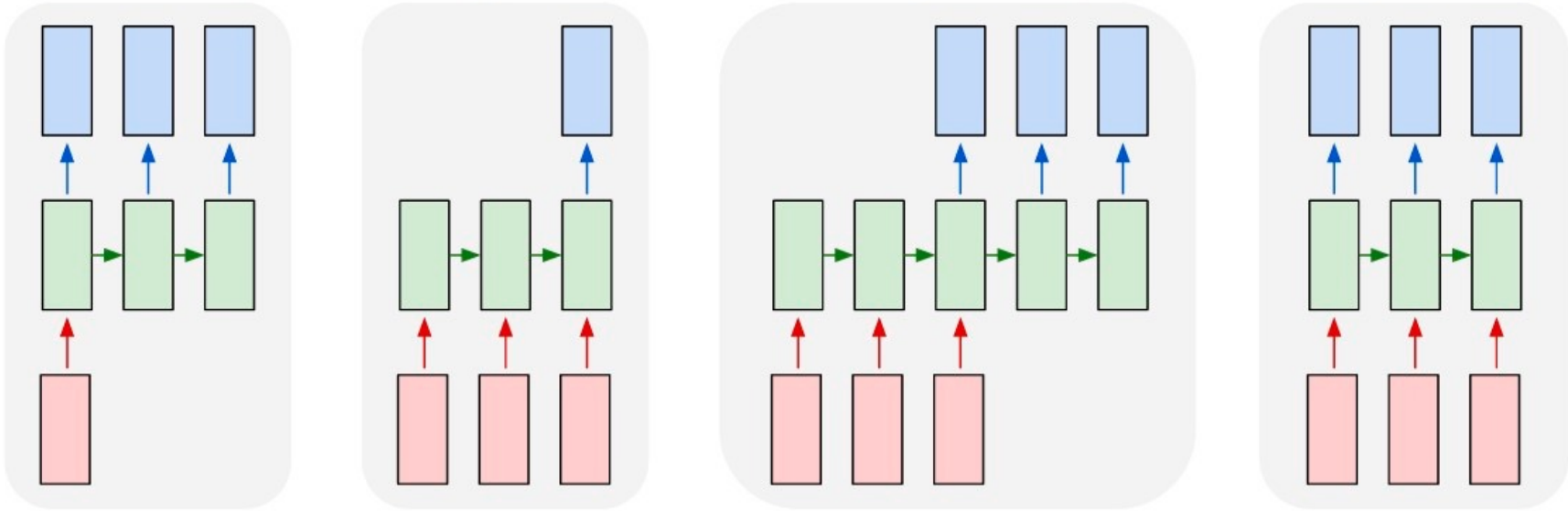
# Recurrent networks



Many slides adapted from Arun Mallya and Justin Johnson (and Stanford CS231n)

Image source

# Outline

- Sequential prediction tasks
- Common recurrent units
  - Vanilla RNN unit (and how to train it)
  - Long Short-Term Memory (LSTM)
  - Gated Recurrent Unit (GRU)
- Recurrent network architectures
- Applications in (a bit) more detail
  - Language modeling
  - Image captioning

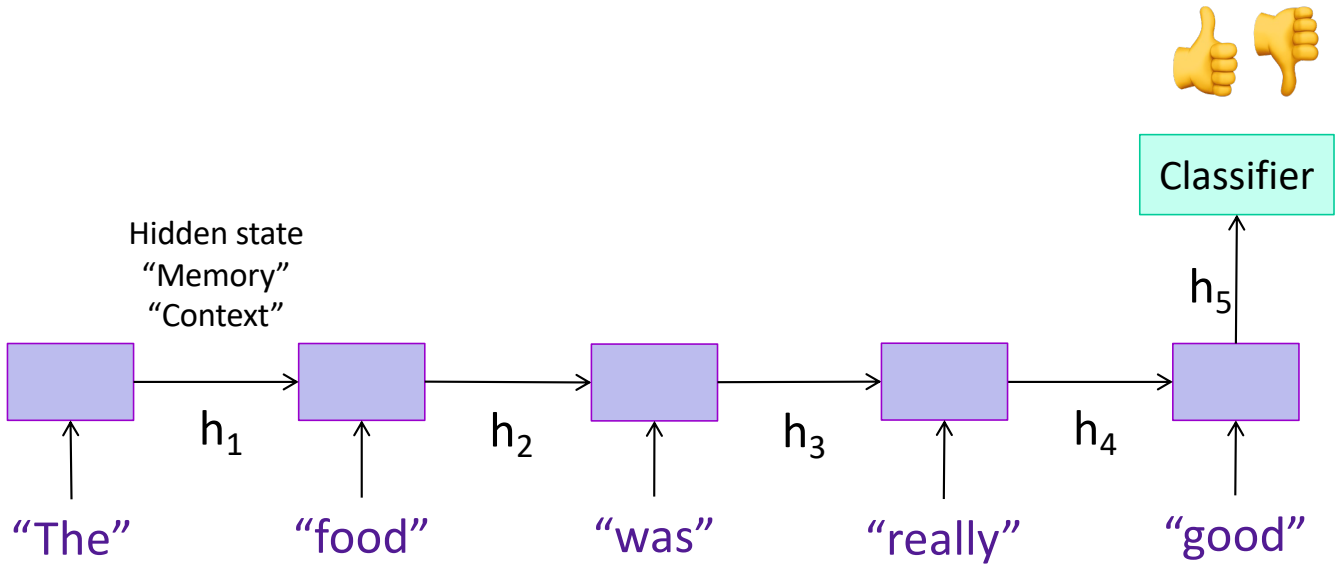
# Sequential prediction example 1: Sentiment classification

- Goal: classify a text sequence (e.g., restaurant, movie or product review, Tweet) as having positive or negative sentiment

  - "The food was really good"
  - "The vacuum cleaner broke within two weeks"
  - "The movie had slow parts, but overall was worth watching"

# Sequential prediction example 1: Sentiment classification

- Recurrent model:



# Sequential prediction example 2: Text generation

- Sample from the distribution of a given text corpus – also known as *language modeling*


# Sequential prediction example 2: Text generation

- Sample from the distribution of a given text corpus – also known as *language modeling*
- Can be done one character or one word at a time:

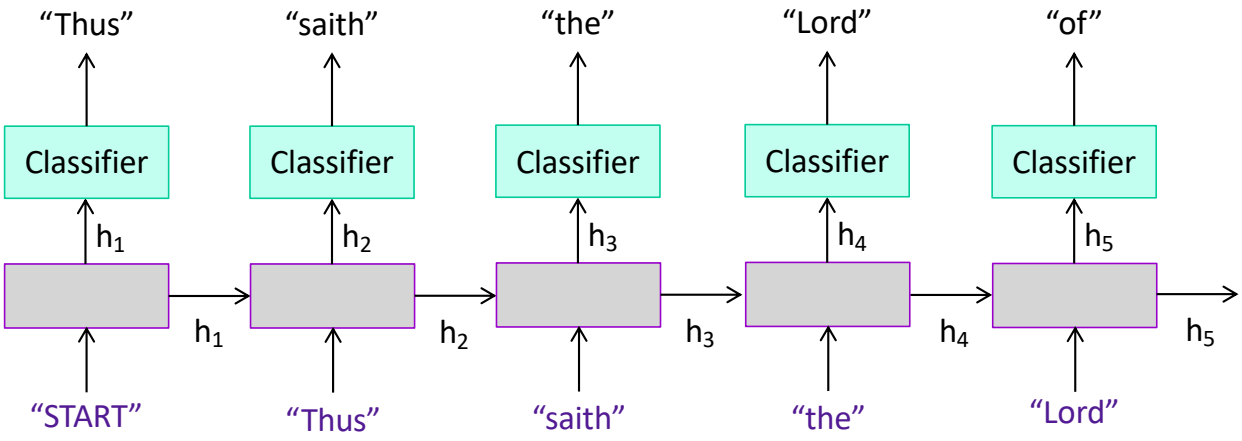



Image source

# Sequential prediction example 3: Image captioning

*A cat sitting on a suitcase on the floor*

*A cat is sitting on a tree branch*

*A dog is running in the grass with a frisbee*

*A white teddy bear sitting in the grass*

*Two people walking on the beach with surfboards*

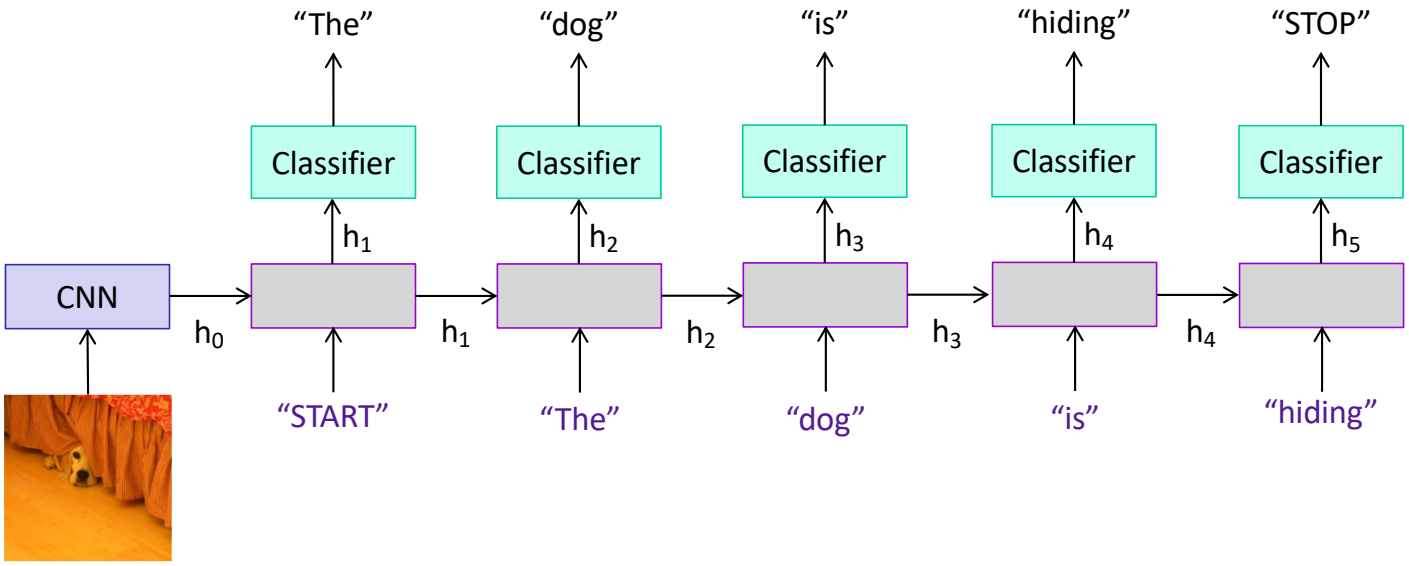

*A tennis player in action on the court*

*Two giraffes standing in a grassy field*

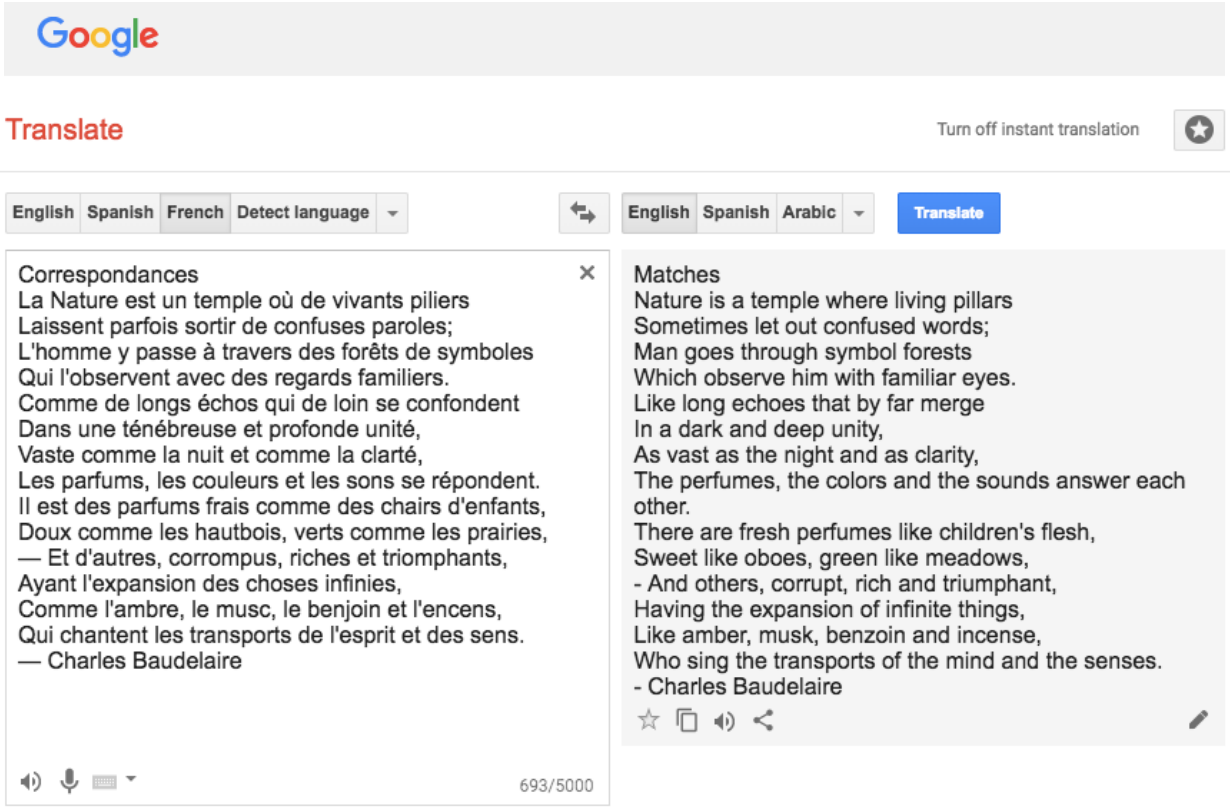
*A man riding a dirt bike on a dirt track*

