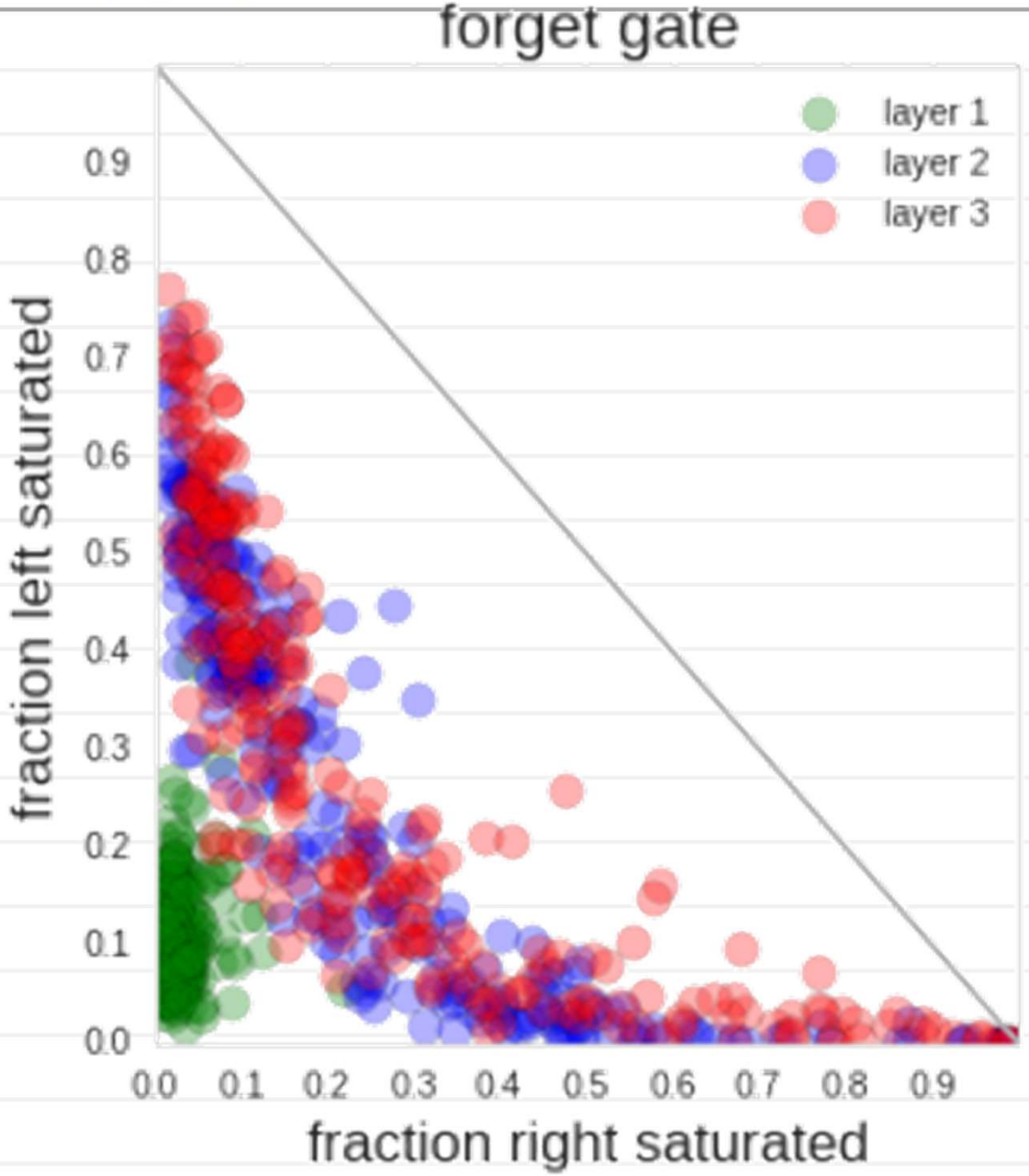
Computer generated "Linux kernel code"

