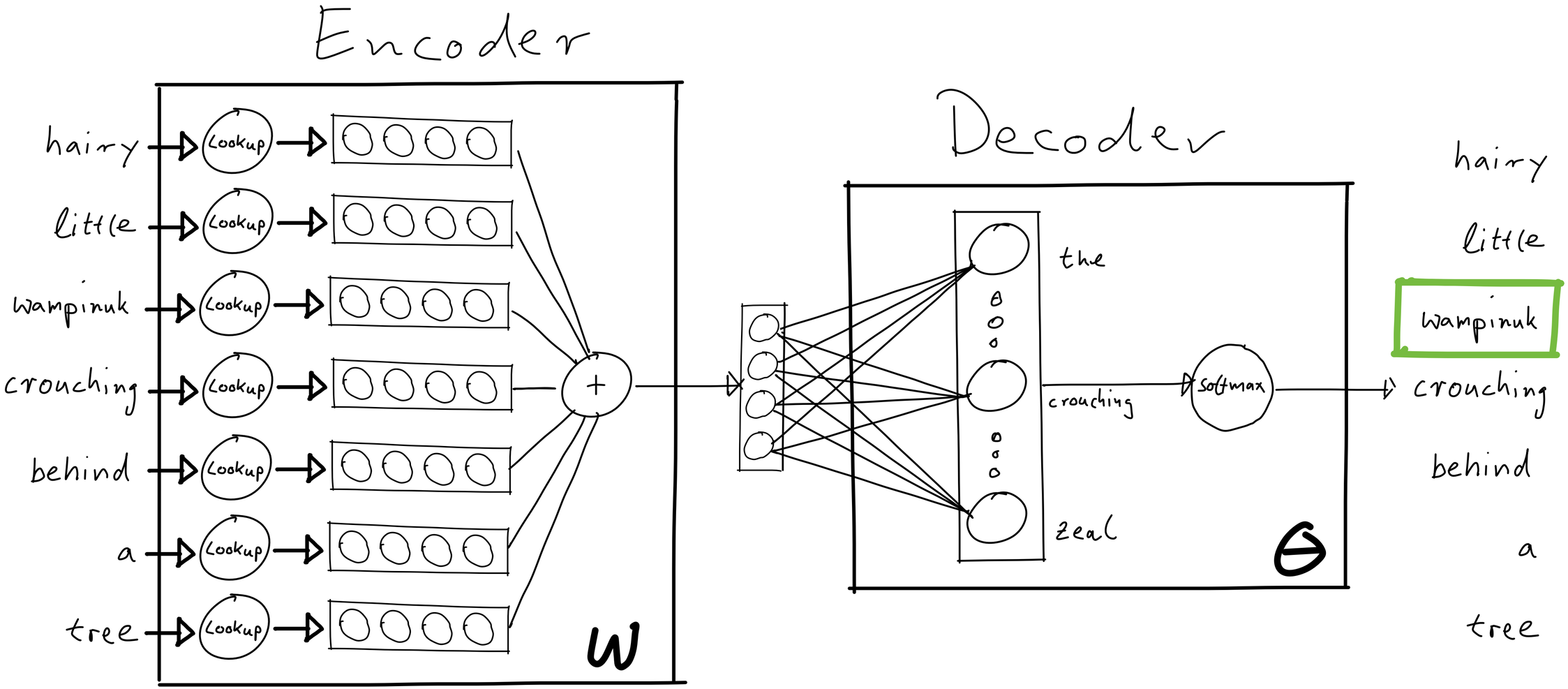
Recurrent Neural Networks

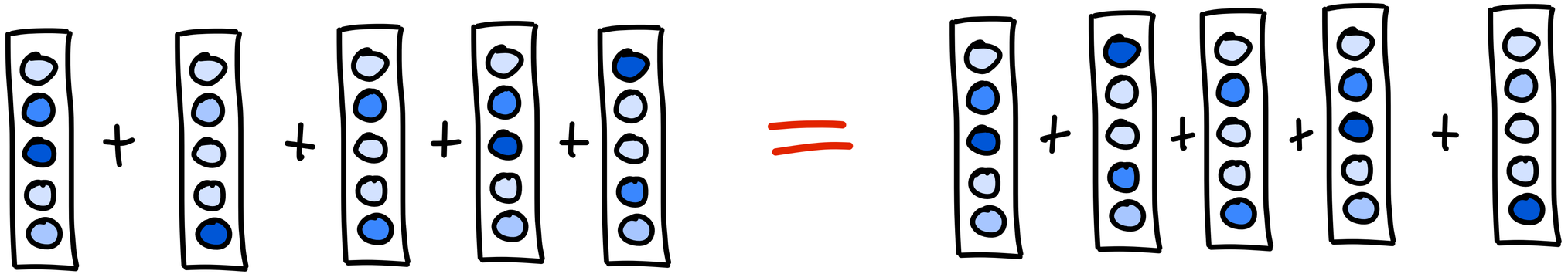
Tim Rocktäschel & Sebastian Riedel

COMP0087 Natural Language Processing

Last Week

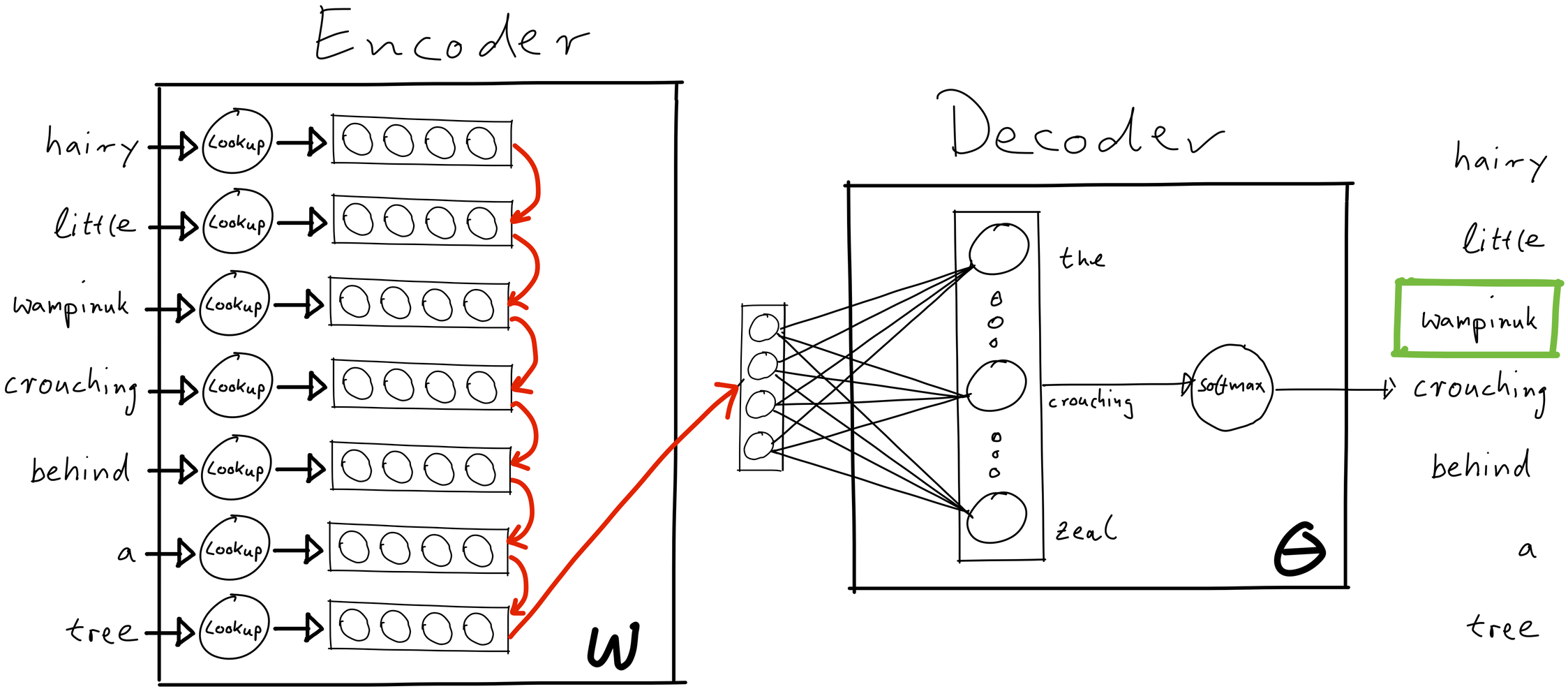

Encoder
hairy
little
wampinuk
crouching
behind
a
tree
Lookup
+
W
Decoder
the
crouching
zeal
softmax
θ
hairy
little
wampinuk
crouching
behind
a
tree