Source: J. Johnson
Captions generated using neuraltalk2

# Sequential prediction example 3: Image captioning



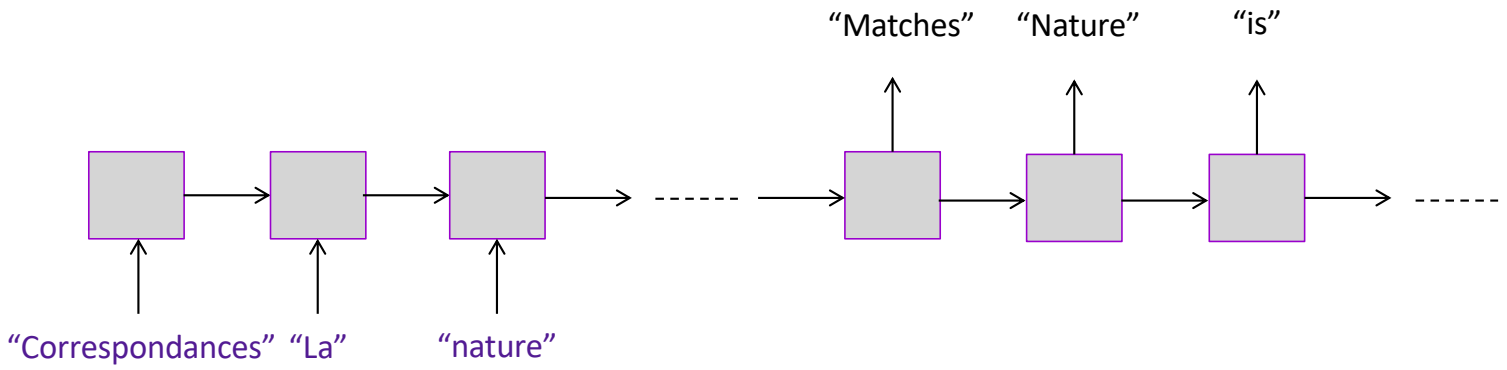
# Example 4: Machine translation



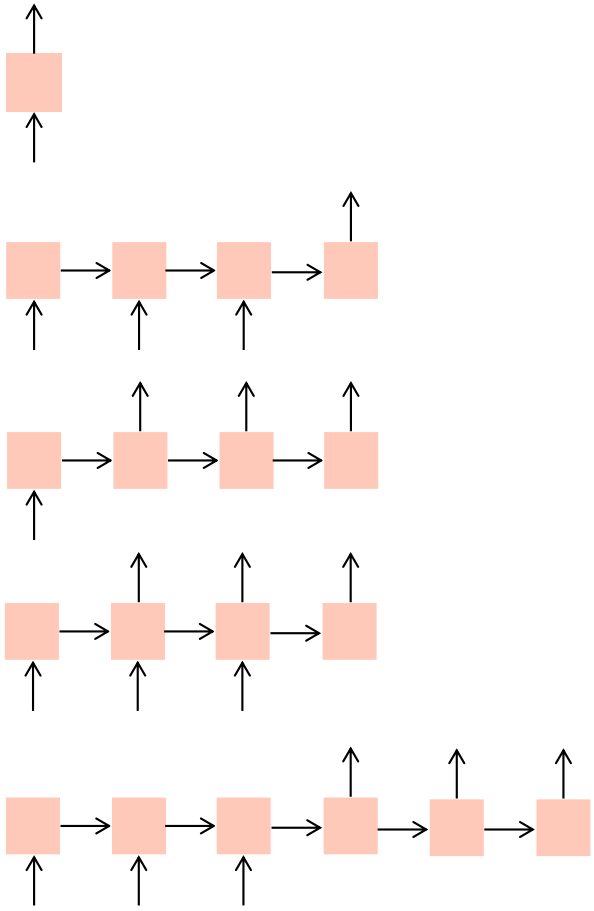

https://translate.google.com/

# Example 4: Machine translation

- Multiple input – multiple output (or sequence to sequence) scenario:



# Summary: Input-output scenarios

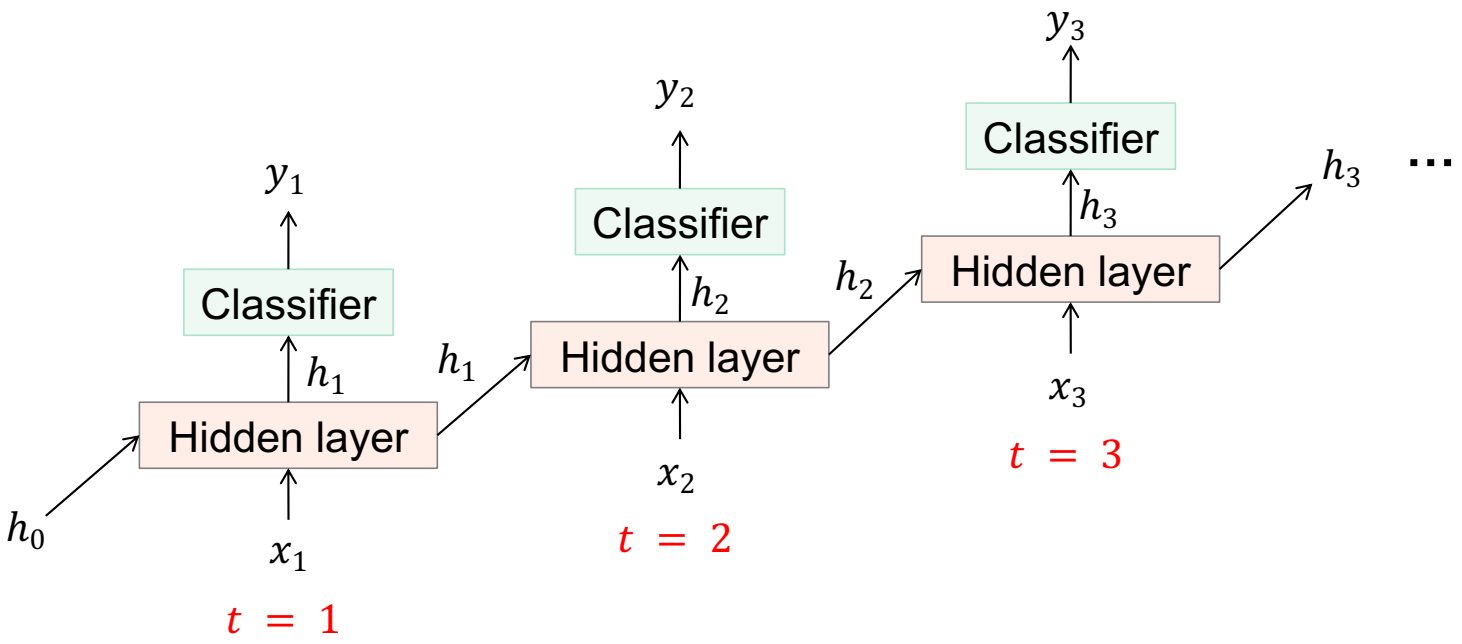


| | |
|---|---|
| Single - Single | Feedforward Network |
| Multiple - Single | Sequence Classification |
| Single - Multiple | Sequence generation, captioning |
| Multiple - Multiple | Sequence generation, captioning |
| Multiple - Multiple | Translation |

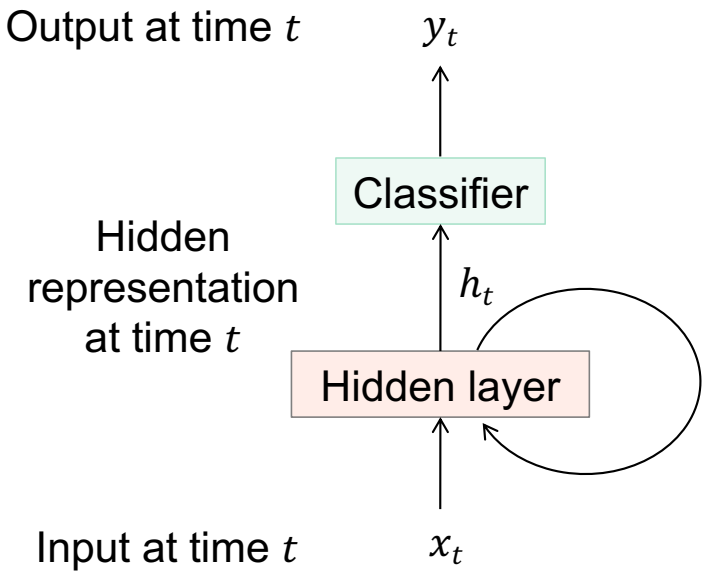
# Outline

- Sequential prediction tasks
- Common recurrent units
  - Vanilla RNN unit
  - Long Short-Term Memory (LSTM)
  - Gated Recurrent Unit (GRU)

# Recurrent unit



# Recurrent unit

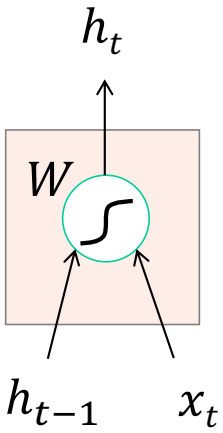



Recurrence:

$$h_t = f_W(x_t, h_{t-1})$$

new state | function of $W$ | input at time $t$ | old state

# Vanilla RNN cell

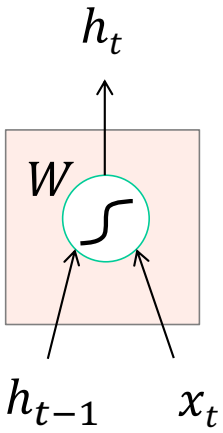



$$h_t = f_W(x_t, h_{t-1})$$

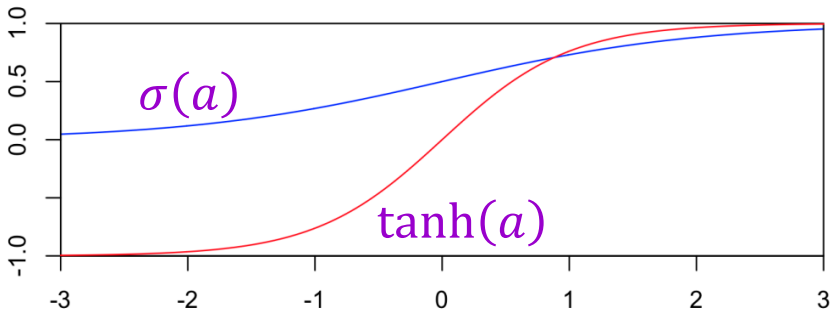
$$= \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

J. Elman, Finding structure in time, Cognitive science 14(2), pp. 179–211, 1990

# Vanilla RNN cell



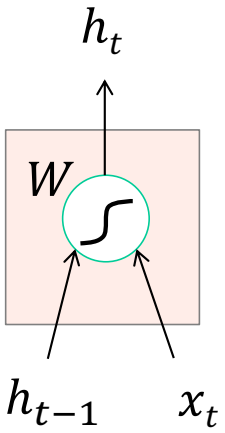

$$h_t = f_W(x_t, h_{t-1})$$
$$= \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$



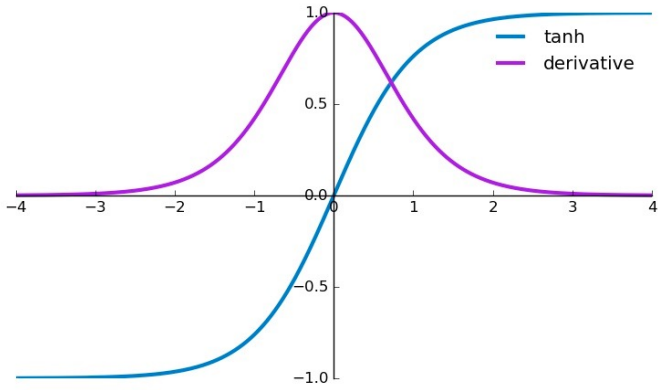

$$\tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}$$
$$= 2\sigma(2a) - 1$$

Image source

# Vanilla RNN cell



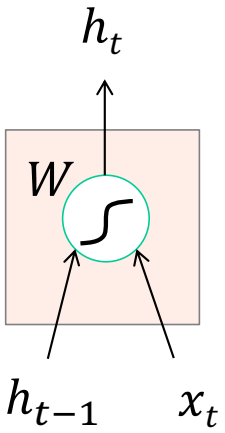

$$h_t = f_W(x_t, h_{t-1})$$
$$= \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$