© Andrej Karpathy:

```
static void do_command(struct seq_file *m, void *v)
{
  int column = 32 << (cmd[2] & 0x80);
  if (state)
    cmd = (int)(int_state ^ (in_8(&ch->ch_flags) & Cmd) ? 2 : 1);
  else
    seq = 1;
  for (i = 0; i < 16; i++) {
    if (k & (1 << 1))
      pipe = (in_use & UMXTHREAD_UNCCA) +
        ((count & 0x00000000fffffff8) & 0x000000f) << 8;
    if (count == 0)
      sub(pid, ppc_md.kexec_handle, 0x20000000);
    pipe_set_bytes(i, 0);
  }
  /* Free our user pages pointer to place camera if all dash */
  subsystem_info = &of_changes[PAGE_SIZE];
  rek_controls(offset, idx, &soffset);
  /* Now we want to deliberately put it to device */
  control_check_polarity(&context, val, 0);
  for (i = 0; i < COUNTER; i++)
    seq_puts(s, "policy ");
}
```

More fun: 1) http://karpathy.github.io/2015/05/21/rnn-effectiveness/ 2) your assignment

Interpretable LSTM Cells

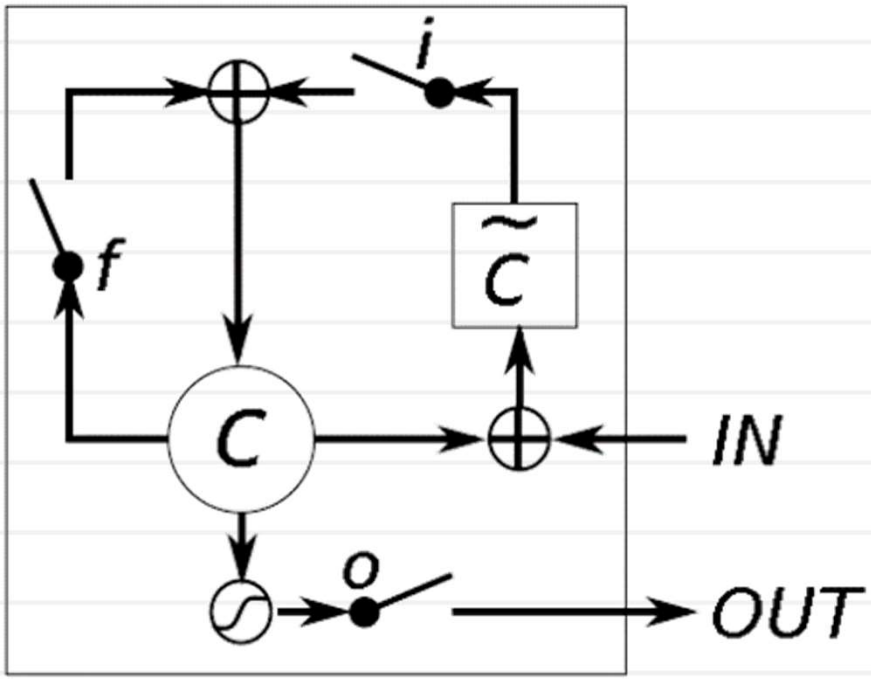


<0.1
(forget info)