Order Matters

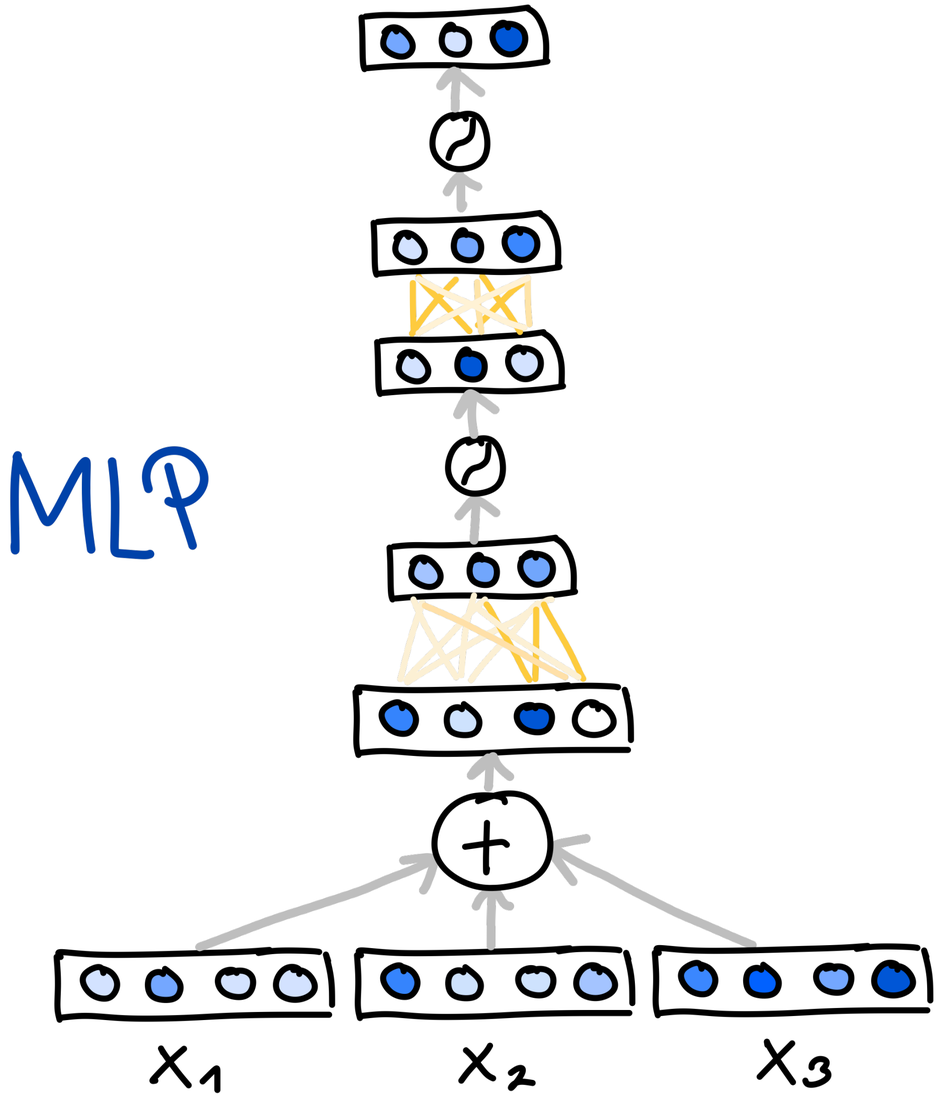


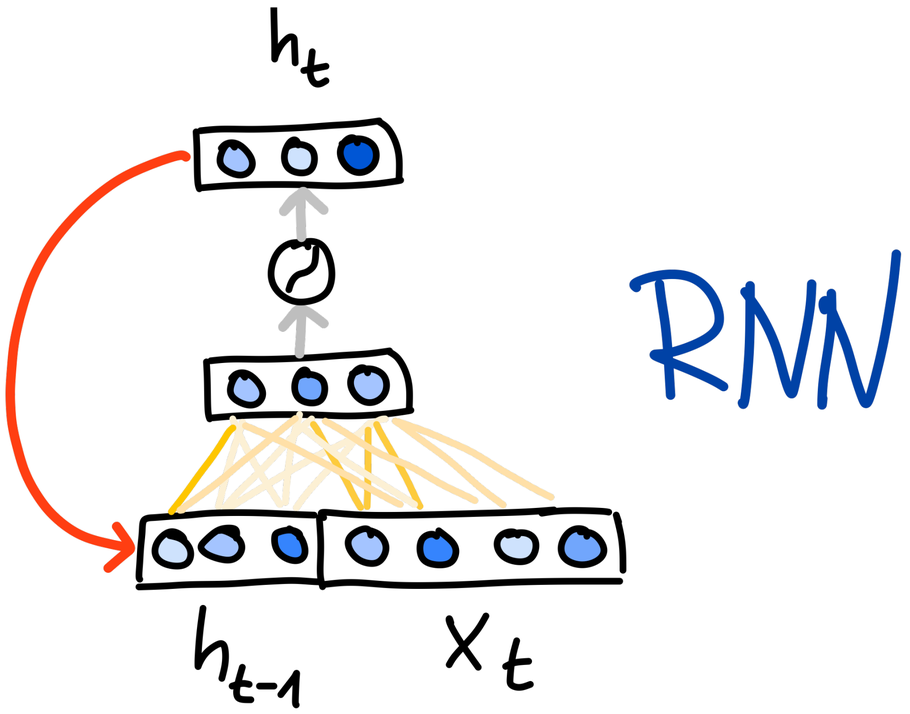
The hunter shot the fox ≠ The fox shot the hunter

This Week

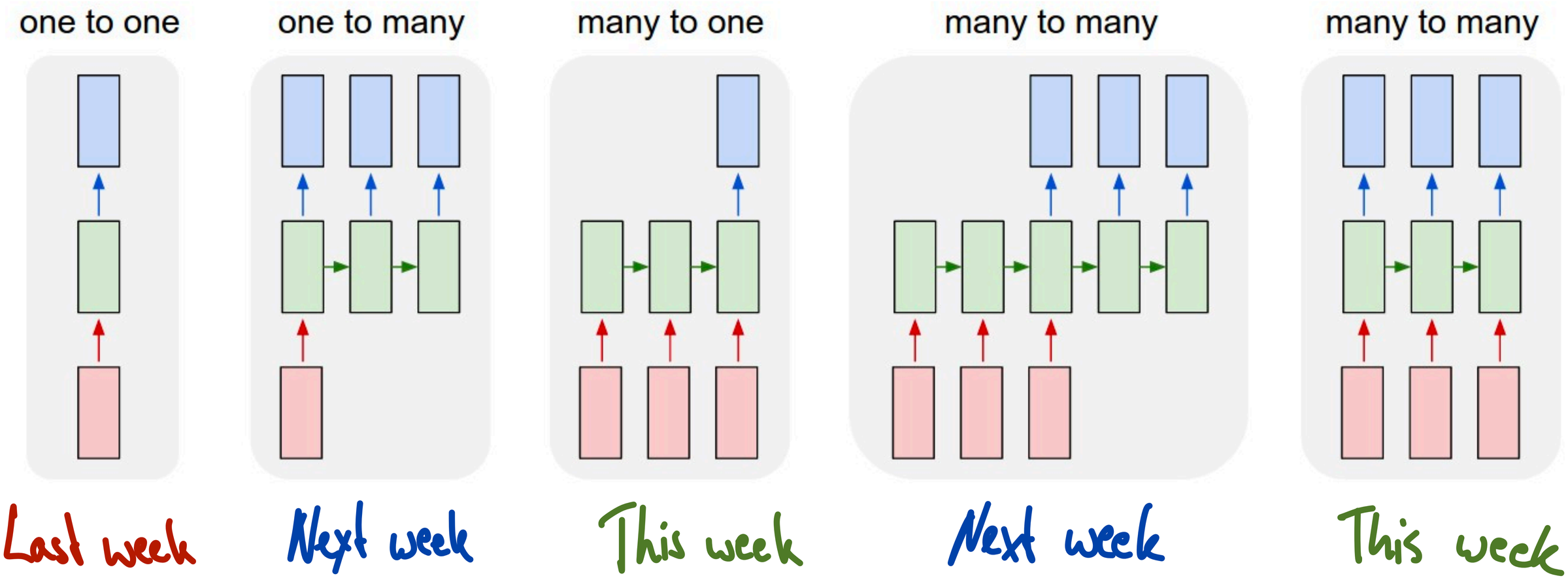

Encoder
hairy
little
wampinuk
crouching
behind
a
tree
Lookup
W
Decoder
the
crouching
zeal
softmax
θ
hairy
little
wampinuk
crouching
behind
a
tree

Recurrence





Sentence to Vector(s)



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

Fully-connected Recurrent Neural Network Cell

$$\begin{aligned}\mathbf{h_t} &= \tanh(\mathbf{W}^x\mathbf{x}_t + \mathbf{W}^h\mathbf{h}_{t-1} + \mathbf{b}) \\ &= \tanh(\mathbf{W}[\mathbf{x}_t; \mathbf{h}_{t-1}] + \mathbf{b})\end{aligned}$$

- RNNs are universal function approximators
  - A network with a single hidden layer containing a finite number of neurons can approximate any continuous function
- Are we done?

Elman. “Finding Structure in Time.” *Cognitive Science*. 1990.

Representation Matters

Representation:
The Earth is fixed center of our Solar System

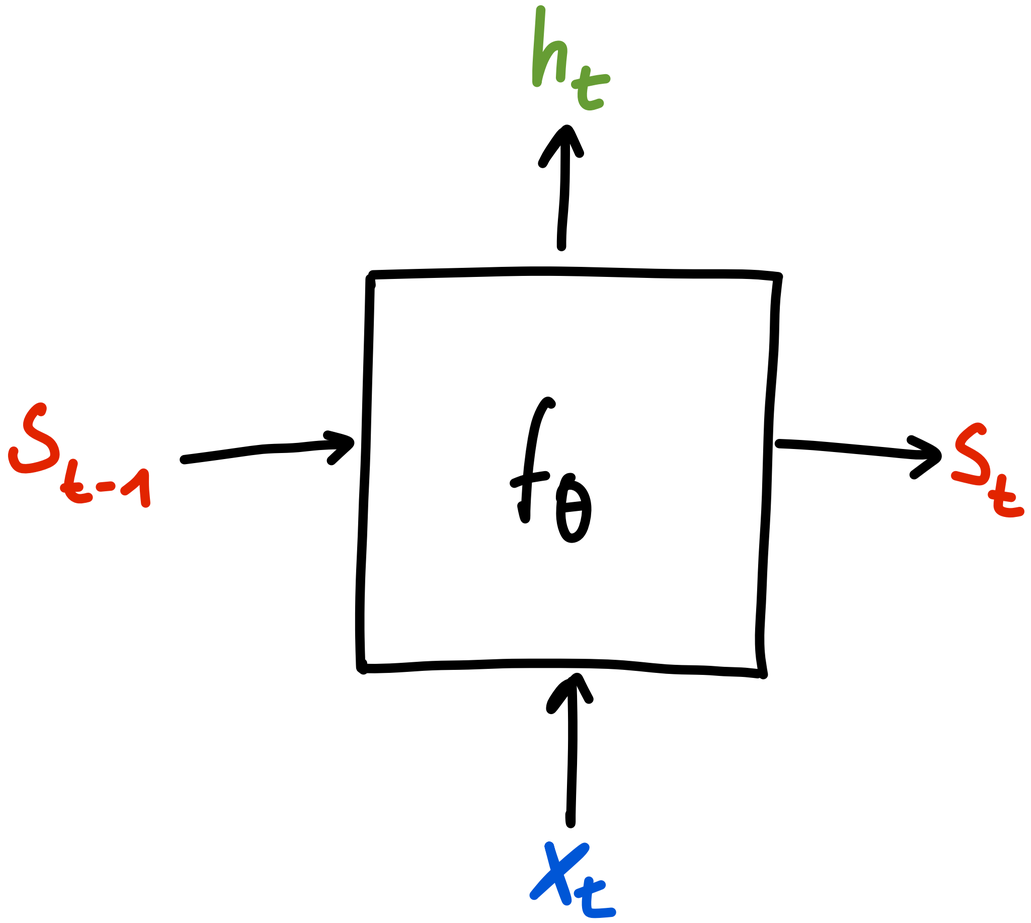
Geocentric Model
(Anaximander, 6th century BC)

Representation:
The Sun is fixed center of our Solar System

Heliocentric Model
(Copernicus, 1543)

Lex Fridman MIT Course 6.S094 https://www.youtube.com/watch?v=O5xeyoRL95U

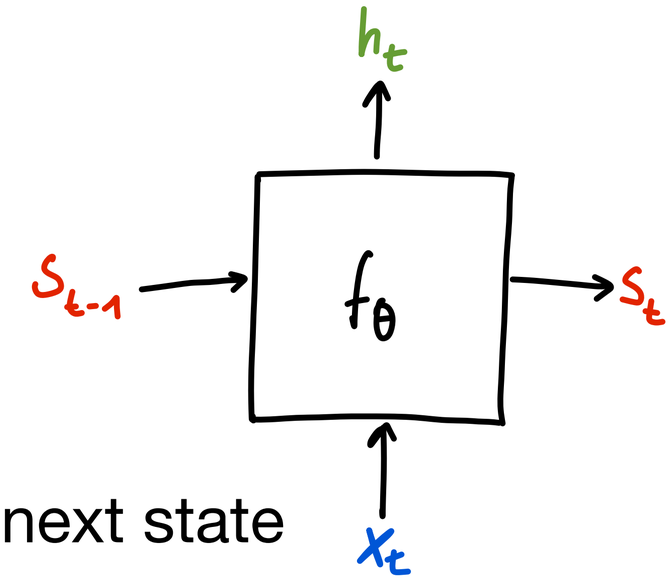
General Recurrent Cell API



General Recurrent Cell API

- Cell function: $f_\theta : \mathbb{R}^k \times \mathbb{R}^s \rightarrow \mathbb{R}^o \times \mathbb{R}^s$
  - Input $\mathbf{x}_t \in \mathbb{R}^k$
  - Previous state $\mathbf{s}_{t-1} \in \mathbb{R}^s$
  - Output $\mathbf{h}_t \in \mathbb{R}^o$
  - Given input and previous state, produce output and next state