$$\frac{d}{da}\tanh(a) = 1 - \tanh^2(a)$$

Image source

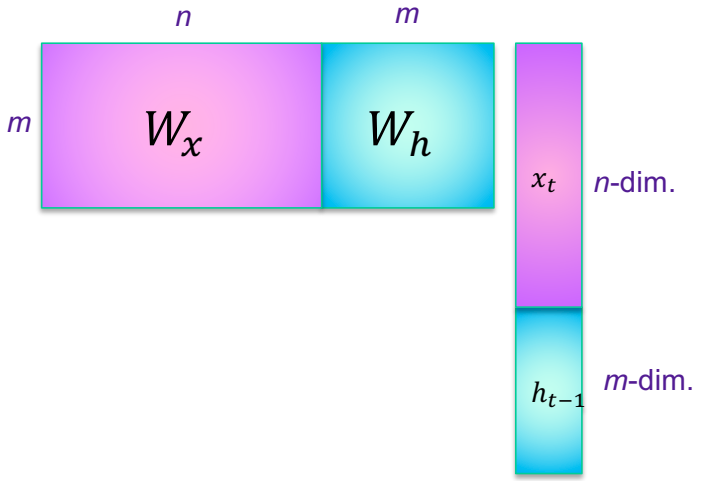
# Vanilla RNN cell



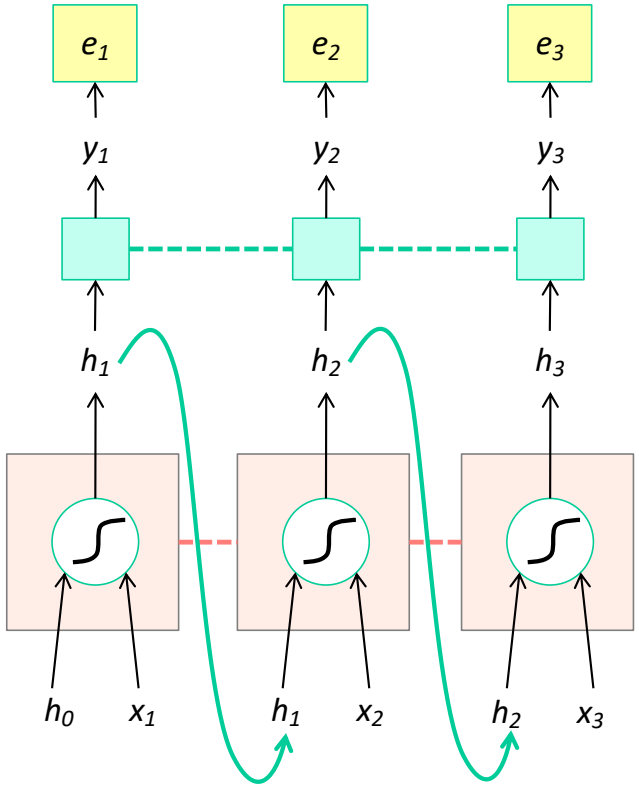

$$h_t = f_W(x_t, h_{t-1})$$

$$= \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

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



# RNN forward pass



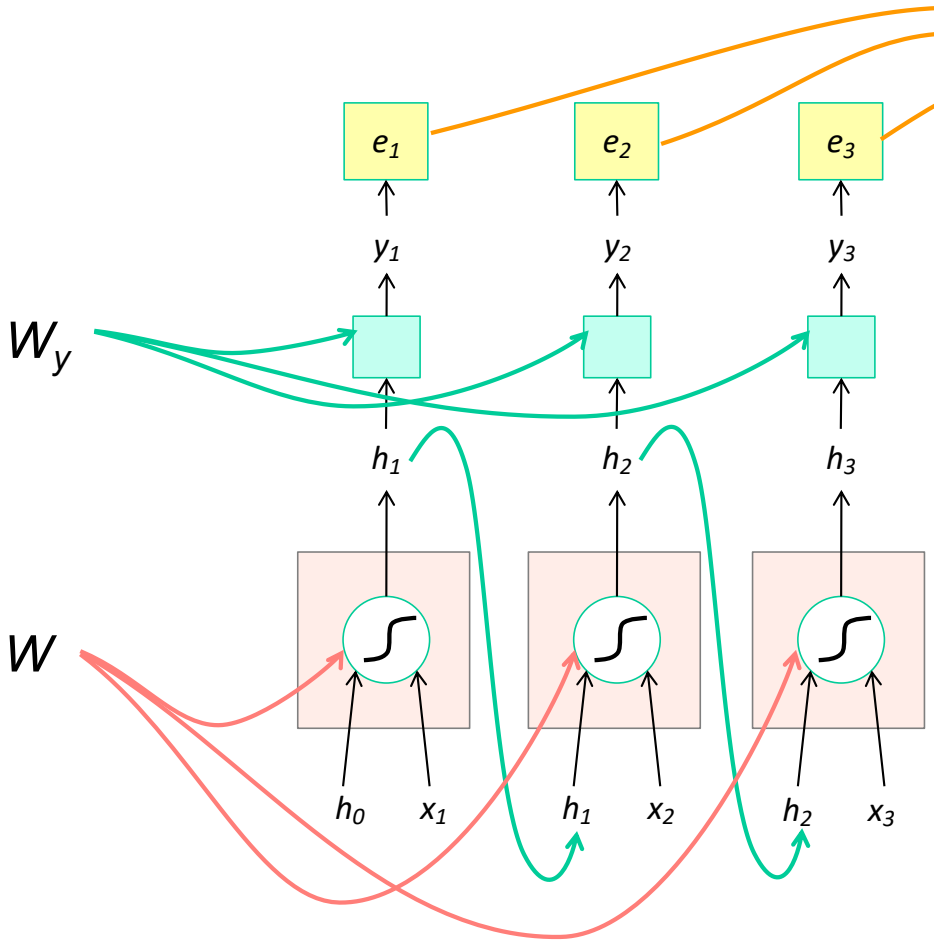

$$e_t = -\log(y_t(GT_t))$$

$$y_t = \text{softmax}(W_y h_t)$$

$$h_t = \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

------ shared weights

# RNN forward pass: Computation graph




$$e_t = -\log(y_t(GT_t))$$

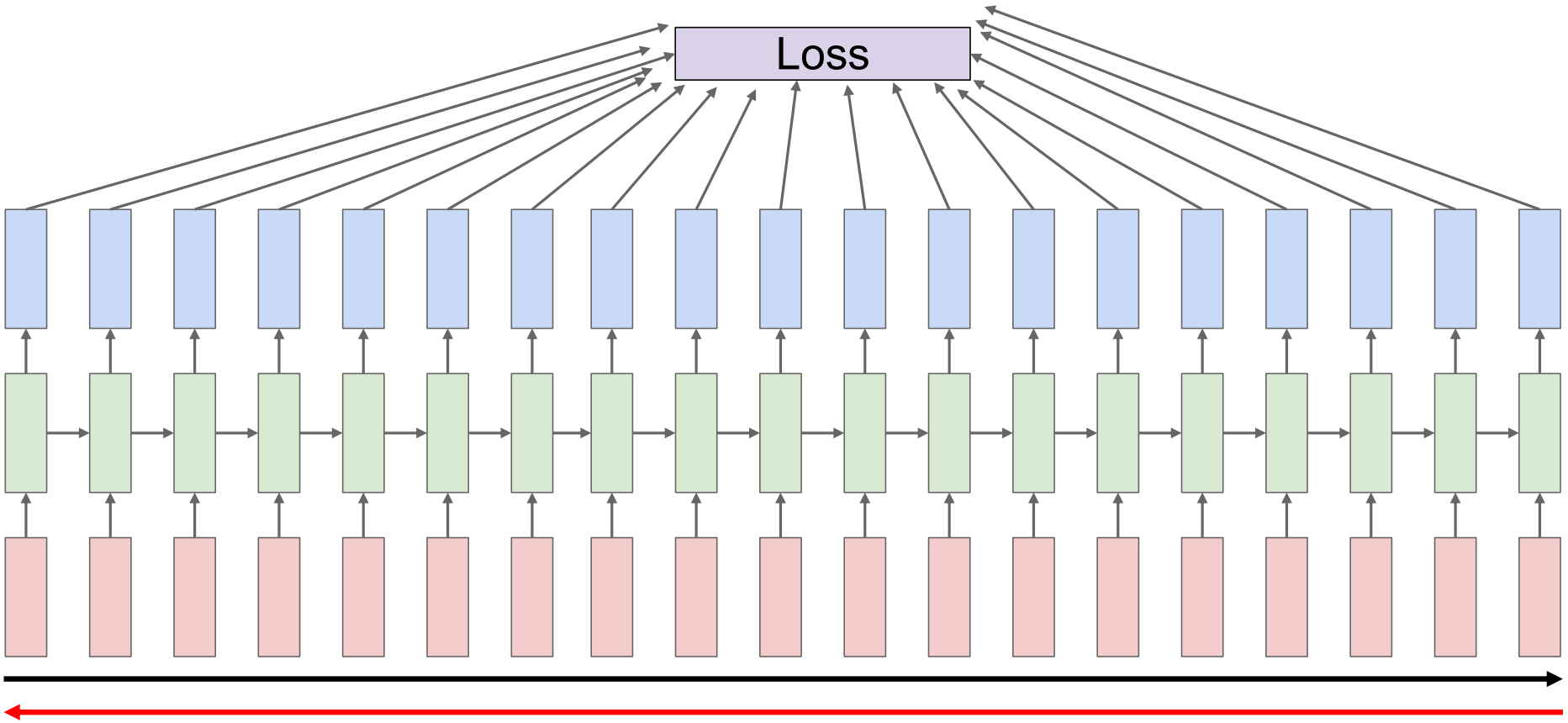
$$y_t = \text{softmax}(W_y h_t)$$

$$h_t = \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

# Training: Backpropagation through time (BPTT)

- The unfolded network (used during forward pass) is treated as one big feed-forward network that accepts the whole time series as input
- The weight updates are computed for each copy in the unfolded network, then summed (or averaged) and applied to the RNN weights

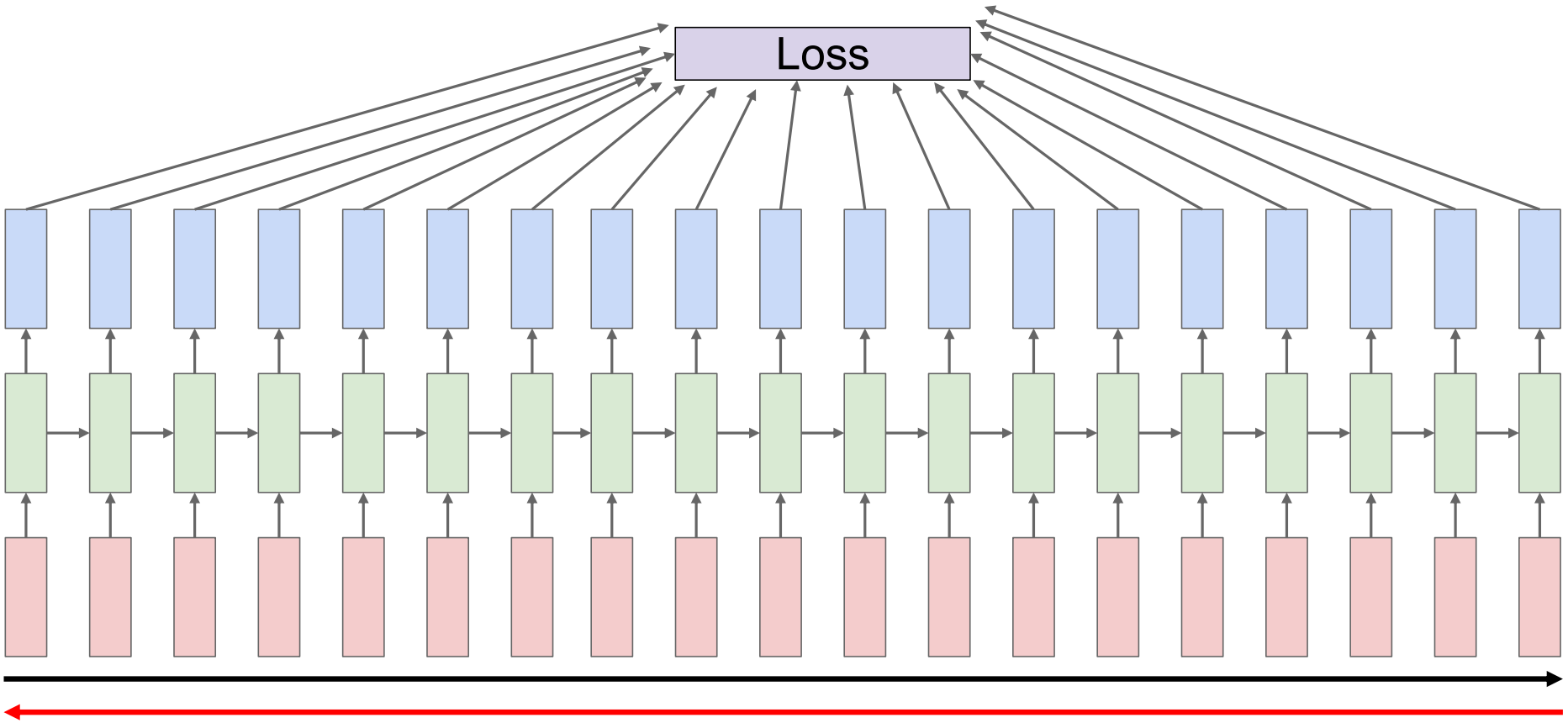
# Backpropagation through time




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

Source: J. Johnson

# Backpropagation through time




Problem: Takes a lot of memory for long sequences!

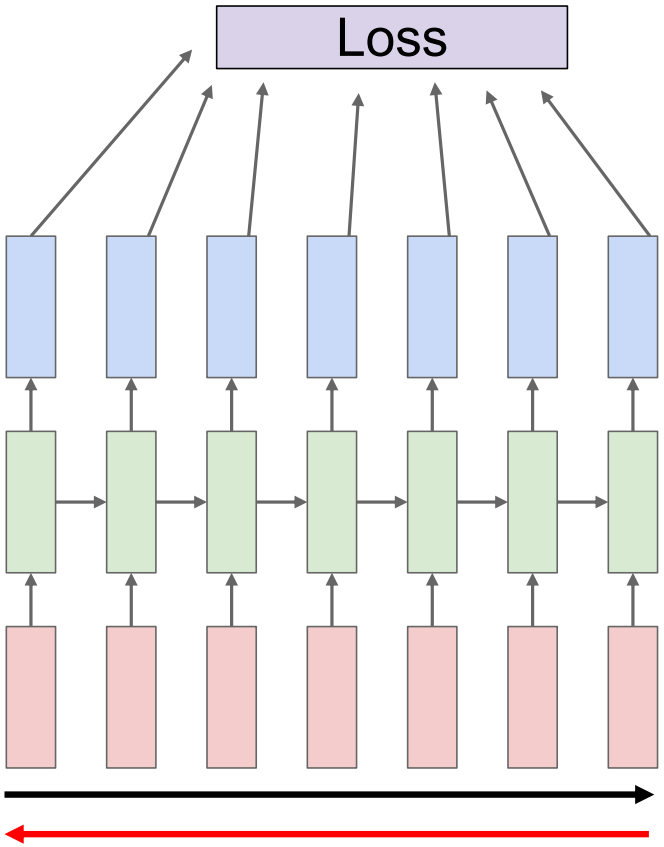
Source: J. Johnson

# Training: Backpropagation through time (BPTT)

- The unfolded network (used during forward pass) is treated as one big feed-forward network that accepts the whole time series as input
- The weight updates are computed for each copy in the unfolded network, then summed (or averaged) and applied to the RNN weights
- In practice, *truncated* BPTT is used: run the RNN forward $k_1$ time steps, propagate backward for $k_2$ time steps

https://machinelearningmastery.com/gentle-introduction-backpropagation-time/
http://www.cs.utoronto.ca/~ilya/pubs/ilya_sutskever_phd_thesis.pdf

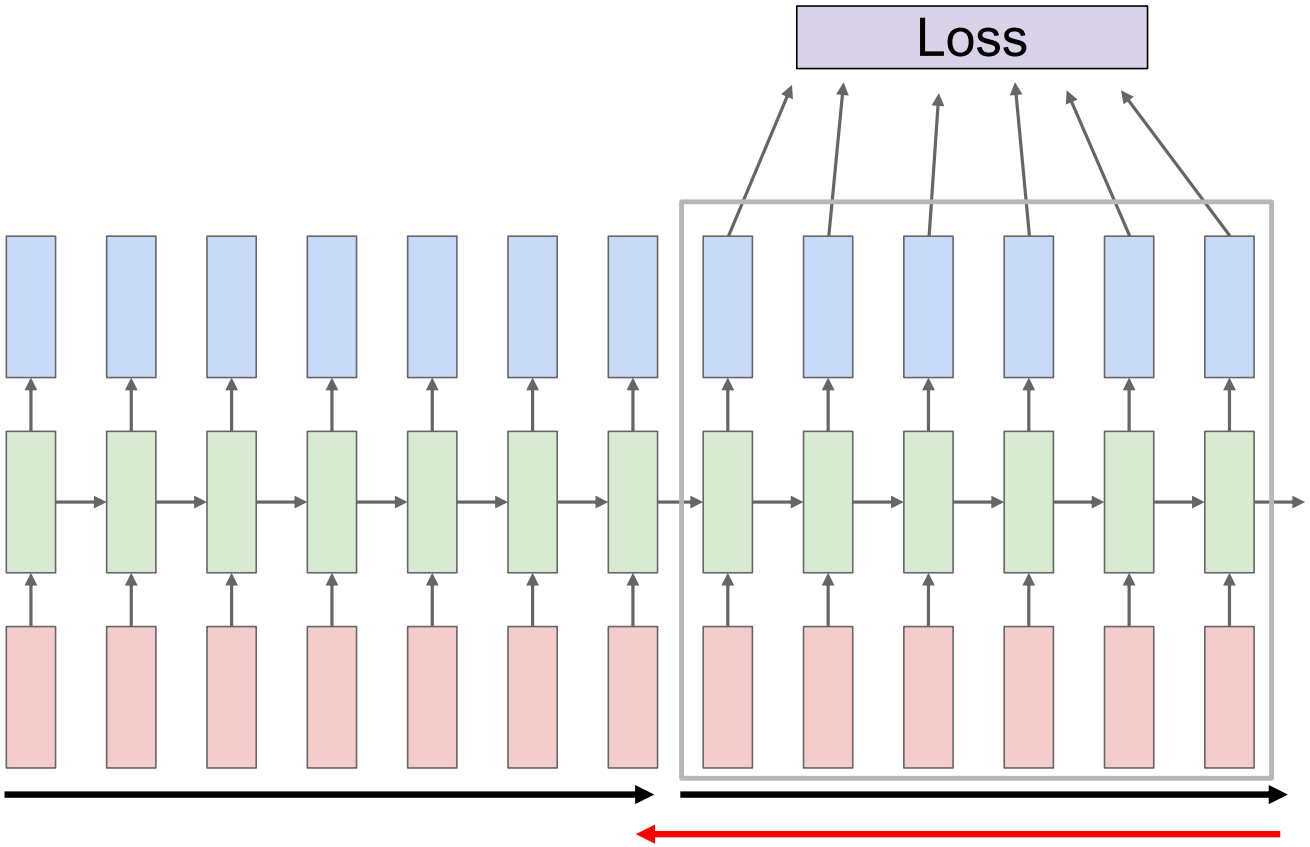
# Truncated backpropagation through time