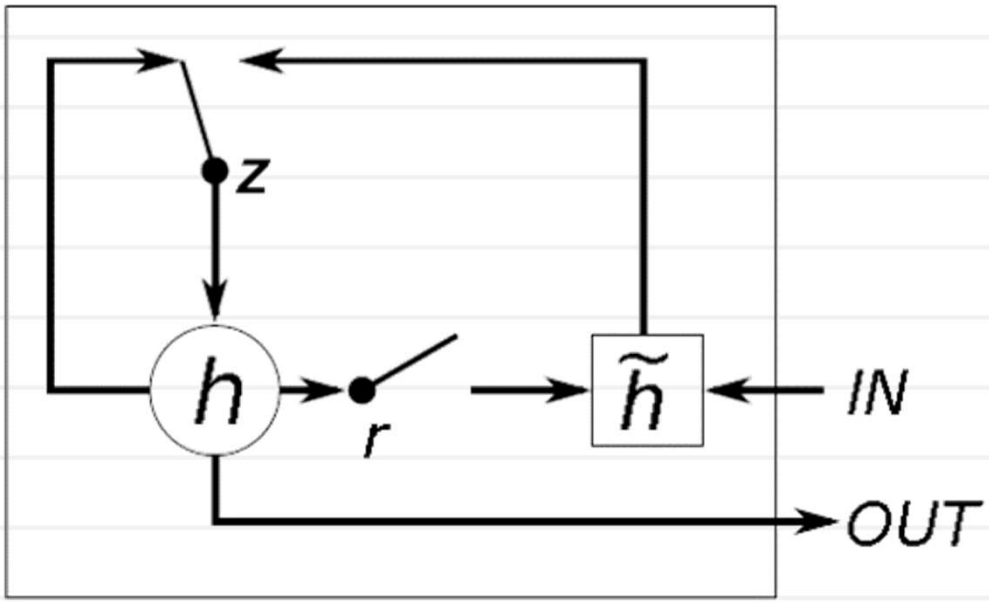
[Karpathy et al. ICLR16]

>0.9 (keep info)

Gated Recurrent Units (GRU)



(a) Long Short-Term Memory



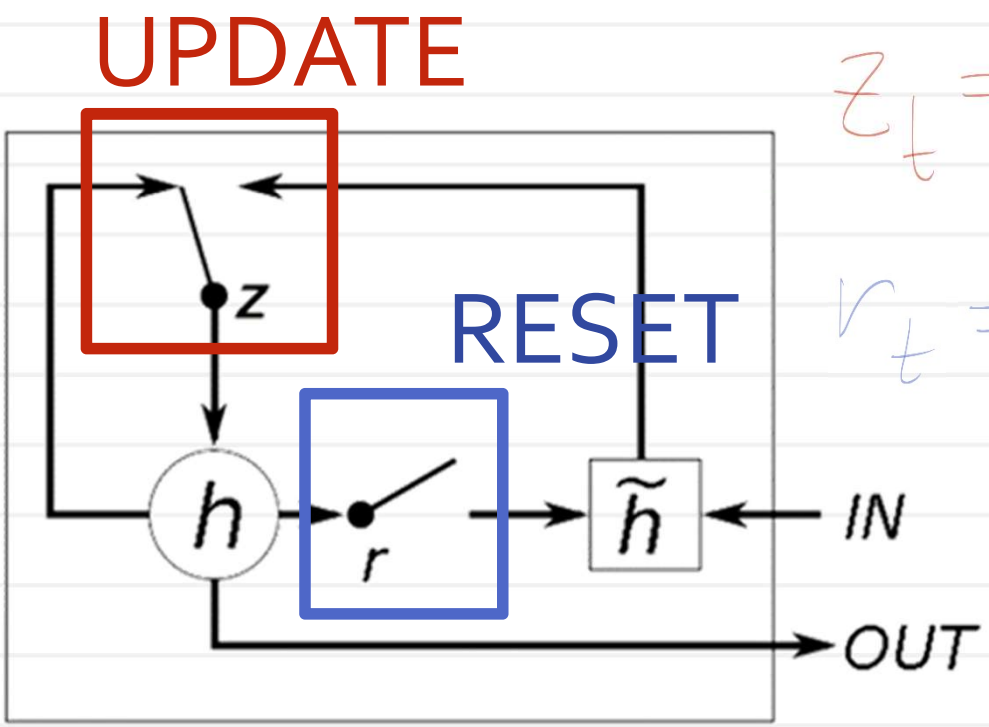
(b) Gated Recurrent Unit

[Cho et al. 14]
[Chung et al. 14]