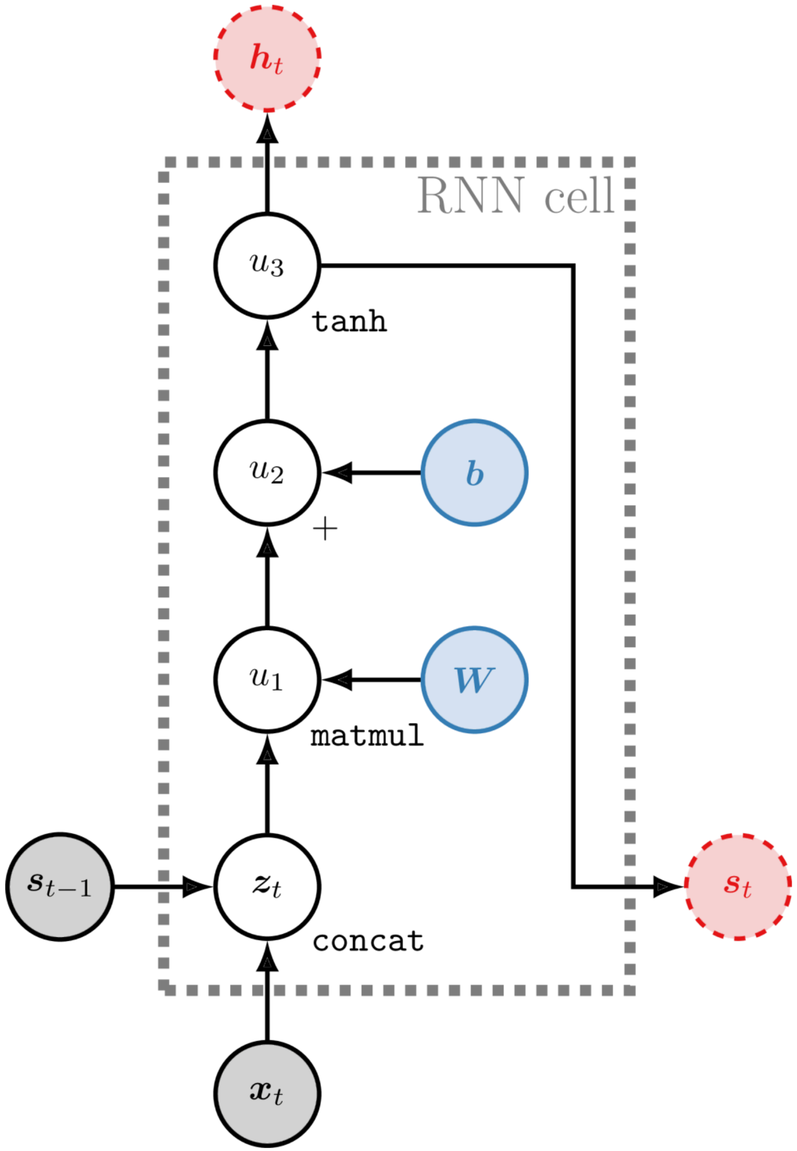
$$\mathbf{h}_t, \mathbf{s}_t = f_\theta(\mathbf{x}_t, \mathbf{s}_{t-1})$$



- Given sequence of inputs $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2 \ldots, \mathbf{x}_N]$ and a start state $\mathbf{s}_0$, produce sequence of outputs and next states

$$\begin{aligned} \textbf{unfold}\left(f_\theta, \mathbf{X}, \mathbf{s}_0\right) &= \left[f_\theta(\mathbf{x}_1, \mathbf{s_0}), f_\theta(\mathbf{x}_2, \mathbf{s}_1), \ldots, f_\theta(\mathbf{x}_N, \mathbf{s}_{N-1})\right] \\ &= \left[(\mathbf{h}_1, \mathbf{s}_1), (\mathbf{h}_2, \mathbf{s}_2), \ldots, (\mathbf{h}_N, \mathbf{s}_N)\right] \end{aligned}$$

Fully-connected Recurrent Neural Network Cell



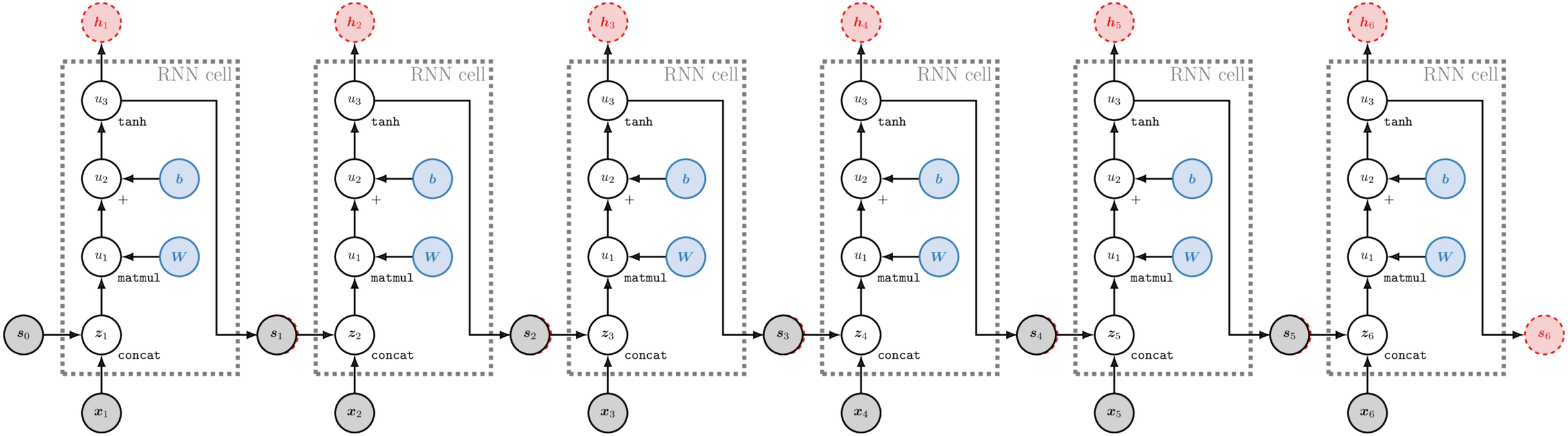
$$\mathbf{s}_t = \tanh(\mathbf{W}[\mathbf{x}_t; \mathbf{s}_{t-1}] + \mathbf{b})$$

$$\mathbf{h}_t = \mathbf{s}_t$$

$$f_\theta^{\mathbf{RNN}}(\mathbf{x}_t, \mathbf{s}_{t-1}) = \left(\mathbf{h}_t, \mathbf{s}_t\right)$$

Elman. “Finding Structure in Time.” *Cognitive Science*. 1990.

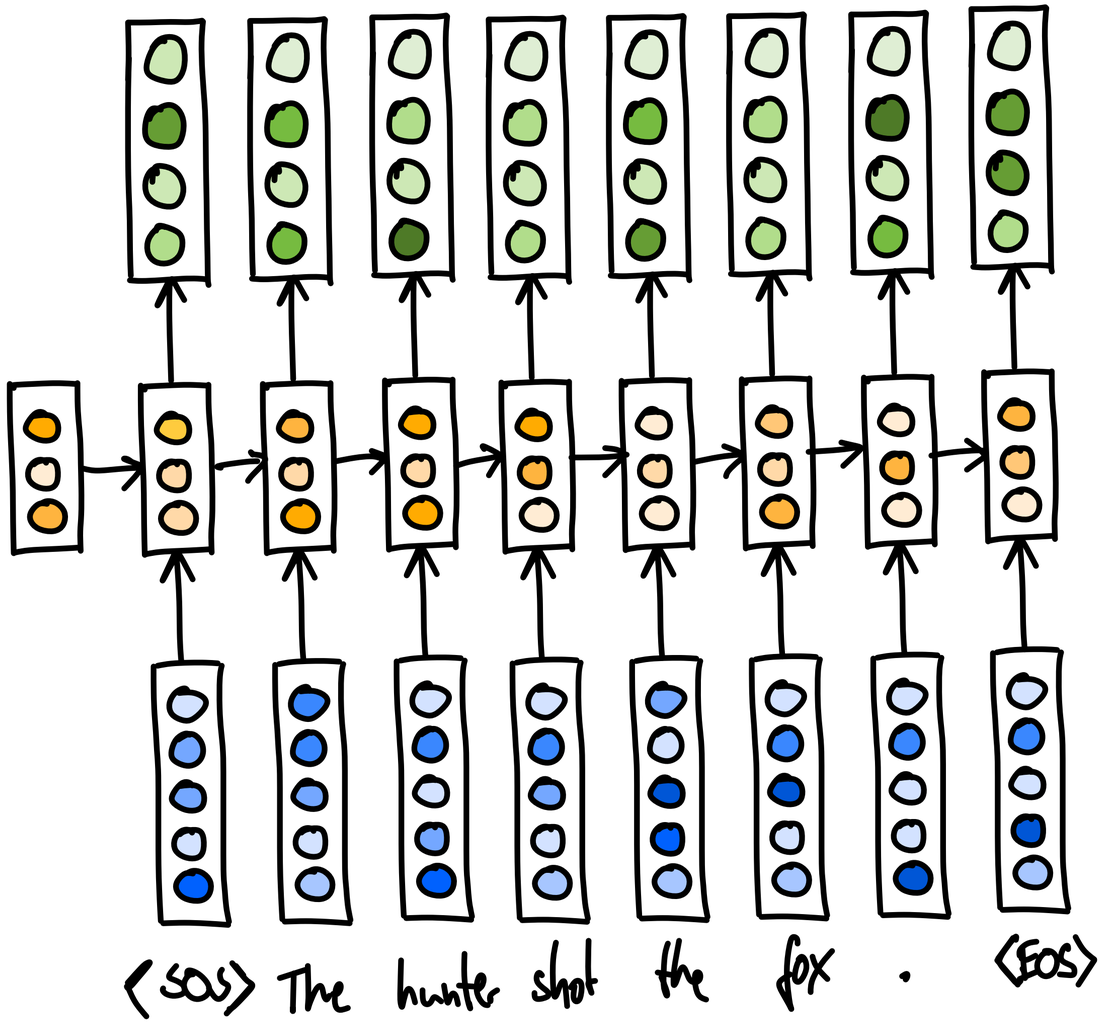
Unfold



$$\textbf{unfold}\left(f_\theta, \mathbf{X}, \mathbf{s}_0\right) = \left[f_\theta(\mathbf{x}_1, \mathbf{s_0}), f_\theta(\mathbf{x}_2, \mathbf{s}_1), \ldots, f_\theta(\mathbf{x}_N, \mathbf{s}_{N-1})\right]$$