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

Source: J. Johnson

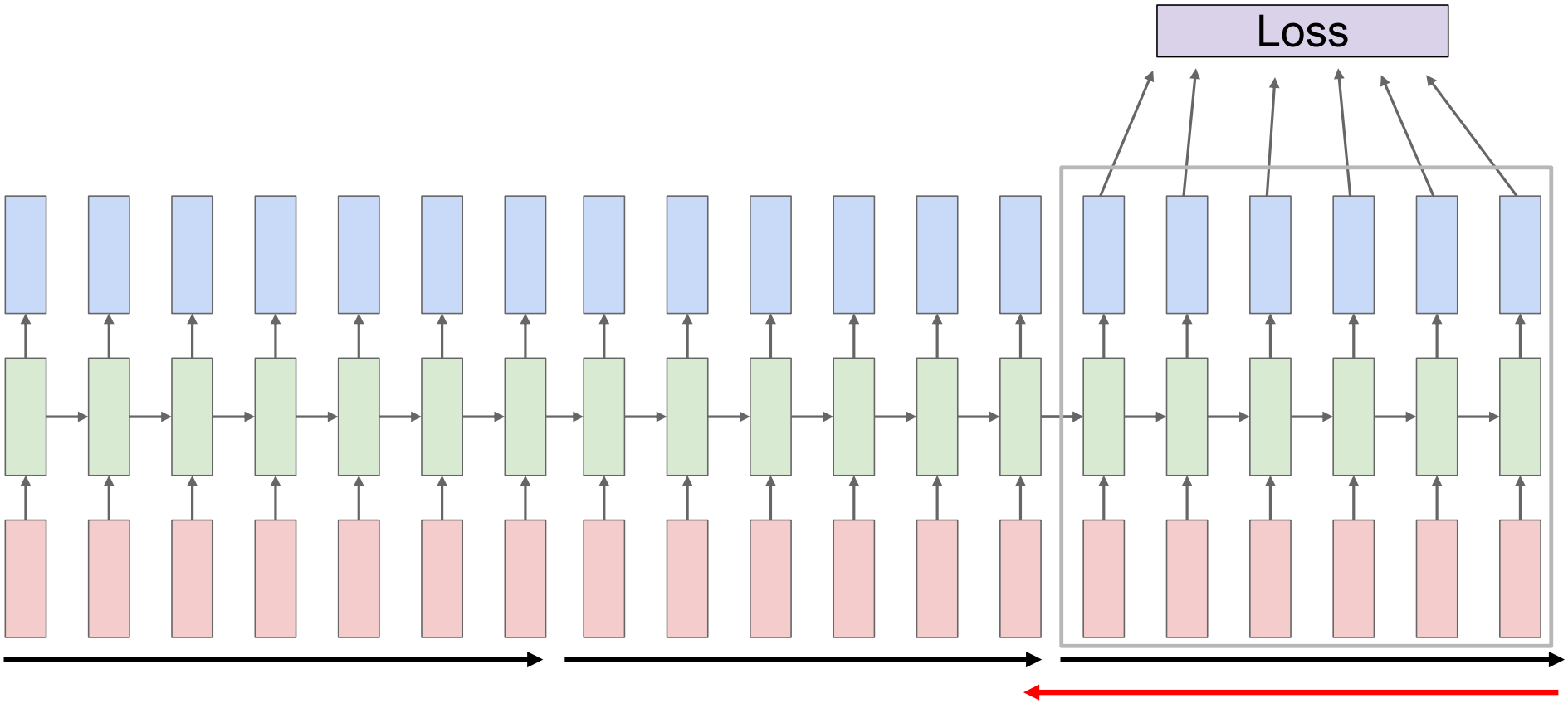
# Truncated backpropagation through time




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

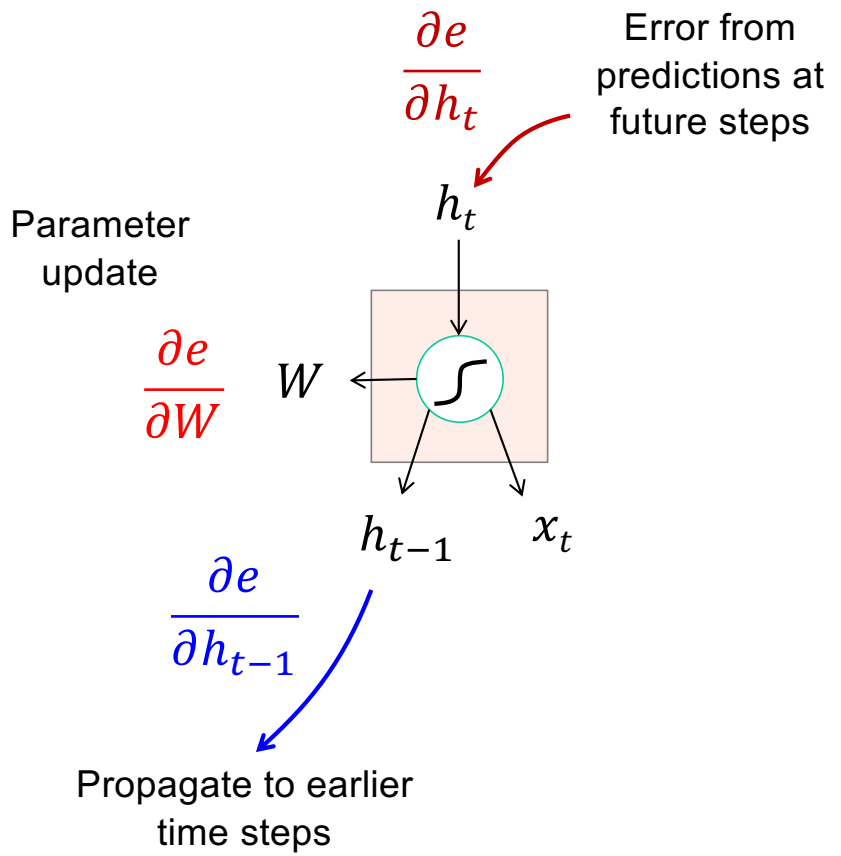
Source: J. Johnson

# Truncated backpropagation through time




Source: J. Johnson

# RNN backward pass



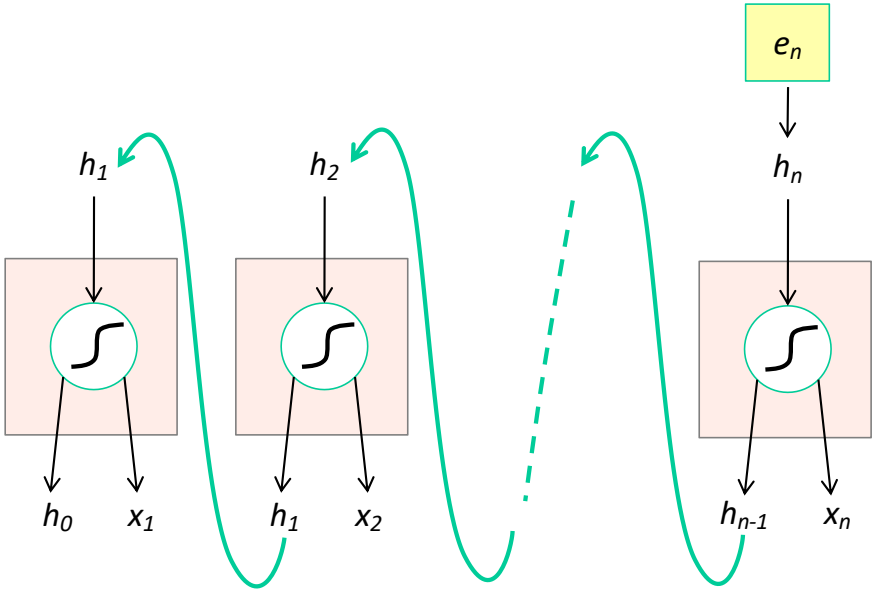

$$h_t = \tanh(W_x x_t + W_h h_{t-1})$$

$$\frac{\partial e}{\partial W_h} = \frac{\partial e}{\partial h_t} \odot \left(1 - \tanh^2(W_x x_t + W_h h_{t-1})\right) h_{t-1}^T$$

$$\frac{\partial e}{\partial W_x} = \frac{\partial e}{\partial h_t} \odot \left(1 - \tanh^2(W_x x_t + W_h h_{t-1})\right) x_t^T$$

$$\frac{\partial e}{\partial h_{t-1}} = W_h^T \left(1 - \tanh^2(W_x x_t + W_h h_{t-1})\right) \odot \frac{\partial e}{\partial h_t}$$

# Vanishing and exploding gradients




$$\frac{\partial e}{\partial h_{t-1}} = W_h^T\left(1 - \tanh^2(W_x x_t + W_h h_{t-1})\right) \odot \frac{\partial e}{\partial h_t}$$

Computing gradient for $h_0$ involves many multiplications by $W_h^T$ and rescalings between 0 and 1

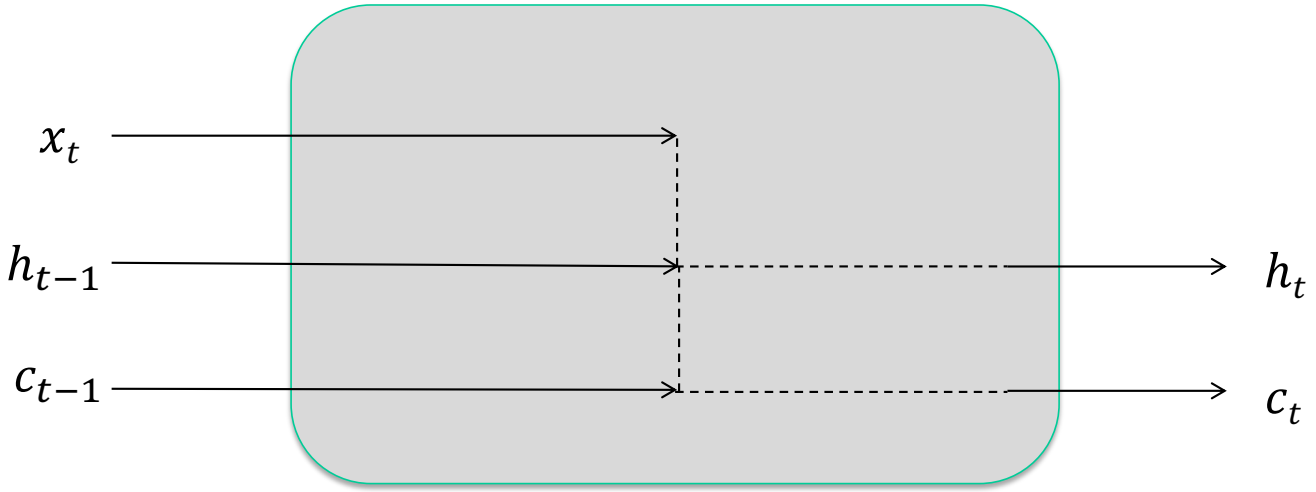
Gradients will *vanish* if largest singular value of $W_h$ is less than 1 and *explode* if it's greater than 1

# Outline

- Examples of sequential prediction tasks
- Common recurrent units
  - Vanilla RNN unit (and how to train it)
  - Long Short-Term Memory (LSTM)
  - Gated Recurrent Unit (GRU)

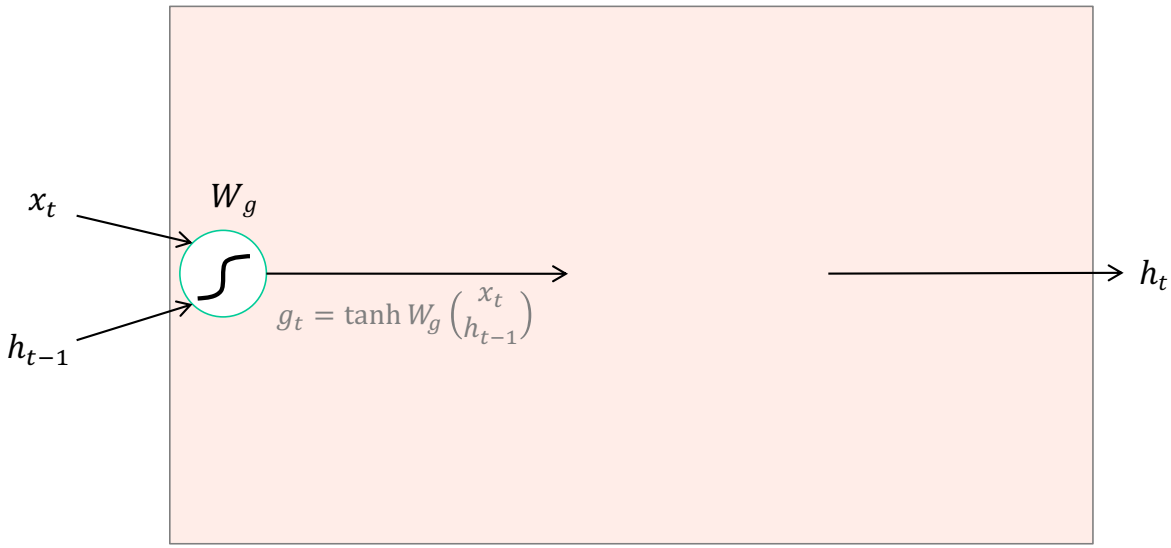
# Long short-term memory (LSTM)

- Add a *memory cell* that is not subject to matrix multiplication or squishing, thereby avoiding gradient decay