Gated Recurrent Units (GRU)

$$h_t = (1 - z_t) \odot h_{t-1} +$$

$$+ z_t \odot \phi(W_x x_t + W \cdot (r_t \odot h_{t-1}))$$

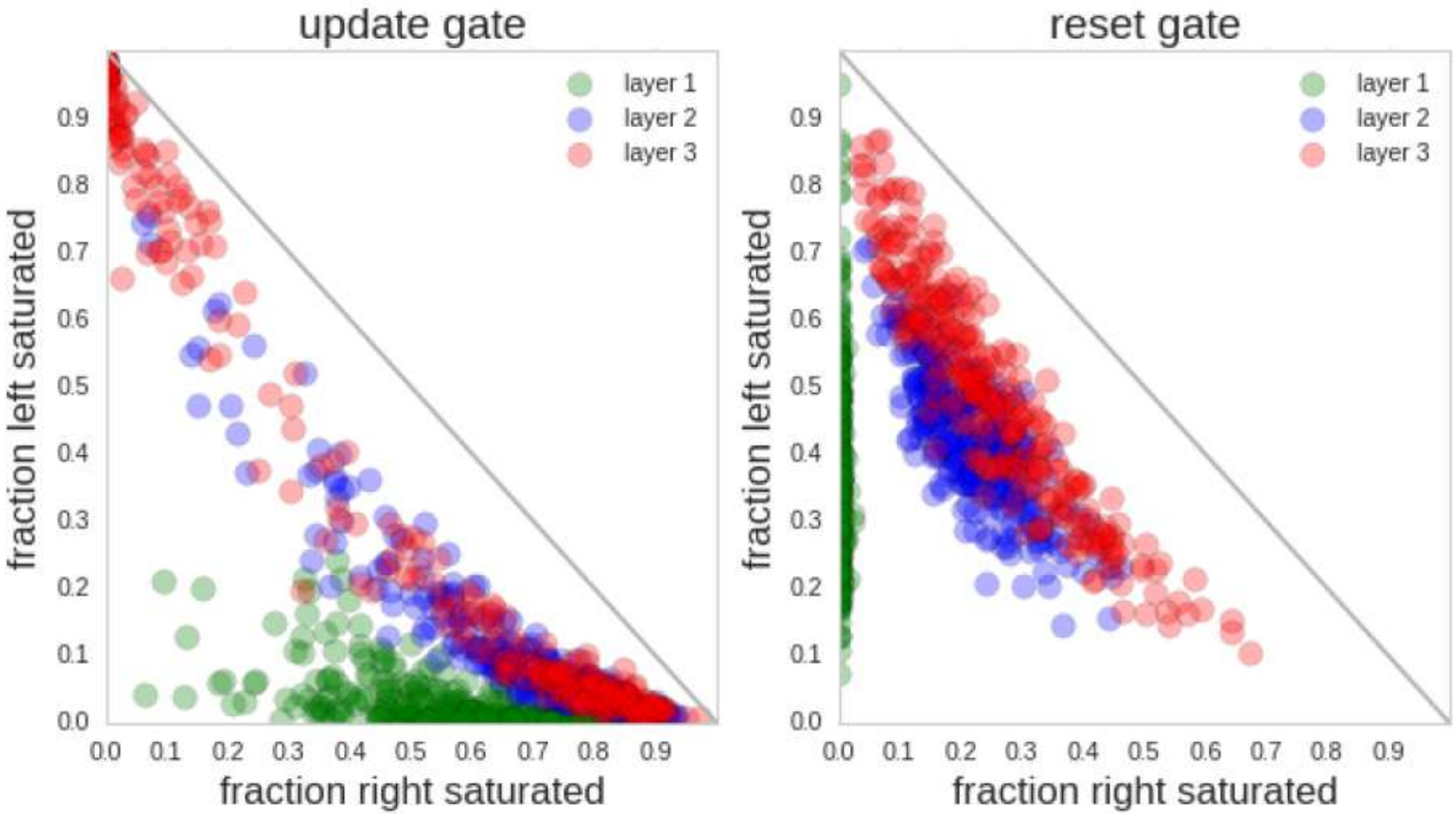


$$z_t = \sigma(W_{xz} x_t + W_{hz} h_{t-1})$$

$$r_t = \sigma(W_{xr} x_t + W_{hr} h_{t-1})$$

[Cho et al. 14]
[Chung et al. 14]