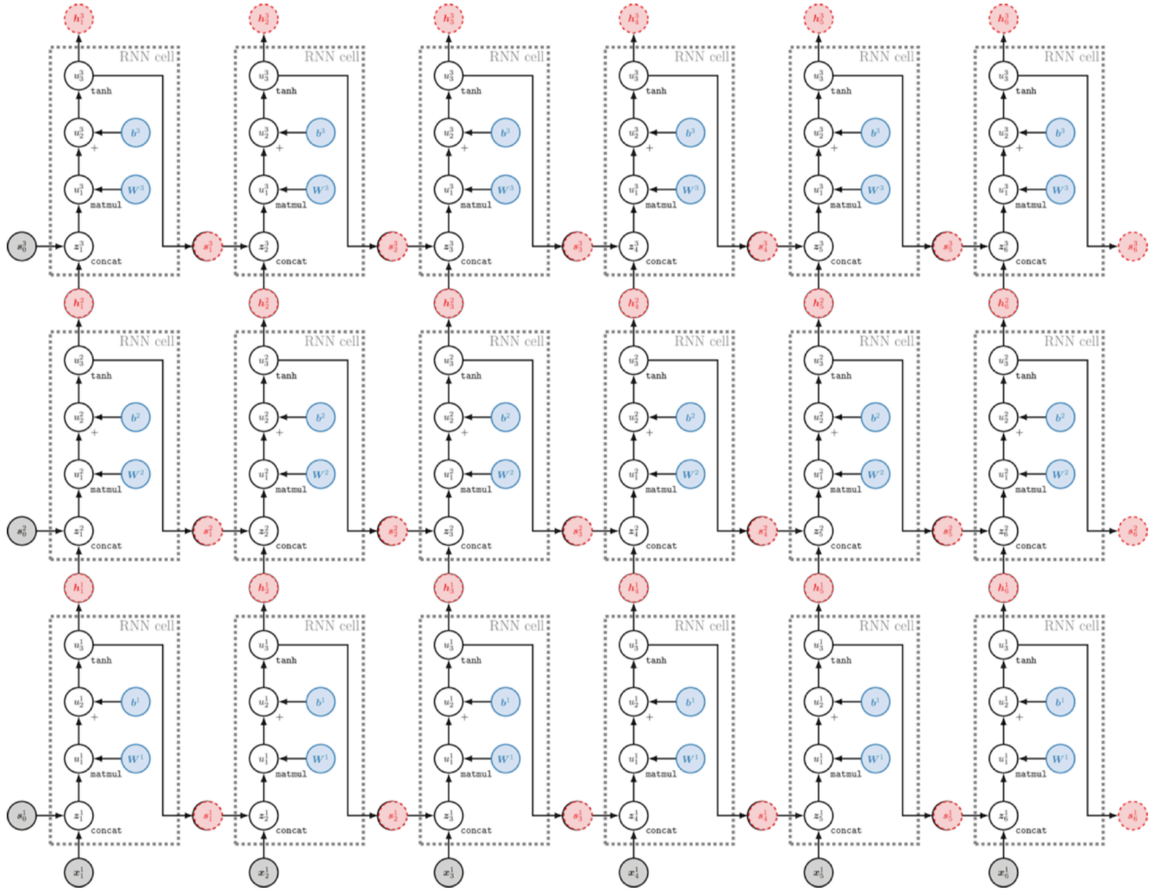
$$= \left[(\mathbf{h}_1, \mathbf{s}_1), (\mathbf{h}_2, \mathbf{s}_2), \ldots, (\mathbf{h}_N, \mathbf{s}_N)\right]$$

Unfold



Stacking RNNs

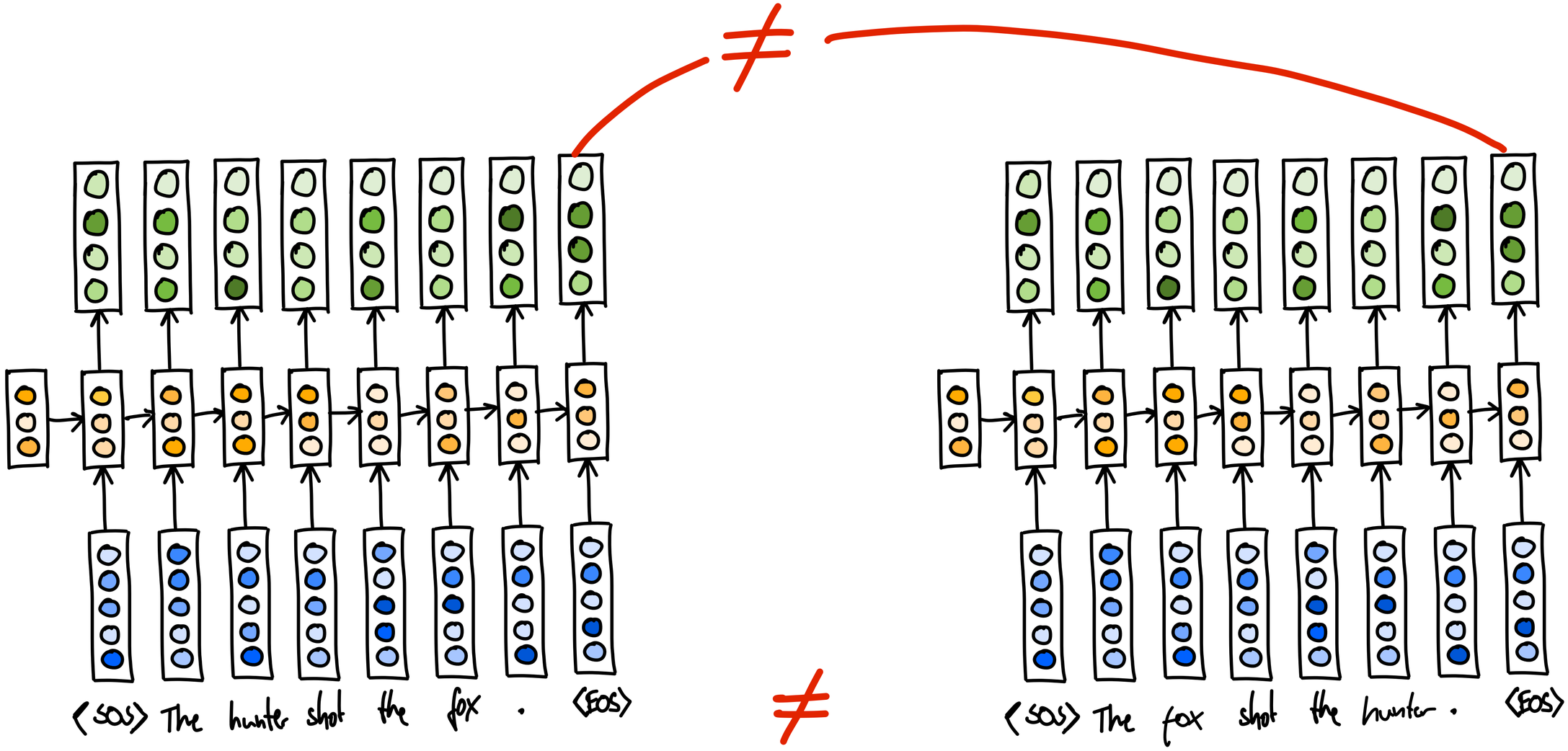
$$
\begin{aligned}
\textbf{unfold}\ \left(f_\theta^1, \mathbf{X}, \mathbf{s}_0^1\right) &= \left[f_\theta(\mathbf{x}_1, \mathbf{s}_0^1), f_\theta(\mathbf{x}_2, \mathbf{s}_1^1), \ldots, f_\theta(\mathbf{x}_N, \mathbf{s}_{N-1}^1)\right] \\
&= \left[(\mathbf{h}_1^1, \mathbf{s}_1^1), (\mathbf{h}_2^1, \mathbf{s}_2^1), \ldots, (\mathbf{h}_N^1, \mathbf{s}_N^1)\right] \\
\textbf{unfold}\ \left(f_\theta^2, \mathbf{H}^1, \mathbf{s}_0^2\right) &= \left[f_\theta(\mathbf{h}_1^1, \mathbf{s}_0^2), f_\theta(\mathbf{h}_2^1, \mathbf{s}_1^2), \ldots, f_\theta(\mathbf{h}_N^1, \mathbf{s}_{N-1}^2)\right] \\
&= \left[(\mathbf{h}_1^2, \mathbf{s}_1^2), (\mathbf{h}_2^2, \mathbf{s}_2^2), \ldots, (\mathbf{h}_N^2, \mathbf{s}_N^2)\right] \\
&\vdots \\
\textbf{unfold}\ \left(f_\theta^m, \mathbf{H}^{m-1}, \mathbf{s}_0^m\right) &= \left[f_\theta(\mathbf{h}_1^{m-1}, \mathbf{h}_0^{m-1}), f_\theta(\mathbf{h}_2^{m-1}, \mathbf{s}_1^{m-1}), \ldots, f_\theta(\mathbf{h}_N^{m-1}, \mathbf{s}_{N-1}^{m-1})\right] \\
&= \left[(\mathbf{h}_1^m, \mathbf{s}_1^m), (\mathbf{h}_2^m, \mathbf{s}_2^m), \ldots, (\mathbf{h}_N^m, \mathbf{s}_N^m)\right]
\end{aligned}
$$



PyTorch Example

```
class SimpleRNN(nn.Module):
    def __init__(self, k):
        super(SimpleRNN, self).__init__()
        self.W = nn.Parameter(torch.randn(k, 2*k))
        self.b = nn.Parameter(torch.randn(k))
    def f(self, x, s):
        x = torch.cat([x, s])  # -- [2*k]
        s = torch.tanh(torch.einsum("ij,j->i", [self.W, x]) + self.b)  # -- [k]
        h = s
        return h, s
    def forward(self, X, s):  # what we called "unroll"
        H = torch.zeros(X.shape)
        for i in range(len(X)):
            h, s = self.f(X[i], s)
            H[i] = h
        return H
rnn = SimpleRNN(3)
X = torch.randn(7, 3)
s0 = torch.randn(3)
rnn(X, s0)
```

Order Matters



Language Modeling

- p("The hunter shot the fox") > p("The fox shot the hunter")
- Factorize using word history

$$p(w_1, w_2, \ldots, w_N) = p(w_1)p(w_2 \mid w_1)p(w_3 \mid w_2, w_1) \times \cdots \times p(w_N \mid w_{N-1}, \ldots, w_1)$$
$$= \prod_{t=1}^{N} p(w_t \mid w_{t-1}, w_{t-2}, \ldots, w_1)$$

- Approximate using NGrams, e.g., TriGrams

$$p(w_1, w_2, \ldots, w_N) \approx p_\theta(w_1)p_\theta(w_2 \mid w_1)p_\theta(w_3 \mid w_2, w_1) \times \cdots \times p_\theta(w_N \mid w_{N-1}, w_{N-2})$$
$$= \prod_{t=1}^{N} p_\theta(w_t \mid w_{t-1}, w_{t-2})$$

Neural Language Modeling

- No theoretical boundary on the length of the word history

$$p(w_1, w_2, \ldots, w_N) = \prod_{i=1}^{N} p_\theta(w_t \mid w_{t-1}, w_{t-2}, \ldots, w_1)$$