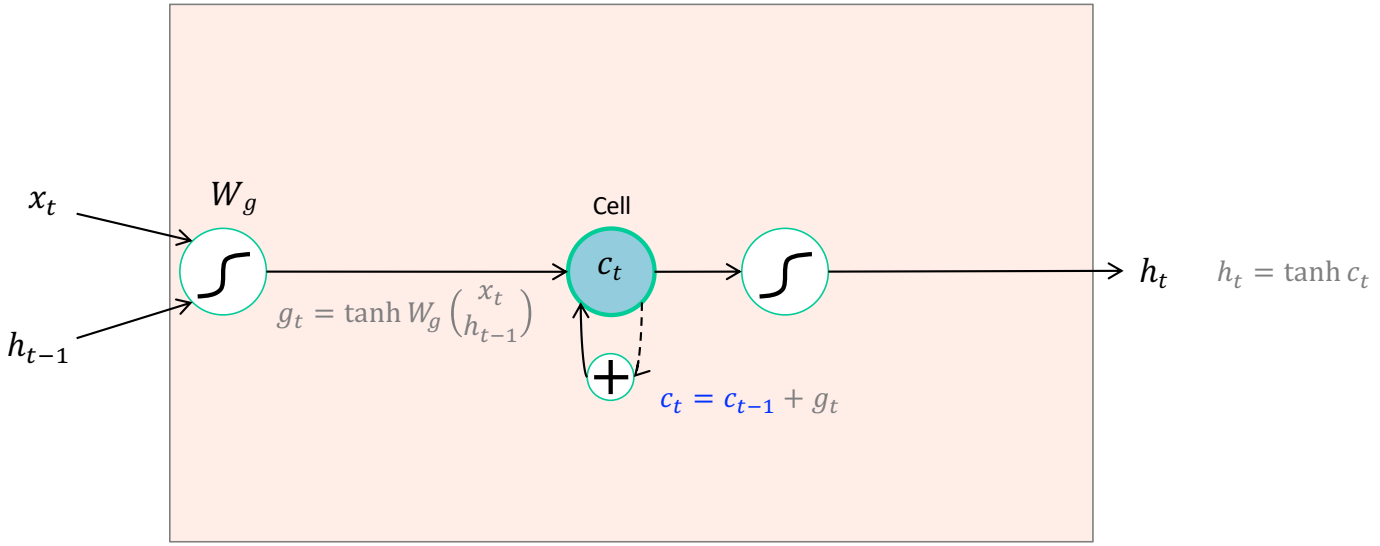



S. Hochreiter and J. Schmidhuber, Long short-term memory, Neural Computation 9 (8), pp. 1735–1780, 1997

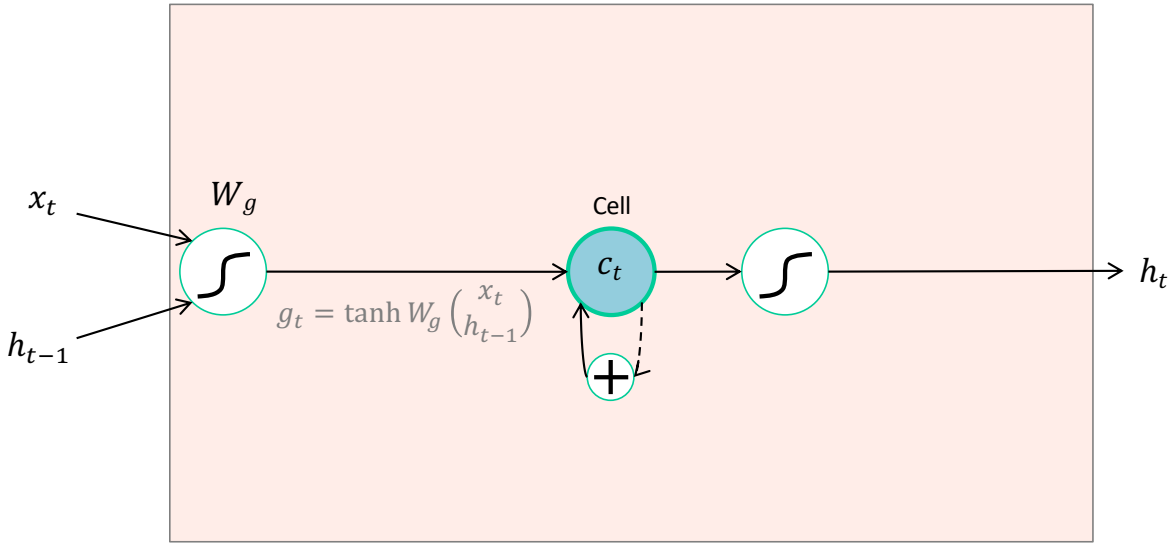
# The LSTM cell



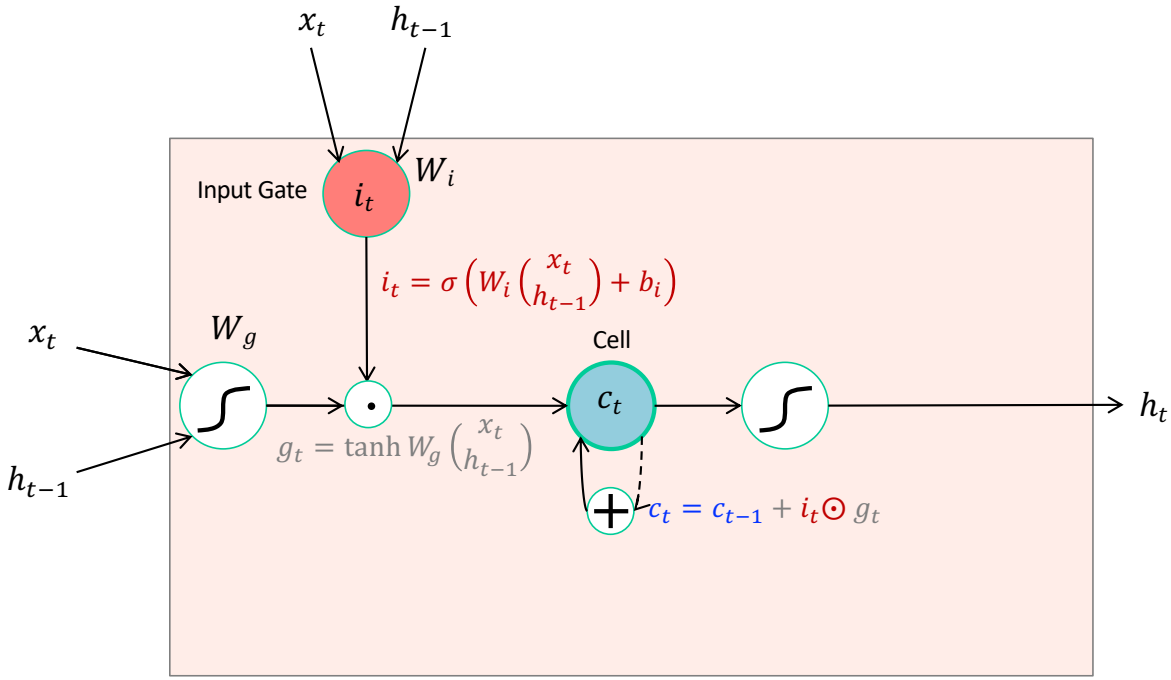
# The LSTM cell



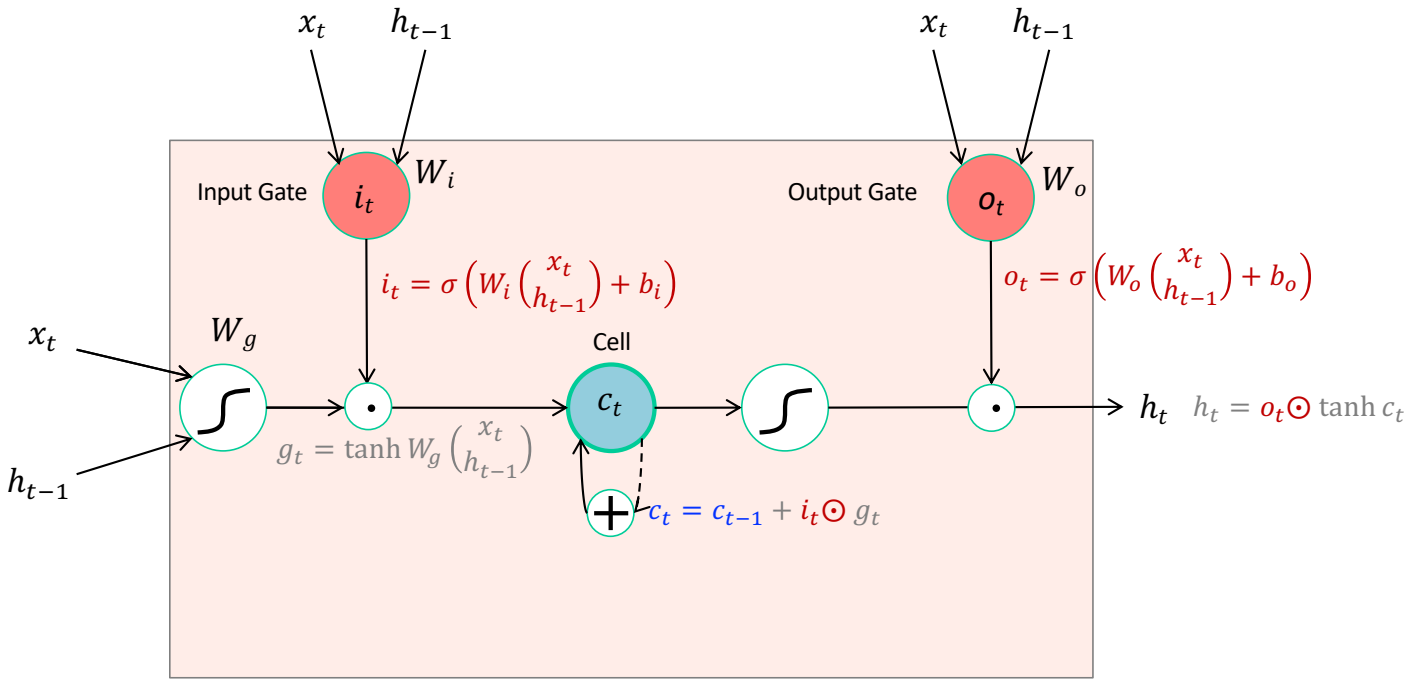
# The LSTM cell



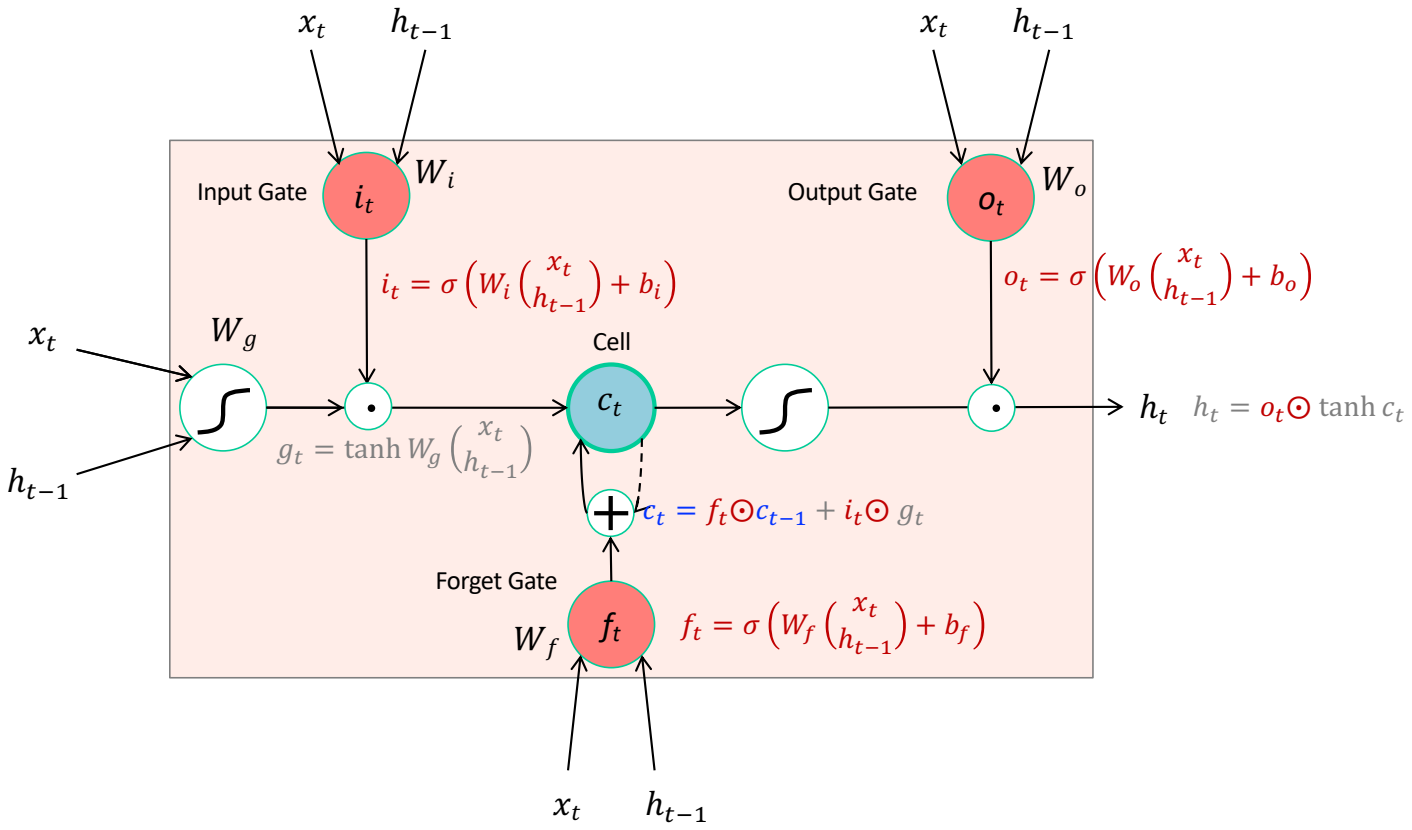
# The LSTM cell



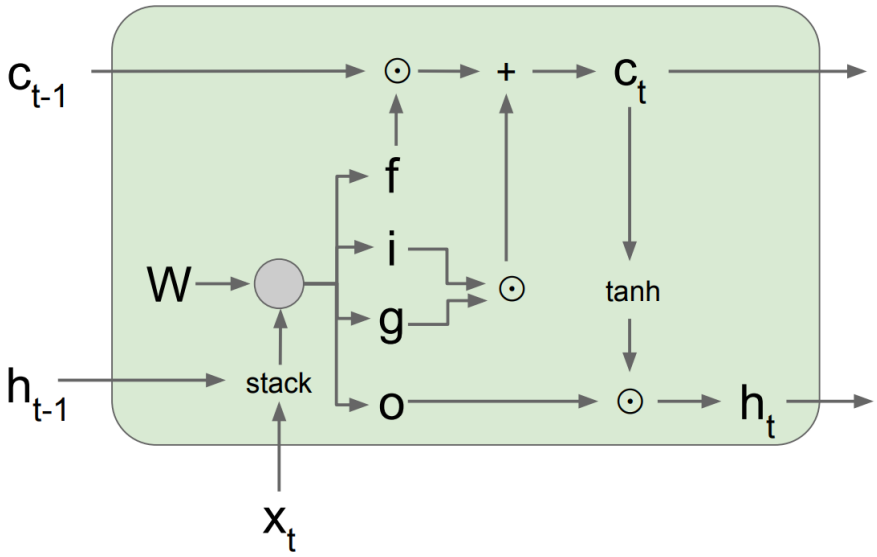
# The LSTM cell



# The LSTM cell



# LSTM forward pass summary



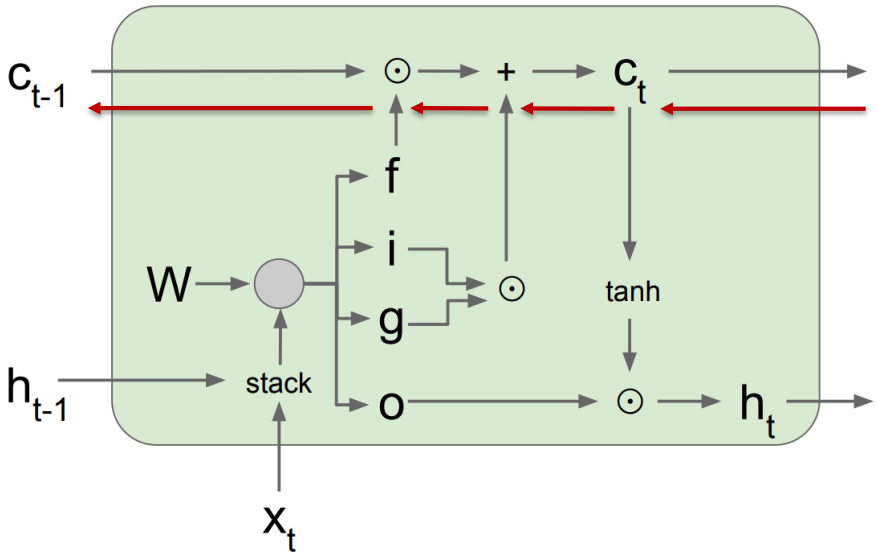

$$\begin{pmatrix} g_t \\ i_t \\ f_t \\ o_t \end{pmatrix} = \begin{pmatrix} \tanh \\ \sigma \\ \sigma \\ \sigma \end{pmatrix} \begin{pmatrix} W_g \\ W_i \\ W_f \\ W_o \end{pmatrix} \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

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

$$h_t = o_t \odot \tanh c_t$$

Figure source

# LSTM backward pass

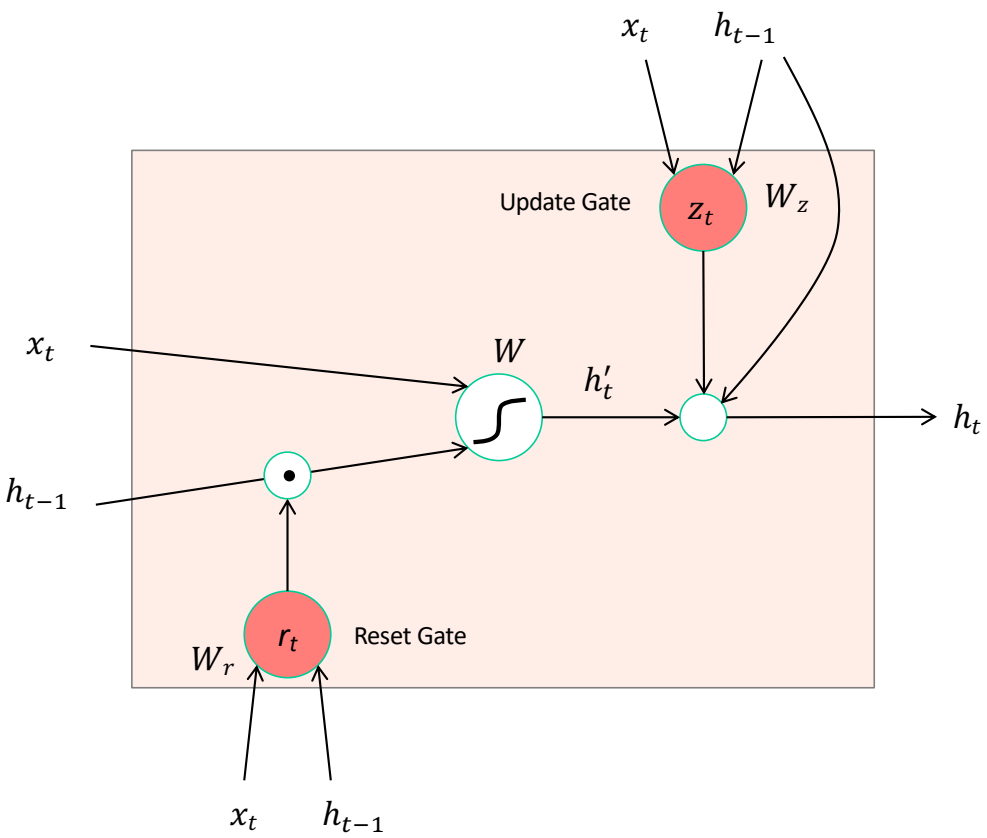



Gradient flow from $c_t$ to $c_{t-1}$ only involves back-propagating through addition and elementwise multiplication, not matrix multiplication or tanh