GRU gate statistics



[Karpathy et al. ICLR16]

Plain vs LSTM vs GRU

Success at predicting next characters in the test sequence (cross-entropy loss) [Karpathy et al. ICLR16]:

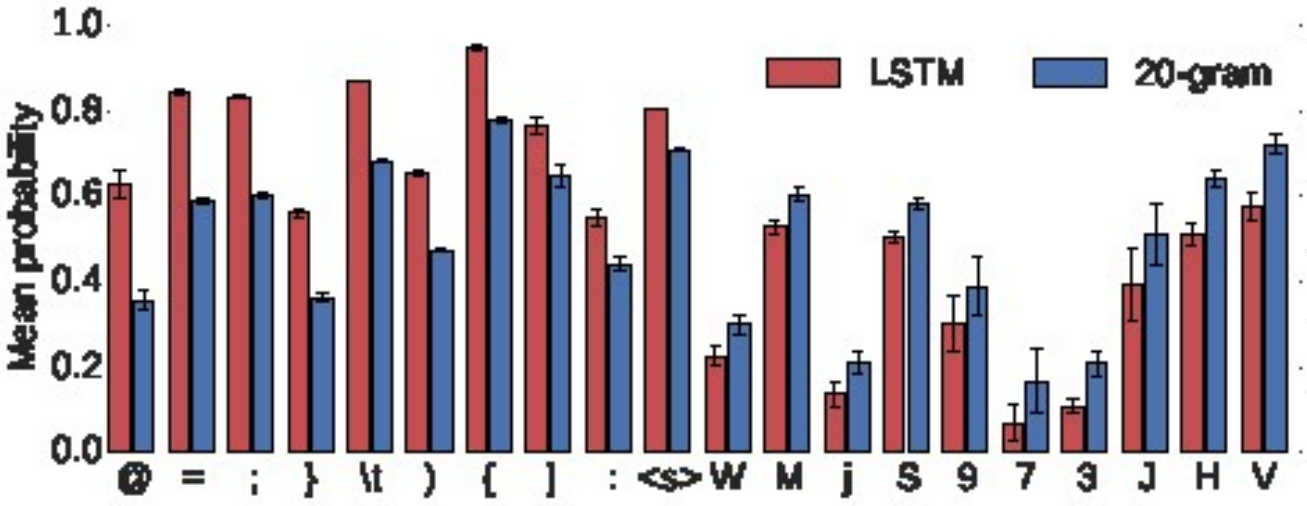
	LSTM			RNN			GRU		
Layers	1	2	3	1	2	3	1	2	3
Size	War and Peace Dataset								
64	1.449	1.442	1.540	1.446	1.401	1.396	1.398	**1.373**	1.472
128	1.277	1.227	1.279	1.417	1.286	1.277	1.230	**1.226**	1.253
256	1.189	**1.137**	1.141	1.342	1.256	1.239	1.198	1.164	1.138
512	1.161	1.092	1.082	-	-	-	1.170	1.201	**1.077**
	Linux Kernel Dataset								
64	1.355	**1.331**	1.366	1.407	1.371	1.383	1.335	1.298	1.357
128	1.149	1.128	1.177	1.241	**1.120**	1.220	1.154	1.125	1.150
256	1.026	**0.972**	0.998	1.171	1.116	1.116	1.039	0.991	1.026
512	0.952	0.840	0.846	-	-	-	0.943	0.861	**0.829**

Model \ n	1	2	3	4	5	6	7	8	9	20
	War and Peace Dataset									
n-gram	2.399	1.928	1.521	1.314	1.232	1.203	**1.194**	1.194	1.194	1.195
n-NN	2.399	1.931	1.553	1.451	1.339	**1.321**	-	-	-	-
	Linux Kernel Dataset									
n-gram	2.702	1.954	1.440	1.213	1.097	1.027	0.982	0.953	0.933	**0.889**
n-NN	2.707	1.974	1.505	1.395	**1.256**	1.376	-	-	-	-