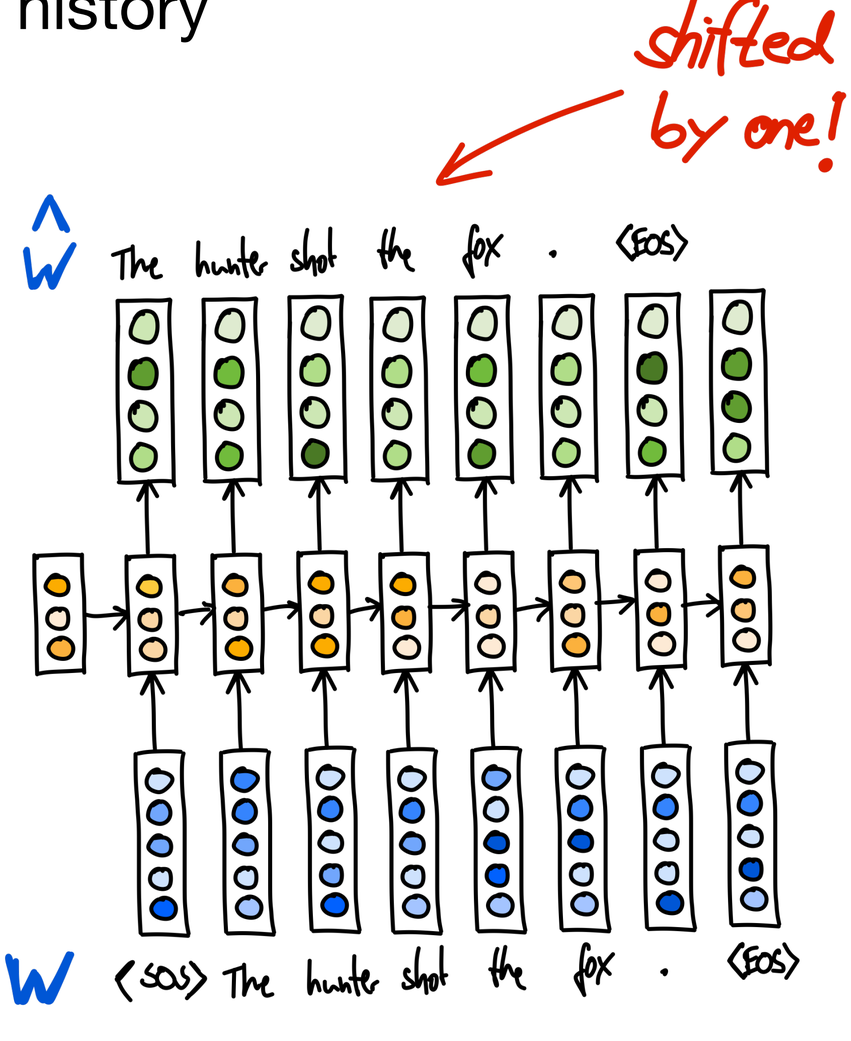
$$p_\theta(\,. \mid w_{t-1}, w_{t-2}, \ldots, w_1) = \textbf{softmax}(\mathbf{V}\mathbf{h}_t + \mathbf{b})$$

$$\textbf{softmax}_i(\mathbf{y}) = \frac{e^{\mathbf{y}_i}}{\sum_j e^{\mathbf{y}_j}}$$

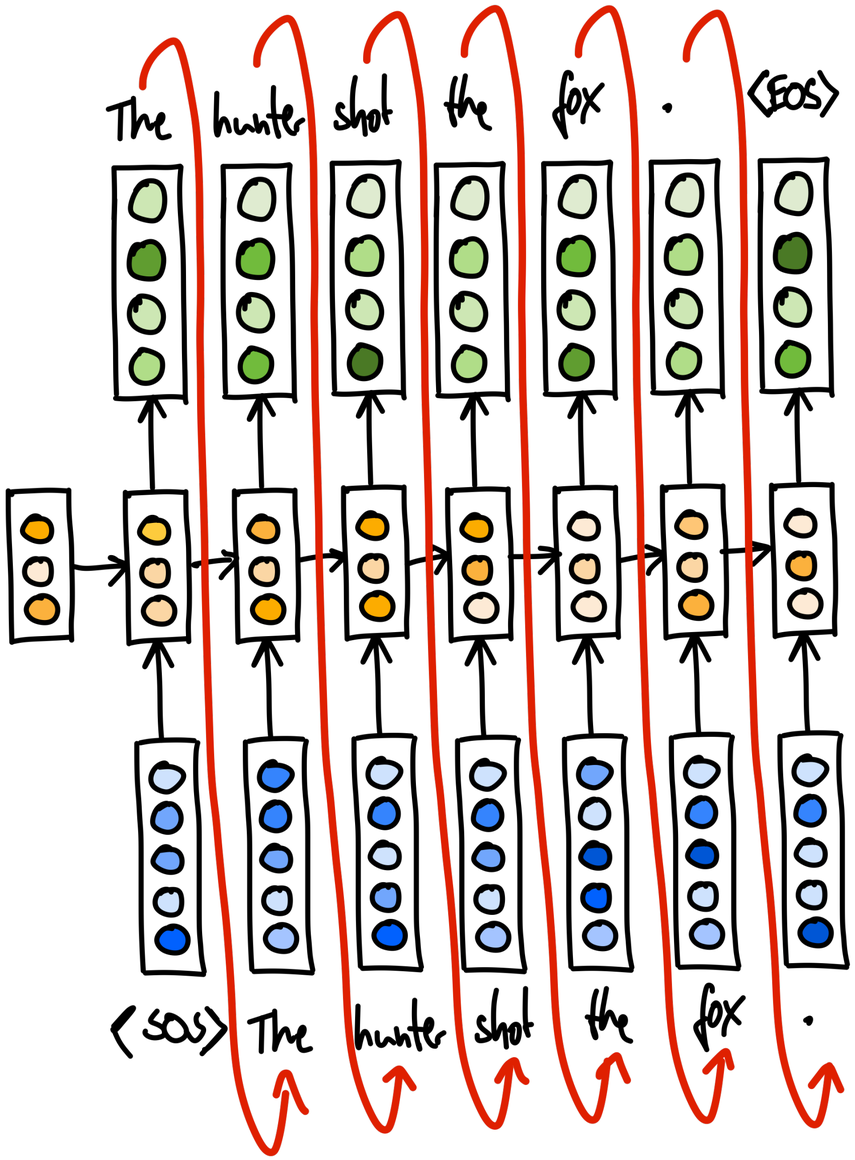
- Training objective given target sequence

$$L_{p_\theta}(\hat{w}) = -\frac{1}{N}\sum_i \hat{y}^T_{\hat{w}} \log p_\theta(\hat{w} \mid w_{i-1}, \ldots, w_1)$$

one-hot encoding of word $\hat{w}$



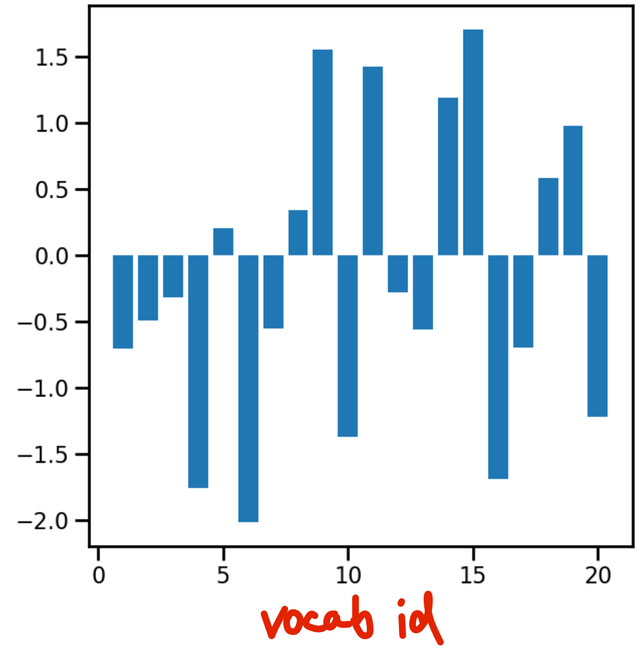
Generating from a Language Model

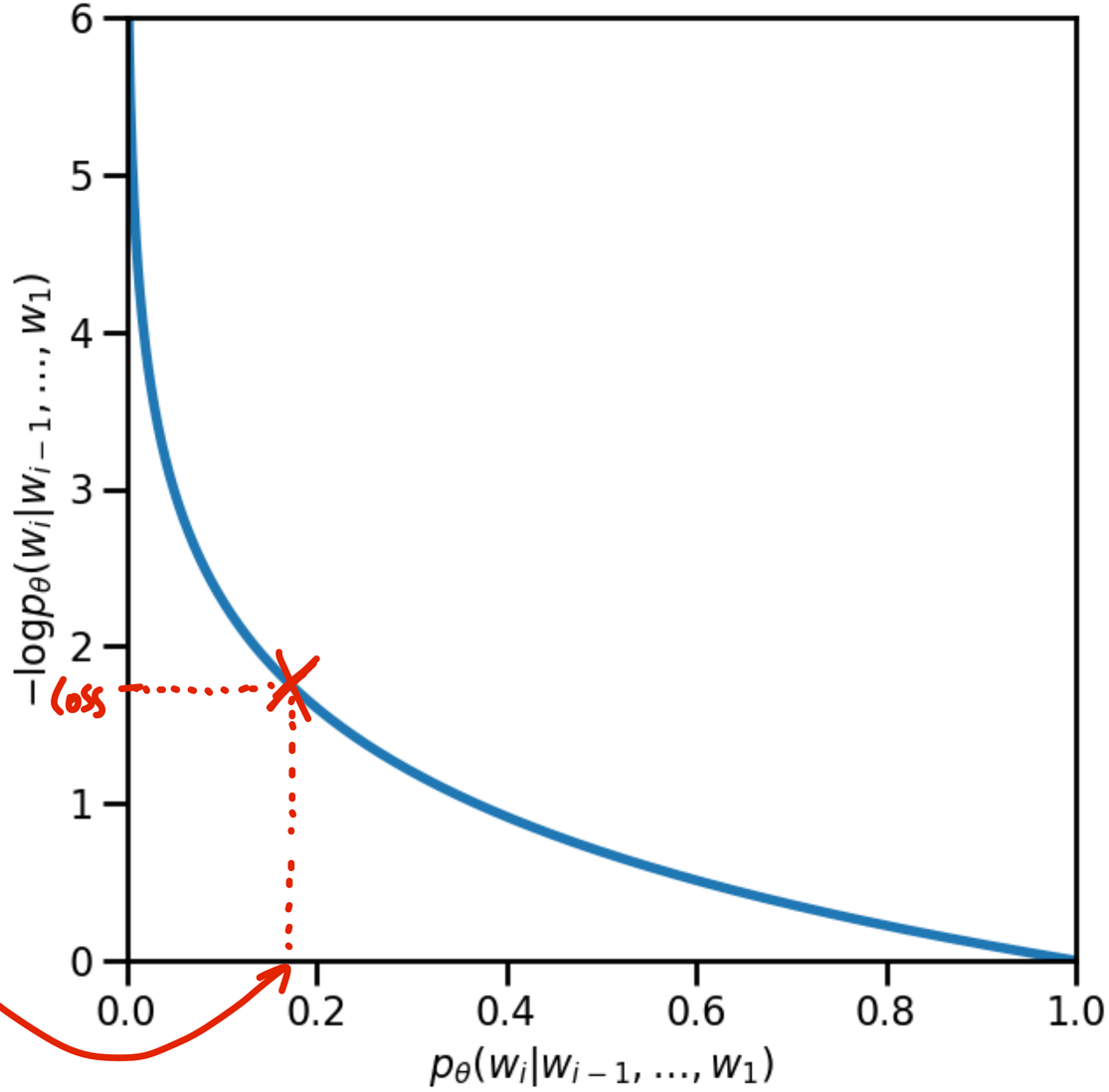


Generating “Wikipedia”

Naturalism and decision for the majority of Arab countries' capitalide was grounded by the Irish language by **John Clair**, **An Imperial Japanese Revolt**, associated with Guangzham's sovereignty. His generals were the powerful ruler of the Portugal in the **Protestant Immineners**, which could be said to be directly in Cantonese Communication, which followed a ceremony and set inspired prison, training. The emperor travelled back to **Antioch, Perth, October 25|21** to note, the Kingdom of Costa Rica, unsuccessful fashioned the **Thrales**, **Cynth's Dajoard**, known in western **Scotland**, near Italy to the conquest of India with the conflict. Copyright was the succession of independence in the slop of Syrian influence that was a famous German movement based on a more popular servicious, non-doctrinal and sexual power post.