For complete details: Illustrated LSTM Forward and Backward Pass

Figure source

# LSTM variant: Gated recurrent unit (GRU)




- Get rid of separate cell state
- Merge "forget" and "output" gates into "update" gate

$$r_t = \sigma\left(W_r \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_t\right)$$

$$h'_t = \tanh W \begin{pmatrix} x_t \\ r_t \odot h_{t-1} \end{pmatrix}$$

$$z_t = \sigma\left(W_z \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_z\right)$$

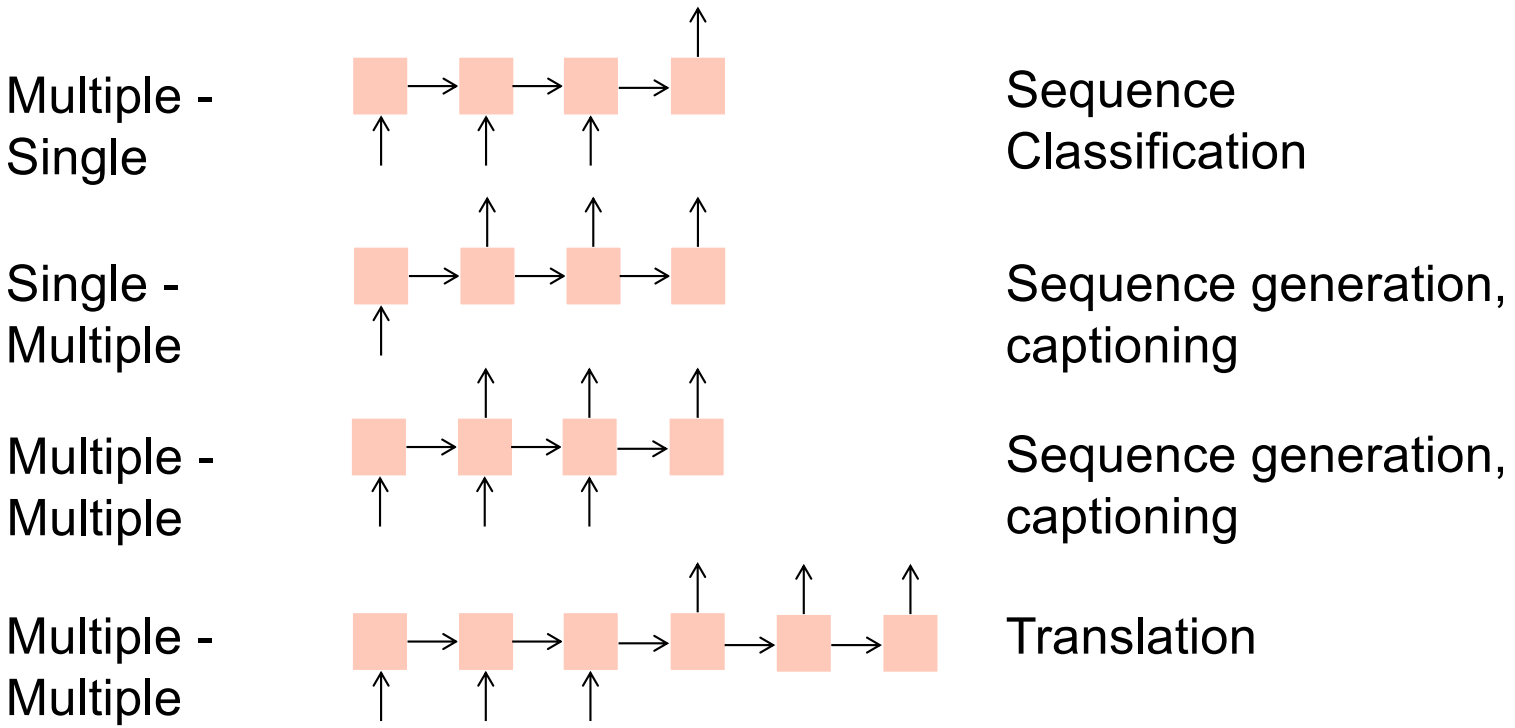
$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot h'_t$$

K. Cho et al., Learning phrase representations using RNN encoder-decoder for statistical machine translation, ACL 2014

# Outline

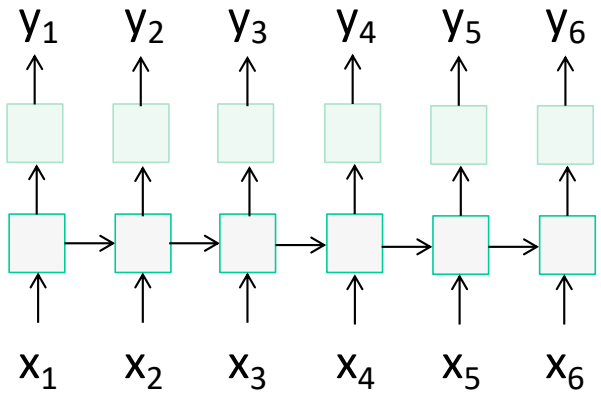
- Examples of sequential prediction tasks
- Common recurrent units
  - Vanilla RNN unit (and how to train it)
  - Long Short-Term Memory (LSTM)
  - Gated Recurrent Unit (GRU)
- Recurrent network architectures

# Recall: Input-output scenarios



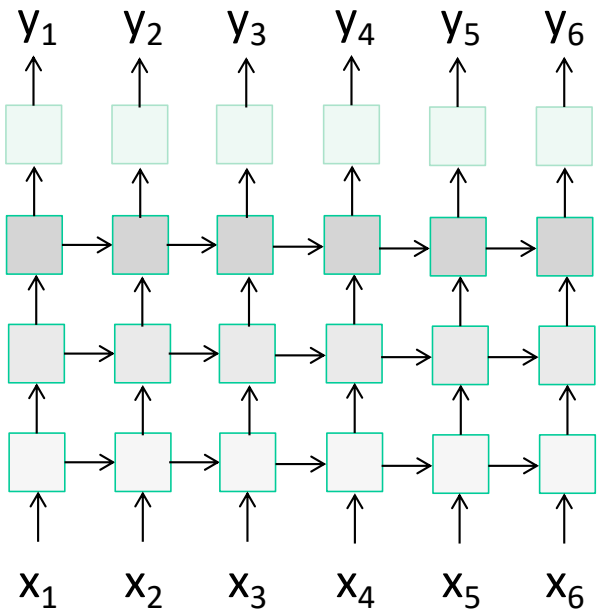
# RNN architectures

- Most general configuration:



# Multi-layer RNNs

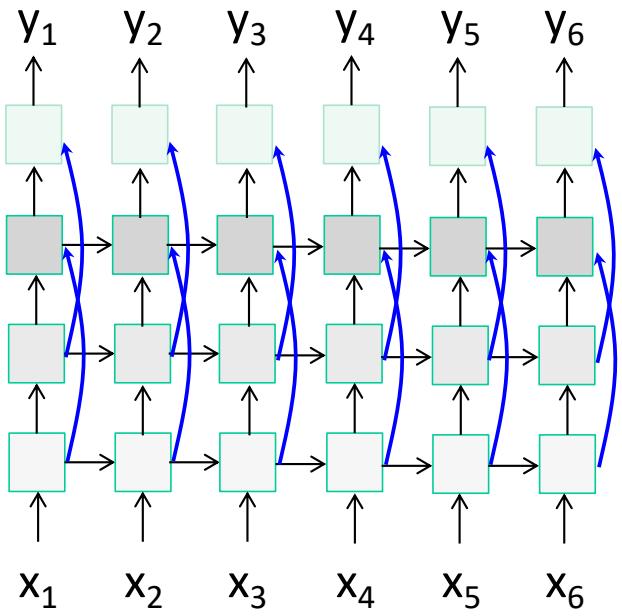
- We can of course design RNNs with multiple hidden layers




- Anything goes: skip connections across layers, across time, …

# Multi-layer RNNs

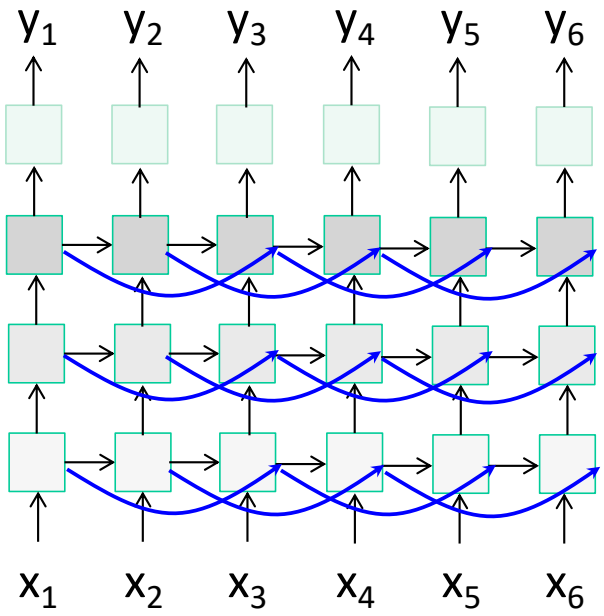
- We can of course design RNNs with multiple hidden layers




- Anything goes: skip connections across layers, across time, …

# Multi-layer RNNs

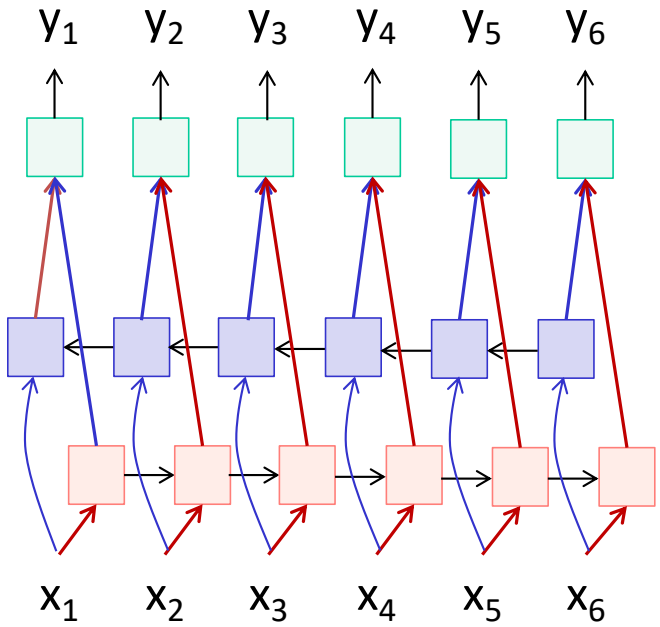
- We can of course design RNNs with multiple hidden layers



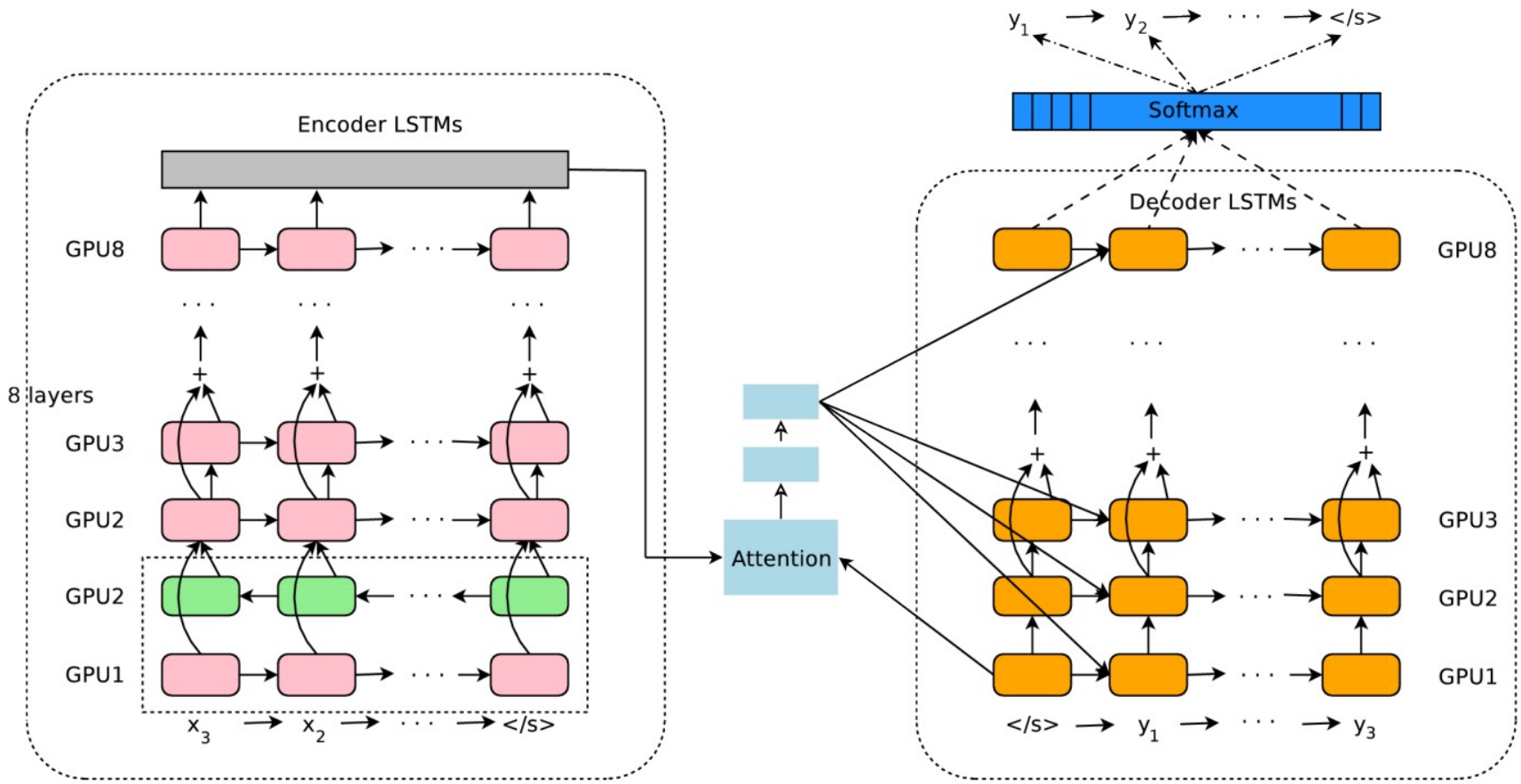

- Anything goes: skip connections across layers, across time, …

# Bi-directional RNNs

- RNNs can process the input sequence in forward and in the reverse direction (common in speech recognition)