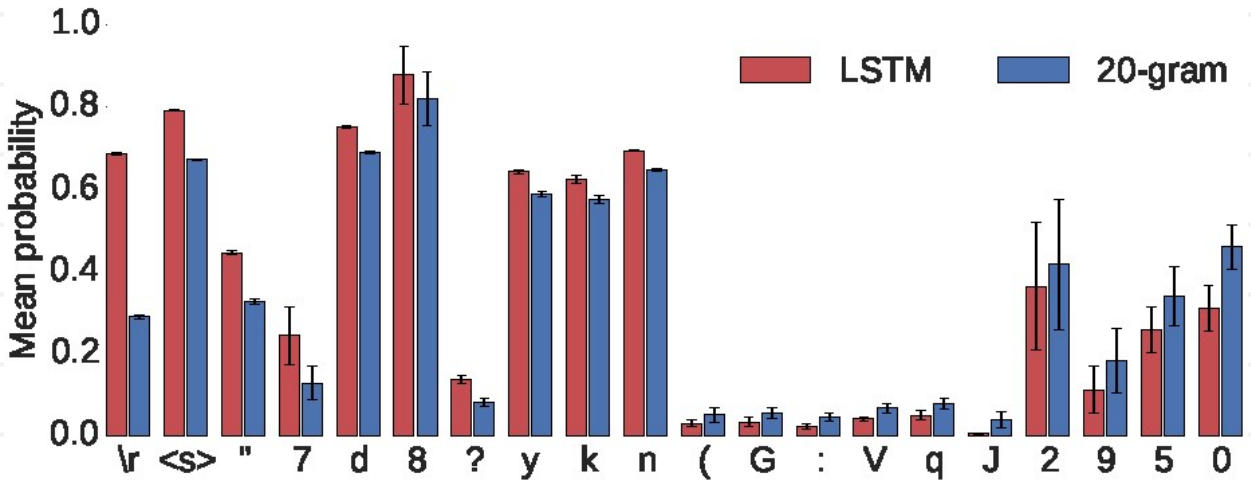
RNN success cases

Success at predicting next characters in the test sequence (cross-entropy loss) [Karpathy et al. ICLR16]:

Linux kernel:

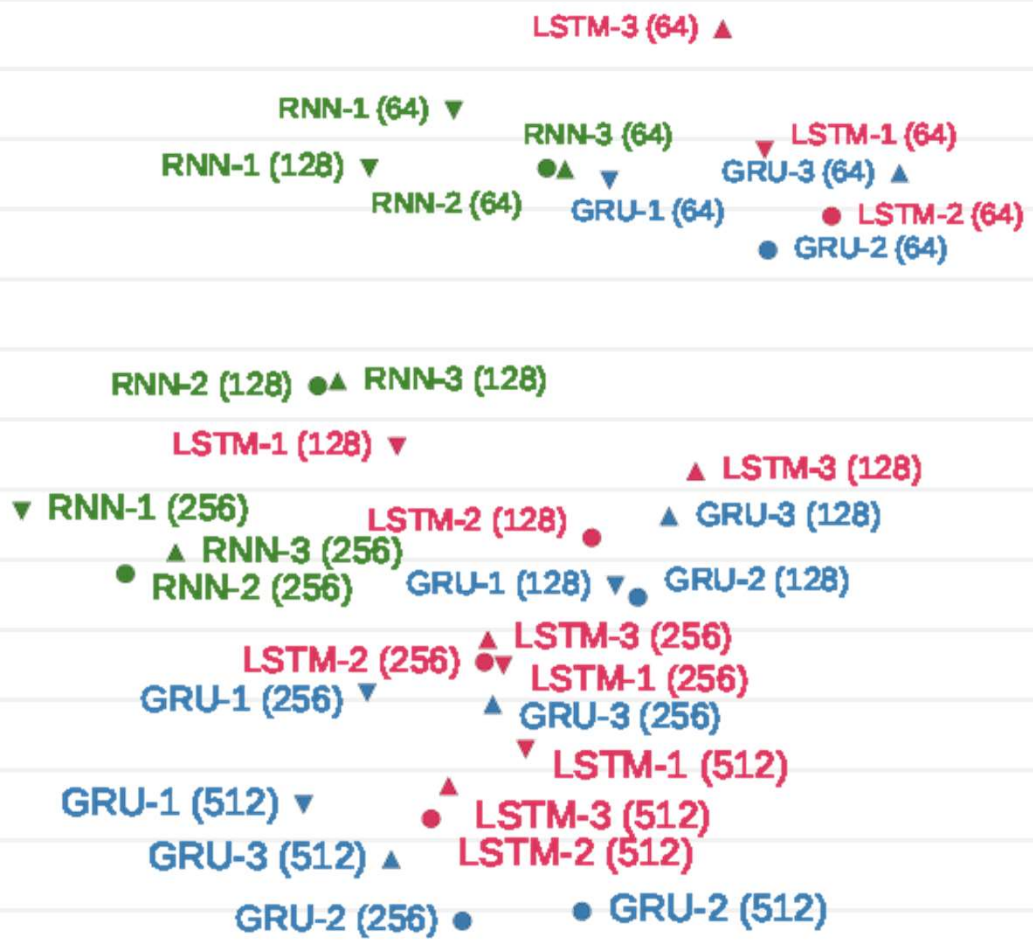


War and peace:



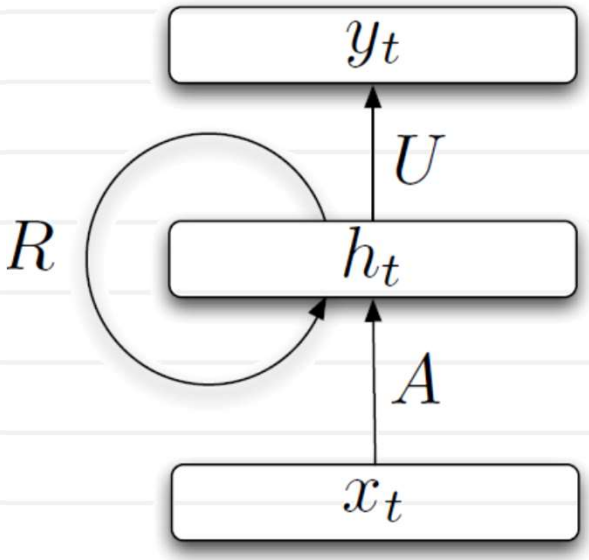
Plain vs LSTM vs GRU

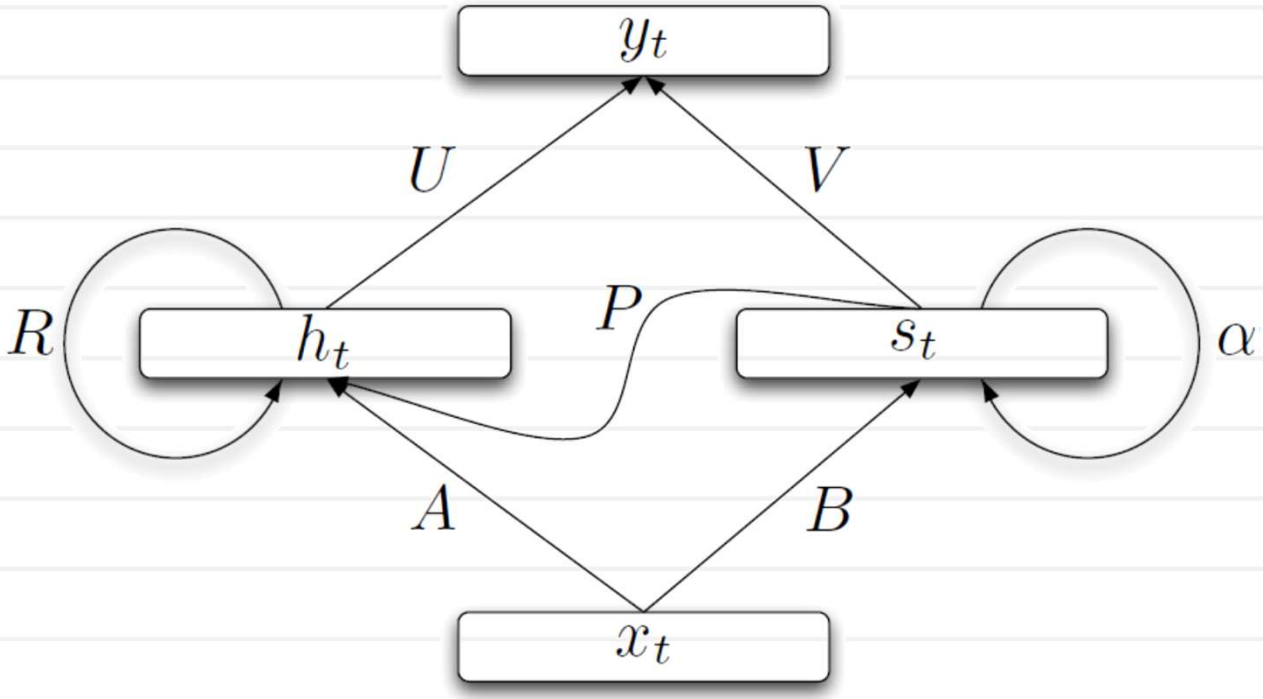
Model similarity (t-SNE embedding of character probabilities):