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


unnormalized
softmax
normalized
vocab id
target word $\hat{w}$


$-\log p_\theta(w_i|w_{i-1},\ldots,w_1)$
$p_\theta(w_i|w_{i-1},\ldots,w_1)$
loss

Intrinsic Evaluation of a Language Model

$$p(w_1, w_2, \ldots, w_N) \approx p_\theta(w_1)p(w_2 \mid w_1)p_\theta(w_3 \mid w_2, w_1) \times \cdots \times p_\theta(w_N \mid w_{N-1}, w_{N-2})$$

$$= \prod_{i=1}^{N} p_\theta(w_i \mid w_{i-1}, w_{i-2})$$

- Perplexity of model p_θ given a real input sequence w = $w_1,\ldots,w_N$

$$\textbf{ppl}(p_\theta, w) = 2^{-\frac{1}{N}\sum_{i=1}^{N} log p_\theta(w_i)}$$

Examples

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;
}
```

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.
   */
```

Cell that turns on inside comments and quotes:

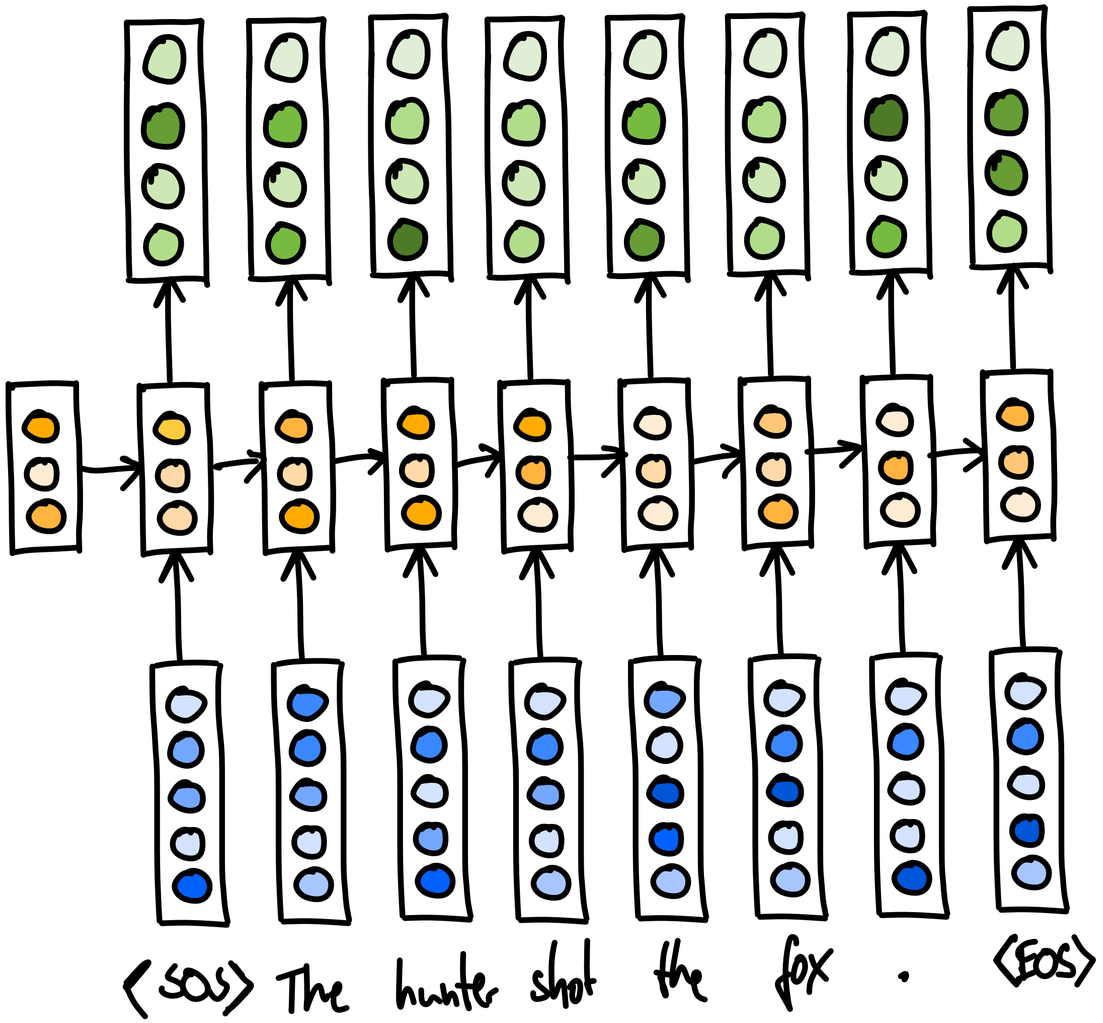
```
/* Duplicate LSM field information.  The lsm_rule is opaque, so
 * re-initialized. */
static inline int audit_dupe_lsm_field(struct audit_field *df,
           struct audit_field *sf)
{
  int ret = 0;
  char *lsm_str;
  /* our own copy of lsm_str */
  lsm_str = kstrdup(sf->lsm_str, GFP_KERNEL);
  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;
}
```

Cell that is sensitive to the depth of an expression:

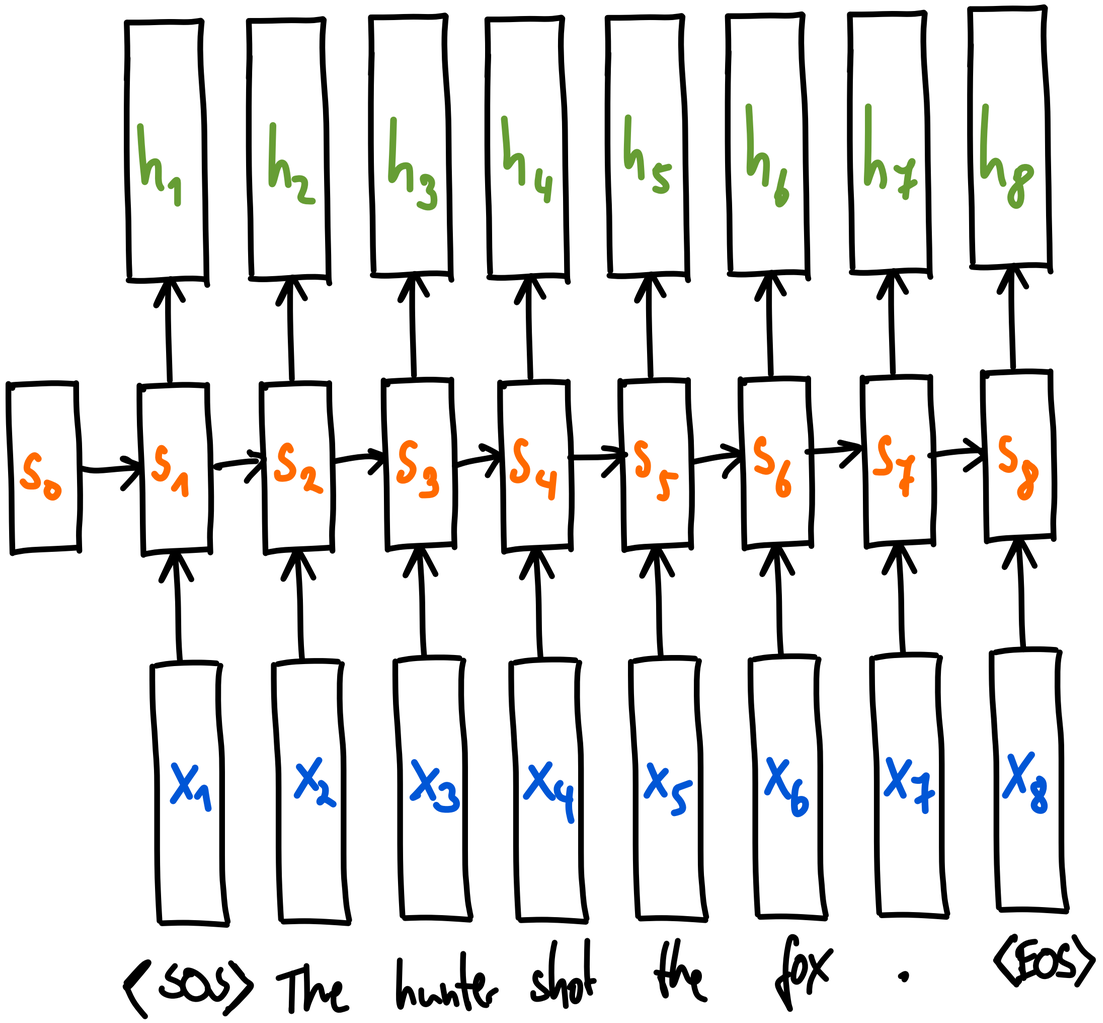
```
#ifdef CONFIG_AUDITSYSCALL
static inline int audit_match_class_bits(int class, u32 *mask)
{
  int i;
  if (classes[class]) {
    for (i = 0; i < AUDIT_BITMASK_SIZE; i++)
      if (mask[i] & classes[class][i])
        return 0;
  }
  return 1;
}
```

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

Backpropagation Through Time (BPTT)



Backpropagation Through Time (BPTT)



$$\mathbf{s}_t = \tanh(\mathbf{W}[\mathbf{x}_t; \mathbf{s}_{t-1}] + \mathbf{b})$$