# Google Neural Machine Translation (GNMT)

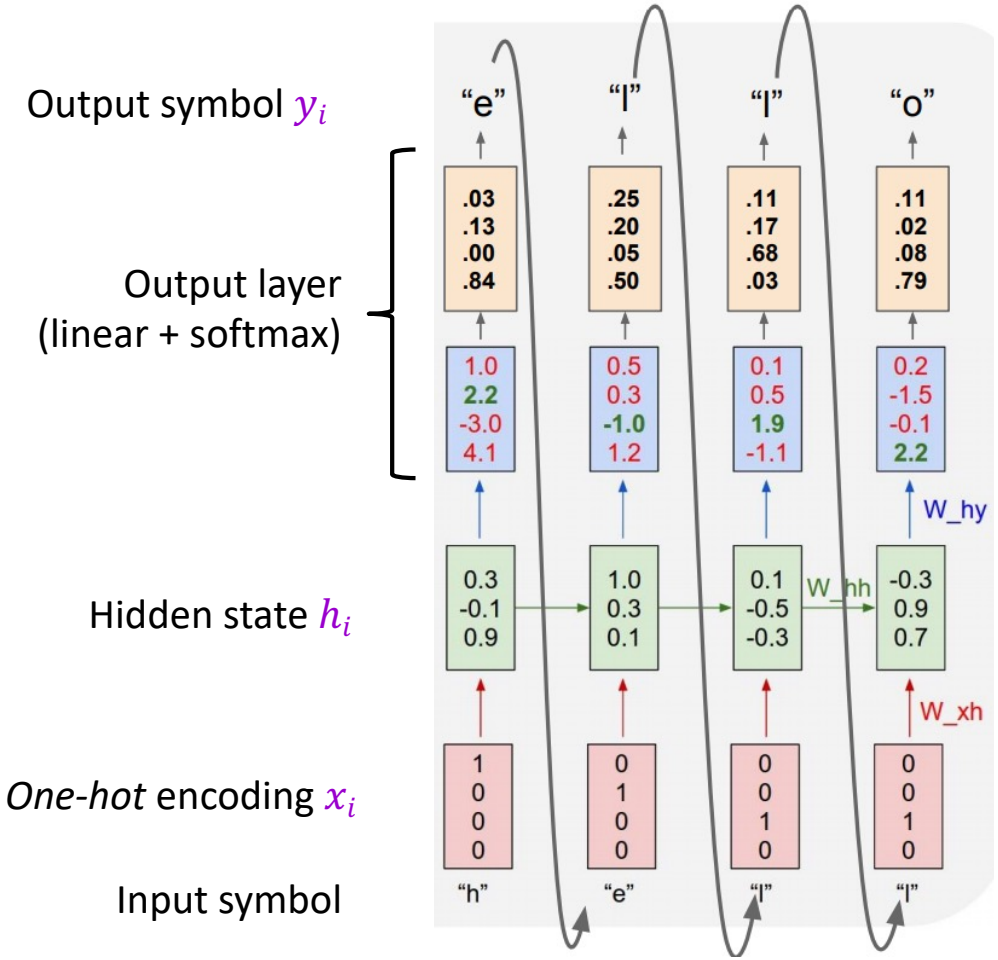



Y. Wu et al., [Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation](), arXiv 2016

# Outline

- Examples of sequential prediction tasks
- Common recurrent units
  - Vanilla RNN unit
  - Long Short-Term Memory (LSTM)
  - Gated Recurrent Unit (GRU)
- Recurrent network architectures
- Applications in (a bit) more detail
  - Language modeling
  - Image captioning
  - Machine translation

# Language modeling: Character RNN




$$p(y_1, y_2, \ldots, y_n)$$
$$= \prod_{i=1}^{n} p(y_i | y_1, \ldots, y_{i-1})$$
$$\approx \prod_{i=1}^{n} P_W(y_i | h_i)$$

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

# Language modeling: Character RNN

100th iteration

```
tyntd-iafhatawiaoihrdemot  lytdws  e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e
plia tklrgd t o idoe ns,smtt   h ne etie h,hregtrs nigtike,aoaenns lng
```

train more

300th iteration

```
"Tmont thithey" fomesscerliund
Keushey. Thom here
sheulke, anmerenith ol sivh I lalterthend Bleipile shuwy fil on aseterlome
coaniogennc Phe lism thond hon at. MeiDimorotion in ther thize."
```

train more

700th iteration

```
Aftair fall unsuch that the hall for Prince Velzonski's that me of
her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort
how, and Gogition is so overelical and ofter.
```

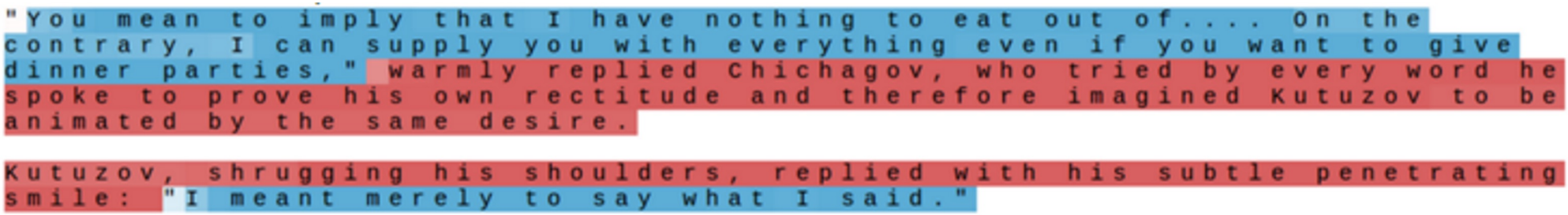
train more

2000th iteration

```
"Why do what that day," replied Natasha, and wishing to himself the fact the
princess, Princess Mary was easier, fed in had oftened him.
Pierre aking his soul came to the packs and drove up his father-in-law women.
```

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

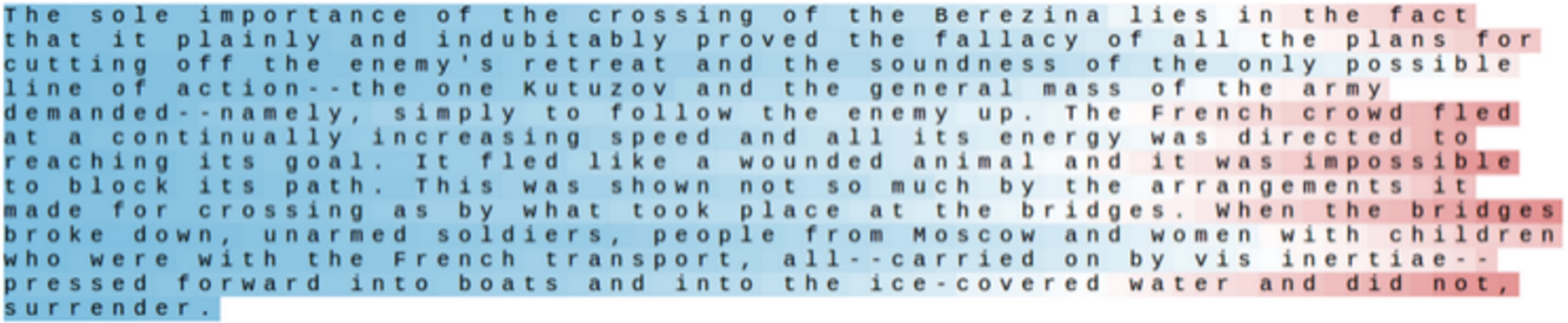
# Searching for interpretable hidden units




quote detection cell

A. Karpathy, J. Johnson, and L. Fei-Fei, Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016

# Searching for interpretable hidden units




line position tracking cell

A. Karpathy, J. Johnson, and L. Fei-Fei, Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016

# Searching for interpretable hidden units

```
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;
}
```

if statement cell

A. Karpathy, J. Johnson, and L. Fei-Fei, Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016

# Searching for interpretable hidden units


A. Karpathy, J. Johnson, and L. Fei-Fei, Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016

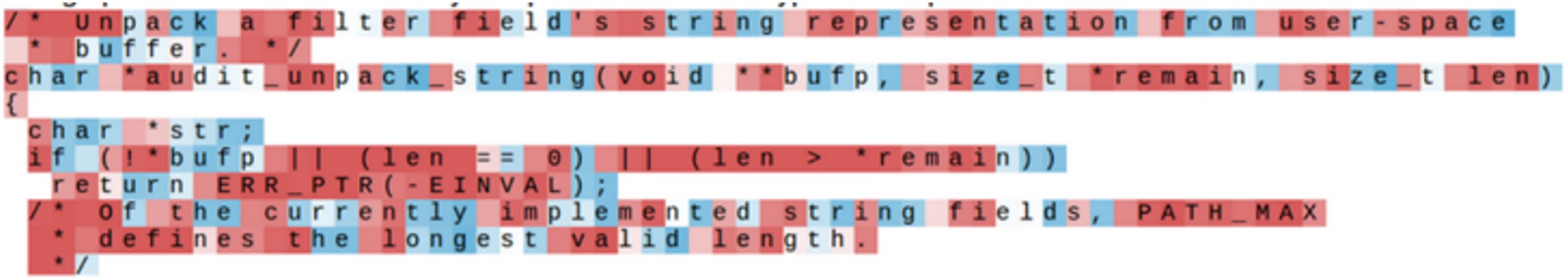
# Searching for interpretable hidden units

```
#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;
}
```

code depth cell

A. Karpathy, J. Johnson, and L. Fei-Fei, Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016

# Searching for interpretable hidden units



```
/* 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.
   */
```

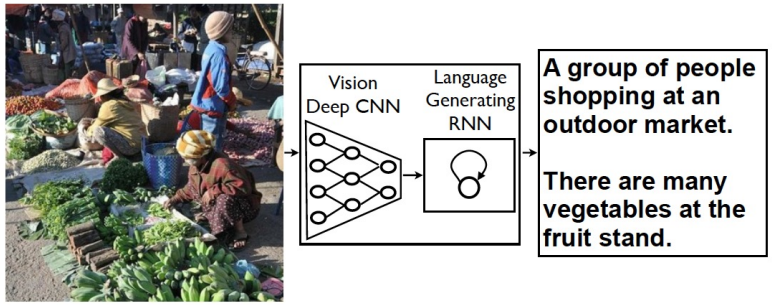
¯_(ツ)_/¯

A. Karpathy, J. Johnson, and L. Fei-Fei, Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016

# Recurrent models: Outline

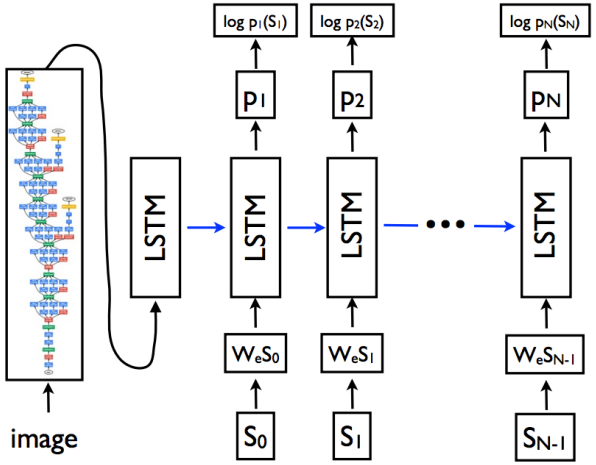
- Examples of sequential prediction tasks
- Common recurrent units
  - Vanilla RNN unit
  - Long Short-Term Memory (LSTM)
  - Gated Recurrent Unit (GRU)
- Recurrent network architectures
- Applications in (a bit) more detail
  - Language modeling
  - Image captioning

# Image caption generation




## Training time

- Maximize likelihood of reference captions




Log-likelihood of next reference word

Softmax probability over next word

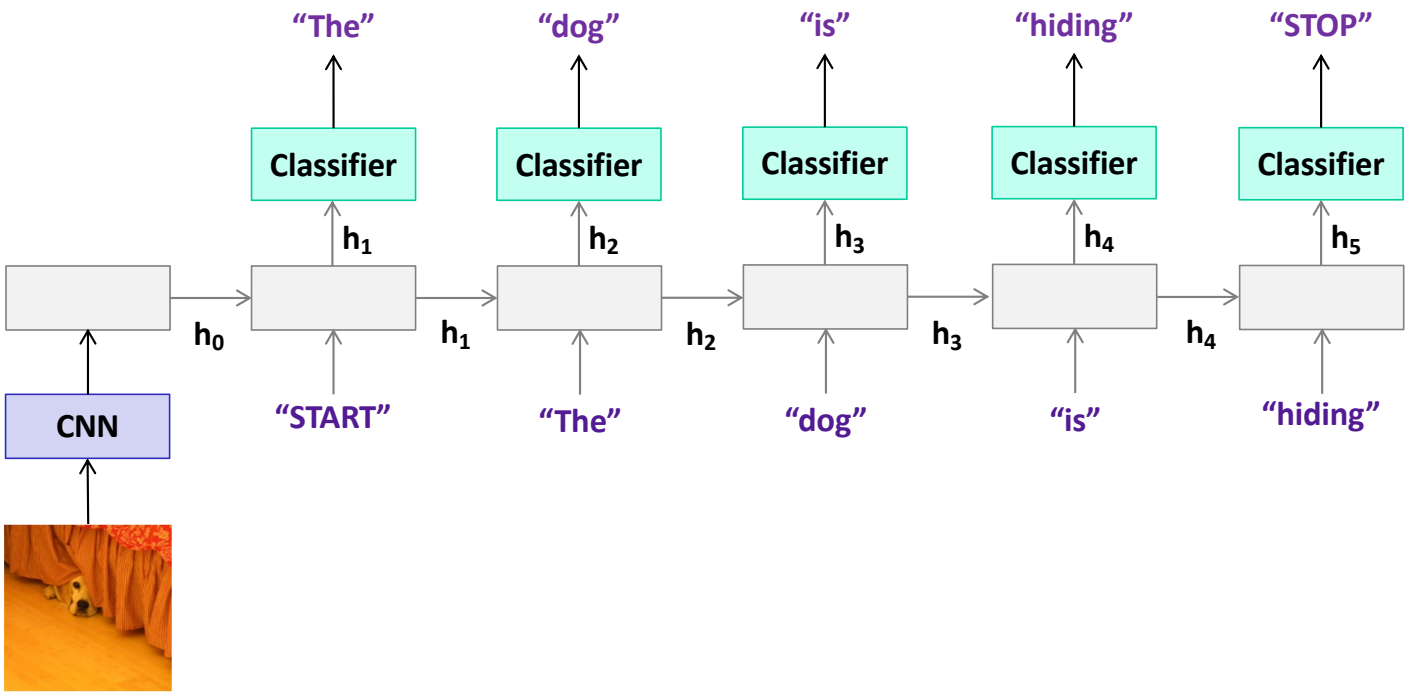
Word embedding

Words of reference caption (one-hot encoding)

O. Vinyals, A. Toshev, S. Bengio, D. Erhan, Show and Tell: A Neural Image Caption Generator, CVPR 2015

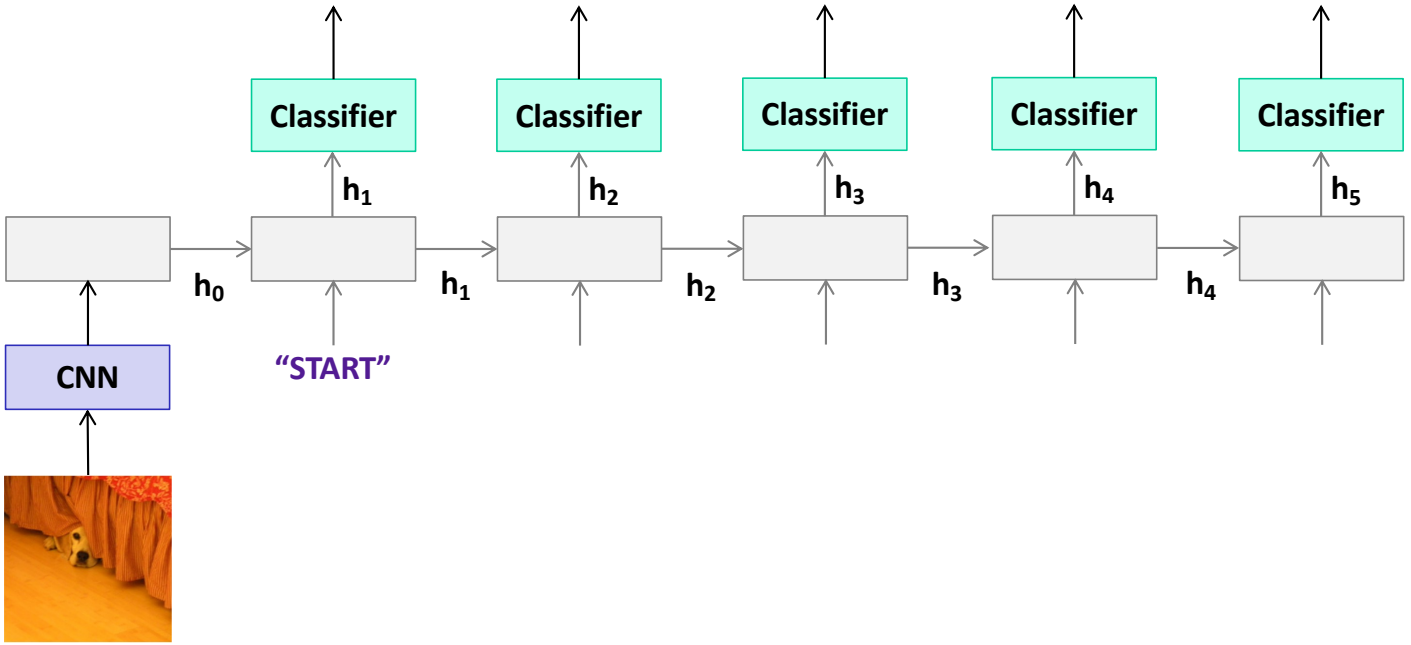
# Image caption generation: Test time

- How do we produce a caption given a test image?
  - How about always choosing the highest-likelihood word?