[Karpathy et al. ICLR16]

Structurally constrained RN





$$\left[\begin{array}{c|c} R & P \\ \hline 0 & \alpha I_p \end{array}\right]$$

$$\begin{aligned} s_t &= (1-\alpha)Bx_t + \alpha s_{t-1}, \\ h_t &= \sigma\left(Ps_t + Ax_t + Rh_{t-1}\right), \\ y_t &= f\left(Uh_t + Vs_t\right) \end{aligned}$$

[Mikolov et al. 15]

Structurally constrained RN

Model	#hidden	#context	Validation Perplexity	Test Perplexity
Ngram	-	-	-	141
Ngram + cache	-	-	-	125
SRN	50	-	153	144
SRN	100	-	137	129
SRN	300	-	133	129
LSTM	50	-	129	123
LSTM	100	-	**120**	**115**
LSTM	300	-	123	119
SCRN	40	10	133	127
SCRN	90	10	124	119
SCRN	100	40	**120**	**115**
SCRN	300	40	**120**	**115**

- Penn-bank word prediction
- For the other dataset (Text8), LSTM is better

[Mikolov et al. 15]

Recap and outlook

- Sequence prediction
- Recurrent nets are SOA in wide variety of domains (NLP, speech/signal, bioinformatics)
- Gating in RNN makes its memory longer
- Sequence prediction extends to other tasks (fixed->seq, seq->fixed, seq->seq, fixed->fixed) – *next lecture*

Bibliography

Elman, Jeffrey L. Finding structure in time. Cognitive science, 14(2):179–211, 1990.

A. Karpathy
The Unreasonable Effectiveness of Recurrent Neural Networks
http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Razvan Pascanu, Tomas Mikolov, Yoshua Bengio:
On the difficulty of training recurrent neural networks. ICML (3) 2013: 1310-1318

A. Graves. Supervised Sequence Labelling with Recurrent Neural Networks. Textbook, Studies in Computational Intelligence, Springer, 2012

Sepp Hochreiter, Jürgen Schmidhuber:
Long Short-Term Memory. Neural Computation 9(8): 1735-1780 (1997)

Bibliography

Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, Yoshua Bengio: Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. EMNLP 2014: 1724-1734

Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, Yoshua Bengio: Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. CoRRabs/1412.3555 (2014)

Andrej Karpathy, Justin Johnson, Fei-Fei Li:
Visualizing and Understanding Recurrent Networks. ICLR 2016

Mikolov, T., Joulin, A., Chopra, S., Mathieu, M., & Ranzato, M. A. (2014). Learning longer memory in recurrent neural networks. arXiv preprint arXiv:1412.7753.