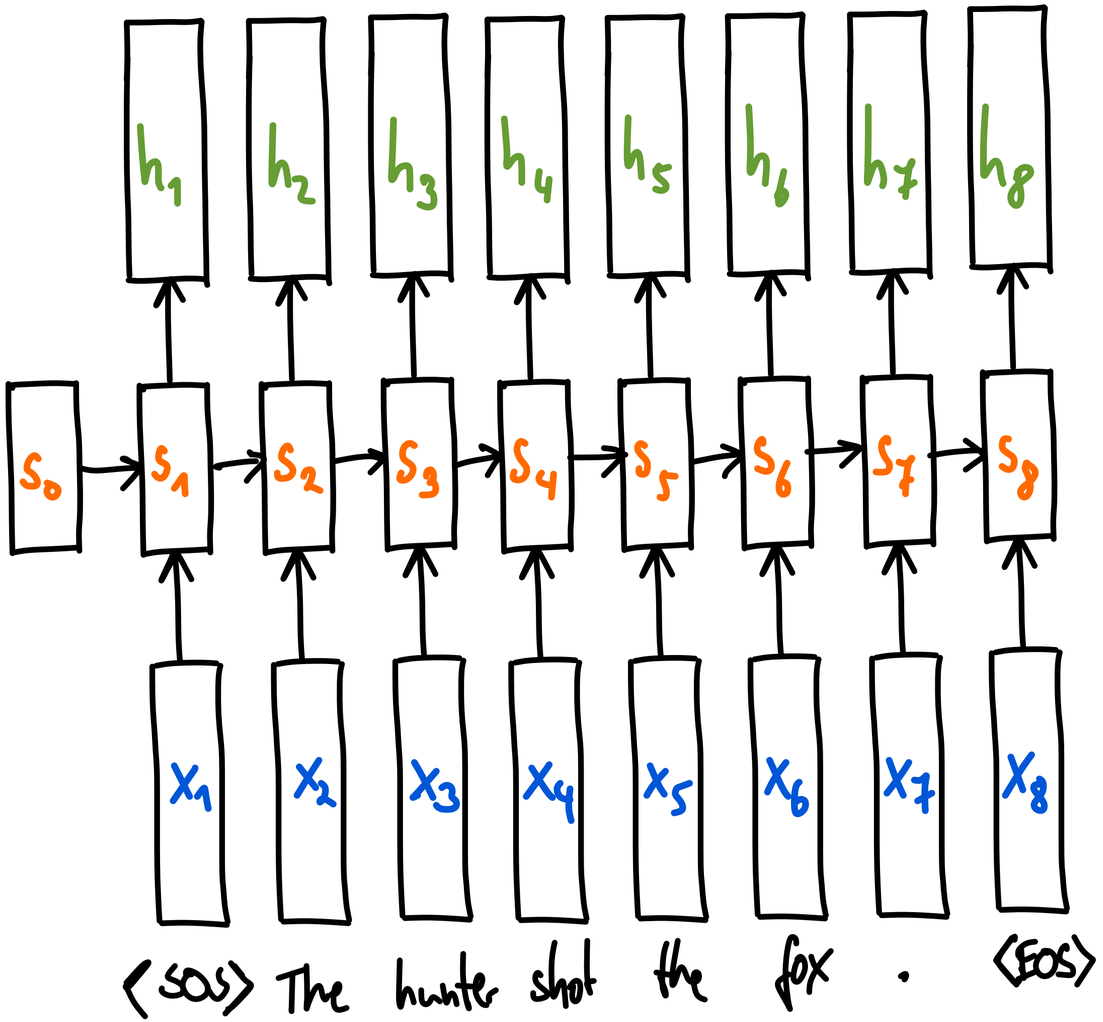
$$\mathbf{h}_t = \mathbf{s}_t$$

$$\frac{\partial L(w_N, \hat{w}_N)}{\partial \mathbf{s}_t} = \frac{\partial L(w_N, \hat{w}_N)}{\partial \mathbf{s}_N} \frac{\partial \mathbf{s}_N}{\partial \mathbf{s}_{N-1}} \cdots \frac{\partial \mathbf{s}_{t+1}}{\partial \mathbf{s}_t}$$

$$= \frac{\partial L(w_N, \hat{w}_N)}{\partial \mathbf{s}_N} \prod_{i=t+1}^{N} \frac{\partial \mathbf{s}_i}{\partial \mathbf{s}_{i-1}}$$

$$\frac{\partial L(w, \hat{w})}{\partial \mathbf{s}_t} = \sum_{j=t}^{N} \frac{\partial L(w_j, \hat{w}_j)}{\partial \mathbf{s}_j} \prod_{i=t+1}^{j} \frac{\partial \mathbf{s}_i}{\partial \mathbf{s}_{i-1}}$$

Vanishing and Exploding Gradients



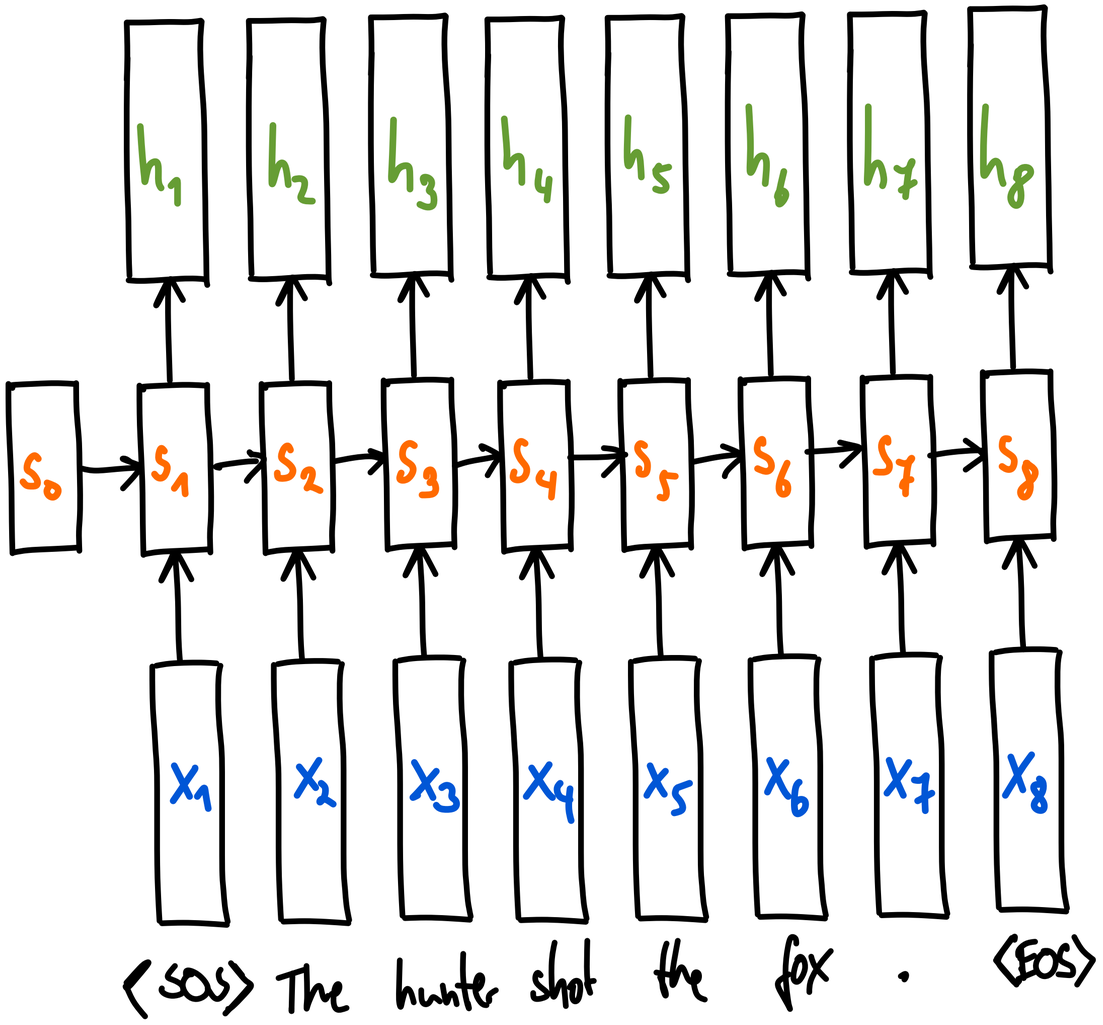
$$\mathbf{u}_t = \mathbf{W}^x \mathbf{x}_t + \mathbf{W}^s \mathbf{s}_{t-1} + \mathbf{b}$$
$$\mathbf{s}_t = \tanh(\mathbf{u}_t)$$
$$\mathbf{h}_t = \mathbf{s}_t$$

$$\frac{\partial L(w_N, \hat{w}_N)}{\partial \mathbf{s}_t} = \frac{\partial L(w_N, \hat{w}_N)}{\partial \mathbf{s}_N} \prod_{i=t+1}^{N} \frac{\partial \mathbf{s}_i}{\partial \mathbf{u}_i} \frac{\partial \mathbf{u}_i}{\partial \mathbf{s}_{i-1}}$$

$$\frac{\partial \mathbf{s}_i}{\partial \mathbf{u}_i} = \tanh'(\mathbf{u}_{i-1})^T, \quad \frac{\partial \mathbf{u}_i}{\partial \mathbf{s}_{i-1}} = \mathbf{W}^s$$

$$\frac{\partial L(w_N, \hat{w}_N)}{\partial \mathbf{s}_t} = \frac{\partial L(w_N, \hat{w}_N)}{\partial \mathbf{s}_N} \prod_{i=t+1}^{N} \tanh'(\mathbf{u}_{i-1})^T \mathbf{W}^s$$

Vanishing and Exploding Gradients



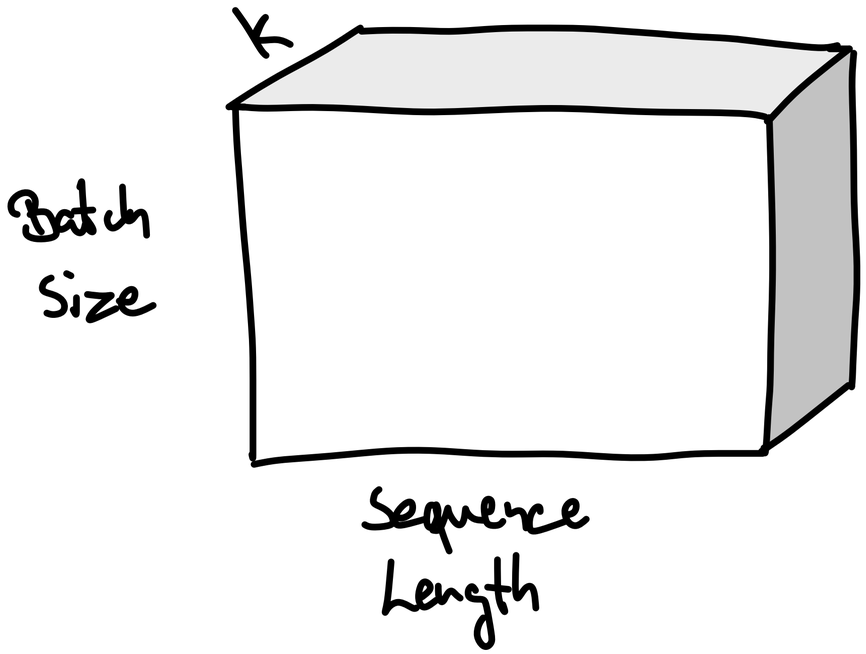
$$\mathbf{u}_t = \mathbf{W}^x \mathbf{x}_t + \mathbf{W}^s \mathbf{s}_{t-1} + \mathbf{b}$$
$$\mathbf{s}_t = \tanh(\mathbf{u}_t)$$
$$\mathbf{h}_t = \mathbf{s}_t$$

$$\frac{\partial L(w_N, \hat{w}_N)}{\partial \mathbf{s}_t} = \frac{\partial L(w_N, \hat{w}_N)}{\partial \mathbf{s}_N} \prod_{i=t+1}^{N} \tanh'(\mathbf{u}_{i-1})^T \mathbf{W}^s$$

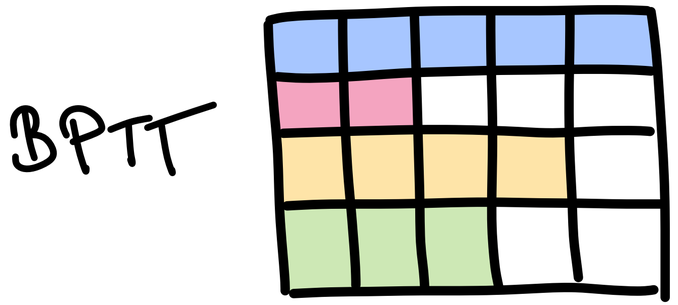
Note the repeated multiplication of $\mathbf{W}^s$
Let λ be the largest eigenvalue of $\mathbf{W}^s$

- **Gradient explodes if λ > 1**
- **Gradient vanishes if λ < 1**
- **Gradient is stable if λ = 1**