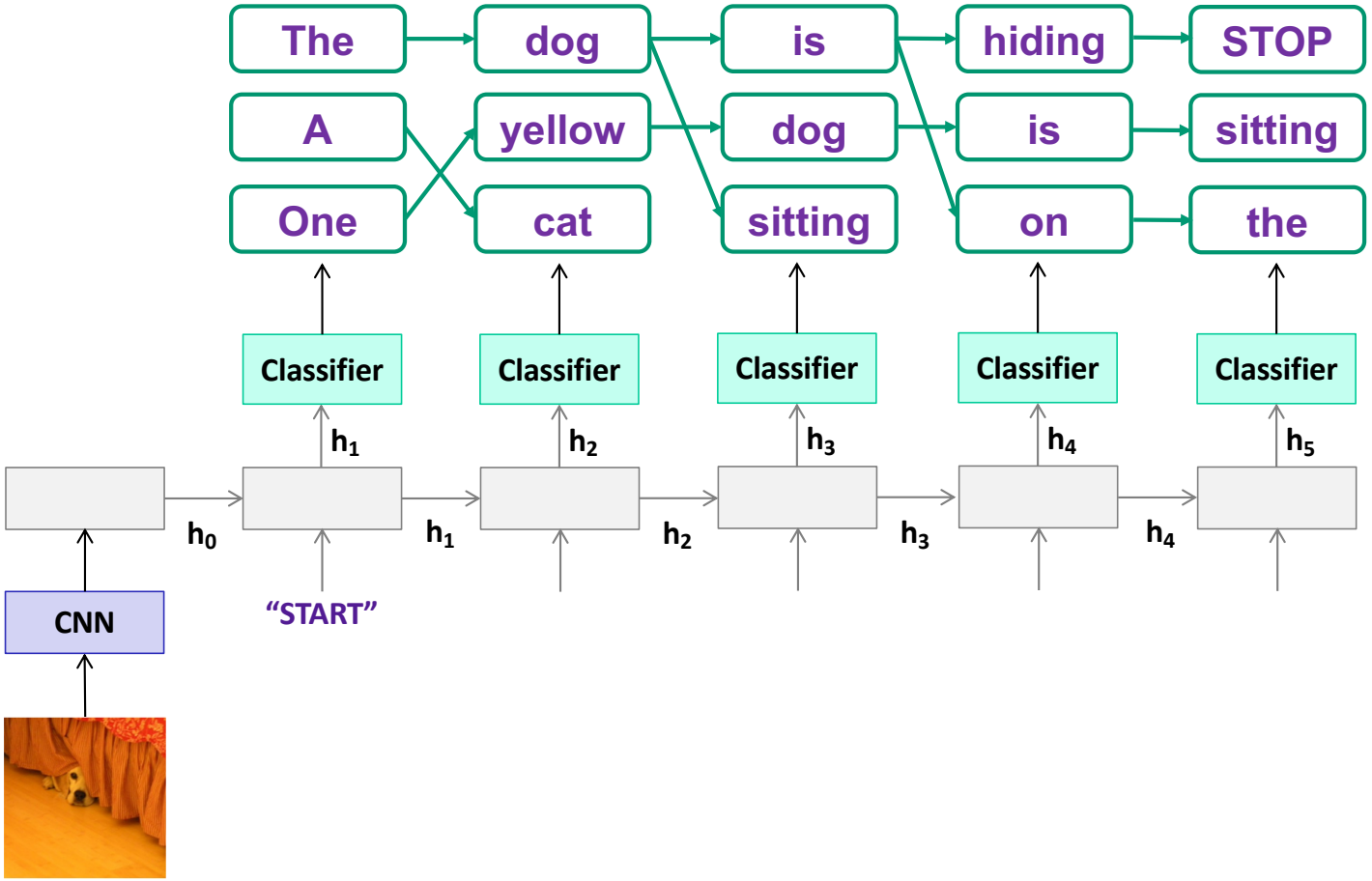


# Image caption generation: Beam search

- Maintain $k$ (*beam width*) top-scoring candidate sentences according to sum of per-word log-likelihoods (or some other score)
- At each step, generate all their successors and keep the best $k$



# Image caption generation: Beam search



# Image caption generation: Example outputs

# How to evaluate image captioning?

Reference sentences (written by human annotators):

- “A dog hides underneath a bed with its face peeking out of the bed skirt”
- “The small white dog is peeking out from under the bed”
- “A dog is peeking its head out from underneath a bed skirt”
- “A dog peeking out from under a bed”
- “A dog that is under a bed on the floor”

Generated sentence:

- “A dog is hiding”

# BLEU: Bilingual Evaluation Understudy

- **N-gram precision**: count the number of n-gram matches between candidate and reference translation, divide by total number of n-grams in candidate translation
  - Clip counts by the maximum number of times an n-gram occurs in any reference translation
  - Multiply by *brevity penalty* to penalize short translations

- Most commonly used measure for image captioning and machine translation despite multiple shortcomings

K. Papineni, S. Roukos, T. Ward, W.-J. Zhu, BLEU: a Method for Automatic Evaluation of Machine Translation, ACL 2002

# BLEU: Bilingual Evaluation Understudy

Original (French): J'ai mangé la pomme.

Reference translation: I ate the apple.

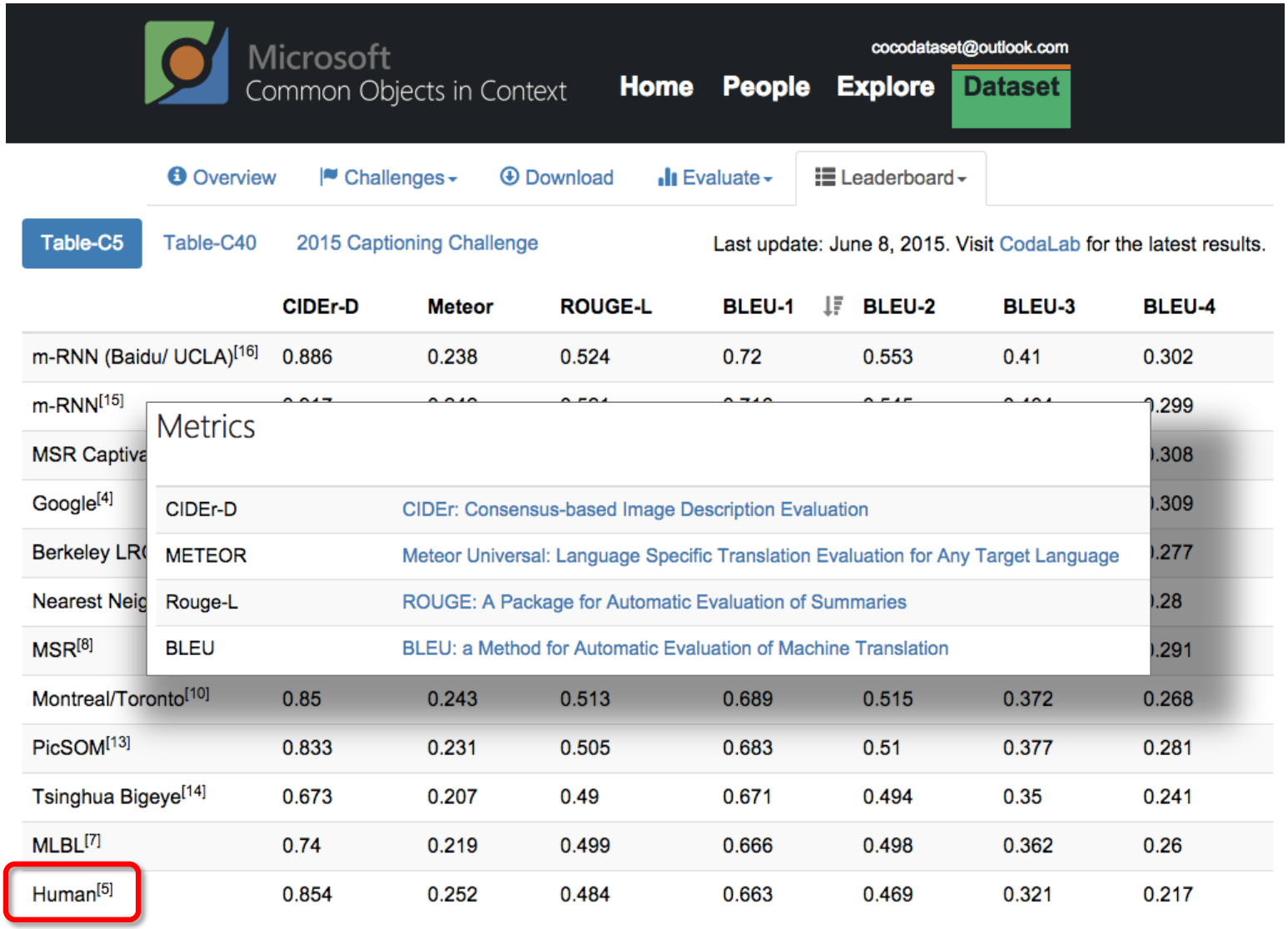
Based on BLEU, these are all "equally bad" output sentences.

I consumed the apple.

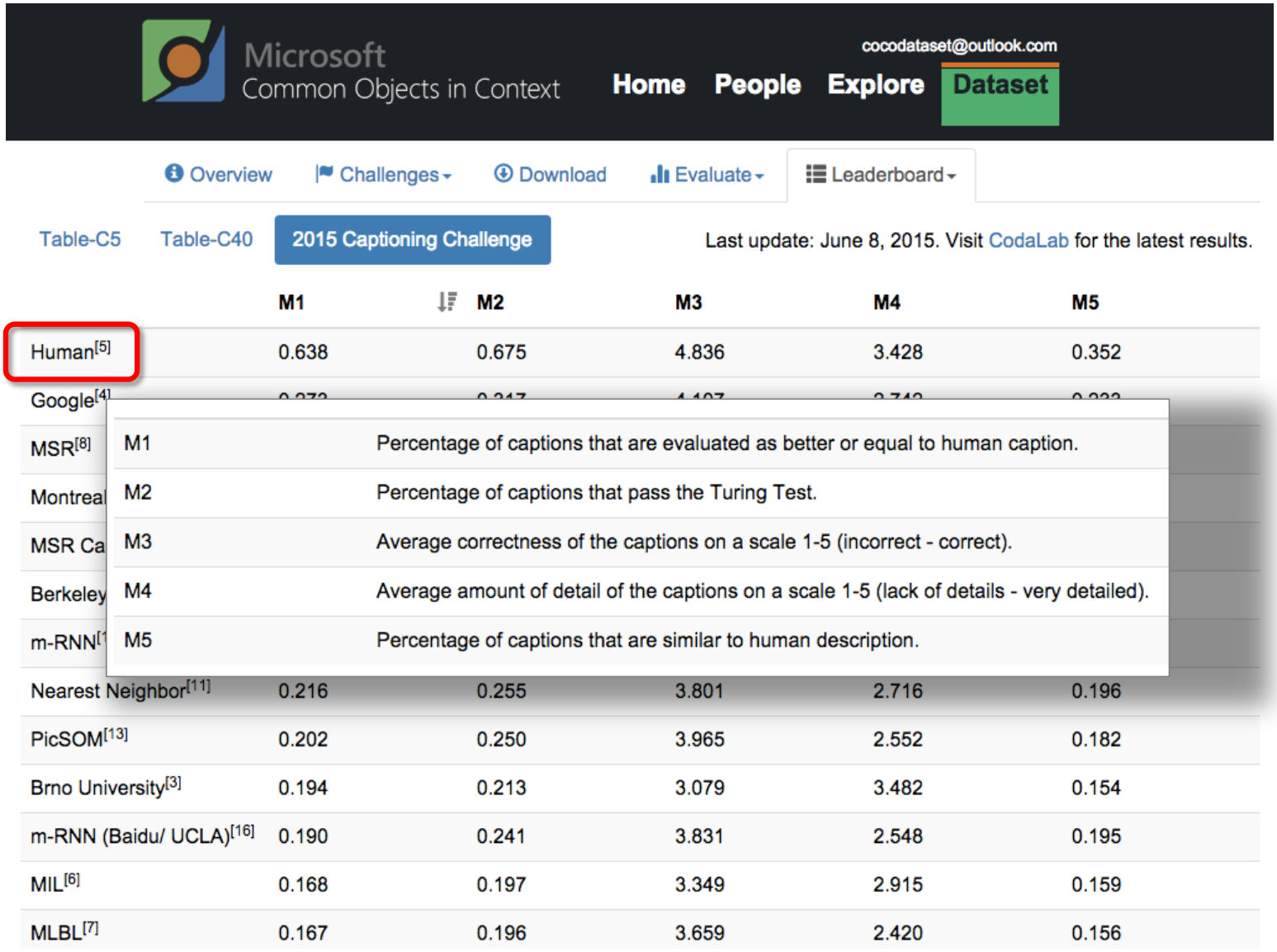
I ate an apple.

I ate the potato.

https://towardsdatascience.com/evaluating-text-output-in-nlp-bleu-at-your-own-risk-e8609665a213




http://mscoco.org/dataset/#captions-leaderboard




| | M1 | M2 | M3 | M4 | M5 |
|---|---|---|---|---|---|
| Human[5] | 0.638 | 0.675 | 4.836 | 3.428 | 0.352 |
| Google[4] | | | | | |
| MSR[8] | | | | | |
| Montreal | | | | | |
| MSR Ca | | | | | |
| Berkeley | | | | | |
| m-RNN[ | | | | | |
| Nearest Neighbor[11] | 0.216 | 0.255 | 3.801 | 2.716 | 0.196 |
| PicSOM[13] | 0.202 | 0.250 | 3.965 | 2.552 | 0.182 |
| Brno University[3] | 0.194 | 0.213 | 3.079 | 3.482 | 0.154 |
| m-RNN (Baidu/ UCLA)[16] | 0.190 | 0.241 | 3.831 | 2.548 | 0.195 |
| MIL[6] | 0.168 | 0.197 | 3.349 | 2.915 | 0.159 |
| MLBL[7] | 0.167 | 0.196 | 3.659 | 2.420 | 0.156 |