Batch Processing

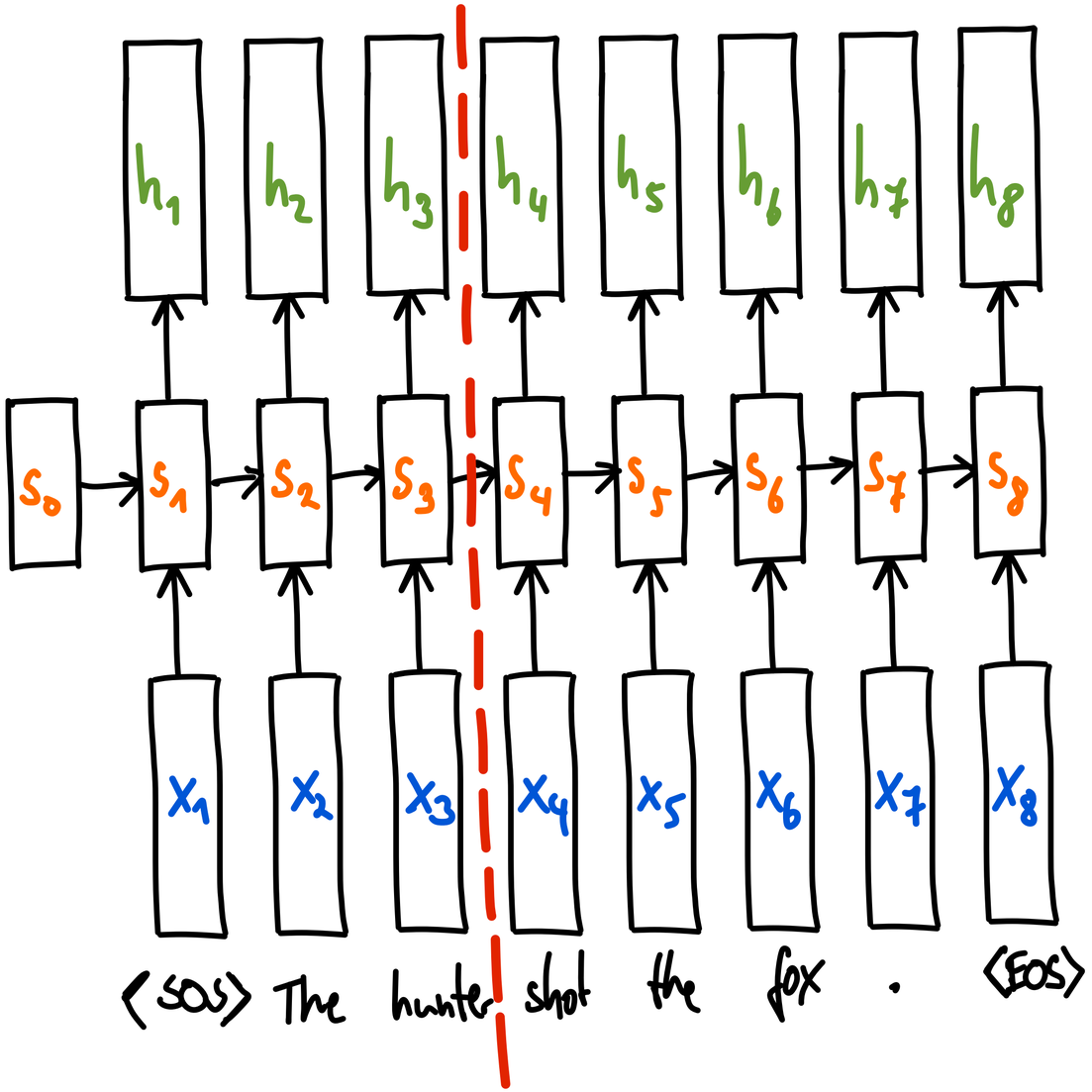


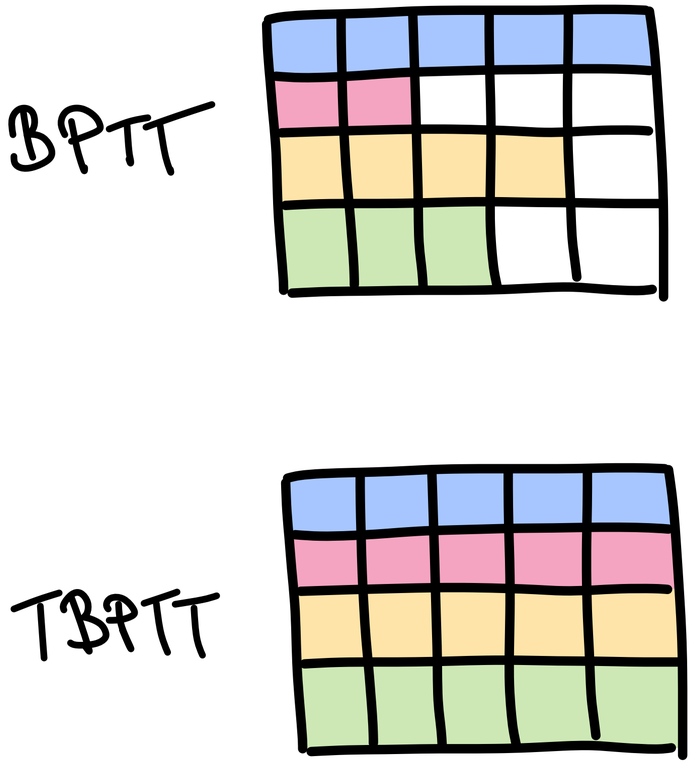
$$\mathbf{S}_t = \tanh(\mathbf{X}_t \mathbf{W}_x^T + \mathbf{S}_{t-1} \mathbf{W}_s^T + \mathbf{b})$$



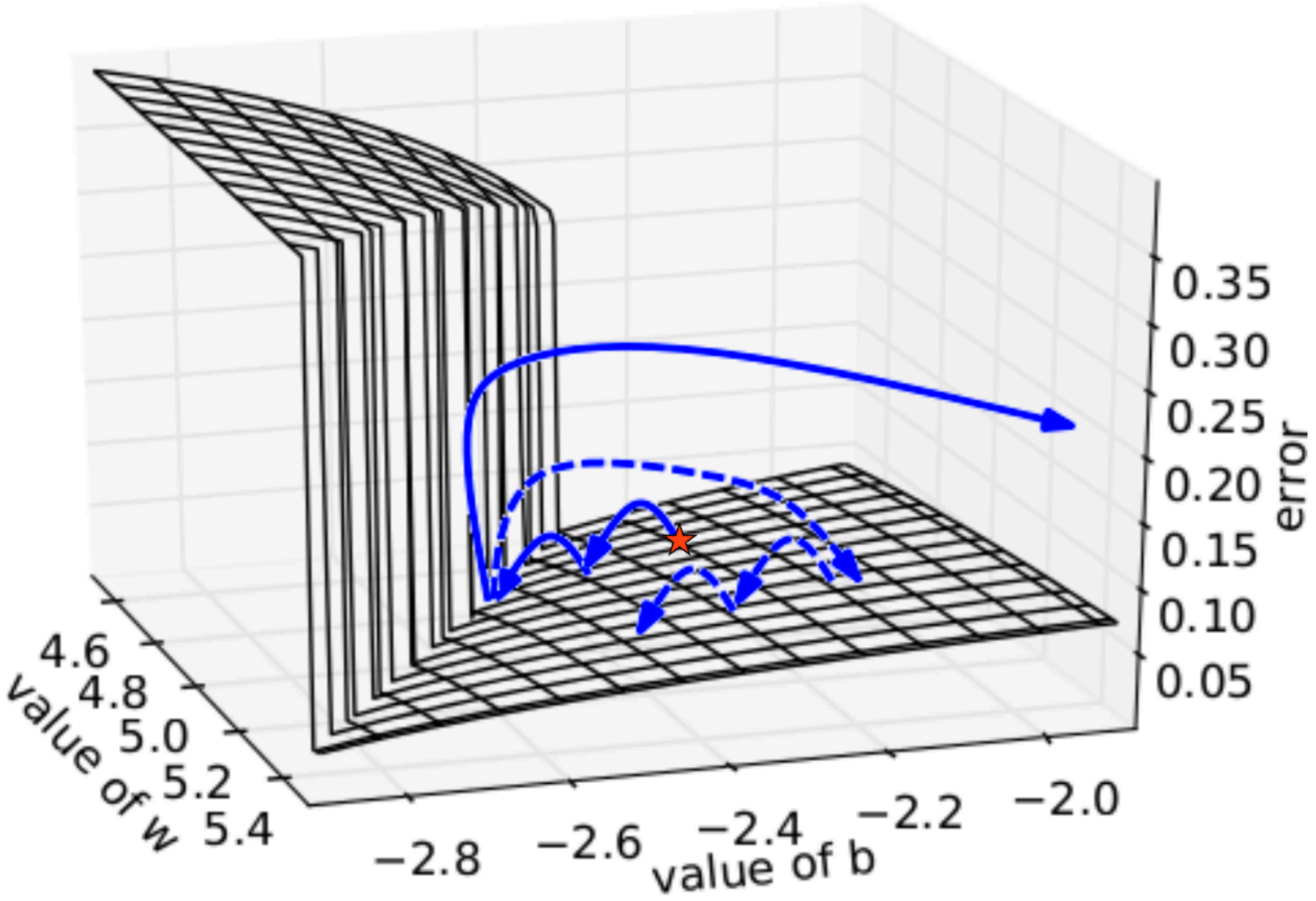

```
def f(self, X, S):
  X = torch.cat([X, S])  # -- [batch_size x 2*k]
  S = torch.tanh(torch.einsum("ij,kj->ki", [self.W, x]) + self.b)  # -- [batch_size x k]
  H = S
  return H, S
```

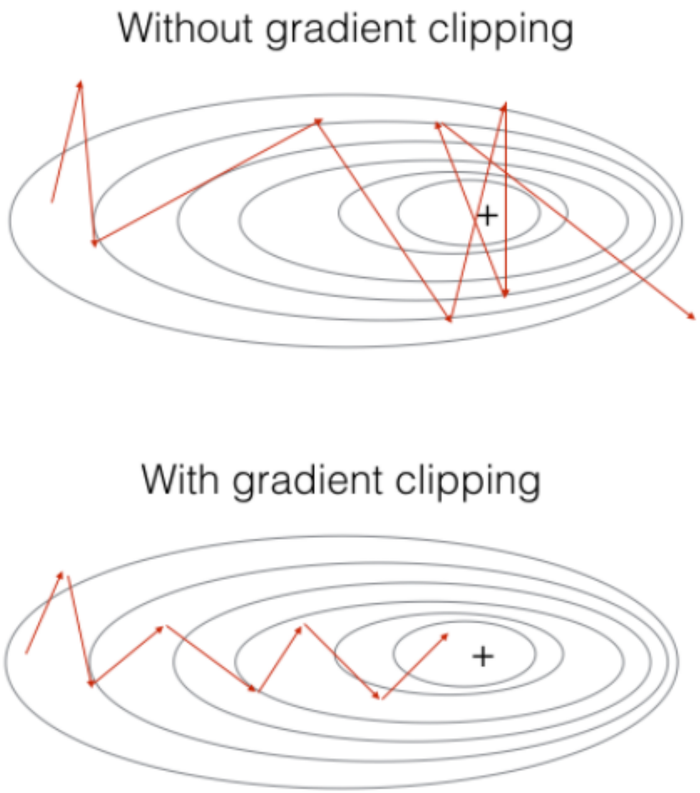
Truncated BPTT





Gradient Clipping





https://www.cnblogs.com/hezhiyao/p/8488888.html